Karlsruhe Institute of Technology

Tuning the Computational Effort: An Adaptive Accuracy-aware Approach Across System Layers

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Michael Bromberger
aus Bietigheim-Bissingen

Tag der mündlichen Prüfung:	20. Juli 2018
Erster Gutachter:	Prof. Dr. rer. nat. Wolfgang Karl
Zweiter Gutachter:	Prof. Dr. rer. nat. Martin Schulz

Abstract

Accuracy-aware computing provides a new opportunity to tune systems towards given design goals. This methodology uses quality of result (QoR) as an additional design parameter to improve the values of conventional goals. Traditionally, computer systems are designed by finding a compromise among conflicting design goals, such as performance and energy consumption. With accuracy-aware computing, such a compromise is determined by accounting for the QoR of internal and external results.

QoR as a design parameter raises the question of how accurate results must be to fulfill their planned purpose. Famous domain-specific approximation examples include the MP3 audio coding format and IEEE-754 floating-point operations. Compared to existing approximation methods, accuracy-aware computing allows for a certain degree of approximation, which is a desired design parameter, and is not a compelling necessity to solve problems like NP-hard ones. Furthermore, accuracy-aware methods are not domain-specific.

Accuracy-aware computing can be divided into *approximate computing* (AC) and *high-precision arithmetic* (HPA). Applications such as image processing, recognition, mining, and synthesis (RMS), machine learning, and data analysis can benefit from AC. In contrast to AC, HPA provides benefits for other domains such as numerics in minimizing the effect of numerical instabilities caused by computational errors.

The corresponding research question of this thesis is: How can we design and tune accuracy-aware applications running on existing computer systems? This requires a holistic approach using a unified manner to combine accuracy-aware methods. Selecting approaches from a single system layer ignores substantial potential benefits but is quite common in literature. It is inevitable that a holistic approach uses the existing accuracy-aware and conventional capabilities of computer systems. This thesis also investigates remaining potential benefits at hardware level regarding accuracy-aware computing. Therefore, an essential aspect of this work is to select, combine, and tune accuracy-aware methods from different layers (a *vertical view*) to maximize gains for an application executed on the system. However, this dramatically increases the configuration space consisting of all possible combinations and parameter settings of accuracy-aware methods. Hence, a vertical view increases the complexity of finding appropriate configurations for given design goals and constraints, which is aggravated by the fact that these configurations often must be extracted during runtime to adapt to a certain situation. The rationale ist that design-time decisions may lead to conservative gains. Therefore, the research aim of this thesis is to present a general, holistic approach to realize an accuracy-aware system. An additional main objective is to increase the scope of accuracy-aware computing, which is an important step to ensuring that accuracy-aware computing gains broad acceptance in the research community.

In this thesis, I propose an innovative adaptive accuracy-aware approach across system layers. This includes a methodology for system designers to realize accuracy-aware

systems. On this theme, a main aspect of this thesis is investigating methods for a configuration layer to deploy and tune accuracy-aware methods. My approach proposes a tuning mechanism that tunes accuracy-aware methods on a software layer, architecture layer, and hardware layer. A layer can include different layers; for instance, the software layer consists of a task, an algorithmic, and a data layer. For this thesis, I develop innovative accuracy-aware methods for a number of applications to increase the scope of accuracy-aware computing. Accuracy awareness is beneficial for these applications but has not been considered. Moreover, these methods located on three different layers can be applied to other domains than the one for which they are designed.

On the hardware layer, this thesis considers converting data before transferring it to memory, which is useful for algorithms working with floating-point values (for instance, image processing) and leads to energy savings and an overhead reduction for transfers while providing a higher accuracy compared to a native execution on a smaller data type.

On the hardware architecture layer, I propose a set of AC-based design patterns (ACPs) that help developers implement designs for accelerators. These patterns target dynamic programming algorithms that pose high challenges for porting them to an accelerator efficiently. A combination of an ACP-based accelerator and approximate data compression further improves performance.

On the software layer, it is essential to have a generally applicable software-only accuracy-aware method. Therefore, I introduce two novel methods in this thesis. One is an innovative *fuzzy memoization* technique based on *locality-sensitive hashes*. The goal of fuzzy memoization is to store function results within a table and use these results, when the function is called with similar inputs. Locality-sensitive hashes improve the identification of results within the table and thus they improve the hit rate for the table and provide a useful QoR. Furthermore, I employ *contract algorithms* in a new way to enable a clear, fine-grain, and predictable correlation between required budget and achievable QoR. When we vary the budget of these tasks, we also adapt the resulting QoR. I present several methods to improve the performance of these algorithms and provide best practices to transfer tasks into contract-based tasks.

On the configuration layer, it is important to design an approach that allows for controlling the degree of internal and external QoR either by maximizing the QoR for the available budget or by minimizing the required budget while achieving the desired QoR. For the budget, this approach can use the execution time, energy consumption, and a combination of both based on an energy delay product metric. The last option makes it possible to deal with multiple objectives: execution time, consumed energy, and QoR.

Integrating accuracy-aware methods into a system results in a large configuration space. In this space, efficient parameter settings for the various methods must be extracted during runtime to handle different inputs, system states, and constraints. To reduce this effort, I present an approach that is divided into a design-time step and a runtime mechanism. Additionally, I provide a hierarchical approach to further reduce the number of considered configurations. Therefore, I consider suitable configurations per task and combine this knowledge to extract global configurations. For this, my approach generates *merged performance profiles* by accounting for integrated accuracy-aware methods and other conventional tuning possibilities of the system such as dynamic voltage and frequency scaling (DVFS) or exploiting more cores. Using the local performance profiles, my approach determines (near-)Pareto-optimal global configurations during design time, which are exploited during runtime. To integrate input awareness, my approach uses the policy to adapt the approximation degree on a per-task level dynamically.

To conclude, accuracy-aware computing is a novel and promising methodology to tune a system towards design goals. During design time, the methods or parameter settings that best suit an application in a certain situation are often unpredictable. The proposed adaptive accuracy-aware approach across system layers presents an innovative methodology to realize dynamic accuracy-aware computer systems. This approach reacts of changing situations during runtime by tuning the global configuration of the system. These configurations offer better value than conventional application-dependent tuning strategies for traditional design goals while having the same or less design effort. The present thesis has the following novel contributions; it:

- Introduces a conversion unit that reduces the overhead of floating-point data transfers. This unit exploits the memory more efficiently while providing a higher QoR than performing an algorithm using precision scaling. Integrating this unit into a RISC-V processor reduces the total dynamic energy consumption of a 2D fast Fourier transformation by 2.5×.

- Proposes a set of AC-based design patterns that leads to the efficient use of heterogeneous units. These patterns reduce resource use and improve the performance of kernels running on a heterogeneous hardware unit. Applying it to time series analysis improves the classification time by up to 29.4× compared to complex similarity measures (complexity class: $O(n^2)$) while having a higher QoR than measures lying in $O(n)$.

- Outlines a general accuracy-aware software-only method based on contract algorithms that can be applied to a wide variety of applications. Furthermore, it presents a set of best practices to convert a task to a contract-based task. In general, this method outperforms other accuracy-aware methods on the software layer. For instance, this method only requires 30% of the baseline execution time to provide good quality (signal-to-noise ratio of 20 db) for debayering.

- Provides a novel fuzzy memoization library. This library provides routines to introduce fuzzy memoization into applications and different fuzzy memoization methods. For instance, the locality-sensitive hash method offers a high hit rate and high visual quality for a JPEG benchmark.

- Investigates an adaptive tuning approach for applications that require HPA. While the conclusion is that HPA is useful for certain applications such as the Lanczos approach, numeric algorithms rely on domain expertise to provide better solutions. Hence, an adaptive tuning approach for HPA cannot be usefully integrated into the adaptive accuracy-aware approach proposed in this thesis.
- Combines accuracy-aware methods at different layers (a vertical view) for a novel domain. Using these methods, the numerical Jacobi algorithm is accelerated by a factor of 6 compared to a 32-threaded version while allowing a QoR of 90%.
- Proposes an innovative execution ordering algorithm for classifying objects within an image. This algorithm orders the execution order of classification tasks so the total information gain is highest after executing the next step of the task leading the order. This algorithm makes it possible to stop the execution at any time and receive an (approximate) classification result for each input.
- Provides an innovative adaptive accuracy-aware approach across system layers, designed to
  - combine and fine-tune accuracy-aware methods including the methods designed for this thesis during runtime;
  - has a vertical view;
  - account for multiple conflicting design goals during runtime;
  - provide input awareness regarding QoR during runtime;
  - exploit conventional methods such as dynamic voltage and frequency scaling and parallelization;
  - allow a user to continue execution if the QoR is not sufficient;
  - fine-tune the execution of several accuracy-aware applications; and
  - be able to be applied to a real-time sensor-based sorting application. This makes it possible to meet all deadlines, even under tight constraints and accelerates the conventional application by a factor of 4 while keeping the same QoR.

Zusammenfassung

Das Konzept der genauigkeitsbewussten Berechnung (engl.: accuracy-aware computing) stellt eine neuartige Lösung dar, um Systeme bezüglich vorgegebener Optimierungsziele einstellen zu können. Diese Vorgehensweise betrachtet die Qualität der Ergebnisse (engl.: quality of result, QoR) als einen weiteren Entwurfsparameter. Während des herkömmlichen Entwurfs von Computersystemen muss ein Kompromiss zwischen sich ausschließenden Entwurfszielen gefunden werden. Zu diesen Zielen zählen die Leistungsfähigkeit und der Energieverbrauch. Unter Zuhilfenahme der genauigkeitsbewussten Berechnung wird für einen Kompromiss auch die benötigte Qualität der internen und externen Ergebnisse berücksichtigt. Somit stellt sich beim Entwurf die Frage, wie hoch die Qualität der Ergebnisse sein muss, um eine gewünschte Aufgabe erfüllen zu können.

Bekannte domänenspezische Methoden sind die MP3-Audiokodierung und der IEEE-754 Standard für Gleitkommazahlen. Der Hauptunterschied zwischen existierenden Approximationsmethoden und dem Konzept der genauigkeitsbewussten Berechnung ist die ausdrückliche Berücksichtigung des Approximationsgrades auf allen Ebenen des Systems. Darüber hinaus sind genauigkeitsbewusste Methoden nicht auf einzelne Anwendungsbereiche beschränkt. Das Konzept der genauigkeitsbewussten Berechnung kann in die Bereiche approximative (engl.: approximate computing) und exaktere Berechnung (engl.: high-precision arithmetic) eingeteilt werden. Anwendungen aus dem Bereich Bildverarbeitung, Maschinelles Lernen oder der Datenanalyse profitieren von approximativen Berechnungen. Gewisse Anwendungen aus dem Bereich des wissenschaftlichen Rechnens benötigen hingegen eine erhöhte Präzision, um den Einfluss von Berechnungsfehlern zu minimieren.

Die zentrale Frage der vorliegenden Dissertation ist: Wie können genauigkeitsbewusste Systeme entworfen und entsprechend zur Laufzeit eingestellt werden? Eine geeignete Lösung benötigt einen allumfassenden Ansatz, der eine Möglichkeit darstellt, um genauigkeitsbewusste Methoden miteinander kombinieren zu können. Die Berücksichtigung von nur einer einzigen Systemebene lässt vorhandenes Potenzial von Anwendungen, die Approximation tolerieren können, ungenützt. Der zu entwickelnde Ansatz nützt idealerweise die Fähigkeiten der verfügbaren genauigkeitsbewussten und konventionellen Methoden der Hardware. Zudem wird in dieser Dissertation verbleibendes Potenzial der genauigkeitsbewussten Berechnung auf Hardwareebene untersucht und ausgenutzt. Ein bedeutender Aspekt für diese Dissertation ist die Auswahl, die Kombination und die richtige Einstellung von Methoden unterschiedlicher Systemebenen, um den höchsten Gewinn für eine Anwendung bezogen auf die Optimierungsziele zu erhalten. Dies stellt eine vertikale Sichtweise dar. Jedoch bedeutet diese Sichtweise eine deutlich erhöhte Komplexität für die Identifikation geeigneter Konfigurationen bezogen auf vorhandene Entwurfsziele und Zwangsbedingungen. Erschwerend kommt hinzu, dass idealerweise

Konfigurationen zur Laufzeit gefunden werden sollten, insbesondere, wenn das Systemverhalten nicht zur Entwurfszeit genau bestimmt werden kann. Daher ist das Forschungsziel dieser Dissertation der Entwurf eines generellen und allumfassenden Ansatzes für die Realisierung eines genauigkeitsbewussten Systems. Ein weiteres Ziel ist die Ausdehnung des Anwendungsbereiches des Konzepts der genauigkeitsbewussten Berechnung. Dies stellt einen essenziellen Schritt dar, um die Akzeptanz dieses Konzeptes in der Forschungscommunity zu vergrößern.

In der vorliegenden Dissertation wird ein neuartiger, innovativer und adaptiver genauigkeitsbewusster Ansatz präsentiert, der über Systemebenen hinweg agiert. Dies beinhaltet eine innovative Vorgehensweise für Systementwickler für die Realisierung von genauigkeitsbewussten Systemen. Hierfür ist ein Hauptbestandteil die Erforschung von Lösungen, die es ermöglichen genauigkeitsbewusste Methoden anhand bestehender Anforderungen entsprechend zu nutzen und einzustellen. Die gefundenen Lösungen werden in eine Konfigurationsschicht (engl.: configuration layer) integriert. Der entwickelte Ansatz stellt einen Tuningmechanismus bereit, der genauigkeitsbewusste Methoden für eine bestimmte Anwendung entsprechend einstellt. Hierfür werden Methoden auf der Softwareebene, der Architekturebene, und der Hardwareebene betrachtet. Zu beachten ist, dass eine Ebene wiederum unterteilt sein kann. Beispielsweise besteht die Softwareebene aus einer Task-, Algorithmen- und Datenebene. Zudem werden für diese Dissertation innovative genauigkeitsbewusste Methoden für neue Anwendungsgebiete entwickelt. Es zeigt sich, dass für Anwendungen dieser neuen Gebiete der Einsatz des Konzepts der genauigkeitsbewussten Berechnung nützlich ist. Diese entwickelten Methoden erlauben auch die Anwendung für andere Domänen.

Auf der Hardwareebene wird eine Methode eingeführt, die Gleitkommazahlen vor dem Transfer in einen Speicher so approximiert, dass der Transferaufwand und der benötigte Speicherplatz reduziert werden. Dadurch kann die benötigte Energie der Ausführung verringert werden. Des Weiteren wird eine höhere Genauigkeit der Ergebnisse gegenüber einer nativen Ausführung mit einem kleineren Datentyp erreicht. Diese Methode ist insbesondere für gewisse Bildverarbeitungsalgorithmen sinnvoll.

Auf der Hardware-Architekturebene werden Entwurfsmuster identifiziert, mithilfe derer effizientere Beschleuniger realisierbar sind. Algorithmen basierend auf der dynamischen Programmierung lassen sich dadurch effizient auf Hardwarearchitekturen wie Field-programmable Gate Arrays (FPGAs) portieren. Diese Methode lässt sich mit einer approximativen Datenkompression kombinieren, um die Leistungsfähigkeit weiter zu steigern.

Auf der Softwareebene ist es essenziell eine generell anwendbare genauigkeitsbewusste Methode zu haben. Daher werden zwei neue Methoden entwickelt. Zum einen eine Methode für die unscharfe Memoisation, die das Konzept der *Locality-sensitive Hashes* ausnützt. Die unscharfe Memoisation nützt zwischengespeicherte Ergebnisse von Funktionen aus, die mit ähnlichen Eingaben berechnet wurden. Mittels Locality-sensitive Hashes lässt sich die Identifikation ähnlicher Ergebnisse in der Memoisationstabelle ver-

bessern. Somit wird neben der erhöhten Trefferrate von Ergebnissen auch der Qualitätsverlust (QoR) verbessert. Zum anderen wird eine Methode eingeführt, die das Konzept der *Vertragsalgorithmen* in einer neuen Art und Weise ausnutzt. Diese Methode erlaubt es, einen feingranularen Zusammenhang zwischen Berechnungsaufwand und erreichbarem Qualitätsverlust (QoR) zu ermitteln. Somit kann mittels eines vorgegebenen, erlaubten Berechnungsaufwand der Qualitätsverlust eingestellt werden. Es wird eine Reihe von Verbesserungen für das Konzept der Vertragsalgorithmen gezeigt. Zudem werden bewährte Verfahren für die Realisierung von vertragsbasierten Tasks vorgestellt.

Für die Konfigurationsschicht ist es essenziell, einen Ansatz zu entwickeln, der mit folgenden Szenarien umgehen kann. Das erste Szenario bedingt die Maximierung des QoR bei vorgegebenem Budget. Zweitens wird das benötigte Budget minimiert, sodass ein gewünschter QoR erreicht wird. Der vorgestellte Ansatz erlaubt es, die Ausführungszeit, die verbrauchte Energie oder eine Kombination aus beidem als Budget zu verwenden. Letzteres wird über die Verwendung eines Energie-Verzögerungs-Produkts (engl.: energy delay product) realisiert. Somit können mehrere Entwurfsziele, Ausführungszeit, verbrauchte Energie und QoR, gleichzeitig berücksichtigt werden.

Wie bereits erwähnt führt die Integration mehrerer genauigkeitsbewusster Methoden, die jeweils (mehrere) Parameter besitzen, zu einem großen Konfigurationsraum. Es ist zentral, gute Konfigurationen zur Laufzeit in diesem Raum zu finden, sodass auf verschiedene Eingaben, Systemzustände und Randbedingungen geeignet reagiert werden kann. Um diese Komplexität deutlich zu reduzieren, ist der vorgestellte Ansatz aufgeteilt in einen Entwurfsschritt und in eine Logik, die zur Laufzeit die Steuerung übernimmt. Des Weiteren wird ein hierarchischer Ansatz entwickelt, der eine weitere Reduktion der zu betrachtenden Konfigurationen erreicht. Hierfür werden sinnvolle (lokale) Konfigurationen auf Taskebene ermittelt mit denen letztlich globale Konfigurationen ermittelt werden. Zu diesem Zweck werden sogenannte fusionierte Leistungsprofile (engl.: merged performance profiles) für eine Task erstellt. Für die Generierung dieser Profile werden sowohl genauigkeitsbewusste Methoden als auch konventionelle Methoden wie Parallelisierung und dynamische Spannungs- und Frequenzskalierung mit einbezogen. Ein Punkt in einem Profil repräsentiert eine (nahezu) Paretooptimale Konfiguration bezüglich dem QoR und dem benötigten Budget. Diese lokalen Profile lassen sich ausnützen, um globale, (nahezu) Paretooptimale Konfigurationen zur Entwurfszeit zu ermitteln. Die ermittelten globalen Konfigurationen können sinnvoll zur Laufzeit eingesetzt werden. Eine Eingabesensitivität wird dadurch erreicht, dass jeder Task den Approximationsgrad selbstständig anhand von QoR-Vorgaben für die aktuelle Eingabe einstellen kann.

Zusammenfassend ist zu erwähnen, dass das innovative Konzept der genauigkeitsbewussten Berechnung eine vielversprechende Möglichkeit darstellt, Systeme hinsichtlich Entwurfszielen einzustellen. Häufig ist zur Entwurfszeit nicht absehbar, welche Methoden und Parametereinstellungen für eine bestimmte Situation am geeignetsten sind. Daher bietet der in dieser Arbeit entwickelte adaptive, genauigkeitsbewusste Ansatz über Systemebenen hinweg eine neue Vorgehensweise, um dynamische, genauigkeitsbewusste Systeme zu realisieren. Mithilfe dieser innovativen Vorgehensweise können sinnvolle globale Konfigurationen zur Laufzeit ermittelt werden. Die ermittelten Konfigurationen führen zu besseren Werten für herkömmliche Entwurfsziele verglichen mit konventionellen, zum Teil anwendungsabhängigen Tuningstrategien. Dabei ist der Entwurfsaufwand gleich oder sogar geringer. Die vorliegende Dissertation beinhaltet die folgenden neuen wissenschaftlichen Beiträge:

- Den Entwurf einer neuartigen *Conversion Unit*, die den Aufwand für den Datentransfer von Gleitkommawerten merklich reduziert. Dadurch wird erreicht, dass der vorhandene Speicherplatz der höchsten Hierarchieebene (in der Regel L1 Cache) effizienter genutzt wird. Trotz der Reduzierung der Genauigkeit von Werten bei der Übertragung in diese Hierarchieebene wird eine höhere Genauigkeit verglichen mit der Nutzung eines kleineren Datentyps für einen Algorithmus erreicht. Durch die Integration der *Conversion Unit* in einen RISC-V-Prozessor wird eine Reduktion der dynamischen Energieaufnahme von 2,5 für eine schnelle 2D-Fouriertransformation erreicht.

- Die Identifikation von Entwurfsmustern zur Realisierung von effizienten Beschleunigern. Mithilfe dieser Muster kann ein Hardwareentwickler approximative Strukturen einführen, die eine Erhöhung der Leistungsfähigkeit mit sich bringen. Die Anwendung der Muster führt zu einem geringen Ressourcenverbrauch und einer höheren Leistung für Berechnungskernels, die auf heterogenen Einheiten ausgeführt werden. Beispielsweise führt die Anwendung dieser Entwurfsmuster zu einer Beschleunigung um den Faktor 29,4 für die Klassifikation von Zeitreihen. Diese Beschleunigung wird gegenüber einer Klassifikation mithilfe komplexer Distanzmaße erreicht. Neben dieser massiven Beschleunigung wird eine Qualitätssteigerung der Klassifikationsgenauigkeit gegenüber einfachen Distanzmaßen erreicht.

- Eine innovative Methode wird entwickelt, die einen generellen Einsatz von genauigkeitsbewussten Berechnungen auf Softwareebene ermöglicht. Diese Methode basiert auf dem Konzept der Vertragsalgorithmen. Es wird eine Reihe bewährter Verfahren zur Realisierung von vertragsbasierten Tasks vorgestellt. Im Allgemeinen bieten diese neuartigen Tasks eine bessere Lösung als die bekannten genauigkeitsbewussten Verfahren aus der Literatur. Dieser Lösungsansatz benötigt beispielsweise nur ca. 30 % der Ausführungszeit eines konservativen Algorithmus, um eine gute Qualität (Signal-Rausch-Verhältnis von 20 db) für den Benchmark *debayering* zu erreichen.

- Eine neuartige Softwarebibliothek, um den Einsatz der unscharfen Memoisation für Entwickler zu ermöglichen. Die Bibliothek stellt Softwareroutinen zur Verfügung, die eine Integration der unscharfen Memoisation in bestehende Anwendungen ermöglicht. Hierfür kann zwischen unterschiedlichen Memoisationmethoden gewählt werden. Die entwickelte Methode basierend auf *locality-senstive hashes* liefert eine gute visuelle Qualität auf den Ausgaben des JPEG Benchmarks. Darüber hinaus wird mit dieser Methode der Grad der Wiederverwendung von Ergebnisse in der Memoisationtabelle erhöht.

- Die Untersuchung eines adaptiven Ansatzes für die Einstellung der Präzision von Datentypen für Anwendungen, bei denen eine Erhöhung der Präzision genauere Ergebnisse erbringt. Die Untersuchung führt zum Ergebnis, dass eine Erhöhung der Präzision für das Lanczos-Verfahren sinnvoll ist. Jedoch wird Expertenwissen benötigt, um den hohen Aufwand, der durch die erhöhte Präzision eingeführt wird, deutlich zu reduzieren. Daher wird eine Erhöhung der Präzision nicht weiter für den generellen, adaptiven genauigkeitsbewussten Ansatz, der für diese Dissertation entwickelt wird, berücksichtigt.

- Eine Evaluation über das Potenzial der Kombination genauigkeitsbewusster Methoden, die auf unterschiedlichen Schichten liegen. Diese Evaluation zeigt, dass diese Kombination von Methoden nützlich ist. Insbesondere wurde dieses Potenzial auf einer Domäne betrachtet, auf der das Konzept der genauigkeitsbewussten Berechnungen bisher nicht gründlich untersucht wurde. Mithilfe der Kombination von Methoden lässt sich das Jacobi-Verfahren um den Faktor 6, gegenüber einer konventionellen Methode, die auf 32 Kerne ausgeführt wird, beschleunigen, wenn eine zehnprozentiger Qualitätsverlust akzeptiert wird.

- Ein innovativer Algorithmus zur Bestimmung der Ausführungsreihenfolge von Objektklassifikationen. Der Algorithmus ordnet die Ausführungsreihenfolge, so dass der nächste Schritt das globale Wissen bezüglich der Klassifikationen am meisten verbessert. Dieser Algorithmus ermöglicht es, dass zu jeder Zeit die Ausführung unterbrochen werden kann (anytime algorithm) und für jede Klassifikation das bestmögliche Wissen, das bis zu diesem Zeitpunkt möglich war, über die Art des Objekts zurück gibt. Somit kann der Algorithmus auch bei variierenden Echtzeitbedingungen sinnvoll eingesetzt werden.

- Der Entwurf eines innovativen, neuartigen und adaptiven genauigkeitsbewussten Ansatzes, der über Systemebenen hinweg agiert. Dieser Ansatz
  - kombiniert genauigkeitsbewusste Methoden unterschiedlicher Ebenen miteinander und stellt die dazugehörigen Parameter sinnvoll ein,
  - nutzt das Wissen aus, das vorherige Beiträge dieser Dissertation erzielten und berücksichtigt die entwickelten genauigkeitsbewussten Methoden beim Tuning,
  - berücksichtigt mehrere, sich gegenseitig ausschließende Optimierungsziele zur Laufzeit,

- berücksichtigt aktuelle Eingaben für die Steuerung des QoR zur Laufzeit,
- schöpft das Potenzial konventioneller Methoden wie DVFS und Parallelisierung, aus,
- erlaubt Nutzern die Fortsetzung der Ausführung, um den QoR zu verbessern,
- stimmt die Ausführung mehrerer genauigkeitsbewusster Anwendungen gemeinsam ab,
- kann für eine echtzeitbasierte und sensorgestützte Sortieranwendung verwendet werden. Durch den adaptiven Ansatz können feste Echtzeitbedingungen eingehalten werden, auch wenn diese eng gefasst sind und eine hohe Auslastung im System vorliegt. Zudem erreicht der Ansatz eine vierfache Beschleunigung der konventionellen Anwendung, ohne den QoR zu beeinflussen.

Acknowledgments

Doing a PhD is a long way full of ups and downs. It is not successfully viable without the great support of people and thus I want to express my heartfelt thanks to these people.

First and foremost, I am profoundly grateful to my advisor, Prof. Dr. Wolfgang Karl, for his valuable guidance, support, and useful critiques. He provided me an enjoyable working place. Research is not only identifying problems and finding new solutions in a “dark room”, it is the mediation of research results. Therefore, he taught me how to present my research results to other researchers in the right way. Heartfelt thanks go to my co-referee, Prof. Dr. Martin Schulz, for agreeing to review my thesis without hesitation and providing me valuable feedback. I would like to express my sincere thanks to Prof. Dr. Vincent Heuveline and Klaus Tschira. They made it possible to provide me with a scholarship from the Heidelberg Institute of Theoretical Studies (HITS). Prof. Dr. Vincent Heuveline gave me valuable comments and feedback.

My research would have been impossible without the aid and support of my current and former colleagues at the Chair of Computer Architecture and Parallel Processing (in alphabetical order): Thomas Becker, Markus Hoffmann, Mario Kicherer, Oliver Mattes, Fabian Nowak, and Anas Toma. Furthermore, I would like to thank the people from the Data Mining and Uncertainty Group, the Chair of Embedded Systems, and Chair of Dependable Nano Computing. Especially, I want to thank Dr. Lars Bauer and Tanfer Alan for the valuable feedback and fruitful discussions during lunch and coffee breaks. I express my heartfelt thanks to Prof. Dr. Thomas Längle and Georg Maier for our fruitful meetings about approximate computing.

What is research without inspiration? Therefore, special thanks go to Ravi Nair for his keynote about approximate computing. This keynote was the main driving force for the topic of my work. I am profoundly grateful to HiPEAC and Movidus for giving me the opportunity to do an internship. It was a special time in my live and I met wonderful people that supported my research and had a significant contribution to the success of this work. Heartfelt thanks go representatively to David Moloney, Fergal Connor, Colm Keane, and Francesco Paci. Moreover, I would like to thank the people from Myestro, namely Jens Schick, and from BIOQUANT, namely Karl Rohr. It was a pleasure for me to work with them and transferring our results into publications successfully. My sincere thanks to my students whom I supervised in the scope of this thesis. Additionally, I would like to thank Martin Buchty and the secretaries for assisting me with my regular duties.

I am extremely thankful to my parents (Else and Kurt) for their support and unlimited love. They always believed in me and gave me the certainty to complete my PhD journey. They have been a permanent fount of love and strength since ever. My research would have been impossible without the support and the welcome diversions of my friends. Doing things that are not related to research helped me survive the PhD adventure. I would like to thank the parents of my partner for their frequent encouraging words.

Finally and without a doubt, this would have been hardly possible without the support of my girlfriend, Rebecca, who was exceptional patient during my work. I am exceptionally and truly grateful to her.

Contents

List of Own Relevant Publications

[BHR18] Michael Bromberger, Markus Hoffmann, and Robin Rehrmann. “Do Iterative Solvers Benefit From Approximate Computing? An Evaluation Study Considering Orthogonal Approximation Methods”. In: *International Conference on Architecture of Computing Systems (ARCS)*. 2018.

[MBL$^+$16] Georg Maier, Michael Bromberger, et al. “High-throughput sensor-based sorting via approximate computing”. In: *Forum Bildverarbeitung 2016*. 2016, p. 99.

[BNK15] Michael Bromberger, Fabian Nowak, and Wolfgang Karl. “Combined hardware-software multi-parallel prefiltering on the Convey HC-1 for fast homology detection”. In: *Parallel Computing* 42 (2015), pp. 4–17.

[NBS$^+$13] Fabian Nowak, Michael Bromberger, et al. “Multi-parallel Prefiltering on the Convey HC-1 for Supporting Homology Detection”. In: *Proceedings of the 20th European MPI Users’ Group Meeting*. (EuroMPI). International Workshop on Parallelism in Bioinformatics. Sept. 2013, pp. 169–174.

[BN13] Michael Bromberger and Fabian Nowak. “Parallel Prefiltering for Accelerating HHblits on the Convey HC-1”. In: *Mitteilungen*. Vol. 30. Gesellschaft für Informatik E.V., Parallel-Algorithmen und Rechnerstrukturen (PARS), 2013, pp. 47–57.

[BSS$^+$17] Michael Bromberger, Michael Scharrer, et al. “OPTICAR: Design Aspects and Challenges of Stereo Vision for Autonomous Driving”. In: *ACM Chapters Computer Science in Cars Symposium*. 2017. (Visited on 03/21/2018).

[BK15] Michael Bromberger and Wolfgang Karl. “Integration of Approximate Computing in today’s Systems: From Hardware to Algorithmic Level”. In: *Workshop on Approximate Computing (AC)*. Oct. 2015.

[B18] Michael Bromberger. “Tuning the Computational Effort: An Adaptive Accuracy-aware Approach Across System Layers”. In: *Organic Computing: Doctoral Dissertation Colloquium*. (submitted). 2018.

[BHK16] Michael Bromberger, Vincent Heuveline, and Wolfgang Karl. “Reducing Energy Consumption of Data Transfers Using Runtime Data Type Conversion”. In: *International Conference on Architecture of Computing Systems (ARCS)*. 2016, pp. 239–250.

[BBB+16] Michael Bromberger, Pascal Bastian, et al. “FPGA-accelerated Richardson-Lucy deconvolution for 3D image data”. In: *IEEE International Symposium on Biomedical Imaging (ISBI)*. 2016, pp. 132–135.

[B14] Michael Bromberger. “Supporting Big Data Applications Using Hybrid Systems”. In: *Tagungsband Informatiktage, GI-Edition Lecture Notes in Informatics (LNI)*. Mar. 2014.

[BKH15] Michael Bromberger, Wolfgang Karl, and Vincent Heuveline. “Exploiting Approximate Computing Methods in FPGAs to Accelerate Stereo Correspondence Algorithms”. In: *Workshop On Approximate Computing (WAPCO) in conjunction with HiPEAC conference*. Jan. 19, 2015. (Visited on 03/20/2018).

[BES+17] Michael Bromberger, Steffen Ehrle, et al. “Design Space Exploration Including Approximate Computing for OpenCL-based Stereo Vision Hardware”. In: *Mitteilungen*. Gesellschaft für Informatik E.V., Parallel-Algorithmen und Rechnerstrukturen (PARS), 2017.

[BSE+17] Michael Bromberger, Michael Scharrer, et al. “OpenCL-Based 6D-Vision on Heterogeneous System on Chips”. In: *International Conference on Architecture of Computing Systems (ARCS)*. 2017, pp. 33–46.

[BK18] Michael Bromberger and Wolfgang Karl. “A Transparent View On Approximate Computing Methods For Tuning Applications”. In: *Approximate and Transprecision Computing on Emerging Technologies Workshop (ATCET) at ISC High Performance*. (to appear). 2018.

[BKM+18] Michael Bromberger, Wolfgang Karl, et al. “CADIS: Contract-based Approximate Computing Design of Systems”. In: *ACM Transactions on Architecture and Code Optimization (TACO)* (2018). (submitted).

[BHH17] Michael Bromberger, Markus Hoffmann, and Andreas Hampp. “Evaluating the Influence of Data Type Precision On Numerical Algorithms”. In: *Mitteilungen*. Gesellschaft für Informatik E.V., Parallel-Algorithmen und Rechnerstrukturen (PARS), 2017.

Other Publications

[NBK14] Fabian Nowak, Michael Bromberger, and Wolfgang Karl. “An Architecture Framework for Porting Applications to FPGAs”. In: *The 11th Workshop on Parallel Systems and Algorithms (PASA)*. Feb. 2014.

List of Figures

List of Tables

List of Algorithms

Part I

Introduction and Approach

CHAPTER

ONE

INTRODUCTION

According to philosopher and logician Carveth Read *"It is better to be vaguely right than exactly wrong"*. This quote is often misattributed to the economist John Maynard Keynes. Similar to the existing uncertainty about the origin of this quote, there is uncertainty what future computer systems and software will look like.

This uncertainty is caused by the end of *Dennard Scaling* [1]. Dennard Scaling enabled the efficient use of an increasing number of transistors in an area in line with Moore's Law [2]. Until that point, performance was increased per processor generation, an improvement mainly reached through frequency scaling. Thus, performance improvement for applications occurred concurrently. In response to the end of the Dennard Scaling, hardware developers chose to conduct multicore scaling, but increasing the number of cores per processor generation cannot continue arbitrarily [3, 4]. Additionally, performance improvement only occurs for applications that provide a high degree of parallelism, compare with Amdahl's Law [5].

"Far better an approximate answer to the right question, which is often vague, than the exact answer to the wrong question, which can always be made precise." [John Turkey] Transferred to computer systems, is it necessary to always calculate the best results? Are there situations in which we should calculate a "good enough" result depending on the actual purpose?

Accuracy-aware computing deals with these questions and offers a new opportunity to tune a system towards given design goals by using the accuracy of results as an additional design parameter. Computer systems are traditionally designed and tuned by finding a compromise among conflicting design goals such as performance, energy consumption, and real-time capability. Now, we also consider the accuracy of the results while reaching this compromise, although we must differentiate between the quality of result (QoR) and the solution quality for a problem (QoS). For instance, the perceived QoS of an MP3 can vary among listeners even if the QoR is the same.

Accuracy-aware computing can be divided into *approximate computing* (AC) and *high-precision arithmetic* (HPA). The main difference between AC and existing approximation methods, such as heuristics, is that AC allows some degree of approximation, which is a desired design parameter. It is not a compelling necessity to solve certain problems such as NP-hard ones. HPA provides methods to increase the precision of integer and floating-point operations within the system. The objective of accuracy-aware methods is that they can be applied more generally.

Applications such as image processing, recognition, mining, and synthesis (RMS), and data analysis can benefit from AC, the underlying rationale behind being that these applications work with noisy input data, have no golden output (i.e., similar results are still useful), exploit the perceptual limitations of humans, or iteratively improve their results [6]. In contrast to AC, HPA benefits domains such as numerics by minimizing the effect of numerical instabilities caused by computational errors.

In the literature, the same applications are often used to demonstrate the benefits of accuracy-aware methods. This makes sense in terms of comparability, but new domains must be investigated to increase the scope of accuracy-aware computing. To identify these new domains, new accuracy-aware methods must be designed.

In the literature, there exists accuracy-aware methods for different layers. The main focus of AC is at hardware and architectural layers; for HPA, it is at the software and algorithm layers. AC methods can be grouped according to four layers:

- *Hardware layer* methods [7] often deal with approximating processing units. Imprecise adders or multipliers reduce the required hardware area and are more energy-efficient compared to their precise counterparts. This also includes providing different hardware-supported data types or exploiting precision scaling.

- *Architecture layer* methods introduce AC into the hardware architecture. This includes neural processing units, approximated memory components [8], or entire designs that have integrated dynamic accuracy and voltage scaling [9, 10]. Programmers can use these components through an extended instruction set architecture. Some methods use graphics processing units (GPUs) [11] or field-programmable gate arrays (FPGAs) [12].

- *Algorithmic layer* methods sometimes use loop perforation [13] or loop tiling [14]. Others rely on an automatic transformation of the code into a neural network [15], which requires hardware support to be efficient, or they sample the input data. There are automatic ways to reason about the required data type [16] and approximate versions of certain algorithms are also used [17]. Furthermore, programming language methods exist that enable programmers to exploit the underlying approximation methods [18].

- *Task layer* methods comprise skipping tasks, relaxing synchronization points [19], or exploiting approximate parallel patterns [14].

To fully utilize the potential of accuracy-aware computing, these methods must be tuned individually or in combination during design time [20] or during runtime. However, only few approaches exist for this purpose.

Runtime approaches select an implementation version of a task from different approximate versions [17] or tune the corresponding knobs of the approximation methods [11, 21, 22] - for instance, the perforation rate of a loop. These approaches assume a streaming behavior and thus that subsequent inputs behave according to the current input.

Input-aware runtime methods decide if an approximate result is sufficient or if a re-execution using the original algorithm is required [23, 24]. Laurenzano et al. present an input-aware approach using a calibration phase per input, which tunes all available accuracy-aware knobs with a given QoR constraint [25] but cannot handle constraints other than QoR; furthermore this approach is limited to the algorithmic layer and provides only a few value settings per knob. These tuning approaches only tune methods of a single layer (a *horizontal view*).

Selecting methods from a single layer misses substantial potential for an approximation-tolerant application. Previous work shows that considering various layers offers substantial benefit [26] - for instance, a reduced and rank kernel exploiting three complementary methods. This ad hoc approach does not provide a general solution. Therefore, this thesis addresses an essential challenge: How to combine and tune methods from different layers (a *vertical view*) in a proactive way to achieve the greatest possible gain for an application? Considering methods from multiple layers dramatically increases the configuration space (which represents all possible combinations of methods and respective knobs' settings). Hence, the vertical view dramatically increases the complexity of finding appropriate configurations to achieve design goals. This *tuning approach* handles energy or execution time constraints since a user or an external system often restricts these resources.

In addition to the fact that accuracy-aware methods influence the values of conventional design objectives, conventional methods exists for tuning systems. For instance, exploiting more cores can accelerate an application but also increases energy consumption; dynamic voltage and frequency scaling (DVFS) can reduce required energy consumption for memory-bound tasks by decreasing the voltage and frequency of a core. Existing tuning approaches in the domain of accuracy-aware computing do not exploit these conventional methods' potential. Hence, one challenge is determining how to control these conventional methods using the accuracy-aware *tuning approach* to better satisfy constraints.

While reducing the degree of internal accuracy is valuable for approximation-tolerant applications, we also know that other algorithms rely on higher internal precision. This higher precision can reduce computational errors and thus improve numerical stability [27]. It is an open question how higher precision will improve QoR and influence the values of other objectives. Additionally, determining how to combine this aspect with the accuracy-aware *tuning approach* is challenging.

The research aim of this thesis is to introduce an innovative methodology to realize accuracy-aware systems, which will help designers integrate accuracy awareness into their systems. I propose an adaptive accuracy-aware approach across system layers that addresses the challenges discussed in this section, combining and tuning accuracy-aware methods on different system layers. The methods are located at the software layer, architecture layer, and hardware layer, and a layer can be composed of further layers; - for instance, the software layer can include a task, an algorithmic, and a data layer. To widen the scope of accuracy-aware computing for other domains, this thesis presents innovative accuracy-aware methods and techniques for different system layers.

The required tuning of the accuracy-aware methods is integrated into a *configuration layer* that tunes the available knobs of the accuracy-aware methods integrated into a system. This includes the methods proposed in this thesis. In the configuration layer, a suitable configuration specifying the knobs' settings is determined in a proactive way during runtime depending on the current input, system state, and constraints. The thesis also highlights that increasing internal precision is valuable for certain numerical algorithms, although a control mechanism would require a completely different approach than that used in the configuration layer. The rationale is that an application-specific approach and implementation is mandatory to achieve the best performance.

In sum, the proposed methodology exploits the inherent tolerance of applications for approximating internal and external results to improve values of other important design goals. This includes computation under time and energy constraints during runtime.

1.1 Thesis Organization

This thesis includes four parts. Part I includes the introduction, background, and overall proposed approach. Chapter 2 provides a brief background on the current challenges in designing systems, presents conventional optimization concepts (such as multicore programming and heterogeneous computing) and novel alternative concepts, and thoroughly discusses related work in the field of accuracy-aware computing.

Chapter 3 introduces the central research questions addressed in this thesis and provides a detailed proposal for a novel adaptive accuracy-aware approach across system layers that tunes the computational effort to reach a certain desired QoR.

Part II introduces innovative accuracy-aware methods designed for this thesis. Chapter 4 introduces a novel conversion-based approach for memory transfers, as well as explains the design choices and analyzes the benefits of such an approach for different applications. Additionally, it integrates the conversion unit into an RISC-V processor; a simulation-driven execution of the applications leads to a realistic evaluation of the new hardware component.

Chapter 5 introduces AC design pattern to help hardware programmers efficiently use the available resources of an hardware accelerator. It demonstrates these patterns' use on different applications, using FPGA-based designs for applications in biology, stereo vision, and time series analysis. The evaluation reveals that exploiting these patterns reduces the required resources and increases performance, as well as that these patterns are valuable for other hardware accelerators.

Chapter 6 presents different novel AC methods on the software layer, including the Locality-sensitive hash-based fuzzy memoization framework and a novel approach to exploit contract-based algorithms, and applies the AC methods to a set of known approximation-tolerant applications and new domains for AC.

In Chapter 7 shows that increasing internal data type precision can reduce the numerical instability of certain algorithms caused by computational errors, using the well-known Lanczos algorithm for calculating eigenvalues as a benchmark. Furthermore, the chapter considers first methods to reduce the effort introduced by higher precision and clearly demonstrates that algorithmic-specific optimizations are required to reduce computational effort.

Part III presents the approaches designed for the configuration layer. Chapter 8 highlights why it is extremely beneficial to exploit a combination of AC methods on various layers (a vertical view). Then, it discusses the drawbacks of existing static tuning approaches for controlling AC methods. Based on these shortcomings, it presents a novel static tuning approach based on performance profiles.

Chapter 9 completes the novel runtime tuning approach for AC; this approach relies on the information generated by the static tuning approach.

Part IV concludes this thesis. Chapter 10 summarizes this thesis' findings on the novel structure for an accuracy-aware computing system, reviews the benefits of using the new AC methods, and discusses future directions to extend this work.

1.2 Collaborations

Parts of this thesis were developed during collaborations with other researchers. For the sake of completeness, briefly, the related projects include:

In collaboration with Markus Hoffmann (Karlsruhe Institute of Technology), I have investigated the use of accuracy-aware computing in the domain of scientific computing - more precisely, a certain class of pre-conditioned iterative methods for solving systems of linear equations needed to solve an additional system per iteration, which does not influence the final result but poses the issue of significant overhead. The corresponding evaluation was published [BHR18].

A collaboration with Georg Maier and Thomas Längle (Fraunhofer Institute of Optronics, System Technologies, and Image Exploitation) have aimed to investigate novel concepts for sensor-based sorting, particularly accelerating required data processing. A result of this collaboration was published [MBL+16].

FPGAs provide an interesting platform to accelerate data-intensive applications. In collaboration with Fabian Nowak (former Karlsruhe Institute of Technology), I have designed FPGA-based solutions for an application from computational biology. The outcomes of this research were detailed in several publications [BNK15, NBS$^+$13, BN13]. With Martin Schäler (Karlsruhe Institute of Technology), I have investigated the use of accuracy-aware computing for time series analysis.

1.3 Previously Published Content

This thesis includes work that has been published or submitted for publication:

Chapter 4: [BHK16]

Chapter 5: [BBB$^+$16, BNK15, NBS$^+$13, B14, BKH15, BES$^+$17, BSE$^+$17, BN13]

Chapter 6: [BK18, BKM$^+$18]

Chapter 7: [BHH17]

Chapter 8: [BHR18, BKM$^+$18]

Chapter 9: [MBL$^+$16, BKM$^+$18]

CHAPTER TWO

BACKGROUND AND RELATED WORK

This chapter introduces the background of the present thesis. It describes the current challenges for designing computer systems and presents conventional and alternative concepts to address these challenges. Then, it discusses the state-of-the-art in accuracy-aware computing and how it relates to the present work.

2.1 Design Goals and Challenges for Designing Computer Systems

A system can be defined as a set M of at least two members. Each member has a path to every other one [28]. A system can consist of components and other systems called subsystems [29]. In the context of computer science, the term computer systems

> "includes system architectures, operating systems, distributed systems, and computer networks" [30, page 1].

The design of computer systems is a very complex task and relies on expertise in several fields. To reduce this complexity, the design is split into several layers [31], see Figure 2.1.

Each expert is in charge for a certain layer, for instance, the computer architect implements the instruction set architecture in hardware as efficient as possible. During the design, several aspects have to be taken into account. These aspects are the design goals, the constraints, the design principles, and the different requirements. The latter compromises the application area, the level of software compatibility, the requirements of the operating system and the different standards [32]. Optimal design decisions depend on metrics, such as performance, energy efficiency, reliability, fault tolerance, and costs.

Currently used applications, ranging from machine learning and data analysis to image processing, pose huge challenges to computer systems. Especially, the world's

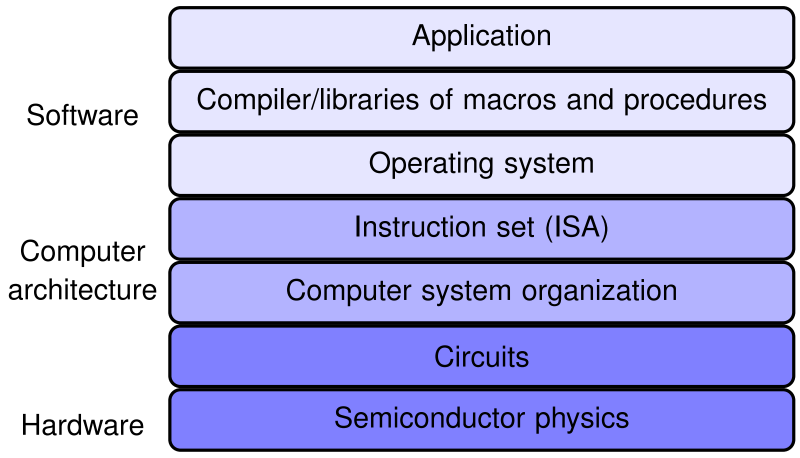


Figure 2.1: Structure of layered computer systems according to [31].

amount of data is doubling every two years and is in the zetabyte range currently [33]. This huge amount of data has to be efficiently processed in order to find valuable insights. Often short response times and real-time processing is important for applications such as autonomous driving, time series analysis, and industrial applications. A high throughput is crucial for data analysis tasks. Besides the performance aspects, a low energy consumption is often inalienable. Computer architects, system designer, and application developers are currently facing the so-called Brick Wall [34].

2.1.1 Brick Wall

Performance is one of the primary design goals of computer systems. The performance of a system depends among others on the algorithm, the used programming language, the compiler, the instruction set architecture, and the hardware. In the past, the main performance improvement for CPU-intensive applications was achieved by scaling the frequency and by increasing the instruction level parallelism (ILP). A useful metric for the performance strongly depends on the application area. The execution time of an application is considered as the only valid and reliable metric [35]. A relative performance can be determined by using benchmarks [36]. Benchmarks are a set of real world programs and are selected according to the type of the domain [37, 38, 39].

Until 2004, computer architects were able to exploit the increasing amount of transistors (cf. Moore's Law [2]) in almost the same rate for performance improvements of general purpose processors. The driving force behind it was Dennard scaling [40]. Dennard scaling enabled the decrease of the power used by transistors in the same range than the area shrinking. However, the Brick Wall consisting of *Memory Wall* [41], *Power Wall*, and *ILP Wall* [32] stopped this free lunch of performance improvement [42].

The *Memory Wall* describes the widening gap between the performance of processing units and the memory. A main approach to overcome this wall is the usage of a cache hierarchy. This is valuable for applications that have a spatial and temporal data locality. However, a programmer has the burden to optimally write code in order to fully exploit the potential of caches [43, 44, 45]. The arise of new applications coming from the area of big data poses further requirements to a cache hierarchy. These applications use a much higher instruction working set. This set does not fit into current L1 instruction caches of high-end processors [46].

The *ILP Wall* describes the challenge to extract more parallel instructions from a sequential stream. This results in underutilized processing units [32].

The *Power Wall* is strongly related to the end of Dennard Scaling [1]. This end is caused by the high impact of leakage current and the limitation of scaling gate oxide thickness below 90 nm [47]. Hence, further increasing the frequency highly impacts the power consumption in a exponential fashion. Additionally, this causes serious heat problems. Too high temperature leads to reliability issues, slowing down transistor switching, or hardware damage. Therefore, we are in a situation, where we cannot power on all transistors on a chip currently. This situation is often referred as dark silicon [3, 48, 4].

Design decisions that improve a design value often negatively impact the other ones. Hence, a compromise between different design goals has to be found. This compromise leads to Pareto-optimal design point. The optimum value regarding a single design objective specifies a Pareto-optimal point. Hence, this specific value is smaller or equal for all other design points. Several approaches exist that consider design goals together to improve the overall system state [49, 50]. In this thesis, the focus is to find a compromise between quality of result (QoR), energy consumption, and performance.

2.1.2 Current and Future Trends to Break the Brick Wall

In the following, current and future trends and approaches to overcome the *Performance* and *Power Wall* are presented. These approaches do not impact the QoR.

Multicore scaling

The industry has responded to the failure of Dennard scaling by building multicore architectures. These architectures include cores replicating existing core designs instead of designing a very complex single core. This was a paradigm change from latency-oriented to throughput-oriented designs. Programmers have a high effort to program multi-threaded applications. This task is more error-prone than programming applications for single cores due to concurrency issues.

Not all applications can benefit from the performance capability of multicores. Many algorithms have an inherent sequential part (Amdahl's law [51]), which limits the maximum speedup. Revisiting Amdahl's law in the multicore area, researchers emphasize to conduct research on parallel and sequential ways to increase performance of applications [5]. Researchers stated that dark silicon probably leads to the end of the multicore scaling [3, 4]. They predict an optimistic performance increase of only 7.9 for highly parallel workloads and a pessimistic increase of 3.7× [3].

Modern processors can adapt their power consumption itself or allow the operating system and the user to do this. For example, Intel's SpeedStep or AMD's PowerNow. Dynamic voltage and frequency scaling can trade off performance for energy [52, 53]. Reducing the voltage, hence the frequency leads to less power consumption of a processor, but reduces performance. To improve performance for a short period, hardware vendors have introduced methods such as Intel TurboBoost [54, 55] and AMD Turbo Core. These methods increase the frequency of a core compared to the base frequency in case that power, temperature, and current are below certain limits.

Heterogeneous Computing

Approaches were proposed to overcome dark silicon in multicores, for instance, by exploiting heterogeneous and specialized cores [48]. In general, the term heterogeneous computing refers to a system or a single chip integrating different types of processing units. These units can significantly differ in their architecture or just slightly differ in certain aspects. Since different applications have different requirements on the underlying hardware, specialized cores for certain tasks are beneficial. Heterogeneous systems [56, 57] combine units such as manycores [58], field-programmable gate arrays (FPGAs) [59, 60], and graphics processor units (GPUs) [61].

It is common known that FPGAs are more energy-efficient than GPUs [62]. GPUs often perform better on streaming applications that require a high bandwidth. There exist FPGA-based platforms that are optimized for non-linear memory accesses. These platforms are beneficial for highly parallel applications that require a low memory bandwidth [63]. Sliding-window applications such as 2D convolution performs well on FPGAs [64]. The best architecture depends amongst others on the input size [64]. In [65], the authors propose guidelines for finding the best hardware unit according to the properties of an application. Another approach for providing heterogeneity is to adapt certain characteristics of a single core during runtime. This includes the change of the frequency or the adaptation of the bandwidth for the decode, issue, commit, and fetch stage [66].

Software Improvements

Another way to improve performance is to find optimizations regarding the way, how we develop software and execute the software on the hardware. Nowadays, programmers assemble application code from a large number of reusable libraries and frameworks. Many abstraction layers were invented to improve the productivity of software. Each abstraction layer causes a performance gap, which was closed by the frequency scaling.

The end of the free lunch raises the question how we can run software more efficiently. This requires solutions to deal with the aforementioned software bloat [67]. A better integration of methods between different stack layers can improve the performance. For instance, a garbage collector that works between the operating system and the Java virtual machine allows an in-memory execution and thus improves the performance [68].

The performance and energy issues do not represent an isolated problem for the compiler and computer architecture community. These issues has to be considered by all kind of software designers, since a runtime bloat causes an inefficient usage of the hardware [69, 70]. Therefore, tools and approaches have to exist that find performance bugs within the application. This includes performance bugs from memory leakage [71], memory bloat [72, 73, 74], and execution bloat [75]. The resulting consensus should be that performance analysis is an important aspect for all developers [76, 77].

To further increase performance, hardware-specific optimizations are a promising solution. These approaches adapt the software according to the current hardware [78]. They are called (online) empirical code optimization or auto tuning [79, 80]. For instance, finding the best tile size for loop tiling improves the exploitable parallelism and data locality and thus reduces the execution time [81]. Auto tuning has to find an optimal solution in a huge search space. Therefore, approaches exist that reduce the search space before applying a search algorithm [82]. Optimal auto tuners require knowledge of the applied domain, hence it is important to have domain-specific tuners. Solutions exist that help developers to implement tuners for their purpose [83]. Another optimization approach is to select the best version from a set of code versions during runtime[84].

A strategy called dynamic concurrency scheduling (DCT) controls the number of active threads to save energy and to improve performance concurrently [85]. This is applicable for a single parallel programming model, such as MPI or OpenMP [86], but also for hybrid MPI/OpenMP models [87].

Device Trends

A further research direction is to improve the device technology for gates and memory components. The intention is to find a breakthrough for technological devices similar to the breakthroughs that were achieved by going from vacuum tubes to bipolar transistors and from these transistors to complementary metal-oxide-semiconductor (CMOS) designs. Near threshold computing (NTC) or sub threshold computing is a solution to improve the energy efficiency of CMOS-based processors with the drawback of a (significant) performance degradation [88, 89]. Evolutionary approaches extent the conventional CMOS technology by new structures or materials to overcome the failure of Dennard scaling [90]. Besides extensions to classical CMOS devices, there are plenty of ideas for novel technologies often summarized as beyond CMOS [91, 92]. To get mainstream, such device technologies need to provide significant improvements regarding energy and delay of standard cells. Furthermore, they have to provide a logic and memory family. Beyond CMOS devices can be categorized into charge and non-charge approaches [93]. Examples are tunneling field effect transistors [94] or carbon nanotube field effect transistors (CNT-FET) [95].

There is a huge research effort regarding novel memory technologies such as non-volatile memory (NVM) [96]. NVM is mostly used for secondary storage, but there is a trend to develop NVM devices acting as main memory. Spin transfer torque RAM (STT-RAM) is an example [97]. Intel and Micron developed a NVM called 3D Xpoint and a stack-based RAM called Hybrid Memory Cube (HMC). HMC provides a theoretical bandwidth of up to 420 GB/s [98]. These approaches use 3D integration [99, 100, 101]. 3D integration exploits a third dimension to achieve a power reduction and a higher bandwidth. Thus, it aims to overcome the memory wall. However, several challenges have to be solved before applying this technology to main stream. These challenges include amongst others yield rate, financial costs, design complexity, and testing.

Alternative Computing

There exist new alternative computing concepts [102] inspired by biology, the human brain [103, 104, 105, 106, 107], or new technologies [108, 109]. For instance, quantum computing exploits the quantum-mechanical phenomena to realize quantum bits. Such bits can represent 0, 1, or any superposition between it. Currently, most of these methods are still under research or in a work in progress state.

Stochastic computing is an unconventional computing method and represents numbers as random binary bit streams [110]. This representation allows real numbers between zero and one. The numbers are specified by the amount of ones in a bit stream. Stochastic computing has the benefit to build simple hardware and is more fault tolerant. The latter results from the point that there is no positional weighting for single bits. Therefore, each bit flip change the result by the same amount. Stochastic computing is applied among others to image processing, neural computations, and reliability analysis [7]. In addition, stochastic computing is used for error-resilient designs [111]. However, a major issue with stochastic computing is the accuracy of results. Correlated random bit streams result in an inaccurate output. Another main drawback is the long latency of operations caused by an exponential increase with respect to the accuracy [110].

A further recent trend is optical computing. Optical computing exploits photons for computations or memory transfers. On-chip optical interconnection is an approach to reduce the issue regarding the memory wall [112, 113].

2.1.3 Discussion

The end of Dennard scaling has raised two important questions for system designers. How can we still improve design values for applications? How can we satisfy the requirements of applications?

Multicore scaling is only suitable for applications that provide a certain degree of parallelism. *Heterogeneous computing* is a promising approach to provide significant benefits for applications. However, not all applications have the capabilities to fully exploit the potential of heterogeneous systems. Radically changing the way of implementing systems is not a feasible option, since it would drastically impact the design time negatively. The future situation regarding the used device technology is unclear and no winner is in prospect. The same holds for most of the alternative computing concepts.

The present thesis focuses on solutions to improve systems that are based on existing hardware technologies. The exploited computing methodology poses an alternative way to find a better compromise between conventional design goals. This methodology called accuracy-aware computing uses the approximation degree of computations as a further design parameter. Accuracy-aware computing represents an orthogonal solutions and thus allows designer to combine it with the aforementioned approaches in this section. Compared to stochastic computing, accuracy-aware computing does not assume any stochastic nature and provides deterministic approximations [7].

2.2 Accuracy-aware Computing

In this section, the related work regarding accuracy-aware computing is discussed. This includes the state-of-the-art in approximate computing and high precision arithmetic. Before doing this, the term Quality of Result (QoR) as used in this thesis is defined.

The Quality of Result (QoR) is a measure for the accuracy loss of the final output of an application. A numerical value represents this loss. The metric must reflect the result quality of the application that matters for the user or the system. Generic metrics, such as the error rate, are often inadequate for this modeling. Moreover, there exist no general approach to model the relationship between a generic and an application-specific metric. Defining a suitable and application-dependent metric for the QoR is a very complex challenge [114]. Akturk et al. present a classification for useful metrics for the QoR and corresponding example applications [115]. Example metrics are the mean square error, classification accuracy, Signal to Noise Ratio (SNR), or the number of mismatches between a exact and an inexact query to a database. It can be worth to consider different metrics for an application and select a suitable subset of the considered metrics [114].

2.2.1 Approximate Computing

Approximate computing (AC) comprises different approaches that trade off the internal and external QoR for improved performance or energy consumption, while still retaining adequate results [116, 117, 118, 119, 120, 121, 122]. Moreau et al. presented a taxonomy for AC [123]. AC is applicable to a wide-range of applications such as Recognition, Synthesis, and Mining (RMS), machine learning, image processing, and big data applications [124, 125, 126, 127, 128]. Iterative algorithms such as the Jacobi method benefit from executing iterations on low-power but unreliable hardware units [129]. There exist AC approach for different layers which I group into four layers.

Overview of Methods on The Task and Algorithmic Layer

Deploying neural networks to approximate imperative code snippets is a generally applicable method [15, 128]. The neural network-based method is limited to a small amount of inputs. McAfee et al. proposed a method for decomposable algorithms [130]. To improve performance or energy consumption of such networks, approximating the network itself is a possible solution [131, 132] For instance, AXNN finds approximable neurons through backtracking and applies precision scaling to the found neurons[131]. Such a quantization method is further improved by a dynamically scaling together with avoiding operations that has zero as input [133]. But the issue with such a solution is the variable data size of processing units. To accelerate the evaluation of a convolution neural network, the usage of per-layer perforation reduces the amount of calculated outputs [134]. Missing output values per layer are interpolated using nearest neighbors. Similar to neural networks, the evaluation of support vector machines are approximable [135].

Loop perforation is a concept to skip certain iterations within a loop [13]. An automatic program transformation exploits loop perforation [136]. In [124], Yazdanbakhsh et al. conclude that the neural network-based method outperforms loop perforation on the used benchmarks. They claim that neural networks are close to optimal in terms of the computation. There is no gain in terms of memory accesses, since the access patterns are the same.

Beside perforating loops, the number of tasks can be reduced [137]. A corresponding method is relaxing synchronization points for applications running on parallel architectures [19]. Replacing exact parts with approximated versions within a program is a further method [138, 139]. Another method transfers serial to parallel loops without the requirement to calculate exact results [138].

Anytime algorithms correlate the execution time with QoR [140]. Such algorithms can be applied to image processing algorithms [141]. Recent work considers such algorithms on GPUs[142]. In [143, 144], the authors propose such a method to represent approximation-tolerant applications as a parallel pipeline of anytime sampling stages.

ApproxHadoop [127] provides an approximation framework for map-reduced applications, while minimizing the total error. A different sampling strategy improves the per-key error [145]. Programs modeled as a tree of computation and reduction nodes can be transformed to an AC-based program [146]. ApproxIt [147] is a framework to approximate iterative methods. Combining incremental computing and AC leads to the framework IncApprox. This enables a higher throughput as shown by experiments using a Twitter data stream and compared to a native Spark Streaming [148].

Well-known parallel patterns such as mean, minimum, sum, and linked lists offer the opportunity to apply AC [149]. Parapox [14] considers different approximation methods for data parallel patterns. Often algorithms themselves have static configuration parameters that influence the QoR. These parameters can be used as dynamic knobs to trade off QoR for execution time [150].

Overview of Methods on The Architecture and Hardware Layer

Function units such as adders [151, 152, 7, 153], multipliers [154, 155, 156], or compressors used for Dadda multipliers [157] were considered for an approximated design. Lazy pipelines exploit the slack of underutilized imprecise function units realized by voltage over scaling to improve the QoR [158]. GPUs are more power-efficient by using approximated floating point units [159].

To reduce the design effort for AC hardware, an extension of a hardware description language (Verilog) was proposed [160]. Furthermore, hardware synthesis for approximate circuits was considered [161, 162, 163, 164]. It is also important to analyze the approximated circuits [165].

While dynamic voltage and frequency scaling enables a reduction of the energy consumption, dynamic voltage an accuracy scaling keeps the QoR in mind [9]. There exist processor designs that are extended by approximated instructions [166, 10, 166]. Furthermore, there exist AC-based processors [167]. For instance, a quality-configurable reduce and rank hardware design was proposed [168].

To reduce the energy consumption and the latency caused by cache misses, a load-value approximation unit returns a historically estimated value [169]. A similar method targeting GPUs additionally drops a certain amount of missing load requests to reduce pressure regarding the off-chip memory bandwidth [170]. A last level cache often contains similar values. Miguel et al. exploit this fact for applications that can tolerate approximations [171, 172]. Shoushtari et al. [173] favor a two cache policy for storing exact and approximate data separately. Flikker is a method to distribute exact and approximated data into different modules of a DRAM memory. For the modules that contain approximable data, the refresh cycle time is increased. This reduces the energy consumption of the modules [8]. The time required for storing data into a phase change memory can be reduced for approximable data [174]. Approximating SSD accesses accelerates applications such as WordCount [175].

Memoization is a technique to store calculated results within a table for later reuse. For instance, the result of a costly function inside a loop [176]. A fuzzy memoization approach stores results of floating point operations [177]. This approach also uses stored results, when the input parameters are similar. A crucial point is to find a measure for the similarity to avoid high errors. The significance distance can pose such a measure [178]. Results of function units within a GPU can be cached inside an associative memory [179]. To indicate an entry in the memoization table, mapping functions such as cryptographic hashes were considered [180]. However, this approach does not use application-specific knowledge. A FPGA-based memoization framework reduces the energy consumption of function unit [181].

Since a software-based calculation of a neural network is slow, a hardware-based execution was proposed [182]. GPUs connected to neural accelerators were also considered [183]. An approximation of the accelerator itself is also possible [184]. The combination of precise and approximate accelerators is useful [185]. The approximate accelerators vary in performance and QoR executing the same task.

Main Approximate Computing Issues

An important aspect before applying AC methods is to decide how to quantify the QoR [115]. Especially, the difference between the QoR and the validity of a result is important. Crowdsourcing allows programmers to get feedback about the tolerable approximation from users directly [186].

Finding approximable algorithms or parts within a complex application is an important task [187]. Domain experts can indicate such parts [18, 188], or tools exists that find such parts under certain prerequisites [6]. To reduce the effort of programmers for annotating their programs, the solution of Park et al. is applicable [189]. Debugging and monitoring of AC applications delivers further insights [190].

There exist first compiler approaches which exploit approximable code regions [191, 192, 193, 194]. Significance analysis [195] or sampling strategies [196] find parameters or inputs that are not suitable for an approximation. A profiling method investigates the QoR by applying loop perforation [197]. In total, sampling approaches are very accurate, but slow. Statistic or compiler approaches are fast, but can miss potential due to their non-data sensitivity. A useful configuration of AC methods also depends on the current program phase [20]. Based on the significance value for a task, the decision is made where the task is executed, i.e. whether it is executed on exact or inexact hardware [198].

Since the resulting QoR is very input dependent, runtime decisions greatly improves the benefits of AC and avoids a pessimistic assumption. There exist frameworks that selects a version of a task during runtime [17, 199]. There also exist approaches that create such versions for GPUs [11] and FPGAs [12]. CoAdapt is a mechanism that tries to meet certain constraints during runtime. It can consider up to two constraints in parallel [22]. But this approach only works for streaming applications. JouleGuard guarantees a user-specified energy consumption of a system while providing near optimal QoR [21]. Scheduling of hard or soft real-time tasks for approximation-tolerant applications was investigated on heterogeneous multicore architectures [200, 201].

Light-weight checks [202] or models [203, 23, 204] check the QoR during runtime.

Topaz checks if a result is outside of a tolerable error margin and induces a re-execution of the task on accurate hardware [204]. Combining several base checkers increases the prediction accuracy and hence reduces wrong rollbacks [205]. Predictors decide about the usage of an approximate accelerator depending on the input data [24].

A proactive control approach configures an AC-based application per invocation according to a QoR constraint [206]. A useful configuration can be found by comparing different settings of AC methods on a small subset of the input data. The found configuration likely results in the best achievable acceleration, while keeping the desired QoR [25]. However, the authors only evaluate their approach using AC methods from the algorithmic layer and for a small configuration space.

Discussion

Considering the state-of-the-art in approximate computing, we can see that the set of used applications to design novel methods is limited. Therefore, this thesis makes a step forwards to increase the scope of AC by evaluating AC on novel applications. To further exploit the potential of AC, we rely on novel and more generally usable AC methods on different layers. Such methods are designed, implemented, and evaluated on novel application areas for the present thesis.

All of the mentioned control approaches do not satisfy the requirements of a holistic methodology for realizing an accuracy-aware system. Especially, the combination and tuning of methods on the different layers is important. Moreover, the control approach has to adapt the parameter setting for the different AC methods depending on the current system state. A detailed summary of the missing points in the AC domain and the derived research statements are discussed in Chapter 3.

2.2.2 High Precision Arithmetic

64-bit IEEE-754 floating-point arithmetic is often sufficient for scientific applications. On the other side, there exist a considerable amount of applications that rely on high precision. For instance, ill-conditioned linear systems, large summations, or large-scale simulations [27, 207]. Note that a computer system never provides infinite precision.

Floating-point arithmetic causes four main issues regarding the accuracy of results. Firstly, rounding floating-point values introduces numerical errors. They are not necessarily bad in each situation [208]. Secondly, not each real number is representable in a binary system correctly. Thirdly, arithmetic errors occur after simple operations. Cancellation of significant bits and absorption count to these errors. Finally, the associative and distributive property are not valid for floating-point arithmetic.

Software libraries are usually used for high precision arithmetic.These libraries differ amongst others in the used number representation. Examples are GNU Multi Precision

Floating-Point Reliable (MPFR), Multiple Precision Integers and Rationals (MPIR), Arbitrary Precision Computation Package (ARPREC), and MPFUN. MPFR is based on GNU Multi Precision (GMP). A comparison between these libraries is missing in the literature.

A suitable data representation is important to avoid incorrect results and unnecessary overhead for scientific applications [209]. Tool support is available for programmers to find operations that can be performed in single precision [16, 210]. This possibly reduces the computational effort. There exist GPU-based libraries that provide double-double, quad-double, and arbitrary precision [211, 212, 213]. Reconfigurable architectures exist that support high precision arithmetic [214, 215, 216]. It was claimed that exact addition and multiplication in-line with an exact dot product is crucial for high precision arithmetic [217]. A radical approach was proposed to replace IEEE-754 arithmetic [218]. The main issue with this approach called *UNUM* is the hardware complexity. It was argued that the format is only useful for low-precision applications [219].

This thesis gives an answer to the following question: Is it possible to build an accuracy-aware systems that tunes the precision inside a system for a given numerical algorithm? This question is further discussed in Chapter 7.

CHAPTER
THREE

AN ADAPTIVE ACCURACY-AWARE APPROACH ACROSS SYSTEM LAYERS

This chapter discusses the central dissertation statements, which where identified and established according to the latest standards and methods in accuracy-aware computing. Hence, they represent aspects missing in the relevant literature. Using these statements, this chapter presents this thesis' contributions in detail.

3.1 Dissertation Statements

This thesis' main research aim is to develop an accuracy-aware multi-layer approach that considers a horizontal view and a vertical view. The corresponding tuning approach configures available accuracy-aware methods for approximation-tolerant applications on the various layers. It is important to note that it is unnecessary to have at least one method for each layer. This thesis includes research objectives to address its overarching aim:

I. Increasing the scope of accuracy-aware computing and thus finding its limitations.

   Increasing the application field of accuracy-aware computing is important to building acceptance of it. Therefore, this thesis surveys novel application domains on the applicability of accuracy-aware computing. This also builds understanding on application characteristics that make accuracy-aware computing the methodology of choice.

II. Designing accuracy-aware methods that can be used for different applications.

   IIa. Developing a general method to reduce costly data transfers for the floating-point unit while providing a deterministic behavior.

IIb. Providing a set of AC-based design patterns for heterogeneous units.

IIc. Developing a general accuracy-aware method on the software layer that overcomes the limited scope of current methods.

To improve the conventional design values of applications coming from the newly found domains, innovative methods are required that can be applied to a broad spectrum. This also makes it possible to find further domains for accuracy-aware computing. Furthermore, a control method is needed to vary the approximation degree during design time or runtime. Thus, it is necessary to investigate the limitations of current accuracy-aware methods and design new ones that make more general applicability possible. According to the vertical view, innovative accuracy-aware methods are designed for various layers such as the hardware layer, architecture layer, and software layer.

III. Evaluating HPA and the achievable benefit for applications.

IIIa. Evaluating an algorithm that is numerically unstable.

IIIb. Considering the possibility of developing a tuning system for applications that benefit from HPA.

Certain algorithms require higher internal precision than offered by hardware-supported data types. This thesis takes a step forward and investigates whether it is possible to tune a system by using a higher precision degree. It investigates the HPA's potential to overcome the issues caused by internal computational errors, as well as considers how the method competes in terms of performance with algorithmic-specific adaptations.

IV. Solving the problem of providing a runtime tuning approach for approximation-tolerant applications that:

IVa. Is more generally usable than existing approaches.

IVb. Applies a horizontal view and a vertical view for tuning knobs.

IVc. Provides input awareness for the system. This is important since the quality of result (QoR) loss depends on the current input in addition to the knobs.

IVd. Deals with multiple constraints and system states.

IVe. Exploits the potential of other conventional methods and application-specific approaches.

IVf. Considers competing tasks within and between applications.

IVg. Finds a good compromise among multiple design goals.

One of this thesis' main aspects is designing a configuration layer that selects, deploys, and tunes accuracy-aware methods realized for different system layers (a vertical view). Different system requirements can occur: maximizing the QoR while having a pre-defined budget or minimizing the budget while meeting a certain QoR. Since there is generally no clear mapping between quality of solution and QoR, domain knowledge is used for mapping. This leads to a numerical value for QoR, which is required for tuning the system and thus finding a suitable configuration. A configuration represents the settings of knobs for all accuracy-aware methods, which can also imply that a method is deactivated. The resulting configuration layer tunes the system per input in a proactive way; it is important to stay above certain QoR boundaries, but calculating results that are too accurate means missing opportunities to meet other design goals. Moreover, the configuration layer must control competitive tasks versus only considering a single task inside a system.

3.2 The Central Approach for an Accuracy-aware System Structure

This section describes the holistic approach used in this thesis to build an accuracy-aware system. To apply accuracy-aware computing to a wider scope (Statement I) than related work, this thesis investigates new domains that include applications from time series analysis, sensor-based sorting, stereo vision, computational biology, and scientific computation. For all of these applications, I show that accuracy-aware computing is highly beneficial.

This process has resulted in two lessons: First, it is wise to clarify the necessary QoR for an application. For instance, using stereo vision data in an automotive environment for collision detection does not require the per-pixel accuracy of the real world; this significantly increases the potential gain for other design goals' values by using accuracy-aware computing. Second, it is useful to integrate an accuracy-aware method providing granular adaption of the approximation degree into a software layer; this allows for initial detection of whether accuracy-aware computing is useful for the application reducing the effort needed to evaluate the suitability of accuracy-aware computing.

Considering a set of known approximation-tolerant applications from related work alongside these new domains leads to innovative accuracy-aware methods. They are designed and implemented on various layers (Statement II), represented in Figure 3.1 by the smaller bright rectangles; the corresponding layers (hardware layer, architecture layer, and software layer) are below the thick black line, and the corresponding chapter with a detailed description of each method is provided.

On the hardware layer, this thesis introduces an innovative AC method based on

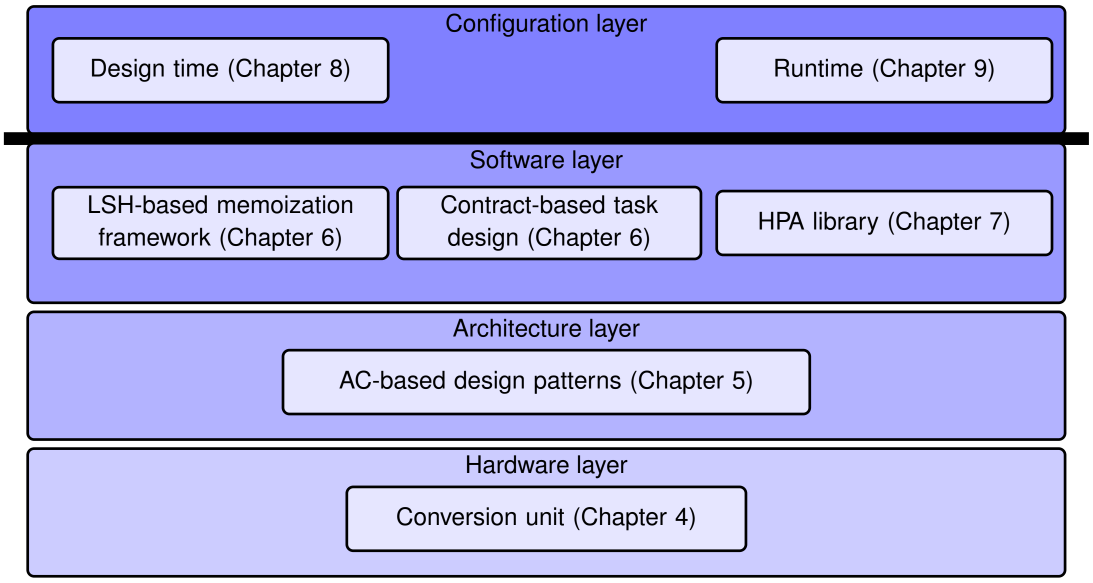

Configuration layer
Design time (Chapter 8)
Runtime (Chapter 9)
Software layer
LSH-based memoization framework (Chapter 6)
Contract-based task design (Chapter 6)
HPA library (Chapter 7)
Architecture layer
AC-based design patterns (Chapter 5)
Hardware layer
Conversion unit (Chapter 4)

Figure 3.1: Overview of the innovative methods introduced in this thesis on various layers and the corresponding configuration layer.

converting and compressing data. This method is applicable to algorithms working with floating-point values - for instance, time series analysis and image processing. This hardware-based method exploits deterministic approximation techniques that make the influence on QoR predictable. The outcome is a hardware-supported conversion unit for floating-point values, which reduces the amount of bits that must be transferred to a first-level memory and thus leads to energy savings and reduced time for data transfers while providing a higher QoR than native execution on a smaller data type.

On the architecture layer, it proposes novel AC-based design patterns for designing accelerators. An approximation of dynamic programming algorithms makes it possible to better exploit the capabilities of reconfigurable architectures and thus leads to more high-performance designs for accelerators. Removing certain data dependencies on the algorithmic level allows for efficient implementation and realizes a streaming-based design. This also results in a low internal memory footprint. Example applications are stereo vision and computational biology. Coupling this innovative approach with compression before sending data to the accelerator further improves performance and allows for varying the degree of approximation even during runtime.

On the software layer, this thesis designs an HPA library. Using this library, I show that increasing the operations' internal precision is beneficial to increase the quality of calculated eigenvalues of a matrix. However, this requires significant computational effort that can only be reduced by exploiting domain knowledge of a certain algorithm (Statement III). Furthermore, tuning precision according to the input remains a problem. Therefore, I do not consider higher internal precision of operations for the proposed adaptive accuracy-aware approach across system layers.

Also on the software layer, I design a novel library for fuzzy memoization that can easily be plugged into existing applications. The motivation of memoization is to cache previously calculated results of a function and re-use them when a similar input occurs. However, this method's inherent overhead makes its application for typical approximation-tolerant applications not useful. The same is true for other state-of-the-art fuzzy memoization methods, and neither fuzzy memoization nor other state-of-the-art software accuracy-aware methods can satisfy statement IIc. Therefore, employing contract algorithms in a new way enables a clear, fine-grain, and predictable correlation between execution time and QoR. Each contract-based component improves the output QoR over time, and hence the final result improves over time. Varying the granted budget leads to tuning QoR. Tasks from different domains can be re-implemented as contract algorithms. Exploiting this contract-based task design in accuracy-aware computing at the software layer leads to the desired software accuracy-aware method that is general applicable. Furthermore, reaching the exact result requires simply continuing to execute the contract-based task. I propose different methods that improve the performance of contract-based tasks. Moreover, I introduce best practices to realize contract-based tasks.

Having many accuracy-aware methods on different layers, a system designer faces the problem of deciding which accuracy-aware methods would be valuable for his or her system. An additional challenge is determining how to tune these different methods. Increasing the difficulty, this decision must be made during runtime to achieve greatest gains for the system. For tuning these methods, a configuration must be found for each method. A configuration specifies the knob settings for the accuracy-aware method and makes deactivating the method possible; for instance, the perforation rate used by loop perforation is such a knob. Setting this rate to a specific value presents a configuration for the accuracy-aware method.

As a hypothetical, if there are ten possible perforation rates and thus 10 possible configurations, further integrating a loop perforation with 10 configurations into the application leads to a global configuration space for the system with 100 global configurations. Thus, integrating many accuracy-aware and conventional methods drastically increases the global configuration space of the accuracy-aware system. According to the Statement IV, a suitable global configuration must be extracted during runtime according to different constraints, system states, and inputs. Therefore, this thesis designs a configuration layer to help system designers to significantly reduce design effort and the complexity of extracting a suitable global configuration during runtime, as well as to activate and tune the integrated accuracy-aware methods in the system. The configuration layer consists of a design-time step and a runtime mechanism (see Figure 3.1, above the thick black line). The runtime mechanism exploits the information determined by the design-time step. The design-time step is required to significantly reduce the number of global configurations that can be selected during runtime to drastically decrease runtime overhead during tuning. It is vital that the runtime mechanism introduces low overhead to maintain the benefits of using accuracy-aware computing. When the application is composed of tasks, error propagation between tasks within the application is application-specific. Therefore, I suggest also determining this correlation during design time.

The remainder of this subsection briefly presents the innovative methodology designed for this thesis to realize an accuracy-aware system, including the steps to achieve this system and their relationships to each other (see Figure 3.2). The starting point is an approximation-tolerant application composed of task, which a system designer has already implemented (Step A). The designer must collect representative input data for the system that are important for the design-time step (Step B), and then integrate accuracy-aware methods per suitable task (Step C); an important property of a suitable task is its significant contribution to the entire execution time or to energy consumption. Steps A to C require the most effort for a designer to realize the accuracy-aware system. However, this thesis presents best practices to help reduce this effort.

The remaining steps involve identifying, designing, and implementing solutions, methods, and algorithms to help the designer integrate and realize the configuration layer. The configuration layer consists of a design-time step and a control mechanism. In the design-time step, the different task flows within the application are determined (Step D). Task flow depends on the application parameters that, for instance, control the ordering of image filters. This is important since the error propagation depends on the task flow. Suitable global configurations for each flow are determined in a later step.

The next step (Step E) involves determining (near-)Pareto-optimal local configurations per task. Here, a local configuration represents knob settings for all integrated accuracy-aware and conventional methods. A Pareto-optimal local configuration means

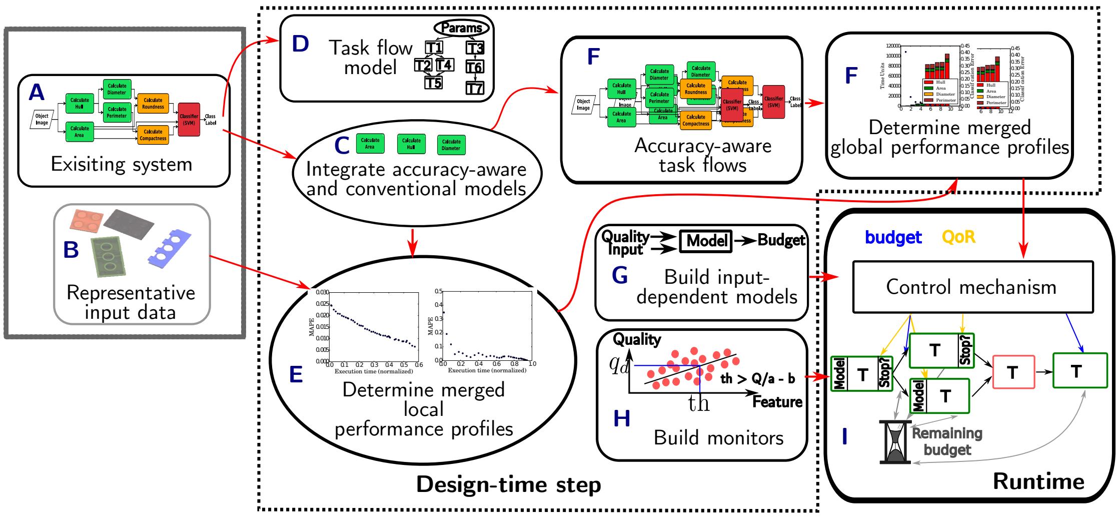

A
Exisiting system
B
Representative input data
D
Task flow model
Params
T1
T2
T3
T4
T5
T6
T7
C
Integrate accuracy-aware and conventional models
E
Determine merged local performance profiles
F
Accuracy-aware task flows
F
Determine merged global performance profiles
Quality
Input
Model
Budget
G
Build input-dependent models
Quality
th > Q/a - b
th
Feature
H
Build monitors
Design-time step
budget
QoR
Control mechanism
Model
T
Stop?
Remaining budget
I
Runtime

Figure 3.2: Overview of steps to realize an accuracy-aware system.

this configuration can only be outperformed in terms of a single design value (either QoR or required budget). As a budget metric, my approach uses execution time, energy consumption, or a combination using energy delay product metrics. This allows for considering multiple objectives - execution time, consumed energy, and QoR - during runtime. Such a Pareto-optimal front is called a *merged performance profile* of the task. This local approach significantly reduces the number of global configurations that must be considered for the entire application. To further reduce the number of evaluated global configurations required to determine a (near-)Pareto-optimal global configuration front significantly, I propose a greedy-based static tuning algorithm. This algorithm exploits the local merged performance profiles (Step F). The extracted front represents the merged performance profile of the entire accuracy-aware system. Each available point in that profile represents a global configuration that can be selected during runtime. This procedure (Step F) is applied for each determined task flow.

The proposed approach introduces two techniques to integrate input awareness into the system. The first creates a model that determines the required budget to reach a pre-defined QoR for a certain input and task (Step G). The model uses the QoR and input-dependent parameters of the function in question to determine the budget. Input-dependent parameters are the size of the input, the mean, the variance, or other statistical features that can be derived from the function's input data. The second method is a monitoring approach that calculates features during execution (such as the amount of considered pixels or the variation from the previous result) and uses these features to estimate the current QoR of a task (step H). If the desired QoR is reached, the execution is terminated, and thus the monitor acts as a termination condition.

During runtime, the control mechanism receives information about the current constraints and system state (Step I), which it uses to select a global configuration from the global merged performance profile. Input awareness is introduced on a local per-task level, and two different control modes exist: *QoR mode* and *budget mode*.

In QoR mode, each task reaches its specified local QoR even when requiring greater spending than initially provided. This decision depends on the input-aware local models, and the information of the local desired QoRs is also part of the global configuration.

For the budget mode, the user or a high-level controller grants a global budget (for instance, a firm deadline that should be met) and thus an initial global configuration is chosen with the highest QoR on average for the given global budget. If several inputs processed by different application instances share a budget, unused local budgets can be redistributed to other tasks; such unused budgets occur when the model determines a lower budget than that initially granted to the task. Finding a universal configuration for independent application instances or accuracy-aware applications is a multiple-choice knapsack problem. The solution parameterized according to the current system state and constraints, leads to suitable global configurations for all instances. This problem must be solved during runtime.

Part II

Design of Accuracy-aware Methods

CHAPTER
FOUR

A NOVEL ACCURACY-AWARE METHOD AT HARDWARE LAYER

This chapter introduces a novel accuracy-aware method at the hardware layer. Such a hardware method constitutes the lowest abstraction level to introduce an approximation into the system in this thesis. The proposed method provides a deterministic way to approximate individual floating-point values. The presented conversion unit can be applied to floating-point applications and provides a knob that varies the degree of quality loss. This knob sets the conversion method which specifies the representation used for storing a single floating-point value.

The conversion unit reduces the required memory space for an application and thus the total energy consumption required for memory accesses. A suitable configuration (knob setting) for the conversion unit can be determined by the configuration layer. Furthermore, the configuration layer can combine the proposed method with other accuracy-aware methods, which are part of the same layer or other layers.

4.1 Introduction

Approximate computing (AC) has been suggested as a possible means of increasing performance per watt at hardware layer [119, 18, 151]. Initial efforts on AC have proposed different versions of approximate multiplier and adder [7, 220, 154, 156, 151]. However, memory accesses consume a considerable amount of the energy in today's computing systems. For instance, an integer operation consumes 1,000$\times$ less energy compared to accessing a location in the L1 Cache [221]. Therefore, this chapter presents an approach that leverages this higher potential for energy savings. The focus lies on image applications that run on low power hardware and work with floating-point values.

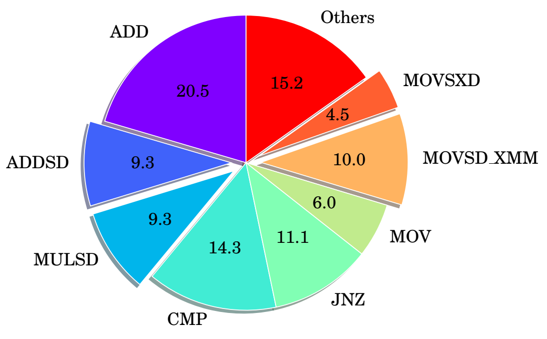
Let us consider two algorithms, 2D Richardson-Lucy deconvolution [222] and fast

Fourier transformation (FFT). For these floating-point algorithms, it is possible to introduce approximation for the data-relevant operations. Approximating these operations only influences the QoR specified as mean square error in this section. Figure 4.1 highlights the amount of operations that can be approximated for the two algorithms in a pie chart. In total, the approximable operations represent ∼33% and ∼62% of the operations, respectively.

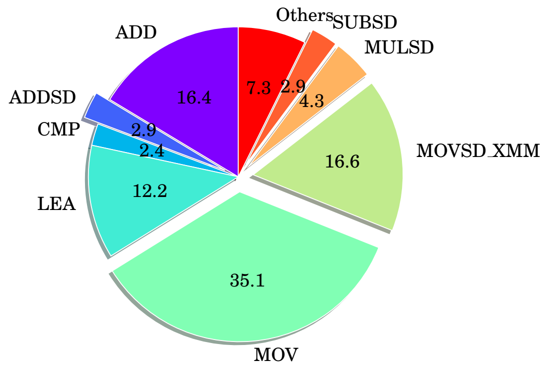
Looking at the consumed energy for the different operations, the approximable operations have a high impact on the energy consumption, see Figure 4.2. Hence, leveraging approximation in these operations is very beneficial to reduce the energy consumption. This statement also applies to arithmetic instructions that implicitly perform memory accesses. The challenge is how can we exploit this opportunity to reduce the energy consumption and keep the effect on the QoR as low as possible.

Current methods in AC exploit approximations to reduce the energy consumption of memories [8, 226, 227, 174]. Loading a value from these memories can result in a completely non-deterministic value. Cache compression methods miss a potential of approximating data [228]. Approximate cache methods are usually located on the last level cache [171, 172].

None of the previous work I mentioned considers the approximation of a single value. Therefore, in this chapter I present an accuracy-aware method that leverages this fine grained adjustment potential. The proposed method allows it to approximate floating-



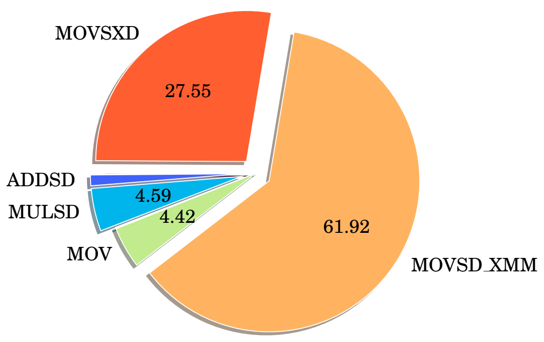
(a) Deconvolution



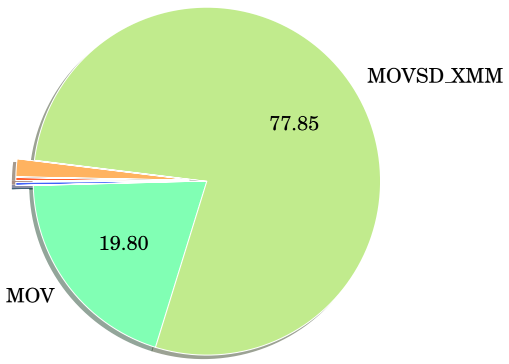
(b) FFT

Figure 4.1: Dynamic instruction mix of a 2D Richardson-Lucy deconvolution (a) and a 2D Fast Fourier Transformation [223] (b). Mnemonics that include *SD* in their name are floating-point relevant operations, hence are approximable for these algorithms. For FFT, the `MOV` operations mainly performs the bit reversal of floating-point values, hence are also approximable[1].

[1] I used the `Opcodemix` tool of Intel Pin [224], which is a dynamic binary instrumentation tool, to determine the dynamic instruction mix for both algorithms. As input, I took a 1024×1024 images with random pixel values. For Richardson-Lucy deconvolution, I specified 10 iterations.



(a) Deconvolution



(b) FFT

Figure 4.2: Estimation of the potential energy savings for the 2D Richardson-Lucy deconvolution (a) and FFT (b). The energy values for each operation are taken from [221] assuming 45 nm technology. According to the measured cache performance for both algorithms using *valgrind* [225], almost all data accesses result in a L1 cache hit. Hence, for each memory access the energy consumption for an L1 hit is assumed.[2]

[2]Energy for controlling instructions is not considered.

point values by converting the values to a representation that requires less bit. The conversion is applied, when a value is transferred to the first-level memory, which is normally the L1 cache. A converted value is used on the entire cache hierarchy and thus reduces the memory footprint of the application. Hence, less data has to be transferred from and to the main memory for converted values. Using an extra unit for the conversion makes it unnecessary to have processing units for different floating-point formats. On that account, the contributions of this chapter are:

- A novel runtime accuracy-aware method which approximates floating-point value is proposed. This method reduces the energy consumption and increase the performance of approximation-tolerant applications. A conversion unit internally uses different methods to reduce the amount of data that has to be transferred to the first memory level.
- The design and evaluation of different conversion techniques. Especially, a dynamic selection of these techniques provides a higher QoR, while retaining the benefit of reducing the energy consumption.
- A detailed measurement of the energy savings on different embedded platforms.
- The integration of the conversion unit into a recent RISC-V processor.
- An extensive evaluation of the conversion unit-extended RISC-V processor.

4.2 Preliminary QoR and Energy Measurements

A huge challenge for designing a conversion unit is to find conversion methods which have a low impact on the QoR for different applications. This section compares different possible conversion techniques. I integrate the best conversion methods into the conversion unit as described in a later section. Furthermore, I investigate the potential energy savings that can be achieved by the conversion unit for real hardware.

4.2.1 Accuracy Analysis of Different Conversion Methods

In this section, I identify possible conversion methods and compare the influence of these methods to QoR of different applications. For the required experimental tests, I make the following assumptions. All floating-point (FP) operations are performed in double (64 bit) according to the IEEE-754-based standard. Floating-point values that are stored in internal architecture registers are not converted. The conversion unit converts a floating-point value before transferring it to the L1 cache and deconverts it during a load.

Conversion methods. Table 4.1 summarizes techniques that I have identified for converting a 64-bit floating-point value (*FP64*) into one with 32, 21, or 16 bits.

The first technique (Op 0) converts a *FP64* value to one with less precision bits according to the IEEE-754 standard. Besides double precision (*FP64*), this standard defines single precision (*FP32*) and half precision (*FP16*). Since a 21-bit floating-point format (*FP21*) is not part of the IEEE-754 standard, I define this format using 1 sign bit, 5 bits for the exponent, and 15 bits for the mantissa. The motivation behind *FP21* is the possibility to pack three *FP21* values into a 64 bit word before transferring it to memory.

The second technique (Op 1) increases the value range by a factor of 2 for all FP data formats by not using negative superscripts. This is beneficial for applications, where it is tolerable to use the constant 0 for values between 0 and 1.

Opcode 2 and 3 convert a *FP64* value to a fixed-point representation *QX.Y* or *Q.Y*, where X is the number of bits for the integer and Y for the fractional part. The conversion is achieved by dividing the *FP64* value by an adaptable scalar value.

Table 4.1: Proposed techniques for converting a *FP64* value to one with lower precision. Signed numbers are represented by a sign bit.

Opcode (Op)	Conversion technique	Application's value range
0	IEEE-754 standard (*FP32*, *FP21*, *FP16*)	high value range
1	values < 1 set to zero	small numbers are negligible
2	Unsigned/signed *QX.Y*	small value range
3	Unsigned/signed *Q.Y*	small value range (adapting the scalar value improves accuracy)

Additional to statically defined precision, a runtime technique that dynamically selects between the number of used bits (16, 21, or 32 bit) for the converted value is considered. The programmer can set a threshold j, which specifies the allowed maximum absolute approximation error for a conversion into a certain lower data type. The conversion unit uses the smallest data format for which the resulting conversion error is less than that given threshold. Hence, a programmer can adapt the threshold in order to trade off accuracy for energy consumption and total memory latency. This technique is especially useful if some parts of an algorithm need more accurate calculations. An example are mixed precision methods, such as mixed-precision iterative refinement [229].

Evaluation setup. `Octave`[1] is used for a rapid prototype implementation. Pixel intensities are chosen randomly between 0 and 255 (assuming 8 bit camera sensors) or between 0 and 65536 (assuming 12 bit camera sensors) for an input image of size 1024×1024. The Mean Square Error (MSE) is sufficient for investigating how different conversion techniques influence the QoR.

$$MSE(x, y) = \frac{1}{n} \sum_{i=0}^{n-1} (x_i - y_i)^2,$$

where x_i is the ith result pixel of a pure *FP64* implementation and y the ith result pixel of an execution using a certain conversion technique.

In the following figures, lines named with *FP16*, *FP21*, and *FP32* are based on Opcode 0. Lines with *Q8.8*, *Q8.13*, and *Q8.24* are based on Opcode 2, and *Q.16*, *Q.21*, and *Q.32* are based on Opcode 3. Note that Opcode 1 is not applicable for the first two benchmarks because it relies on numbers between 0 and 1.

Corresponding lines for the dynamic data type are named with *dyn dt (th=j)*, where j specifies the error threshold. A line named with *Full FP32* means that all internal operations are natively executed in *FP32* and not *FP64*. Due to the absence of a *FP16* execution unit in the test system, we do not consider a *Full FP16* execution.

Benchmarks. The first benchmark is a *2D convolution*

$$I^{'}[x, y] = (I * f)(x, y) \equiv \sum_{m=-k/2}^{k/2} \sum_{n=-k/2}^{k/2} I(x - m, y - n) \cdot f(m, n),$$

where I is the original image, f is a $k \times k$ 2D Gaussian filter, $*$ the convolution operator, and $I^{'}$ is the blurred output image. The *2D convolution* is executed up to 10 iterations,

[1]`Octave` is a high-level interpreted language. It is primarily intended for numerical computations and hence poses a rapid way for performing numerical experiments. Moreover, it is an open-source alternative to MATLAB.

where $I^{'}$ is used as input for the following iteration. When the conversion unit is used, the pixels of image $I^{'}$ are converted.

A *2D fast Fourier transformation (2D FFT)* is the second benchmark [223]. A *2D FFT* is performed by row-wise *1D FFT*s followed by column-wise *1D FFT*s.

$$X(n) = \sum_{k=0}^{N-1} x(k) e^{-jk2\pi \frac{n}{N}}, n = 0...N-1$$

is a forward FFT, where $x(k)$ is a complex series with N samples. Opcode 0, Opcode 1, Opcode 3, and the dynamic method are considered for the *2D FFT*.

The maximum absolute value of $x(k)$ is used as a scalar value for Opcode 3. I also investigate to adaptively change the scalar value during the execution of the FFT. Initially starting from a scalar value of 256, the value is multiplied with 2 after each FFT butterfly. Finally, I integrate this *2D FFT* into an algorithm that reconstructs holography images acquired by lens-free microscopy [230]. I use the *spectral method* for the reconstruction.

The last benchmark is a *2D Richardson-Lucy deconvolution*

$$u^{(t+1)} = u^{(t)} \cdot \left(\frac{g}{u^{(t)} * K} * \widehat{K} \right),$$

where $u^{(t)}$ is the latent image, g the observed image, K a point spread function (PSF) and $\widehat{K}$ the flipped PSF. The conversion technique for the PSF values is fixed to *FP16*, *FP21*, and *FP32* (Opcode 0), respectively. Due to the higher value range of the algorithm, the *QX.Y* conversion methods are adapted to *Q11.5*, *Q11.10*, and *Q11.21*.

Results. Since pixel values coming from an image sensor are integers, converting the data to floating-point values does not lead to any approximation of the input data. The results shown in Figure 4.3 demonstrate the case, where the register set is able to store all values of the kernel f during the execution. Hence, the values of f were transferred as *FP64* values to the FPU register set.

The output of the first iteration is equal to the *FP64* execution ($MSE = 0.0$) because there is no approximation of kernel and input values at this point. The first approximation (conversion) is applied while storing the result image of the first iteration. The experiments have shown that the MSE of a *Full FP32* execution is slightly higher than an execution using *Q8.13* or *Q.21*, but about two orders of magnitude higher than the *FP32* and even seven orders of magnitude higher than *Q8.24* and *Q.32*. Using a 32-bit fixed-point method (*Q8.24* or *Q.32*) has resulted in a much smaller MSE than a *Full FP32* execution, but has the same possibility of data reduction. For the dynamic method (*dyn dt (th=0.01)*), the threshold 0.01 implies an usage of more *FP16* data types, hence the MSE is closer to the MSE of the *FP16* execution. More *FP32* data types were used for *dyn dt (th=0.0001)*, therefore the resulting MSE is closer to the MSE of the *FP32* execution.

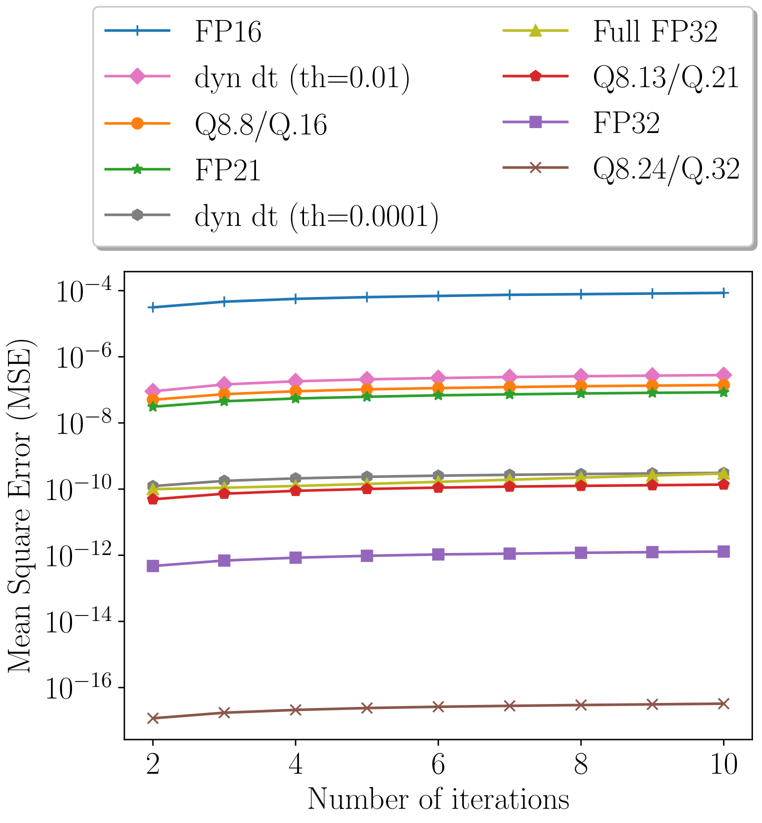


Figure 4.3: MSE values of 2D convolutions using different conversion methods without preconversion. As input, an image of size 1024×1024 with random values is used.

For the second test, the filter values of f were converted into different conversion formats before transferring it to the FPU registers (see Figure 4.4). This test demonstrates the case, where kernel values also have to be replaced from the register set during the execution. The fixed-point conversion formats (*QY.X* and *Q.Y*) have resulted in a higher MSE than their *FP* relatives. The *FP32* approach has slightly outperformed a full *FP32* execution.

The two mentioned tests were also applied to images with 12 bit pixel values.However, the trend was similar to 8 bit values, hence considerations of 12 bit image values are not presented here. To get a visual impression of the results, I have applied different conversion methods to the 2D convolution, that has the well-known “Lena” image as input, see Figure 4.5. As we can see, the visual difference between the exact convoluted image and the approximated one is low. The approximations cause no artifacts in the images, however, one can argue that the smearing effect is stronger for the approximated versions. Nevertheless, the proposed conversion unit allows us to adapt the quality loss for different users.

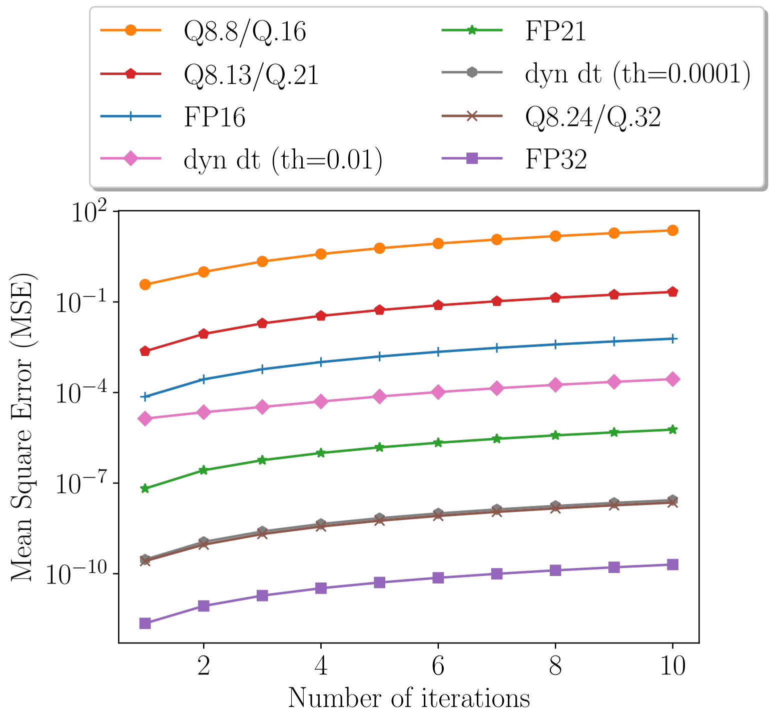


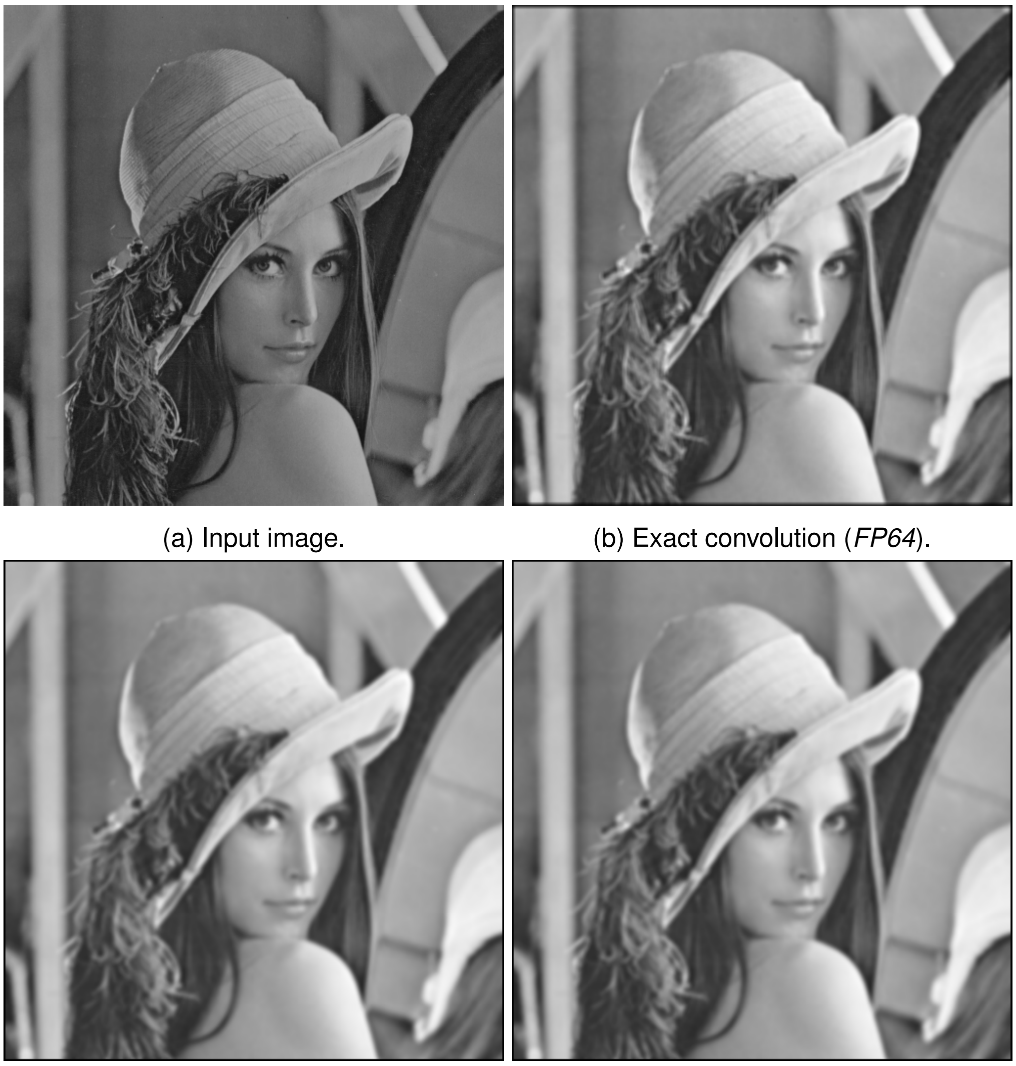
Figure 4.4: MSE values of 2D convolutions using different conversion methods with pre-conversion. As input an image of size 1024×1024 with random values was used.

Table 4.2: MSE values for 2D FFTs using different conversion methods and an image of size 1024×1024 with random values.

	Full FP32		**FP (op 0)**		**FP (op 1)**		**Q.Y (op 3)**		**Q.Y adapt. (op3)**	
	MSE	RD	MSE	RD	MSE	RD	MSE	RD	MSE	RD
16 bit	-	-	∞	4	1.60E+05	4	2.00E+10	4	9.87E+07	4
21 bit	-	-	∞	3	2.84E+02	3	3.66E+09	3	9.53E+04	3
32 bit	1.63E+02	2	1.67E-04	2	-	2	9.02E+02	2	2.27E-05	2
Dynamic method										
Threshold	1E00		1E-01		1E-03		1E-05		1E-07	
	MSE	RD	MSE	RD	MSE	RD	MSE	RD	MSE	RD
	1.16E+03	3.65	1.85E+02	3.14	2.75E-02	2.34	1.67E-04	2.21	1.67E-04	2.21

The third test studies the influence of the conversion techniques for the *2D FFT*. Table 4.2 shows the resulting MSE and the factor of reduced amount of data (RD) for the different techniques.

Most of the static techniques using either 16 or 21 bits are presumably not applicable in real applications. The reason is the large value range of outputs of the *2D FFT* algorithm. The only exception is Opcode 1 using a 21 bit data type. A *Full FP32* ex-



(a) Input image.

(b) Exact convolution (*FP64*).

(c) *FP16*-based (MSE ≈ 0.0005).

(d) *Q8.8*-based (MSE ≈ 5).

Figure 4.5: Visual effect of the *2D convolution* using different conversion methods compared to the *FP64* method.

ecution, where all operations were performed in 32 bit, had a higher MSE compared to *FP32*. Formats based on Opcode 1 have reduced the MSE compared to *FP16* and *FP21*. Adapting the scalar value for *Q.32* has resulted in a smaller MSE compared to *FP32*. Compared to *Full FP32*, the dynamic method with a threshold of 0.1 had roughly the same MSE, but has reduced the amount of data by a factor of about 3. Hence, the dynamic method is usefully applicable for the *spectral method*.

The different conversion techniques were applied to the *2D FFT*s, which are part of the *spectral method*, see Table 4.3. Using *FP32* has resulted in a MSE of 0.0. The

Table 4.3: MSE values for the spectral method that reconstructs holography images using different conversion techniques for the *2D FFT*.

FP16 (Op 0)		**FP21** (Op 0)		**FP32** (Op 0)	
MSE	RD	MSE	RD	MSE	RD
4.93E-02	4	4.78E-02	3	0.00E+00	2

Dynamic approach

threshold=1		threshold=0.5		threshold=0.1		threshold=0.01	
MSE	RD	MSE	RD	MSE	RD	MSE	RD
2.16E-04	3.94	1.63E-04	3.91	8.00E-05	3.77	1.40E-05	3.35

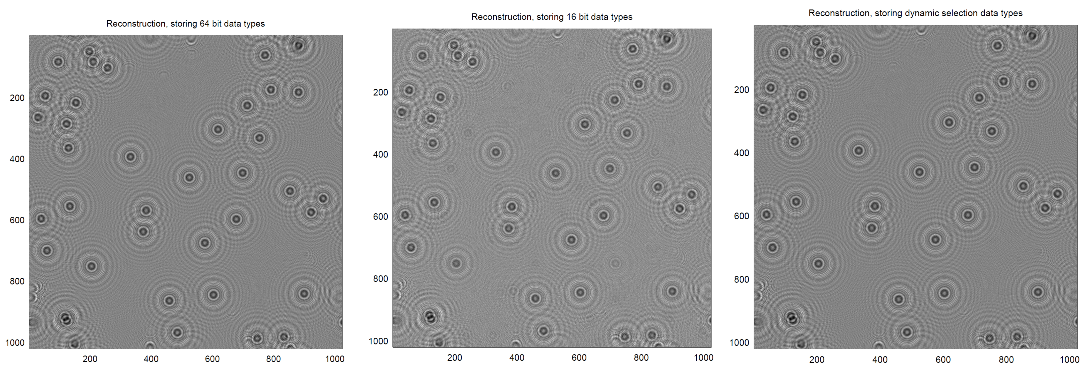

Reconstruction, storing 64 bit data types
Reconstruction, storing 16 bit data types
Reconstruction, storing dynamic selection data types
200
400
600
800
1000

Figure 4.6: Spectral method for holography reconstruction. Images are calculated using *FP64* (left), *FP16* (middle), and *dynamic method* with threshold 0.001 (right).

dynamic method has reduced the MSE by two orders of magnitude, while slightly decreasing the RD compared to *FP16*. Figure 4.6 shows the output of the spectral method using different conversion techniques. While we can see some artifacts for *FP16* (middle image), the *dynamic method* (right) visually matches with the exact image (left).

The last test considers different conversion techniques for *2D Richardson-Lucy deconvolution*. In each case, the same conversion method was used for values of the intermediate results and the output image. The results strengthen the already determined results. A brief summary is given in the following. The MSE of a *Full FP32* execution was higher than our *FP32* approach. The FP approaches (*FP16*, *FP21*, *FP32*) had a smaller error than their fixed-point relatives (*Q.16*, *Q.21*, *Q.32*, *Q11.5*, *Q11.10*, and *Q11.21*).

Findings. According to the results, it is clear that frequently used data such as kernel and filter values should be stored in the register set without conversion in order to get a high QoR. Moreover, it is sufficient to support the conversion methods with an Opcode 0/1 (*FP32*, *FP21*, *FP16*), because they achieve the best results in terms of MSE. It is also shown that *Q.Y* yields a lower MSE for algorithms with a small value range. Additionally, this mode enables a programmer to calculate other parts of an application using fixed-point operations. For example, parts of the algorithm that need a higher QoR can be calculated using *FP64* hardware combined with the conversion unit and the other parts can be calculated using fixed-point arithmetic. The conversion unit avoids to have an explicit cast of variables used in both parts.

4.2.2 Measuring the Potential Energy Savings

In order to measure the energy consumption of an algorithm using different conversion techniques, memory footprints of the *2D FFT* benchmark are created.Because *FP21* is not supported in current computer systems, only *FP16*, *FP32*, and *FP64* are considered. Using these extracted footprints, the energy consumption is measured on existing embedded compute platforms. The function `memcpy`, which is part of the standard C library, is employed for data transfers.

The used platforms are the Odriod-XU [231], the Parallela board [232], and the Myriad 1 development board [233]. An overview of the integrated compute units in each platform is given in Table 4.4. These platforms offer some of the lowest energy operations among commercially available platforms.

The Odriod-XU includes a Samsung Octa processor, which integrates a Cortex-A15 processor and a Cortex-A7 processor. This processor is based on the ARM big.LITTLE architecture. An operating system can switch between both processors depending on the current workload and the required performance, but the processors cannot be used concurrently. The frequency of the Exynos microprocessor is decided based on which processor the benchmark runs.

A script is used to measure the energy consumption of the Odriod-XU. This script reads out hardware performance counters that include values for voltage, current, and power of the A7 processor, A15 processor and the main memory. To measure the energy consumption for all four available cores, I execute the extracted memory footprints on each core. An infinite loop around the `memcpy` calls are used to get a stable value of the electric power. The execution time is measured in multiple runs of the original implementation (without the infinite loop) and averaged.

The Parallela board that includes a Dual-core ARM A9 (600 MHz) processor and

Table 4.4: Overview about the considered platforms.

Platform	**Host processor** [Technology]	**Coprocessor** [Technology]
Odriod-XU	Exynos 5 Octa (5410) [28 nm]	PowerVR SGX544MP3 GPU [28 nm]
Parallela	Zynq-7010 [28 nm]	E16G301 [65 nm]
Myriad 1	Leon3 [65 nm]	SHAVEs [65 nm]

a 16-core Epiphany (666 MHz) was developed for energy-efficient execution of high-performance applications. The Odriod Smart Power that is a deployable power supply is employed to specify a voltage and to measure the current and the electric power of the Parallela board. The Odriod Smart Power measures the values of the entire board.

I run two threads on the A9 processor to get the energy consumption for the A9 processor exclusively. To measure the electric power of the A9 processor together with the Epiphany, I start two threads on the A9 processor and 16 threads on the Epiphany. Each thread executes the memory footprint. The 2D FFT data is transferred from the host memory to the local memory by Direct Memory Access (DMA). On each Epiphany core, `memcpy` transfers data to another region in the local memory of the core.

The Myriad 1 combines a Leon processor with eight Streaming Hybrid Architecture Vector Engines (SHAVEs). A SHAVE consists of features from Reduced Instruction Set Computing, Very Long Instruction Word processors, Digital Signal Processors and Graphic Processing Units. For measuring the electric power of the Myriad 1, power cables are directly connected to supply the processor with electrical energy. A switched-mode power supply together with an ampere-meter enable it to specify the voltage and to measure the current. To transfer the memory footprints inside the host memory, the function `memcpy` is used on the Leon processor. In the second run, eight DMA units, that are assigned to each SHAVE, transfer data to the local memory. All SHAVES are used to transfer data inside the local memory according to the memory footprints.

The energy consumption is calculated by multiplying the execution time (of the original implementation) with the measured electric power (determined with an infinite loop integrated into the implementation). The results of all setups are visualized in a bar chart (Figure 4.7). The initial expectation is that reducing the amount of the application's data results in a reduction of the energy consumption almost in the same range. According to the measurements, this expectation is valid. The main contribution for the energy reduction is the reduced number of cache misses.

4.3 Conversion Unit

As the former section has revealed, converting individual floating-point values is very beneficial in terms of reducing the required energy consumption for memory accesses. Furthermore, it offers a higher precision than performing the entire algorithm using a native lower precision format such as *Full FP32*. It also has often a negligible impact on the QoR. To exploit the potential of converting values, a conversion unit (CU) is designed.

Figure 4.8 presents the structure of the CU. The CU converts a *FP64* value into a value with less bits performed by the *Converter* before storing it to memory and thus it is a lossy operation. The used conversion technique depends on the opcode which is given to the *Converter*. During a read access, the stored value is converted back to *FP64* without an accuracy loss using the *Deconverter*. Specifying the opcode for both units is done via

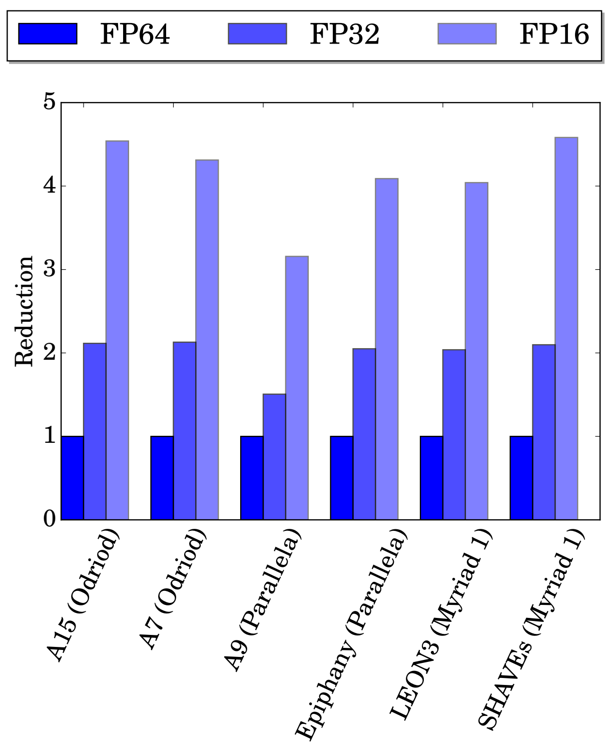


Figure 4.7: Reduction of the energy consumption using data formats with less precision.

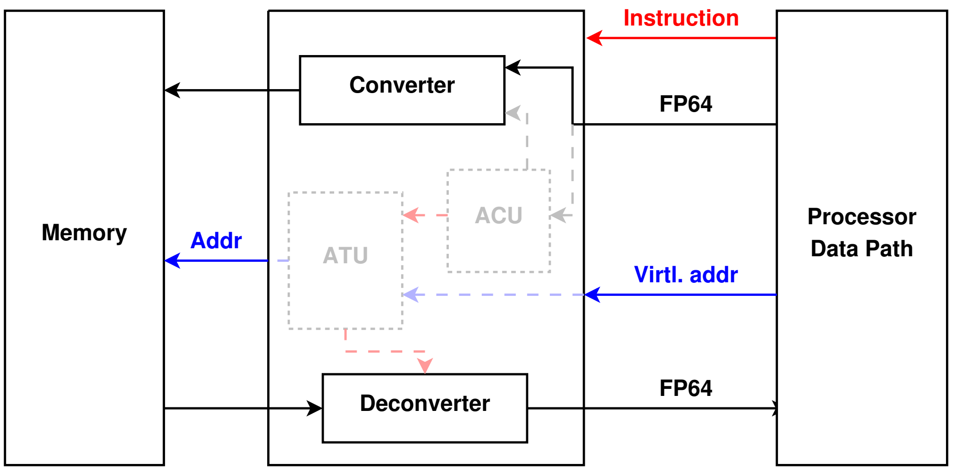


Figure 4.8: Structure of the conversion unit.

an instruction or via memory mapping as described later. Furthermore, when using the dynamic method for selecting the conversion method, then the decision is based on an approximation check unit (*ACU*). The *ACU* decides, which conversion method is used depending on the introduced numerical error that will occur after the conversion. The

Table 4.5: Area and estimated power for the *Converter* and the *Deconverter* for different target frequencies. TSMC 28nm HPM High Speed library is used for the synthesis.

	500 MHz		**600 MHz**	
	Area $[\mu m^2]$	**Static power** $[\mu W]$	**Area** $[\mu m^2]$	**Static power** $[\mu W]$
Converter	1958.681	336.900	2187.844	435.100
Deconverter	1199.812	188.300	1214.325	225.800

Table 4.6: Area and estimated power for the considered approach for the Approximation Check Unit (ACU). TSMC 28nm HPM High Speed library is used for the synthesis.

Consumption			
500 MHz		**1000 MHz**	
Area $[\mu m^2]$	**Power** $[\mu W]$	**Area** $[\mu m^2]$	**Power** $[\mu W]$
97.875	10.043	129.600	25.044

ACU is parametrized using the threshold j as described in Section 4.2.1.

However, supporting the dynamic method would also require an address translation unit (*ATU*), which also has to store further information about the conversion method that was used for a specific stored value. The *ATU* also has to avoid fragmentation inside the memory. To fully exploit the bandwidth, values should be collected before transferring it to memory. These aspects are not further considered in this thesis.

4.3.1 Preliminary Design

As a first step to realize such a CU, the *Converter* and the *Deconverter* are implemented in Verilog. The preliminary CU supports Opcode 0, 1, and 3. Instead of truncation, the round to nearest method is implemented. The *Q.16* and *Q.32* data formats are converted by scaling the *FP64* value. Each conversion is performed in one clock cycle.

Using the Synopsys tools, we get an estimation about the power and the area of the units. The TSMC 28nm HPM High Speed library is used for the synthesis. The results are summarized in Table 4.5. Compared to the measured values in Section 4.2.2, these units do not significantly increase the energy consumption. Table 4.6 shows the area and power overhead for a simple *ACU*.

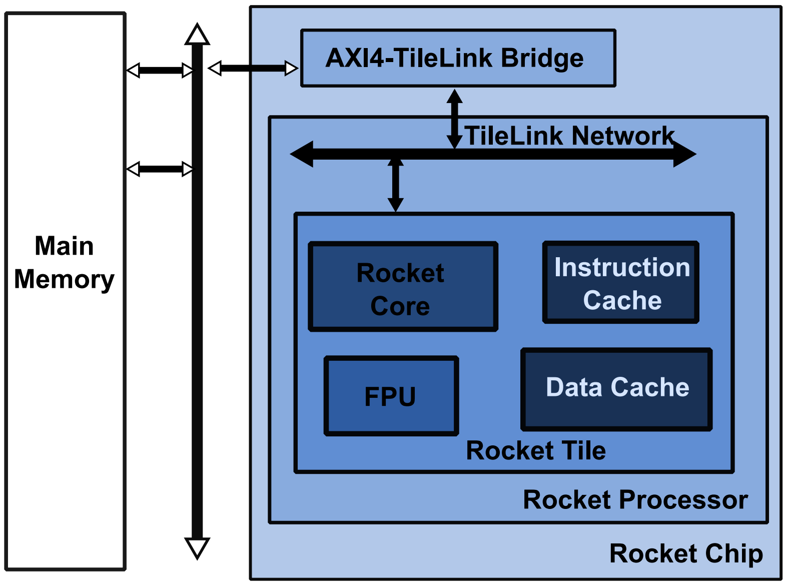


Figure 4.9: Structure of the the used basic Rocket Chip implementation.

4.3.2 Integration Into an Existing Processor

In the following, I consider a conversion unit, which is controlled by a memory mapping scheme. I have integrated the CU into an in-order general purpose processor. I have selected a RISC-V-based system on chip called Rocket chip [234]. A publicly and freely available `Rocket Chip Generator`, which is a system on chip generator, has allowed me to specify a configuration for a Rocket chip [235]. Possible configurations include the address and data width, the cache hierarchy, floating-point unit support and number of cores. Additionally, one can select a support for virtual memory. Besides the hardware description, a simulation environment and a GNU/GCC toolchain to run assembly or C code on the processor is available.

Before the integration of all CU relevant hardware into the processor, I have configured a basic processor for evaluation purposes, see Figure 4.9. I have configured the basic version by setting the number of rocket tiles to one and using a 64-bit address and data width. The rocket tile includes a data and an instruction cache, while I have not integrated other caches. I have deactivated the support for virtual memory.

In a next step, I have integrated the CU into the basic processor. The memory location of the FP value specifies the technique, which is used for the conversion, see Figure 4.10. This approach poses a memory mapping scheme.

All values in a certain memory region named `CU region` are converted before storing it to memory and converted back during a read access. The used CU version for the evaluation supports Opcode 0, i.e. `FP64` to `FP16/32`. That means the CU applies the same conversion method for all values in the `CU region`.

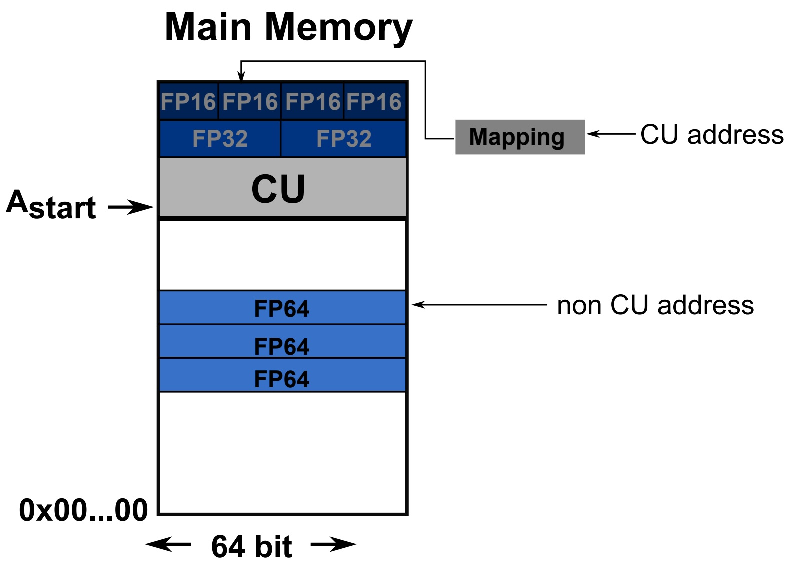


Figure 4.10: A certain address space in the main memory is memory mapped as the `CU region`. This region is located between address A_{start} and the end of the physical address space. Each access to that region activates the address mapping and the CU.

To avoid adaptations of the compiler, the processor supports the required address mapping. Moreover, a write mask specifies which part of a 64 bit memory location stores the converted value. Memory addresses that belong to the `CU region` are mapped to an effective memory address. The following formulas are used for the mapping.

$$A_{phys,CU16} = A_{start} + \frac{A - A_{start}}{4}$$
$$A_{phys,CU32} = A_{start} + \frac{A - A_{start}}{2}$$

The integration of the CU affects two parts of the basic processor, see Figure 4.11. Firstly, there exist a `Store CU` and a `Load CU`, which are implemented in the FPU. Secondly, the controller of the CU (`CU control`) and the address mapping (`Mappping`) are part of the rocket core.

During a store, a `FP64 value` from the register set is converted, when the associated memory address is within the `CU region` The conversion is performed by the `Store CU`. The required indication comes from the `CU control` that activates the `use_scu` signal. Since the data bus is 64-bit wide, the `FP16` value is concatenated with 48 zeros. Then, the rocket core sends the data word, the address, and the write mask to the data cache.

When a FP value within the `CU region` is transferred from the data cache via the core to the FPU, the `CU control` activates a signal to the FPU (`use_lcu`). The transfer is requested by a read access to `CU region`. The `use_lcu` signal is set to one if the FP value is a `FP16/32`.

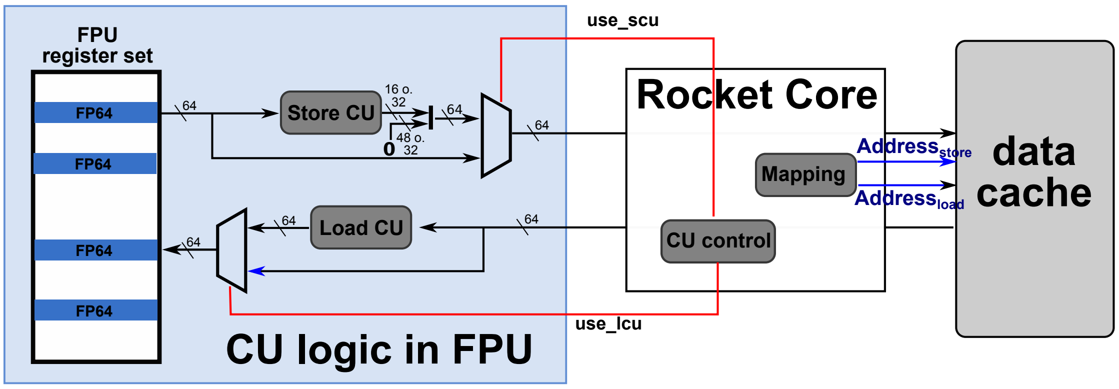


Figure 4.11: Integration of the conversion unit into the rocket processor.

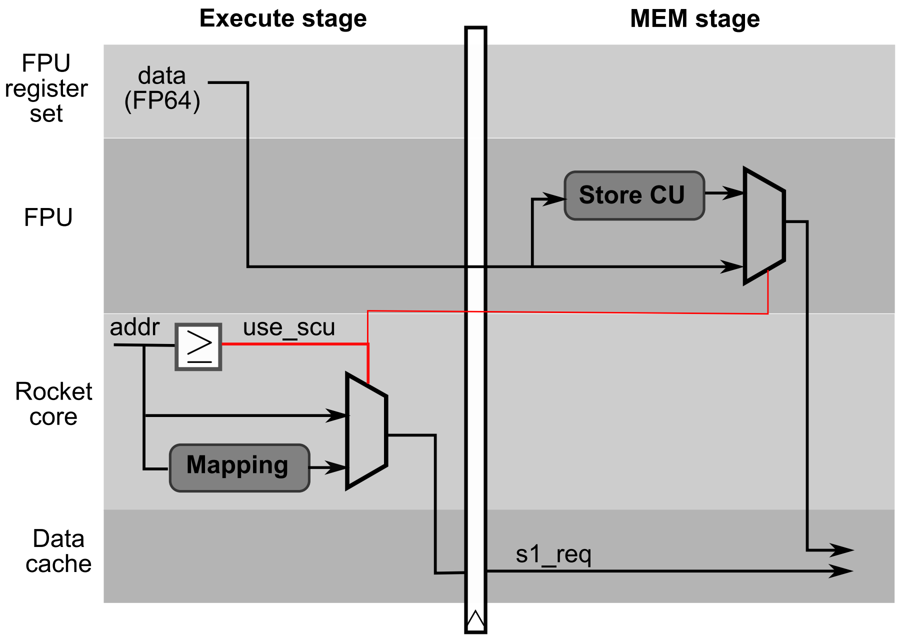


Figure 4.12: Time diagram for a write memory access to the `CU region`.

Figure 4.12 shows a time diagram of storing a `FP16/32` to the `CU region`. In the execute stage of the instruction pipeline, the decision is made whether the mapped address or the original address is required. In case that the address is part of the `CU region`, the signal `use_scu` is set. The effective memory address is sent together with a write mask (`s1_req`) to the data cache. In the MEM stage, the conversion is performed and depending on the `use_scu` signal the converted value is transferred to the data cache.

All extensions to the basic processor are written in Scala. The configuration files of the rocket chip generator are adapted in order to specify the basic version, a version with a `FP16` CU, and a version with a `FP32`.

Synopsys tools are used to simulate and synthesize the different versions. Prime Time PX is employed to apply a power analysis. As the required digital standard cell library, the SAED 32/28nm library is used. This library is freely and publicly available from Synopsys for research purposes. The selected technology file offers gates with a low threshold voltage, hence leads to the highest possible frequency provided by the SAED 32/28nm library. This file is called `ff1p16v25c`.

Each processor is synthesized for 400 MHz, i.e. a timing constraint of 2.5 ns. Table 4.7 shows several information about the basic configuration of the processors. Using the rocket chip generator and the Synopsys design compiler, three different versions of a processor are implemented and synthesized.

- Basic processor
- Processor with `FP16` CU
- Processor with `FP32` CU

4.4 Results and Evaluation

First of all, let us consider the area consumption of the different versions, see Table 4.8. The high-level synthesis tool of Synopsys, design compiler, determines the chip area by taking the area of all gates into account. However, it does not incorporate additional resources such as wires.

The basic version requires $101,211$ cells in total, where the `CU16`-based processor requires $100,321$ cells and the `CU32`-based processor $101,865$ cells. Hence, the amount of cells is roughly the same for all versions. The small differences are caused by the optimization algorithm of the synthesis tool. Due to different optimization strategies during

Table 4.7: Configuration of the processor, the data cache, and the main memory using the rocket chip generator.

Processor		**Main memory**	
Number of Tiles	1	Size	256 MiB
Number of integer register	32	Data size of an access	64 B
Number of floating-point register	32		

Data cache	
Size	16 KiB
Line size	64 B
Number of sets	64
Associativity	4

Table 4.8: Area consumption of the different designs.

	Basic Processor Area [μm^2]	**CU16-based Processor** Area [μm^2]	**CU32-based Processor** Area [μm^2]
Chip	329,076	327,119	330,653
| Processor	298,455	296,470	299,764
| | Rocket Tile	239,713	237,995	241,131
| | | FPU	134,185	132,292	135,181
| | | | `Load CU`	-	191	477
| | | | Reminder FPU	134,185	132,101	134,704
| | | Rocket Core	61,768	61,898	62,216
| | | Reminder Tile	43,760	43,805	43,734
| | Reminder Processor	58,742	58,475	58,633
| Reminder Chip	30,621	30,649	30,889

Table 4.9: Static power consumption of the different designs.

	Basic Processor Power [mW]	**CU16-based Processor** Power [mW]	**CU32-based Processor** Power [mW]
Chip	8.79	8.74	8.82
| Processor	7.82	7.77	7.85
| | Rocket Tile	6.39	6.35	6.42
| | | FPU	3.65	3.60	3.67
| | | | `Load CU`	-	$5.56 \cdot 10^{-3}$	$14.9 \cdot 10^{-3}$
| | | Rocket Core	1.63	1.63	1.64

the synthesis and the distributed implementation of the CU, only the area consumption of the `Load CU` is available in the reports. The 16 bit version consumes roughly 0.2% of the chip and the 32 bit version 0.5%. Table 4.9 shows the static power consumption for the different versions. Similar to the area consumption, the power consumption is roughly the same for all designs.

The power and energy consumption of the caches, which include SRAM modules, and the main memory is determined using CACTI 5.3 [236][3]. The result for area, power, and energy consumption is shown in Table 4.10. The static power consumption of the main memory (63.6 mW) is significantly higher than that of the chip (8.8 mW). Compared to that, the SRAM modules have a negligible static power.

I apply three benchmarks to evaluate the designs: a *2D convolution* with a 3×3 filter and an image of size 64×64, a *2D convolution* with a 7×7 filter and an image of size 512×16, and a *2D FFT* on an image of size 2048×4. The relatively small input size is because of the high consumed time for analyzing the energy consumption using PrimeTimePX. The input images are taken from a subset of the well-known "Lena" image.

[3]I used the web-interface of CACTI5.3 that was available at `quid.hpl.hp.com:9081/cacti/`

Table 4.10: Power and energy consumption analysis of the caches and the main memory using CACTI 5.3.

	Area [μm^2]	**Static power** [mW]	**Energy per access** [nJ]
Data cache	$54,364$	$5.33 \cdot 10^{-4}$	5.09
Instruction cache	$47,871$	$5.78 \cdot 10^{-4}$	2.87
Main memory	$68.6924\,mm^2$	63.594	31.99

Table 4.11: Different performance metrics for the considered benchmarks.

	2D Convolution (3×3)			**2D Convolution (7×7)**			**2D FFT**		
	Basic	CU16	Change	Basic	CU16	Change	Basic	CU32	Change
Execution time	1.508 s	1.445 s	1.04×	10.93 s	8.09 s	1.35×	27.51 s	8.66 s	3.18×
# cache misses	1750	397	4.41×	61.435	892	68.87×	442,166	9159	48.28×
# memory accesses	1741	394	4.42×	61,339	794	77.26×	442,037	9163	48.24×
# read accesses	1292	340	3.8×	56,962	687	82.91×	301,108	6336	47.52×
# write accesses	449	52	8.63×	4377	107	40.91×	140,929	2827	49.85×

Performance. Table 4.11 presents performance metrics of the different benchmarks. It comprises the execution time, the number of cache misses, and the amount of memory accesses. For the first benchmark, the number of memory accesses is reduced by a factor of 4.4 using the 16 bit CU. However, the execution time is only slightly decreased. The reason is that for the basic version and the CU16 version the cache is large enough to store all required pixels for the computation. Increasing the convolution kernel, which is done for the second benchmark, we can see that the number of cache misses drastically decreases ($68.9\times$). The execution time is 4.2% faster for the CU16 in this case. But the main advantage of the CU is for applications that have many memory accesses such as the `2D FFT`. Here, the cache misses are reduced by a factor of 48.2 but the execution time is also drastically reduced (factor of 3.2).

Theoretically, a frequency of 400 MHz allows the execution of 400 M instructions per second. Applications, where many of these instructions are memory accesses on FP values, especially, when the working set is larger than the L1 cache, benefit from the CU regarding performance. For instance, the benchmark 2 has a memory access every 75th cycle on average. Compared to that, the FFT benchmark has an access every 25th cycle. Moreover, this benefit increases for irregular memory accesses.

Energy and power consumption. I investigate dynamic power and consumed energy for the benchmarks. PrimeTime PX determines an averaged dynamic power P_{dyn}. The dynamic energy is $E_{dyn} = P_{dyn} \cdot t_{exec}$. As the used gate library does not allow us to get power and energy values for memories, the values for the SRAM modules and the main memory are determined by using the values from Tables 4.10 and 4.11. Tables 4.12

Table 4.12: Power and energy of the 2D convolution with a 3×3 kernel (benchmark 1).

	Dynamic Power Consumption [mW]			**Dynamic Energy Consumption** [mJ]		
	Basic	CU16	Diff	Basic	CU16	Diff
Chip	88.6	91.4	+3.2%	133.6	132.1	-1.1%
| Processor	86.7	89.5	+3.2%	130.7	129.3	-1.1%
| | Rocket Tile	79.6	83.0	+4.3%	120.0	119.9	-0.1%
| | | I Cache	12.9	13.3	+3.1%	19.5	19.2	-1.2%
| | | | SRAM	$2.2{\cdot}10^{-3}$	$2.18{\cdot}10^{-3}$	-1.0%	$3.32{\cdot}10^{-3}$	$3.15{\cdot}10^{-3}$	-5.1%
| | | FPU	32.1	36.1	+12.5%	48.4	52.2	+7.8%
| | | | Load CU	-	$60 \cdot 10^{-3}$	-	-	$86.7{\cdot}10^{-3}$	-
| | | | Rest of the FPU	32.1	36.0	+12.3%	48.4	52.1	+7.6%
| | | Rocket Core	30.8	30.5	-1.0%	46.4	44.1	-5.1%
| | | Rest of the tile	3.8	3.1	-18.4%	5.7	4.5	-21.8%
| | | | D Cache SRAM	$1.07{\cdot}10^{-3}$	$1.02{\cdot}10^{-3}$	-5.3%	$1.62{\cdot}10^{-3}$	$1.47{\cdot}10^{-3}$	-9.3%
| | Rest of the processor	7.1	6.5	-8.5%	10.7	9.4	-12.3%
| Rest of the chip	1.9	1.9	0.0%	2.9	2.7	-4.2%
Main Memory	0.0369	0.0087	-76.4%	0.0557	0.0126	-77.4%
Total	88.6	91.4	+3.1%	133.6	132.1	-1.2%

Table 4.13: Power and energy of the 2D convolution with a 7×7 kernel (benchmark 2).

	Dynamic Power Consumption [mW]			**Dynamic Energy Consumption** [mJ]		
	Basic	CU16	Diff	Basic	CU16	Diff
Chip	80.9	100.5	+24.2%	884.24	813.05	-8.1%
| Processor	79.0	98.8	+25.1%	863.47	799.29	-7.4%
| | Rocket Tile	71.8	92.2	+28.4%	784.77	745.90	-5.0%
| | | I Cache	11.6	11.4	-1.7%	126.79	92.33	-27.3%
| | | | SRAM	$2.38{\cdot}10^{-3}$	$2.45{\cdot}10^{-3}$	+3.0%	$65.4{\cdot}10^{-3}$	$21.2{\cdot}10^{-3}$	-67.6%
| | | FPU	29.3	46.5	+58.7%	320.25	376.19	+17.5%
| | | | Load CU	-	$90.90 \cdot 10^{-3}$	-	-	0.735	-
| | | | Rest of the FPU	29.3	46.4	+58.4%	320.2	375.4	+7.6%
| | | Rocket Core	26.3	30.3	+15.2%	287.46	245.13	-14.7%
| | | Rest of the tile	4.6	4.0	-13.0%	50.3	32.4	-35.6%
| | | | D Cache SRAM	$2.02{\cdot}10^{-3}$	$1.01{\cdot}10^{-3}$	-50.1%	$55.7{\cdot}10^{-3}$	$8.75 \cdot 10^{-3}$	-84.3%
| | Rest of the processor	7.2	6.6	-8.3%	78.7	53.4	-32.2%
| Rest of the chip	1.9	1.7	-10.5%	20.8	13.8	-33.8%
Main Memory	0.1795	0.0031	-98.3%	1.9623	0.0254	-98.7%
Total	81.1	100.5	+24.0%	886.2	813.07	-8.3%

to 4.14 shows P_{dyn} and E_{dyn} for the designs and benchmarks including parts of the chip and the main memory.

For the benchmark 1, the average dynamic power is 3.1% higher for the design with the CU16 compared to the basic processor. This is mainly caused by the FPU, which consumes 12.5% more dynamic power. However, this increase is not due to the Load CU. Since the Load CU only requires 0.3 μW. It seems that the reduced idle time of the FPU caused by less cache misses is responsible for the higher dynamic power. This trend is similar for all benchmarks. Additionally, the Design Compiler creates different

Table 4.14: Power and energy of a 2D FFT (benchmark 3).

	Dynamic Power Consumption [mW]			**Dynamic Energy Consumption** [mJ]		
	Basic	`CU16`	Diff	Basic	`CU16`	Diff
Chip	63.1	81.4	+29.0%	1735.88	704.92	-59.4%
| Processor	61.2	79.4	+29.7%	1683.61	687.6	-59.2%
| | Rocket Tile	52.3	72.9	+39.4%	1438.77	631.31	-56.1%
| | | I Cache	8.0	8.7	+8.7%	220.08	75.34	-65.8%
| | | | SRAM	$2.45\cdot10^{-3}$	$2.13\cdot10^{-3}$	-13.3%	$26.8\cdot10^{-3}$	$17.2\cdot10^{-3}$	-35.8%
| | | FPU	26.7	42.8	+60.3%	734.52	370.65	-49.5%
| | | | Load `CU`	-	$3.04\cdot10^{-3}$	-	-	$26.3\cdot10^{-3}$	-
| | | | Rest of the FPU	26.7	42.8	+60.3%	734.5	370.6	-49.5%
| | | Rocket Core	11.6	17.7	+52.6%	319.12	153.28	-52.0%
| | | Rest of the tile	6.0	3.7	-38.3%	165.1	32.0	-80.6%
| | | | D Cache SRAM	$1.51\cdot10^{-3}$	$1.26\cdot10^{-3}$	-16.5%	$16.5\cdot10^{-3}$	$10.2\cdot10^{-3}$	-35.8%
| | Rest of the processor	8.9	6.5	-27.0%	244.8	56.3	-77.0%
| Rest of the chip	1.9	2.0	+5.3%	52.3	17.3	-66.9%
Main Memory	0.5140	0.0383	-93.4%	14.1414	0.2931	-97.9%
Total	63.6	81.4	+28.0%	1750.02	705.22	-59.7%

versions for the FPU for both designs as discussed above. The reason is that the synthesis tool choose a different trade-off between area, static, and dynamic power. But the reduced number of cache misses leads to a shorter execution time, hence the processor with CU needs slightly less dynamic energy (-1.2%). The main benefit of the CU is visible for the main memory. Here, the dynamic power and the dynamic energy is reduced by a factor of roughly 4.4. *Rest of the tile* includes among others the consumption of the data cache. We can see that the energy consumption is reduced by 21.8% for this part of the chip. Additionally, *Rest of the processor* and *Rest of the chip* compromise the consumption for the processor-internal bus (TileLink), the processor-external bus (AXI4), and the bridge between both buses. While these values do not not include the wires itself or the main memory connection to the bus, it gives a relative view on the difference between the designs. My method reduces the traffic on the bus and hence reduces the energy consumption of bus system modules by roughly 10%.

Considering the results of benchmark 2, we see that the higher amount of cache misses leads to a smaller power consumption for the basic chip (80.9 compared to 88.6 for benchmark 1). The rationale is, as discussed above, the higher utilization of the system components for the first benchmark. The modules remain idle for less cycles, while waiting for data from the main memory. Compared to that, the power consumption of the CU16 version increases for the second benchmark. This can especially be seen for the FPU and the Rocket Core. This behavior is reversed for the main memory. Here, the power consumption for the basic version is higher for benchmark 1 than for benchmark 2, where the power consumption decrease for the CU16 version. Similar to benchmark 1, the power and energy consumption are mostly reduced for all memory-related parts of the CU16 design. This leads to an entire energy reduction of 8.3%.

The last benchmark, FFT, clearly illustrates the discussed aspects. While the power of components increased due to the higher utilization, the dynamic power significantly decreased for memory-related components. Therefore, the energy consumption of the entire CU32-based design is 2.5 smaller compared to the basic version.

4.5 Related Work

Current AC approaches at the hardware layer often focus on reducing the precision of hardware execution units [237, 153]. Instruction Set Architecture (ISA) extensions are used to specify which operations or memory regions are executed using high or low voltage [10]. Cache misses caused by loads are very expensive in terms of latency and energy consumption in modern architectures. Instead of loading a value that is missing in the cache, an approximated value is generated according to a history of loaded values [169]. An often used method to decrease the memory footprint is to exploit cache compression techniques [238, 228, 239, 240]. However, these methods do neither take advantage of lossy techniques nor show the same benefit for floating-point data [241, 171]. In case of FP values, a FP block format, in which several mantissae use the same exponent, reduces the amount of data [242].

Doppelgänger maps approximately equal cache lines to the same physical location in the LLC to reduce the required cache size [171]. While the performance is almost similar to the baseline architecture, Doppelgänger saves static and leakage energy. The Bunker Cache exploits the fact that often real-world data shows the behavior of *spatio-value similarity* [172]. On that account, San Miguel et al. argue that data that have similar content are located nearby in the memory. They map such memory addresses to the same cache line in order to reduce off-chip transfers and decrease execution time.

The novel approach I presented is orthogonal to the cache related approaches explained above. Therefore, my approach can be used together with them towards higher performance or energy efficiency. The presented CU operates on a single value. A newer work from Jain et al. (published after my approach) also exploits the aspect of using a different data format in computation and storage [243]. This newer work uses an ISA extension to integrate the so-called *asymmetric compute-memory extension*. However, their approach is a pure static or compiler-based approach. The presented CU method introduces a dynamic behavior, since the approximation decision is based on a memory mapping. During runtime, an approximation can be used by storing data into the *CU region*. Another newer work compared to my work integrates an approximated compression method for integer data types such as char, short, int, and so on into the memory controller [244]. This method dynamically controls the used compression method depending on a quality target.

Increasing the refresh rate of a DRAM memory region can reduce the energy consumption, but raises the probability of losing the correct values [8, 226]. In addition, there exist AC approaches for emerging memory technologies such as Solid State Disks (SSD) [227] and Phase-Change Memory (PCM) [174]. There also exist software approaches, which trade-off accuracy for performance by using different data types [16, 245]. However, they only consider single and double precision data types.

4.6 Summary

AC poses an interesting and alternative way to improve design goals such as energy consumption or execution time. A wide-spread applicability requires a controlled way to tune the degree of approximation. Such a knob can be integrated at hardware layer as discussed in this chapter and can be employed and tuned by the configuration layer that is designed in this thesis. While approximating compute units is quite common in the related literature, the contribution to the design values of an entire processor is low. However, the contribution of memory accesses is often quite high in terms of energy consumption and latency for applications running on any processor.

This chapter presented an innovative way to integrate an accuracy-aware method at hardware layer in a controlled way. The proposed conversion unit leads to a design that converts floating-point values before being transferred to memory and hence reduces the required memory space. The results show that the amount of cache misses is reduced. This leads to a reduction of the energy consumption and latency for memory accesses on average. Furthermore, compared to a native execution of an application using a smaller data format, the conversion units yields a higher QoR. An application field that is suitable to be executed on the presented approximate hardware is image processing. Using the dynamic method, the QoR achieved by a `2D FFT` is improved by two orders of magnitude, while retaining the potential gain in the reduction of energy consumption.

The experiments on different applications have shown that applications with high memory access frequency and irregular memory access patterns benefit most from the CU. On the other side, the overhead of the conversion unit is almost negligible in terms of area and static power. Comparing the energy and power values to a basic version of a processor poses some difficulties. Firstly, the integration of the conversion unit and the heuristic search of synthesis algorithms also change the behavior of unaltered components. Lastly, the dynamic power of the various components such as the floating-point unit significantly increases.

The conversion unit can be combined with other accuracy-aware methods at the same layer or with one from higher layers. The next chapters present additional accuracy-aware methods for other layers. Moreover, I present an abstraction layer, configuration layer, that can control the conversion unit in combination with other methods to achieve soft bounds for the QoR dynamically or to meet certain performance constraints.

CHAPTER
FIVE

FPGA-BASED APPROXIMATE COMPUTING PATTERNS FOR DYNAMIC PROGRAMMING

This chapter proposes a novel way to introduce an accuracy-aware method at the architecture layer. It poses another possibility for applications to exploit the paradigm of accuracy awareness. I introduce novel AC design patterns that can be used by hardware developers to build efficient designs for different hardware units such as FPGAs. Exploiting the introduced patterns leads to accuracy-aware knobs for a hardware implementations that can be tuned by the introduced configuration layer in this thesis. Hence, the presented accuracy-aware method can be combined with the other methods.

The presented design patterns target dynamic programming (DP) algorithms. DP is a programming concept and a mathematical optimization method applicable to many domains. An algorithm based on DP often poses a huge challenge in order to port them into an FPGA. Therefore, I introduce several AC-based patterns, which allow a designer to implement efficient FPGA designs for 2D DP algorithms. I show the benefit of these patterns for algorithms coming from different domains.

5.1 Introduction

DP-based algorithms pose significant challenges for hardware designers in order to port them to special architectures. These algorithms are used in image processing, computational biology, or in time series analysis. In particular, 2D DP-based algorithms have a limited internal parallelism due to data dependencies and huge memory footprints. The high memory footprint often prevents to store all required data on fast local FPGA memories and thus introduces additional costly off-chip memory accesses. Furthermore, an often required backtracking step hampers a streaming architecture implementation. Streaming architectures are very effective to achieve high performance using an FPGA.

An often used approach to parallelize such 2D DP algorithms are systolic arrays, which present a set of processing units connected in a certain way. However, they require a huge amount of hardware resources. To reduce the amount of needed resources and further improve the performance of FPGA designs, applying approximate computing is a suitable way. The investigated algorithms that are considered in this chapter allow us to introduce approximations without sacrificing the final quality of result (QoR).

Useful approximation design patterns enable a hardware designer to implement an adapted version of an algorithm in order to better exploit the capabilities of FPGAs. Besides hand-optimized FPGA implementations for certain DP algorithms, there are only few approaches that investigate more general approximation patterns to implement efficient FPGA designs. In the domain of image processing, down-sampling is such a pattern [246]. GRATER [12] is an approach that applies precision scaling to OpenCL kernels in such a way that a certain QoR is met. But these methods only consider a single pattern, whereas considering more patterns can help to find even better designs. For GPUs, SAGE [11] shows the higher improvement of design values by using a set of patterns but these patterns are restricted to GPU kernels. Therefore, I identify and design useful AC design patterns in this chapter, which target the implementation of 2D DP algorithms applied in completely different domains.

These patterns are usable on various programming levels, hence it does not matter if the design is implemented using a hardware description (e.g., VHDL) or a high-level language (e.g., OpenCL). By employing these patterns, a hardware designer builds efficient FPGA designs running on high-end or low-end FPGA-based systems as shown in the remainder of this chapter. Furthermore, these patterns are valuable for other hardware architectures such as GPUs or vector extension units (VEUs, e.g., AVX). Hence, the contributions of this chapter are the following:

- Identification of AC design patterns for 2D DP-based algorithms.
- Exploiting these patterns for efficient implementations of DP designs for different FPGA systems.
- Evaluation of these designs in various domains such as homology protein sequence search, stereo vision, and time series analysis.
- Showing the effectiveness of these patterns for other hardware architectures.

5.2 Fundamentals

This section provides an overview about the used FPGA-based systems. Furthermore, the basics about dynamic programming and several examples are presented.

5.2.1 Considered FPGA-based Systems

For the evaluation of the FPGA designs, I use an FPGA-based server system, *Convey HC-1*, a workstation, *Maxeler Workstation*, and an evaluation board, which integrates an FPGA-based System on Chip.

Convey HC-1. The Convey HC-1 combines an Intel Xeon 5138 processor with a reconfigurable hardware unit (see Figure 5.1). Four FPGAs (Xilinx Virtex-5 LX330) act as four Application Engines (AEs) that can be user-programmed. Eight memory controllers provide a bandwidth of 76.8 GB/s in total. Communication with the host is controlled by the AE Hub. The available Personality Development Kit (PDK) allows a hardware designer to create custom FPGA designs. Such designs can be described using a hardware description language like VHDL. This system is able to outperform 32-threaded applications running on an Intel Xeon E5-2670. More details can be found in [247].

SoCrates II Evaluation Board. The SoCrates II includes a Cyclone V system on chip, which combines two ARM Cortex A9 running at 800 MHz, a reconfigurable FPGA fabric usually running at 100 MHz, and 1GB DDR3 RAM at a frequency of 333 MHz. There are 41910 Logic Array Blocks (LABs), 112 Digital Signal Processors (DSPs), and 5 662 720 Memory Bits on the FPGA. The CPUs and the FPGA share the RAM. The board offers several interfaces like USB or Ethernet and runs a full-fledged Linux operating system. The required time for the compilation and synthesis is around one hour. Configuring the FPGA requires between 100-200 ms. Despite the RAM being shared, data has to be copied between CPU-attached and FPGA-attached buffers. The reason is that the OpenCL framework has to independently deal with cache coherency.

Maxeler Workstation. The Maxeler workstation includes an Intel Core i7-2600S, 16 GB main memory, and a PCIexpress-based FPGA card (Maxeler Card MAX3A Vectis with 24 GB RAM). The FPGA called *Dataflow Engine* is user-programmable by a high-level language called MaxJ. MaxJ is a Java-based description language for FPGA kernels. A programmer has to create a host code, a manager and a kernel code. The manager code specifies the connection between the host and the FPGA, and the kernel

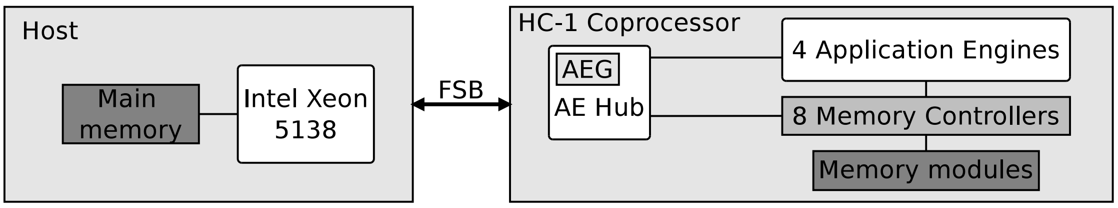


Figure 5.1: Overview of the Convey HC-1 architecture.

code describes the actual design that is implemented on the FPGA. The FPGA gets data from the host via PCI or from a locally attached memory with a bandwidth of 40 GB/s.

5.2.2 Dynamic Programming

The term dynamic programming (DP) describes a mathematical optimization strategy and a programming concept. DP determines the optimal solution for a problem by exploiting solutions of subproblems. Therefore, DP combines solutions of these subproblems to find the optimal solution of the main problem. In case that the problem has several optimal solutions, DP is able to find all of them and thus the number of optimal solutions can be counted. In general, the computation effort depends on the amount of subproblems and the number of choices between possible subproblems to reach the optimum. However, the problem requires a certain structure in order that DP is applicable. The challenge of applying DP is to figure out, whether a problem can be transferred into such a structure. The method was introduced by Richard Bellmann [248]. DP has similarity to divide and conquer approaches, but it relies on subproblems that depend on each other. Bellmann called the needed property of a problem the *Principle of Optimality* and the meaning is

> "An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decisions" [248, Page 4].

An important property of the problem structure is that always the same subproblems occur for solving a main problem, which is called overlapping subproblems. This property makes it possible to cache solutions to these subproblems. Note that subproblems can also occur in other subproblems. Another property of the problem structure is that the optimal solution also requires to solve subproblems optimally. A common way to transfer a problem into one that can be solved by DP is the definition of a recursive structure if possible. Besides 1D DP problems, there exist m-dimensional problems. For them, the computational and storage complexity increase significantly. Such problems require to consider up to m different subproblems.

In this chapter, I consider DP algorithms used in computational biology, stereo vision, and time series analysis. The DP algorithm poses the bottleneck regarding the execution time for these applications. The algorithms are described in the following sections.

5.2.3 A Tool for Finding Homologous Protein Sequences

Based on Hidden Markov Models (HMMs), the tool *HMM-HMM-based lightning-fast iterative sequence search* (HHblits) finds similar (homologous) protein sequences in a database for a given query sequence [249]. The entries in the database are clusters of

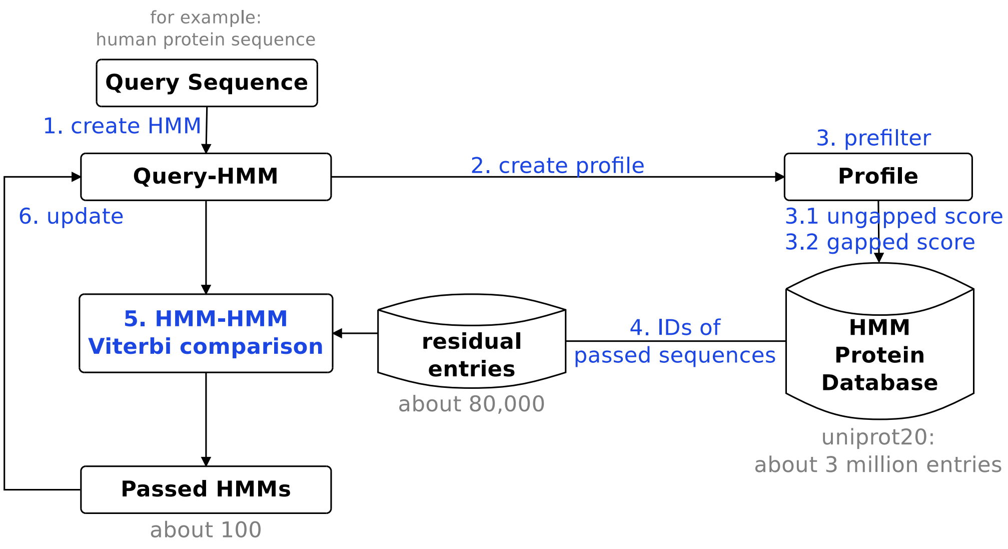


Figure 5.2: HMM-HMM-based lightning-fast iterative sequence search.

sequences that are biologically similar to each other. The clusters are represented by a HMM. The HMM reflects this biological-based similarity or homology through *delete*, *insert* and *match* states. For a consensus column of clustered sequences, the *match* state models the probability for the occurrence of an amino acid at this position. Additionally, a HMM models transition probabilities from one state to another. Goal of HHblits is finding homologous sequences in order to draw inferences on its properties.

Design of HHblits. After starting the application, HHblits first reads the query sequence or a multiple sequence alignment and creates a query-HMM (cf. Figure 5.2, Step 1) from it. All sequences are loaded from the used database into the main memory. Then, a prefilter is initialized. After initialization, the application starts with the main loop. The user can specify how often this loop shall search for homologous sequences. The query-HMM is approximated to a simple profile (Step 2) in order to prefilter the protein database (Step 3). Database entries consist not only of a HMM, but also of a sequence in order to support fast prefiltering.

The first step in the prefiltering is to calculate a score for a locally ungapped similarity of the query profile against each sequence in the database (Step 3.1). For all sequences with a score higher than a threshold, a locally gapped score is calculated afterwards in the second prefilter Step 3.2 by using a striped Smith-Waterman [250] implementation. Both are executed on the VEU. For all HMMs whose corresponding sequence has passed both steps of the prefilter (Step 4), the application performs a comparison using the Viterbi algorithm [251] (Step 5). If the calculated Viterbi score is higher than another threshold then this HMM is considered homologous to the query-HMM. All passed HMMs

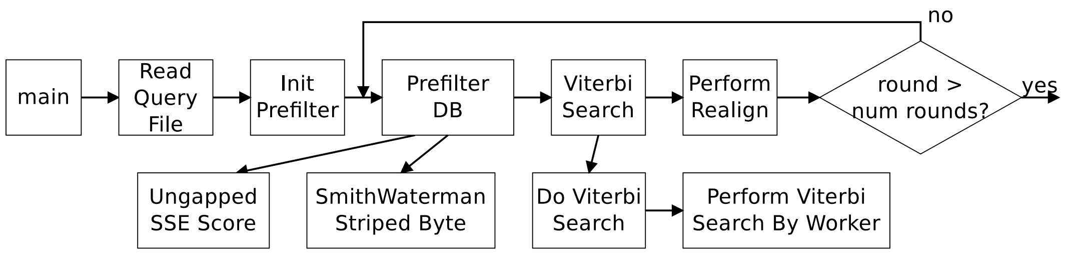


Figure 5.3: Homology detection for a query sequence against a protein database.

are realigned and finally, the query-HMM is updated with all these homologous HMMs (Step 6). The next iteration searches for homologous HMMs with this updated query-HMM. To get a more profound understanding at a lower level of how HHblits works, Figure 5.3 shows a flow chart of the application.

Gene Databases. Equally important to the algorithm under discussion are the layout and size of the data to be processed. For the query, the profile consists of 220 lines, one for each “letter” of the “alphabet” by which consensus amino acid columns of the used query-HMM are represented. A line is at most of length 512 Byte. Thus, a profile requires at most 110 KB of memory space.

The used database uniprot20[1] has roughly 3.1 million sequences. Each entry of this database clusters homologous protein sequences from the uniprot database. An entry is represented as HMM as well as an approximated sequence for prefiltering. The longest sequence has 15,000 letters, while the shortest has only 2 letters. As Figure 5.4 shows, the lengths are not distributed equally. Therefore, it does not make sense to design an architecture explicitly for longer sequences, but rather to handle many short sequences.

Profiling of HHblits. For an efficient software/hardware design, it is important to fully analyze the used algorithm and the used architecture. A widespread method to analyze an application is to use a profiler. Such a profiler gives beneficial hints about the runtime behavior of the entire application and also about the used execution time of different parts of the application.

Therefore, I executed the tool HHblits on the Intel Xeon 5138 processor of the Convey HC-1 and employed gprof as profiler to find the most time-consuming parts. The results are shown in Table 5.1. According to the case studies of Remmert et al. [249], the most common number of used iterations is up to two. For one and two iterations of the tool, most of the calculation time is spent in the function *ungapped_sse_score*. So, this function is the best candidate for accelerating the entire tool HHblits.

[1]ftp://toolkit.genzentrum.lmu.de/pub/HH-suite/databases/hhsuite_dbs/

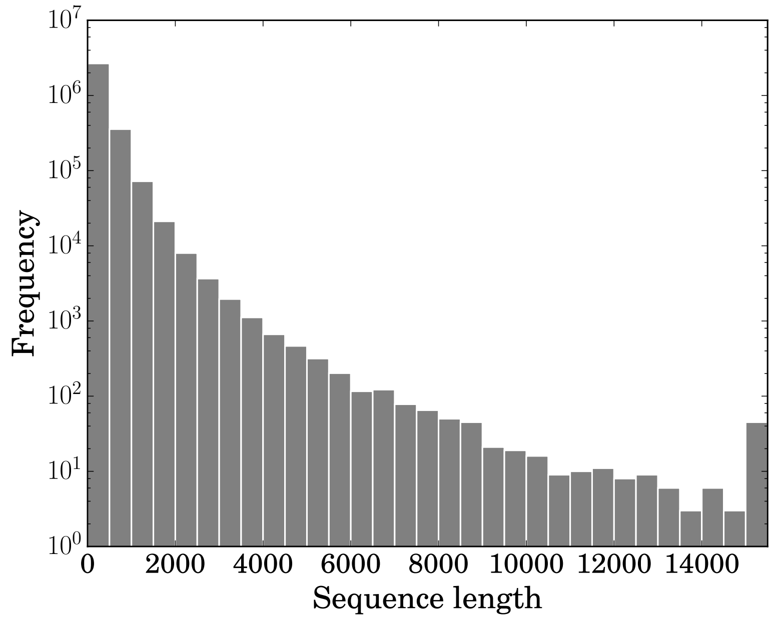


Figure 5.4: Distribution of sequence lengths of the uniprot20 database.

Table 5.1: Profiling HHblits over one, two and four iterations.

Function	1 Iteration			2 Iterations			4 Iterations		
	Time		Calls	Time		Calls	Time		Calls
ungapped_sse.	62.58%	49.77s	3129234	52.52%	99.69s	6258468	11.08%	200.83s	12516936
CalculatePost.	12.10%	9.62s	2155	7.09%	13.46s	3017	1.18%	21.32s	4741
Hit::Viterbi	6.37%	5.07s	601	19.63%	37.27s	3778	64.79%	1174.64s	83760
swStripedByte	3.72%	2.96s	77921	3.88%	7.37s	191721	1.10%	19.90s	452762
Sum	84.77%	67.42s	3209911	83.12%	157.79s	6456984	78.15%	1416.69s	13058199

Another measurement indicated that at most 7.757 GB of sequences data are transferred and processed per second by this function. The query profile of at most 220 lines with a maximum of 512 Byte per line is presumably cached after few runs when all lines have been accessed at least once, while the sequence data is always fetched from main memory. The limiting issues for the throughput seem hence the memory boundedness and the Front Side Bus (FSB) interface with a maximum bandwidth of 8.5 GB/s, because the SSE unit can process data at a rate of 2133 MHz * 16 Byte = 34.128 GB/s, assuming fully pipelined execution. Using more iterations for finding homologous sequences increases the probability to find sequences that are not similar in terms of biological relationships due to over-fitting the query-HMM. Of course, in some cases it is helpful to use more than two iterations. For four iterations, the most time-consuming part (up to about 64.79% of the entire application) is consumed by the generalized Viterbi algorithm.

Dynamic Programming Algorithm: Smith Waterman Algorithm. The function `ungapped_sse` is based on the Smith Waterman algorithm [252]. This algorithm determines the optimal local alignment for two strings, which can be protein sequences. Given two sequences, $X = x_1x_2 \dots x_m$ and $Y = y_1y_2 \dots y_n$, of length m and n, respectively, the following matrix recurrence leads to the optimal local alignment for a substitution matrix S and a gap penalty function W_k.

$$H(i,0) = H(0,j) = 0; 0 \leq i \leq m, 0 \leq j \leq n$$

$$H(i,j) = \max \begin{cases} H(i-1,j-1) + S(x_i,y_i)[\text{(mis)match}], \\ \max_{k\geq 1}\{H(i-k,j), -W_k\}[\text{delete}], \\ \max_{l\geq 1}\{H(i,j-l), -W_l\}[\text{insert}], \\ 0 \end{cases} \qquad (0 \leq i \leq m, 0 \leq j \leq n)$$

The maximum value of H is the similarity score of the local alignment. Starting from that value, a backtracking step determines the best local alignment.

5.2.4 Stereo Vision

Stereo vision is an important task in many applications such as self-driving cars, hence it is an integral part of a computer vision system [253, 254]. By using stereo cameras providing two images of the same scene recorded from different angels, depth information can be extracted. The resulting disparity map represents the horizontal shift between corresponding pixels and can be used to calculate real-world distances. A rectification of the images restricts the search window of corresponding pixels to a horizontal row.

Approaches differ in the way they find corresponding pixel. In general, the calculation is divided into four steps: matching cost calculation, cost aggregation, disparity calculation and optimization, and disparity map refinement. Approaches, distributed in local, semi-global and global ones, exist to calculate a disparity map [255], where global ones often do not use cost aggregation. The accuracy of different approaches can be looked up in several public rankings based on benchmarks such as Middlebury or KITTI [253].

Global approaches, which require to minimize a global energy function, such as graph cut or belief propagation are very accurate, but the high computation and memory costs often rule out a sufficient frame rate [255]. Semi-global (SGM) or DP approaches are a good trade-off between global and local approaches. The disparity calculation and optimization is performed over a subset of pixels. The determined disparity value of a pixel is influenced by the decision for the pixels within this subset. This is not the case for local methods, since they solve the correspondence problem for each pixel separately. Therefore, SGM normally provides a higher accuracy compared to local approaches. However, SGM provide less accurate results compared to global methods in general. While SGM implementations improve the performance compared to global approaches, they do not achieve sufficient performance on embedded multi-core CPUs[256]. Local and SGM approaches require a matching cost function like Sum-of-Absolute-Differences (SAD) or Sum-of-Hamming-Distances (SHD). Larger window sizes tend to improve the accuracy, but have a significant impact on the computational effort.

Local approaches. Local approaches determine the minimum local matching cost for a set A_p of possible correspondence pixels. Since the minimum cost value can occur several times, the local correspondence problem is not well-defined. Therefore, a uniqueness constraint is applied in practice, so that a value is only used if the lowest and second lowest cost value have a certain minimum difference. Such a constraint reduces the density of a depth map but increases the accuracy of the disparity values. Local methods differ in the set A_p as well as in the used matching cost function. A widely used method is SAD, which is defined as

$$cost_{\mathsf{SAD}}(p, \bar{p}) = \sum_{q \in A_q} |l(p+q) - r(\bar{p}+q)|,$$

where l is the left image, r the right image and A_q the set of pixel used for the cost calculation. p and $\bar{p}$ indicate the location of a pixel. Another cost function is SHD defined as

$$cost_{\mathsf{SHD}}(p, \bar{p}) = \sum_{q \in A_q} \mathsf{hamming}(l(p+q), r(\bar{p}+q)),$$

where $\mathsf{hamming}(a, b) = \sum_{i=0}^{m-1} |a_i - b_i|$ is the hamming distance, which returns the number of bits that differ between two given bit vectors a and b of length m.

Dynamic Programming-based Stereo Correspondence Algorithm. The considered SGM algorithm is based on DP[2]. As input, the algorithm uses two rectified images l and r in the RGB space. The first part of the algorithm calculates a cost matrix C_y for each image row $y \in [0, height-1]$, which is calculated using absolute differences in the RGB space for different disparities.

$$\begin{aligned} C_y(x,d) = & |l_r((x,y)) - r_r((x+d,y))| \\ & + |l_g((x,y)) - r_g((x+d,y))| \\ & + |l_b((x,y)) - r_b((x+d,y))| \end{aligned} \quad ,$$

where the red value at pixel $p = (x, y)$ is denoted with $l_r(p)$ and $r_r(p)$ for the left and right image, respectively. Similarly, $g(p)$ specifies the green value and $b(p)$ the blue intensity. The pixel $p = (x, y)$ is addressed by a row y and a column x. Furthermore, $d \in$

[2]Middlebury Stereo Matcher Framework `http://vision.middlebury.edu/stereo/`

$[0, d_{max}]$ is the disparity in x-direction between a pixel in the left and right image. Three DP matrices are calculated for each depth map row (see Figure 5.5). DP_M includes information about matching pixels, DP_L and DP_R about pixels occluded in the left or right image, respectively.

$$DP_{M_y}(x,d) = \min \begin{cases} DP_{M_y}(x-1,d) + C_y(x,d) \\ DP_{L_y}(x-1,d-1) + C_y(x,d) + \lambda(x) \\ DP_{R_y}(x-1,d) + C_y(x,d) + \lambda(x) \end{cases}, \tag{5.1a}$$

$$DP_{L_y}(x,d) = \min \begin{cases} DP_{M_y}(x-1,d) + \text{occl} \\ DP_{L_y}(x-1,d-1) + \text{occl} \end{cases}, \tag{5.1b}$$

$$DP_{R_y}(x,d) = \min \begin{cases} DP_{M_y}(x,d+1) + \text{occl} \\ DP_{R_y}(x,d+1) + \text{occl} \end{cases}, \tag{5.1c}$$

where $\lambda(x)$ are the smoothing costs. The `occlusion cost (occl)` is a factor controlling the smoothness constraint of the line calculation, hence defines an aggregated factor for possible disparity jumps. By design, the DP-Algorithm does not allow valid disparities to “jump” (very high disparity following a very low disparity). But a low `occlusion cost` value allows higher jumps, but increases the error rate.

Three additional matrices store the transitions. The algorithm has a space complexity of O($6 *$ image$_{\mathbf{width}}$ $*$ num$_{\mathbf{disparity}}$). In contrast, local methods have a complexity of O(image$_{\mathbf{width}}$$*m$), where m is the number of rows from A_q. Using the transition information together with the DP matrices, a backtracking approach finds the optimal disparities. The backtracking starts at the cell which contains the smallest value in the rightmost DP Match column. Depending on the transitions, the backtracking algorithm travels through the DP matrices to find the best disparity value for each pixel in the current depth map row. Calculating the three DP matrices and the backtracking step has to be done for each row. Therefore, calculating the DP matrices together with the backtracking is the most time-consuming part. The advantage of a DP-based algorithm is the smoothness of the resulting depth map, but slightly false matches do not change the calculated disparity. Therefore, false disparities are often calculated close to object boundaries.

5.2.5 Time Series Analysis

Comparing time series using a distance function is a central task in the field of time series analysis. A distance function[3] uses as input a query sequence $Q = (q_1, q_2, ..., q_n)$ and a candidate sequence $C_i = (c_1, c_2, ..., c_m)$. It depends on the measure whether both sequences have to be of equal length, i.e., $n = m$. If two sequences are similar, a

[3]I use the term distance function. An often used synonym is dissimilarity measure.

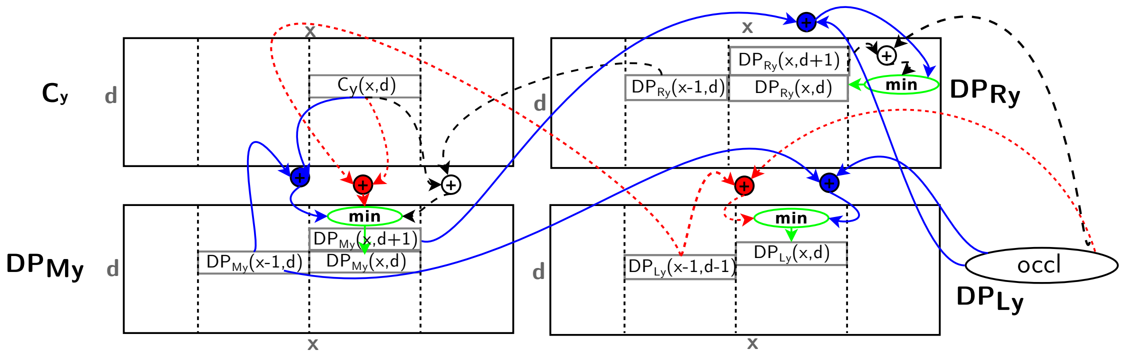


Figure 5.5: This figure shows Equation (5.1) and the corresponding data dependencies graphically. Note that $\lambda(x)$ is excluded due to presentation purposes. The four rectangles represent the three DP matrices (DP_{M_y}, DP_{L_y}, and DP_{R_y}) and the cost matrix C_y with the corresponding dimensions x and y. A matrix element is determined by selecting the minimum value from a set of calculated values. Such a calculated value depends on a certain element (either from the same or a different DP matrix). Additionally, it either depends on the occlusion cost (occl) or a cost matrix value. Data dependencies from DP_{L_y} are shown in red (dotted line), from DP_{R_y} in black (dashed line), and from DP_{M_y} in blue (solid line).

distance function returns a small value. Usually, a distance of 0 indicates that both sequences are equal. By contrast a similarity measure returns a large value in case two sequences are similar.

Euclidean distance. The Euclidean distance: $ED(Q, C_i) = \sqrt{\sum_{i=1}^{n}(q_i - c_i)^2}$ is a simple distance function and often competitively compared to other measures. It requires that both sequences are of equal length and has a computation complexity that lies in $O(n)$. It is easy to compute the single squared sums (i.e., $(q_i - c_i)^2$) in parallel, as there are no data dependencies. Therefore, it is easy to implement a parallel version. ED is not invariant against local scaling (warping), which is crucial for many problems.

Dynamic Time Warping: A dynamic programming algorithm. Dynamic Time Warping (DTW) [257] is a distance function that allows local scaling. This is, for instance, required in case two sensors sample the same fact, but at a different sample rate. As a result, we need to align the sequences. This measure tries to find an optimal global alignment between two sequences. Therefore, we do not require sequences having the same length. DTW relies on the following recurrence to calculate the DP matrix:

$$C_{DTW}(0,0) = d_{0,0}$$

$$C_{DTW}(i,j) = d_{i,j} + \begin{cases} C_{DTW}(i-1,0), & j = 0 \\ C_{DTW}(0,j-1), & i = 0 \\ \min \begin{cases} C_{DTW}(i-1,j)[\text{delete}] \\ C_{DTW}(i,j-1)[\text{insert}] \\ C_{DTW}(i-1,j-1)[\text{(mis)match}] \end{cases}, & else \end{cases},$$

where $d_{i,j}$ is any distance function. Often ED is used. The element $C_{DTW}(n,m)$ represents the DTW distance between Q and C_i. It takes quadratic computing time to get the resulting matching costs by using DP.

An often used method to reduce the complexity of DTW is to restrict the warping path in the DP matrix. Moreover, this restriction avoids pathological paths, hence can increase the accuracy. Typical constraints are the Sakoe-Chiba (SC) band [258], the itakura parallelogram, and the Ratanamahatana-Keogh band (RK band). The SC band restricts the warping path to elements that are only $|i-j| \leq c$ away from the main diagonal of the matrix.

Use case: Distance-based classification. Given a query sequence Q, an often required task is to find the most similar time series for Q called query by content. Another important data-mining method for time series is to perform a classification. Having a new unknown time series, a query, we want to find the class which includes the most similar time series to this query. Classification usually requires labeled data which assigns a class to each training data element.

5.3 Approximate Computing FPGA Design Patterns

Figure 5.6 shows common challenges that arise for hardware developers while porting 2D DP algorithms to an FPGA efficiently. To solve these challenges, I identify five suitable approximations (shown in red) leading to approximate computing-based design patterns. These patterns can help designers to meet their performance and resource constraints while still having an acceptable QoR (classification accuracy, mean square error, etc.).

2D DP algorithms have a two-dimensional input, which can be, for instance, two sequences. First of all, 2D DP algorithms require to calculate one or more 2D DP matrices. The calculation of each matrix element that specifies a state of the DP algorithm depends on two input data items, which span the two dimensions. The input data is required to calculate the cost value. This is important to identify the optimal transition from a previous stage. Cost values can be determined in a preprocessing step and stored in a cost matrix. Besides the costs, the value of a stage depends on the values of previous stages, which can be located in different DP matrices. The considered previous stages vary between DP algorithms. Additionally, determining the stage value needs some constants or regularization costs, which are stored in an array. If an optimal alignment is required, we need to perform a backtracking step. Therefore, we have to store the made transitions and the entire elements of the DP matrices. To conclude, porting DP algorithms into an FPGA poses several challenges which can be roughly grouped into:

C1: Parallelization
C2: Memory consumption
C3: Memory bandwidth
C4: Hardware compute resources

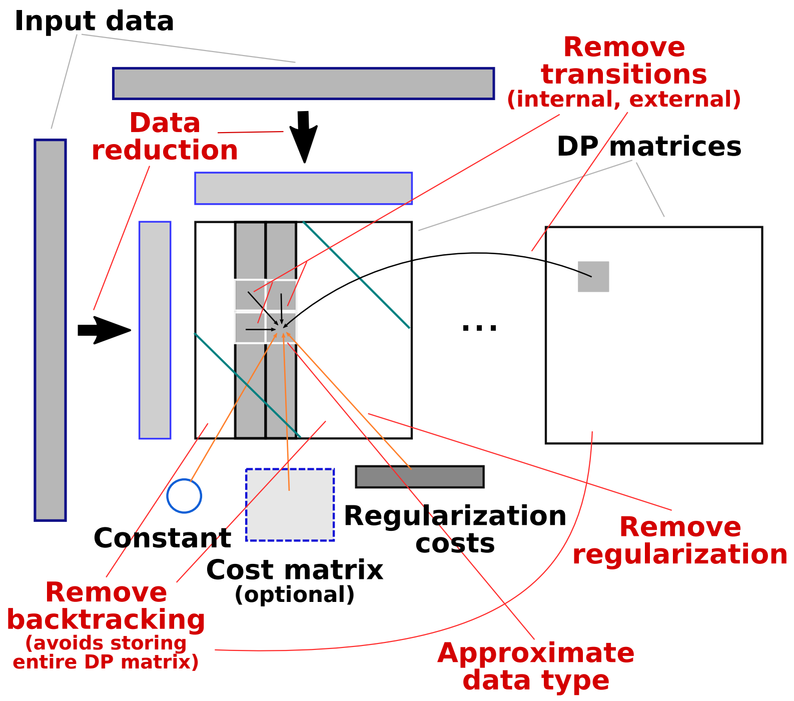


Figure 5.6: While porting 2D DP algorithms to an FPGA, hardware designers may face four challenges. The first challenge (C1) occurs due to the calculation rules for the DP matrices. These rules prevent a parallelization of the algorithms. The second challenge (C2) is related to the presumably high memory consumption caused by DP matrices, the cost matrix, and regularization costs. The third challenge (C3) represents the issues regarding the streaming of input data in an FPGA. Finally, the developer must handle the limited amount of resources available on an FPGA (C4). Therefore, this thesis identifies five suitable target points for applying approximations (shown in red). Such approximations lead to increased performance and reduced resource consumption while having an acceptable QoR (classification accuracy, mean square error, etc.) loss.

There exist data dependencies for calculating a single stage value (DP matrix element). These dependencies occur, because the calculation of the value depends on values from previous stages, which are located in the same or a different matrix. Calculating elements of a row or a column cannot be parallelized (C1), hence the massive parallel capabilities of an FPGA are not exploited. A possible solution is that we parallelize over the anti-diagonals called wavefront parallelization. This is realized by using systolic arrays [259, 260, 261] on an FPGA, but requires a significant amount of resources.

DP algorithms have a high space complexity (C2). Especially, if it is required to store the entire DP matrices for later steps. Moreover, space can be necessary for some of the input data to avoid repeating memory accesses to the same data. In contrast to CPU implementations, it can be worth to calculate costs on demand, thus avoiding the storage for the cost matrix. However, this increases the amount of compute resources (C4). Depending on the cost function, this can require to load more data into the FPGA, hence increases the required memory bandwidth (C3). The required memory bandwidth is accumulated by additional regularization costs that depend on the input data.

In case, we find solutions to the above challenges, we can have issues regarding resource problems due to the number of required processing units, such as floating-point units (FPUs) (C4). FPUs impact the resource consumption of an FPGA significantly.

While porting a function into an FPGA, the designer makes sure that the same results are returned by the FPGA design and the original CPU implementation. For that, the designer uses traditional suitable design patterns [262] including systolic arrays, pipelining, or precision scaling. However, in this chapter I present a different strategy to port algorithms into FPGAs focusing 2D DP algorithms. The motivation is to adapt an algorithm in a way, that it can efficiently exploit the capabilities of an FPGA architecture. The main goals are the improvement of the performance and the reduction of the resource consumption. Such an adaptation is achieved by introducing approximation methods.

I identify five different AC design patterns that lead to an adaptation of the original algorithm and address the aforementioned challenges. These patterns are shown in Table 5.2, together with the challenges they address. By applying these patterns, it is possible that the optimal solution is not found but the considered algorithm still produce useful results, as shown later.

Removing transitions (ACP1), which avoids data dependencies between stages, allows a parallel calculation of stage values (C1). For instance, removing the top neighbor, we can calculate all elements of a column in parallel. However, this requires to take special care of global algorithms, as for them the right bottom element of the DP matrix is the final value. Since this element is only influenced by the left diagonal neighbor after the approximation, we have to deal with this aspect in a algorithm-specific way as described later. In summary, ACP1 allows a designer to exploit the parallel capabilities of an FPGA. A removed transition coming from a different DP matrix gets rid of the synchronization of the calculations for both DP matrices and a connection path which can cause timing is-

Table 5.2: Overview about proposed FPGA compute patterns (CP) including approximate computing patterns (ACP) and the challenges they address.

AC pattern	**Name**	**C1**	**C2**	**C3**	**C4**
ACP1	Remove transitions	✓			✓
ACP2	Remove backtracking		✓	✓	
ACP3	Remove regularization			✓	
ACP4	Data reduction	✓	✓	✓	✓
ACP5	Approximate data type	✓	✓		✓
CP1	Calculate cost matrix on demand		✓	×	

sues. Such data dependencies can cause an additional kernel call when using OpenCL. Moreover, ACP1 reduces the amount of required compute resources (C4).

Reducing the required memory space of the algorithm is very beneficial (C2), because the on-chip memory (block RAM) is a very limited resource on an FPGA. In case, the memory space is not enough, an external off-chip memory has to be used, which causes costly external memory accesses (C3). Therefore, avoiding a backtracking step is beneficial because it makes storing all elements unnecessary (ACP2). Then, it is sufficient to store only two columns for each DP matrix and the DP algorithms under consideration.

Some DP algorithms need a regularization term such as smoothcosts, which takes care that the final disparity values are smooth. However, such costs are adaptive to the current input data, hence have to be determined dynamically. Removing these costs (ACP3) reduces the required bandwidth (C2). Calculating these costs on demand requires a more complex FPGA design.

To address all of the aforementioned challenges, a reduction of the input data (ACP4) is a suitable way. There are two possibilities: reducing the data before sending it to the FPGA or on the FPGA itself. The latter does not address C3. There exist many different ways how to reduce data. A general way, which I will not consider here, but in Chapter 8, is to remove certain data points. Another method is to reduce each single data point, for instance, converting double values to half precision values. Moreover, some DP algorithms allow us to exploit algorithm-specific reduction methods. For example, representing single data points with a symbolic representation requires fewer bits.

This directly leads to ACP5, since a reduced data point requires less compute resources. ACP5 can further approximate individual data items, thus reduces compute resources (C4) and can increase the parallelization degree (C1), because a single computation requires fewer resources. Storing the elements in the DP matrices in an approximate way reduces the memory footprint (C2). The last compute pattern (CP1) that I consider for 2D DP algorithms is not an AC pattern, but it also addresses C2. Calculating the costs on demand prevents the usage of costly memory resources on the FPGA for storing the costs, but this can cause an additional pressure on the memory bandwidth.

5.4 Applying Approximate Design Patterns to FPGA Designs

To show the effectiveness of the AC design patterns (ACPs), they are applied to DP algorithms from various domains. Table 5.3 gives an overview about the used ACPs for an algorithm and the FPGA system(s) used for the implementation. This section shows that these patterns can be applied to completely different FPGA-based systems (cf. Section 5.2.1), which require distinct ways to program the system. This emphasizes the general characteristic of the ACPs.

5.4.1 HHblits

According to Section 5.2.3, *ungapped_sse_score* is the most time-consuming function for low iteration counts of HHBlits, hence it is a very promising candidate to port to an FPGA in order to accelerate the function. As already mentioned, the function is based on the well-known Smith-Waterman (SW)-algorithm. Since this function acts as the first prefiltering step, the maximum local DP value of the DP matrix is sufficient. Hence, a backtracking step is not required (ACP2). The SW algorithm does not require regularization costs and the input data is already represented in a symbolic way (ACP4). A substitution matrix S, which represents the cost matrix, is based on biological conditions, hence cannot be calculated on demand. ACP1 is the AC pattern that can be applied and leads to a parallel calculation of row elements in the DP matrix H. This aspect is already exploited in the original implementation of *ungapped_sse_score* in HHBlits, which also uses the VEU of x86 processors. Since *ungapped_sse_score* requires to process all elements of the database, the design goal is to implement an FPGA design that achieves a high throughput regarding sequence comparisons. Building an efficient and non resource-intensive unit enables integrating more units onto the FPGA. Exactly that is already achieved, since the original implementation exploits ACP1. I have not used the ACPs to redesign *ungapped_sse_score*, but I was able to identify this function as a promising candidate for porting into an FPGA using the ACPs.

Table 5.3: Overview about the application of ACPs to a certain algorithm.

2D DP algorithm	ACP1	ACP2	ACP3	ACP4	ACP5	CP1	FPGA system
Smith-Waterman (Section 5.4.1)	✓						Convey HC-1
DP-based Stereo Vison (Section 5.4.2)	✓	✓	✓		✓	✓	Convey HC-1 & SoCrates II
Dynamic Time Warping (Section 5.4.3)	✓	✓		✓	✓	✓	Maxeler Workstation

Algorithm 5.1: Calculation of the ungapped score.

```
$W = (query\_length + 15)/16$;                          // 16-Byte block count
uint128 workspace[2*W];
s_prev = &workspace[W];
s_curr = &workspace[0];
for $j \leftarrow 0$ to $n-1$ do
    qji = query_profile [ sequence[j]*W ];
    S = s_curr[W-1];
    processing unit SWAP(s_prev, s_curr);
    for $i \leftarrow 0$ to $W-1$ do
        S = S + qji[i] - Soffset;
        Smax = max(Smax, S);
        s_curr[i] = S;
        S = s_prev[i];
    end
end
return component_wise_max(Smax);
```

Calculation of ungapped scores using SSE hardware of the processor. The function *ungapped_sse_score* calculates a locally ungapped score for a query profile against a sequence from the database. To calculate this score, a modified version of the SW algorithm is used in the implementation of HHBlits as depicted in Figure 5.7. Each element in the result matrix depends on the left diagonal neighbor in the matrix, but does not depend on the top or left element (ACP1). The SIMD extensions of an x86 processor are exploited to calculate the elements of the result matrix. The current implementation in HHblits uses Streaming SIMD Extensions 2 (SSE2). Hence, the function works on 16-Byte blocks, with data dependencies removed by means of striping the profile [250].

Algorithm 5.1 gives the corresponding pseudo code. A nested loop processes these 16-Byte blocks where the outer loop iterates over all elements of the reference sequence (`j` loop). In this outer loop, for each element in the sequence an address is calculated to address a line in the query profile. The inner `i` loop iterates over all 16-Byte blocks of a 16-Byte column of the result matrix. Due to striping, only the first block in a workspace column depends on the last block of the previous column. For each other block, the

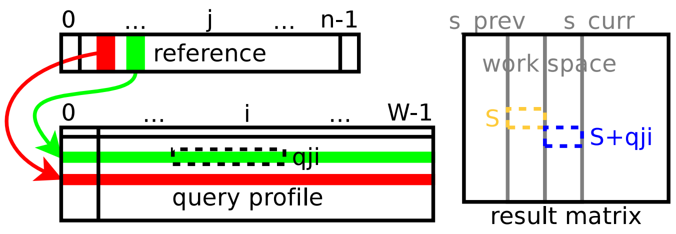


Figure 5.7: Calculation of the ungapped score.

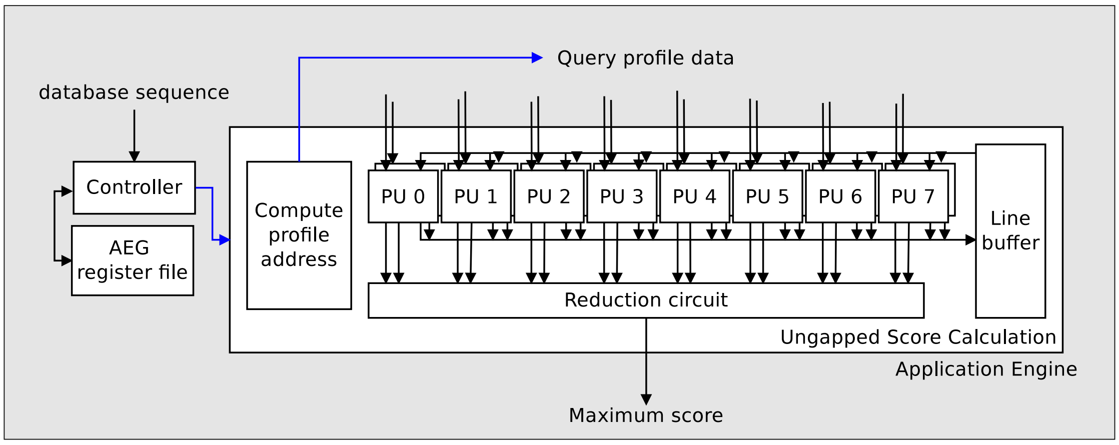


Figure 5.8: Parallel architecture with 16 processing units (PUs) for evaluating the ungapped score between a profile and a sequence on one FPGA.

previously calculated left diagonal neighbor 16-Byte block is added to the corresponding block in the profile. An offset is subtracted from each score in the block. The 16 maximum scores are kept in a 16-Byte variable and at the end of the function, the maximum Byte value in this variable is determined. This is the calculated locally ungapped score.

The SSE-based implementation of the Smith-Waterman algorithm already speeds up the calculation by $24.1\times$ (whole application) to $28.9\times$ (ungapped-score kernel only) against sequential execution. The goal is now to achieve superior speedup over the SSE-based implementation using an FPGA-based 2D DP implementation.

FPGA unit for calculating an ungapped score. The AE design for calculating the locally ungapped score exploits fine-grained data-level parallelism. 128 entries of the result matrix are calculated in parallel for each ungapped score. All internal units are initialized by a controller and each unit executes in a data-driven way. The AE design depicted in Figure 5.8 calculates the maximum locally ungapped score in a pipeline. With a loaded sequence, the compute profile address unit calculates for each element in this sequence the address to request the appropriate line of the profile from the coprocessor RAM. The 16 processing units (PUs) start to calculate the entries of the result matrix when the required data are loaded. Figure 5.9 shows how each PU calculates eight entries of the result matrix in parallel. First, for each Byte from the profile data, the corresponding Byte of the previously calculated line is added. Then an offset is subtracted from this result. Both calculations are done with saturated arithmetic together in one cycle (150 MHz). The resulting 8-Byte values are forwarded to the reduction unit and stored in the line buffer.

The multi-port line buffer needs to store both the previous and the current column of the result matrix because the previous column is required for calculating the current col-

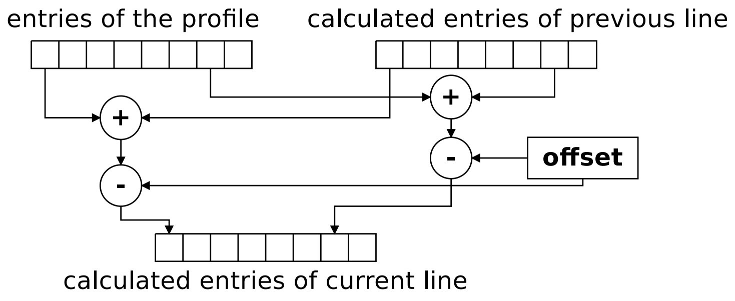

entries of the profile
calculated entries of previous line
+
+
-
-
offset
calculated entries of current line

Figure 5.9: Parallel structure of a processing unit (PU).

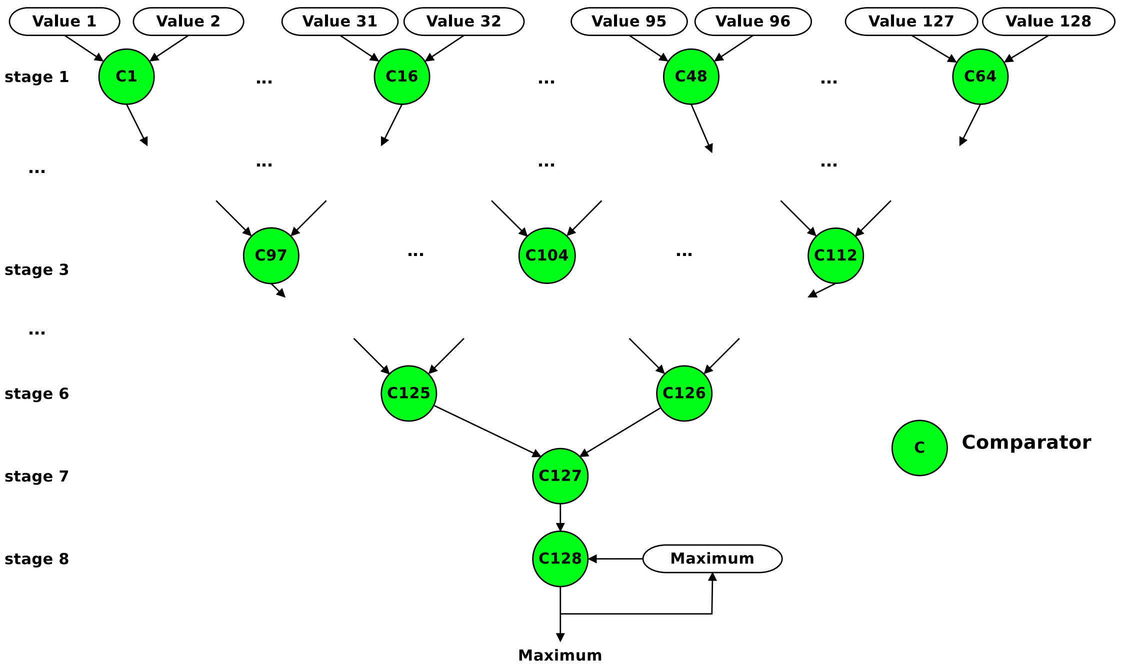


Figure 5.10: Reduction circuit for determining maximum entry of result matrix.

umn. Whenever all PUs have produced valid data, the reduction circuit begins searching the local maximum of the freshly calculated 128 elements of the matrix in a pipelined fashion in the reduction circuit, employing a binary-tree-like approach (see Figure 5.10).

In the first stage, two calculated entries are compared at a time and the greater one is forwarded to the next stage. Same is done in the next stage. Finally, one value is left. In the last stage, this value is compared against the global maximum. In case the local maximum is greater, this value becomes the new global maximum. The reduction circuit pipeline has eight pipeline stages. If all entries of the result matrix have been calculated, the global maximum value is returned to the host system via the AEG register file.

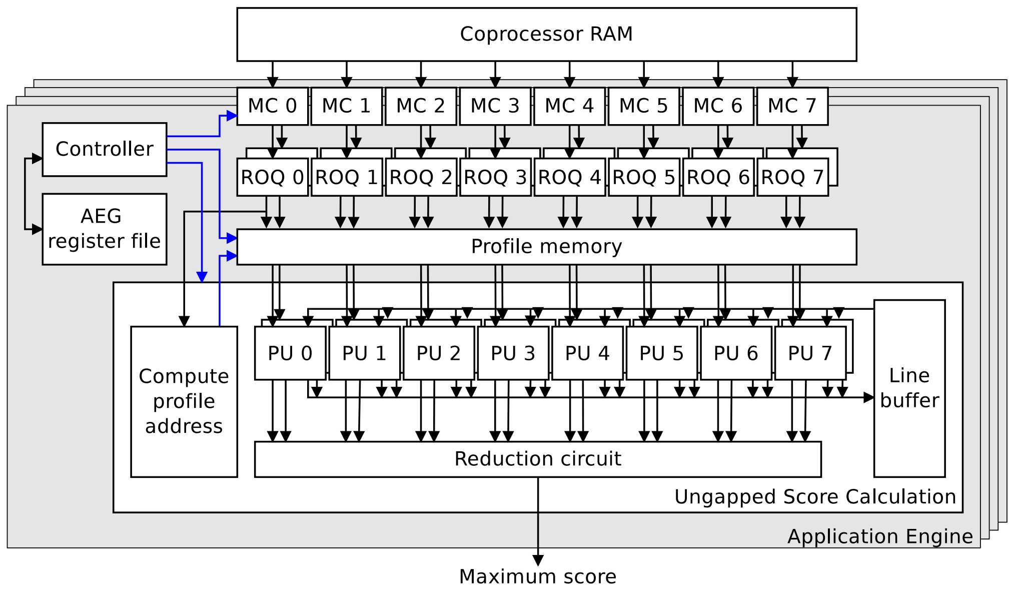


Figure 5.11: Final FPGA design including an ungapped score calculation unit for concurrent use on all four application engines (AEs).

Design of an Application Engine. There is no point in keeping the sequences in FPGA memories for later reuse. Rather, the 110 KB profile that is reused for every sequence comparison should be stored. Then, only small amounts of data will have to be transferred once the profile is loaded. This distinguishes this approach from the memory-bound SSE implementation in HHblits where the profile is cached only implicitly and might be replaced by sequence data.

To exploit the available bandwidth, I decided to transfer the 110 KB query profile in parallel on all memory controller ports first and to store it in FPGA-internal profile memory implemented by Block RAMs (BRAMs) for later reuse. This memory can be seen in Figure 5.11. Due to the weak consistency model of the Convey HC-1, reordering is required. Through this model the ordering of incoming requested data can be different from the read request ordering. A consecutive identification number (ID) is assigned to each request. So, Read-Order Queues (ROQs) are employed to reorder incoming data from the memory controllers according to these IDs. Using such a consistency model allows an efficient implementation of the underlying memory architecture.

The profile memory is split into 16 regions, one assigned to each ROQ (see Figure 5.12). Each PU has its own memory region (M0–M15) in the profile memory. Preloading a query profile enables fast data reuse on FPGA. A query profile is loaded asynchronously, directed by the depicted controller. The query profile is stored in this memory according to Farrar's data striping [250]. This data striping allows highly parallel data access and thereby achieves higher data throughput for PUs. The read address from all

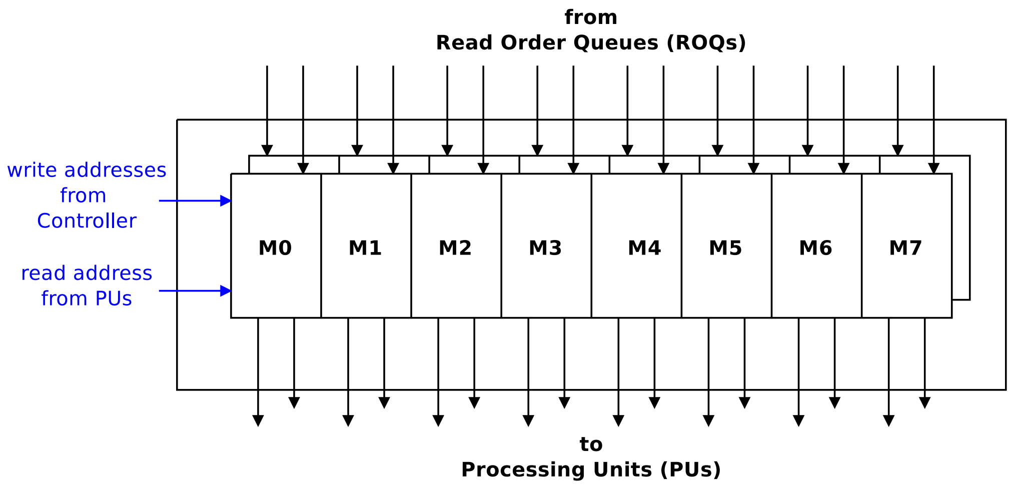


Figure 5.12: Internal Profile Memory, split into 16 memory regions (M0–M15).

PUs is the same, because the calculation of all PUs is synchronous. So, only one collective read address is used for all memory regions. After loading the profile, the reference sequence needs to be streamed to AEs, but will not be consumed at a very fast rate. I use a memory crossbar so that I can exploit a distinct ROQ for reading an entire reference sequence. The remaining ROQs can be used for further ungapped score units.

The calculation of lines 5-15 of Algorithm 5.1 is spread over the ungapped score calculation unit. The kernel (lines 9-10) is calculated in the pipelined PUs. The ROQs, profile memory regions and PUs form logical pipelines whose results are reduced by searching the maximum (line 11) in the custom reduction circuit. Sequence data is not consumed at a very fast rate because for each Byte (line 5), the corresponding 512-element line of the profile needs to be processed (line 6), with each of the 16 PUs processing 32 elements (line 9), 8 of them in parallel, in 4 iterations. Hence, memory bandwidth does no longer pose the limitation. So I have implemented a compute-bound and resource-bound version of the algorithm where the available bandwidth can be exploited for instantiating further units, while the query profile needs to be loaded only once for all instances.

The implementation results are given in Table 5.4. Slice usage is high because I initially required some memory structures to be reset to zero, which inhibited the use of FPGA Block RAMs (BRAMs). Correctness was proved by checking the individual scores against the scores from sequential and SSE implementations and by comparing the number of sequences that passed the prefiltering step.

Optimizing Data Transfer. There are several different methods for allocating and using the coprocessor memory. First of all, data can be allocated on the host side or on the coprocessor. Second, host-allocated data can be migrated implicitly or explicitly toward coprocessor memory or it can be copied explicitly by GLIBC or Convey routines. Copying

Table 5.4: Resource usage of a single ungapped-score calculation unit.

Resource	Count	Percentage
Slices	25,701	49 %
Registers	68,002	32 %
LUTs	70,432	33 %
BRAMs-total	74	25 %
Max. freq. (Synthesis)	5.046 ns	198.196 MHz
Max. freq. (P&R)	6.658 ns	150.20 MHz

data also means occupying more memory and burdening the programmer with memory management. From first measurements during implementation, however, I concluded that allocating on the host and copying each sequence individually and iteratively would be fastest because implicit migration is too slow and migrating explicitly from host to coprocessor memory was not possible for all 3 million sequences.

Transferring all sequences at once requires to pass a pointer for each sequence. However, as it turns out from Table 5.5, a single copy operation ("Block copy") of all sequence data with the Convey-provided copy routine lets coprocessor execution perform much better so that the overhead of calculating and passing an additional pointer to each sequence in coprocessor memory becomes tolerable. As a further option, data can be copied concurrently using one of the two available copy routines to coprocessor execution by performing an asynchronous coprocessor call. Figure 5.13 indicates that data transfer can be hidden completely.

But as Table 5.5 reveals, there is some overhead when transferring a large sequence while calculating the score for a short sequence. So, block copy performs better overall. Therefore, I employ the Convey-provided copy routine and transfer all sequences at once to coprocessor memory. The individual coprocessor calls are each passed the new addresses of the sequences to calculate the scores for. To fully exploit the available bandwidth and the Convey memory architecture of 64 Bytes per memory controller when reading the profiles, the padding is increased from 16 Byte to 512 Byte. Thereby, an

Table 5.5: Time (in seconds) for ungapped score calculations computations and data transfer on an AE.

Way of data transfer	Memcpy routine	Data transfer	Computation	Total
Per iteration	GLIBC	6.627	72.145	78.776
	Convey	42.056	64.207	106.324
Concurrently	GLIBC	–	–	74.569
	Convey	–	–	67.871
Block copy	Convey	0.400	63.802	64.201

entire sequential line of the profile is read by using all 16 memory ports concurrently instead of using only fewer ports.

Exploiting More Parallelism on FPGAs. As the previous synthesis results indicate, there is some area available for further instances of the ungapped score calculation unit. With the highly modular, crossbar-enabled design, each MC-ROQ combination can be used to read a different reference sequence to compare against the query. I modified the software/hardware design so that the memories do no longer need to be reset, which results in using BRAMs after place&route instead of logic and hence reduces the slice usage. Thereby, I was able to instantiate at least two units and a second profile memory, and later on also a third instance and memory. The reference sequences are streamed from the database to the score calculation units via different ROQs so that theoretically up to 16 such units could be instantiated. Additionally, the profile memory has to be replicated for each unit. Due to the available resources only three units fit onto one FPGA using this design.

Therefore, I adapted the profile memory to provide two ports so that two units can share a profile memory. Due to timing problems, I used different options for the vendor tools to create a bitstream. Switching off the synthesis options *resource sharing* and *equivalent register removal* helped to create a working bitstream without timing errors. This additionally reduced the amount of BRAMs because the design now uses logic blocks for on-chip memory. Table 5.6 shows the resource usage of the design for three and four ungapped score units, respectively. Figure 5.14 shows the final design for one

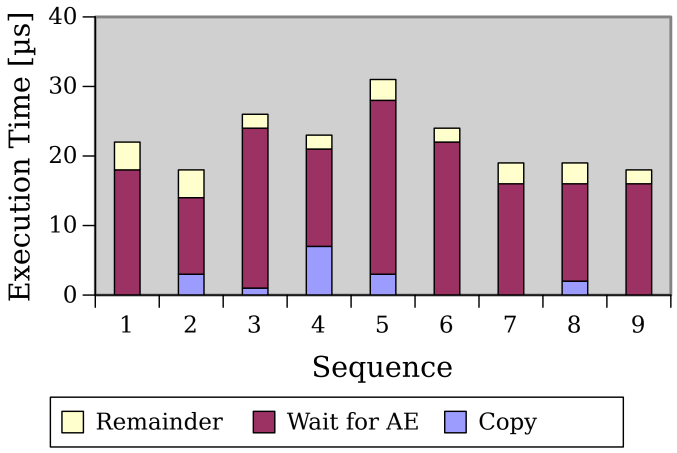


Figure 5.13: Overlapping data transfer with coprocessor execution on one AE for the first sequences. *Copy* indicates the time that is consumed for the data transfer to coprocessor memory initiate by host. *Wait for AE* is the time that the host has to wait for completion of the coprocessor computation. For each sequence, additional computation has to be done on the host. The time for this calculation is labeled *remainder*.

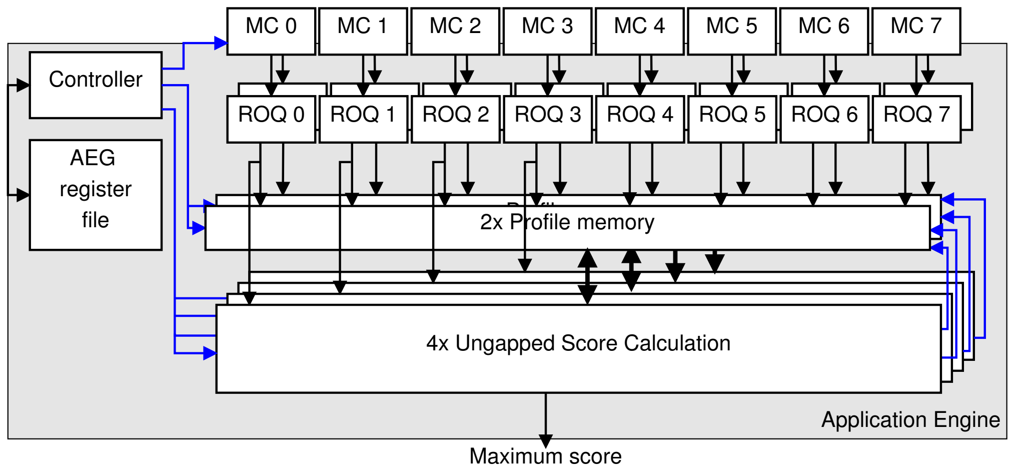


Figure 5.14: Instantiating four ungapped score calculation units on an AE.

AE of the Convey HC-1 including four ungapped score calculation units.

Using heterogeneity of the Convey HC-1. All previous approaches have the same drawback that the host processor is idle while the coprocessor is running. There is much potential in exploiting the host processor and coprocessor in parallel. For this purpose, data-parallel or task-parallel execution can be used. For data-parallel execution, shorter sequences can be calculated on the host and larger ones on the coprocessor. However, it is important to balance the workloads between coprocessor and host processor: in the used database uniprot20[4], there are many sequences smaller than 500 Bytes.

Table 5.6: Resource usage for the previous design including three ungapped score calculation units and three profile memories and the new design including four ungapped score calculation units and two dual-port profile memories. The synthesis options *resource sharing* and *equivalent register removal* were switched off.

	3 Units		**4 Units**	
Ressource	**Count**	**Percentage**	**Count**	**Percentage**
Slices	46,098 /	88 %	50430 /	97%
Registers	140,951 /	67 %	164349 /	79%
LUTs	126,745 /	61 %	143687 /	69%
BRAMs	170 /	59 %	64 /	22%
Max. freq. (Synthesis)	4.801 ns /	208.29 MHz	4.801 ns /	208.29 MHz
Max. freq. (P&R)	6.666 ns /	150.02 MHz	6.661 ns /	150.13 MHz

[4]The uniprot20 released at September 2nd 2011 is used.

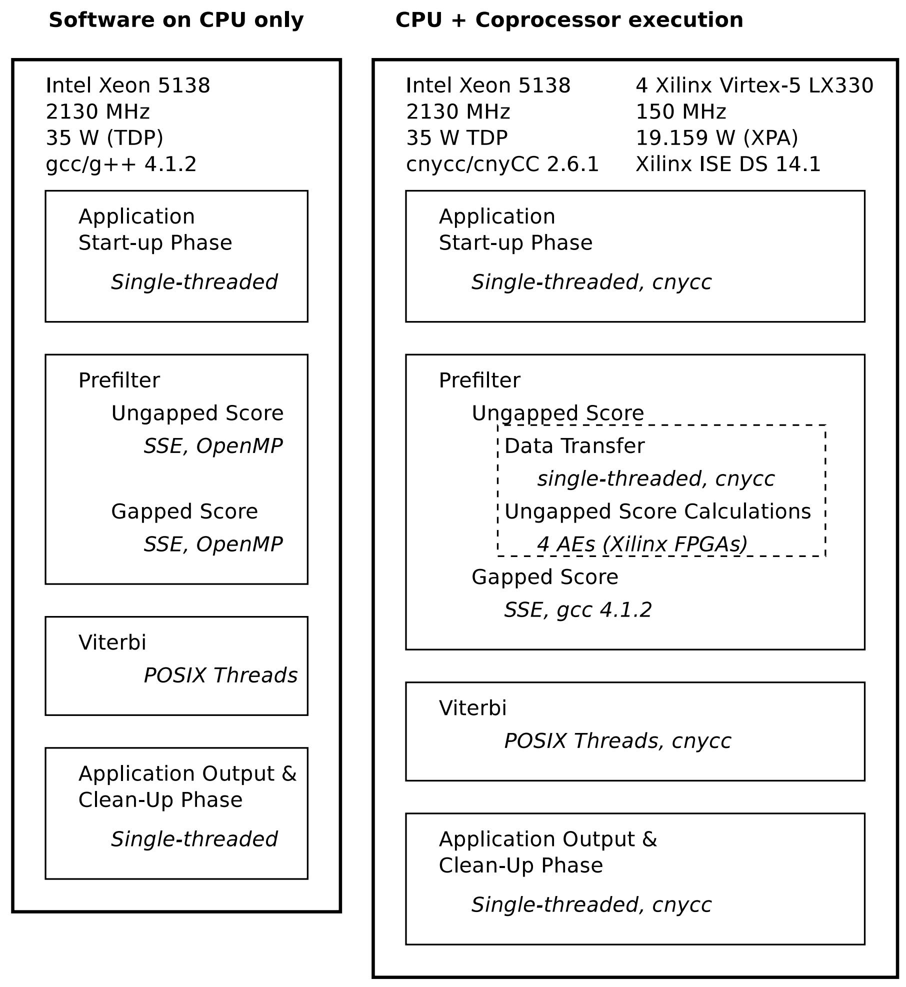


Figure 5.15: Setup of coprocessor integration and used tools.

The final design exploits task-parallel execution for prefiltering the protein database by using a hardware/software pipeline. The first prefilter step is executed by the coprocessor, controlled by a thread on the host processor. Another thread calculates the second stage of the prefiltering on the host processor. This thread is waiting for accepted sequences from stage 1. In addition, a second worker thread is enabled in HHblits to process the Viterbi algorithm in a task-parallel fashion (Step 5 in Figure 5.2, feature of HHblits). I employ the setup as depicted in Figure 5.15 when evaluating the kernel implementation and coprocessor integration. This includes using the dedicated Convey Compiler to access the coprocessor, which also makes comparisons more difficult.

5.4.2 FPGA Design for Stereo Vision

Existing semi-global stereo vision designs for FPGAs are often hand-optimized and implemented using a low level hardware description language. But still, they are very resource and memory demanding. Therefore, I apply the aforementioned design patterns to overcome the resource issues and to increase the performance. By analyzing the selected DP algorithm (cf. Section 5.2.4), three major issues are found, which limit the implementation on an FPGA. The first issue is the required high memory bandwidth. The memory bandwidth poses a bottleneck to stream data in such a way that the design provides a high frame rate. The high memory consumptions poses the second issue. Data dependencies that prevent a parallelization pose the third issue. In the next paragraphs, I discuss general implementation aspects, which do not depend on the actual FPGA-based design. Then, I present an FPGA design targeting low-power FPGAs and a design that is implemented for an FPGA-based server system. Both designs exploit the pattern shown in Table 5.2. Hence, this leads to streaming architectures, which are able to calculate depth maps.

Reducing required off-chip memory bandwidth (ACP3). As input, the original algorithm has the right and left image row data as well as so-called smoothcosts for the current image row. These smoothcosts are removed from the algorithm (ACP3) to avoid that memory transfers are the bottleneck of the resulting implementation and limit the performance. This would require an additional memory stream to the FPGA, which also increases the resource consumption. Furthermore, the new designs are working on gray values instead of RGB values (ACP3), which reduces the amount of image data that has to be transferred to the FPGA by a factor of 3.

Reducing memory consumption (ACP2). The high memory consumption is caused by the backtracking which requires to buffer all the elements in the three DP matrices and the three transition matrices. Removing the necessity to store all elements of the matrices is a crucial point. Therefore, the decision of the optimal disparity is based on local information rather than the global one calculated by the backtracking (ACP2). According to the data dependencies of the algorithm and without backtracking, only a column per DP matrix needs to be stored and the transition matrices are completely removed. This drastically reduces the internal memory consumption and avoids the use of external memories that offer a much lower bandwidth and would increase the execution time. This novel approach decreases the memory complexity from $O(6 \cdot image_{\mathbf{width}} \cdot num_{\mathbf{disparity}})$ to $O(3 \cdot num_{\mathbf{disparity}})$. The three DP columns can be calculated concurrently. An investigation reveals that the best accuracy for the local information approach is given by using the minimum value found in the current calculated DP columns. If the minimum is in column $DP_{L_y}(x)$ or $DP_{R_y}(x)$, the disparity is marked as invalid because the pixel is

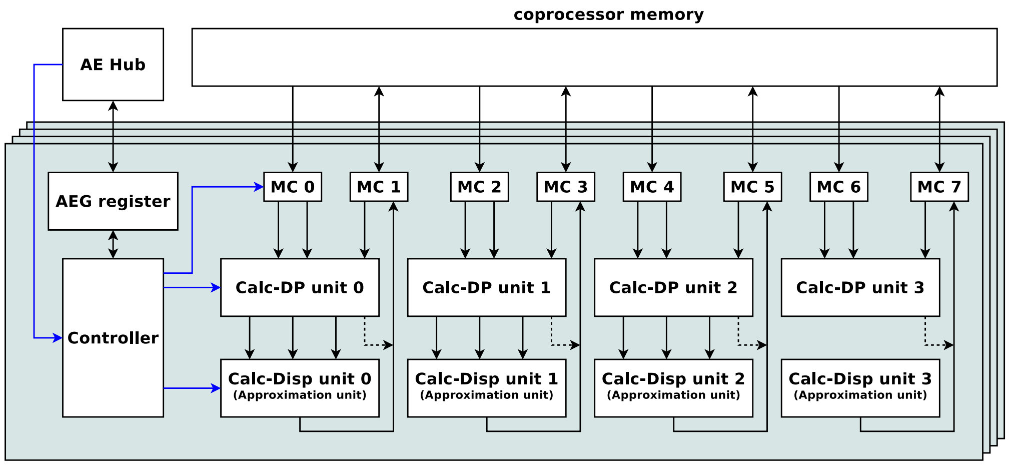

coprocessor memory
AE Hub
AEG register
Controller
MC 0
MC 1
MC 2
MC 3
MC 4
MC 5
MC 6
MC 7
Calc-DP unit 0
Calc-DP unit 1
Calc-DP unit 2
Calc-DP unit 3
Calc-Disp unit 0 (Approximation unit)
Calc-Disp unit 1 (Approximation unit)
Calc-Disp unit 2 (Approximation unit)
Calc-Disp unit 3 (Approximation unit)

Figure 5.16: AE design for calculating four rows of a depth map concurrently and following rows in sequential.

presumably occluded in one of the input images. If not, the disparity is set to the current row index of the cell including the minimum.

FPGA-based VHDL Design for a High-end FPGA-based System (ACP4). The most time-consuming parts of the algorithm, calculation of the DP matrix and determining the disparities, are ported to the AEs of the Convey HC-1. The calculation of the cost matrices is performed on the host CPU. Before the execution starts, some parameters have to be transferred to the AEs. These are the height and width of the depth map, the number of allowed disparities, the occlusion costs and the address of the cost matrix as well as the memory region for the resulting depth map inside the local FPGA off-chip memory. The resulted FPGA architecture calculating four rows in parallel is shown in Figure 5.16. To note, the presented architecture is extensible for more units, which calculate a depth map row each, in case more resources are available. An internal controller based on a state machine controls the execution of each AE. Direct Memory Access (DMA) units (not shown in Figure 5.16) read data from and write data to the coprocessor memory. Each unit for calculating a DP matrix (*Calc-DP unit*) uses three DMA read units connected to three memory controller (MC) ports. Each of the eight MCs provide two 64-bit data ports. Since the values of the cost matrices are 32-bit integer, two of them can be read in one cycle by a port. Internally, the data width is reduced to 20 bit (ACP5). This decreases the consumed resources as well as the latency required for each operation. The latter allows the execution of more additional operations or multiplexing input data in front of an operation during a cycle.

Three columns of each DP matrix (DP_{M_y}, DP_{L_y}, DP_{R_y}) are calculated concurrently by a pipeline structure (see Figure 5.17). A controller sets the correct operands for the

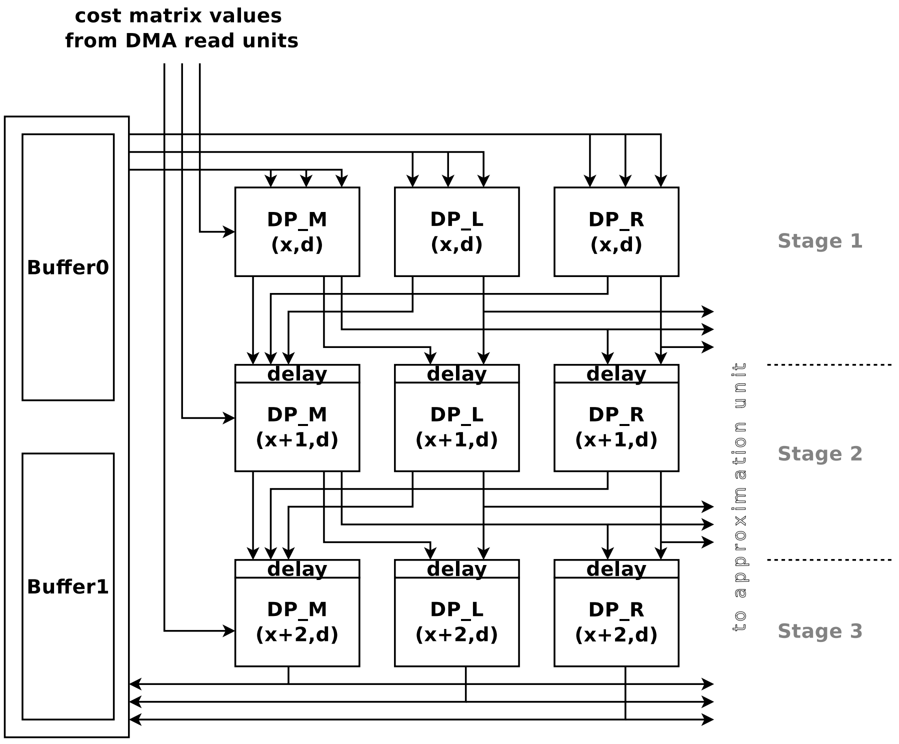


Figure 5.17: Unit for calculating DP matrices called *Calc-DP-unit*.

boundary cases and checks if the required values are valid. Assigning a valid bit to each operation enables it to check which results have to be stored into the buffers and output registers, thus the entire unit always returns valid and correct results. The calculation starts at row 0 and column d_{max} due to Equation (5.1c). According to Equations (5.1a) and (5.1b) some results have to be delayed for further processing in the next pipeline stage. Additionally, result forwarding is integrated to calculate Equation (5.1c) in each of the three *DP_R unit*s. No smoothcosts ($\lambda(x)$) are used for calculating Equation (5.1a) to integrate more *Calc-DP unit*s (ACP3). Integrating smoothcosts would require an additional memory port per unit, hence increases the required bandwidth.

Two buffers are used to store values of a column coming from the last pipeline stages. This data is required to calculate the next columns and transferred to the first pipeline stage if needed. A buffer stores data currently produced by the last stage, another one is used by the first stage for reading previously calculated data. All output data of a *Calc-DP unit* is transferred to an approximation unit *Calc-Disp unit*, see Figure 5.18. This unit calculates the optimal disparity by finding the minimum value in the three current considered DP columns $DP_{M_y}(x)$, $DP_{L_y}(x)$ and $DP_{R_y}(x)$ as well as for column $x+1$ and $x+2$. If the minimum is in column $DP_{L_y}(x)$ or $DP_{R_y}(x)$, the disparity is set to zero, because the pixel is presumably occluded in one of the input images. The disparity is set

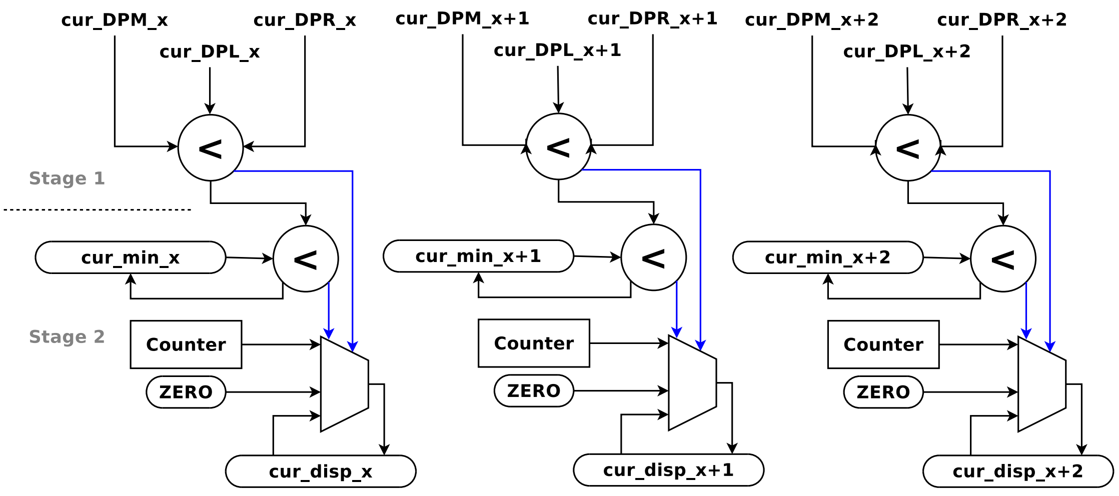


Figure 5.18: Internal structure of the approximation unit determining disparities of three rows concurrently.

to the row index of the cell if the minimum is in column $DP_{M_y}(x)$. This approximation calculates the disparity in a lower accuracy than the backtracking approach but in much less time. The minimum of the current calculated DP values is determined in the fist stage of the *Calc-Disp unit* and passed to the second stage. If the new value is smaller than the minimum, the current disparity (*cur_disp_x*) is updated according to the matrix in which the new minimum lies. The current row index is determined by a counter and is written in the register if the minimum is in $DP_{M_y}(x)$. The register is set to zero for all other cases. The current disparity value is delivered to the *DMA write unit* once the value of the counter is zero.

The amount of units that can be integrated into an FPGA depends on the number of memory ports and the available resources. An AE of the Convey HC-1 allows it to integrate four *Calc-Disp unit*s. To increase performance, the same bitstream is transferred to all AEs. Therefore, four rows of the depth image are calculated in parallel inside an AE, thus 16 rows of the depth map are determined concurrently using all four AEs. A single coprocessor call is invoked to calculate the entire depth map. This avoids additional overhead caused by multiple calls.

Removing data dependencies (ACP1). The next considered FPGA-based system is a System on Chip including a Cyclone V FPGA, which is a low-power FPGA. Instead of VHDL, OpenCL is used to implement the FPGA design. During the implementation and testing phase, I figured out that data dependencies of the algorithm cause an additional OpenCL cycle which doubles the execution time. Therefore, I considered removing some data dependencies on the algorithmic level to avoid this additional cycle. The result is that removing the data dependency from $DP_{M_y}(x-1,d)$ to $DP_{L_y}(x,d)$ enables calculating a depth value per cycle, while only slightly changing the results as shown later.

OpenCL-based Design for a Low-power FPGA-based System on Chip. To reduce the design time and investigate the benefit of the ACPs on a higher programming level, an OpenCL-based design targeting a low-power FPGA is implemented. According to the best practice guide [263], a Single Work-Item Kernel is used. The Intel FPGA SDK for OpenCL [264] Offline Compiler takes care that a fine-grain parallelism is exploited. This is achieved by pipelining the iterations of loops. The pragma `#pragma unroll` leads to a complete unrolling of a loop if sufficient FPGA resources are available. This pragma is used for all *for loops* inside the OpenCL kernel.

The previous VHDL design is transfered to OpenCL C code for the kernel. Avoiding complex source code of the OpenCL implementation is important for the FPGA OpenCL compiler, since the compiler is not able to analyze complex structures efficiently [263], for instance, complex array indexes. Shift registers are used to store the required pixel data of the left and right image. This leads to efficient data access to the needed pixels for the cost calculation within the FPGA. Each of the two shift registers are large enough to store an image row. A subtractor is needed to perform the cost function computation that is based on absolute difference. For the data transfers, two read and a write streamer are used that are connected to the OpenCL kernel required for the DP calculation via so-called channels. Channels avoid costly external memory accesses, because data is passed to the next kernel within the FPGA. This is a unique feature for OpenCL targeting FPGAs. The resulting FPGA design is a streaming architecture that calculates a depth value per cycle.

The SoCrates II board is used as platform of choice [265]. Altera's FPGA SDK for OpenCL compiler v15.1 is used to synthesize the different OpenCL kernels. The measured power consumption of the board is 2W, while running a fully booted but otherwise idle operating system. The maximum power consumption is around 6W for the computation of stereo vision on the FPGA. The execution time of the FPGA design depends on the size of the image. For instance, processing images of size 680×480 require 3.6 ms (277 fps) and images of size 1360×1024 need 13.8 ms (72 fps). The novel FPGA DP algorithm requires more logic for larger images because the internal data width depends on the size of an image row. The design consumes 94% of the available logic resources for 100 possible disparities and images of size 640×480 and 97% for 88 possible disparities and images of size 1360×1024.

A problem that arises with DP-based algorithms is that incorrectly calculated disparities occur between high and low disparities due to smoothing. Smoothing shall increase the density of a depth map. To reduce this boundary effect, the concept of so-called `resets` is introduced, where the previous DP matrix elements are set to zero. To determine reset points, the Sobel filter is used. A reset is triggered if the current image row includes edges. Afterwards, a left-right consistency check is used to further improve the accuracy. Hence, disparities are considered as correct if a depth value is consistent in both calculated depth maps.

5.4.3 Time Series Analysis

My main objective regarding time series analysis is to find a similarity measure that is more accurate than using ED and significantly faster than using DTW. Such a measure shall efficiently exploit the capabilities of FPGAs while getting the best possible results for the similarity.

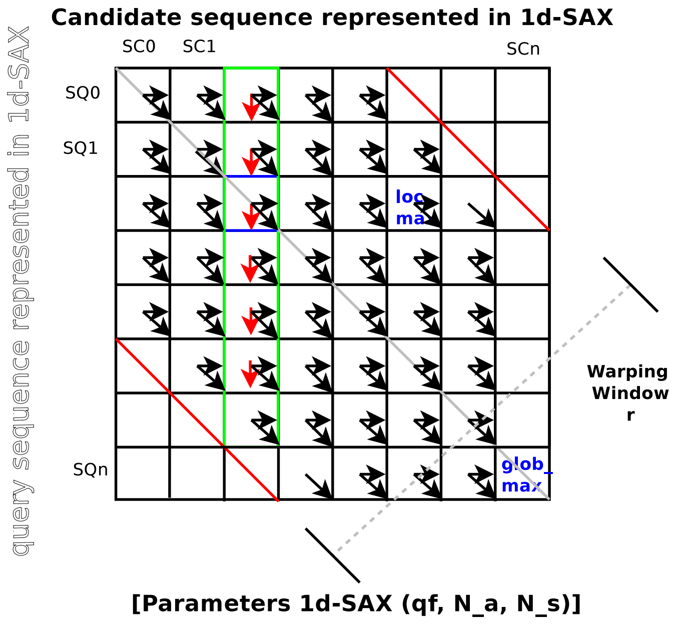
Generating a new similarity measure. By systematically applying the ACPs, I transfer the DTW algorithm into a ACP-based similarity measure. This procedure includes evaluating the impact on the QoR, which is done, for instance, by considering the classification accuracy. Step (1) is to relax data dependencies within DTW to enable data parallelism (ACP1). Normally, complex measures like DTW, SW, or Longest Common Subsequences (LCSS) have time (and space) complexity of $O(n^2)$. Since there exist data dependencies in such measures (cf. Fig. 5.19, the blue element has 3 data dependencies), a parallelization is only possible on the anti diagonals. This aspect is exploited by e.g. systolic arrays or by calculating several measurements concurrently.

Inspired by the approach of Farrar [250] and Rognes [266], I exploit ACP 1 to relax the data-dependencies caused by the top neighbor (red arrows). These authors showed that the SW matrix calculation does not often use the insertion case (red arrows) that prohibits column-wise parallelization during the calculation for biological sequences. Hence, an implementation can calculate all elements in a column concurrently. In Figure 5.19, the elements in the green block can be calculated in parallel now. As the above method is more suitable for local methods (e.g., LCSS and SW) than for global ones (e.g., DTW and ERP), the ACP-based measure is also a local method. This is achieved by integrating the 0 case into the measure. The reason is that values from the upper triangle matrix would have no influence on the global value located at position (n,n) (see Figure 5.19), hence the measure would perform an ED calculation. Step (2) removes the backtracking step for the measure (ACP2), since an alignment is not required for a classification.

Besides relaxing data dependencies, reducing dimensionality or size of time series leads to a more compact representation of a database. Therefore, a query by content is accelerated in two ways: (1) reducing execution time per similarity calculation and (2) better exploiting the memory hierarchy of a computing system [267]. Moreover, it enables us to store time series data in fast but limited internal memory of FPGAs. Quantization methods introduce approximations that enable it to better capture the underlying trend and texture of a time series, hence, e.g., can improve the classification accuracy. Quantization is a method for mapping real values to a smaller set of discrete symbols. Hence, Step (3) is to use a variant of the SAX representation [268] called 1D-SAX [269] (ACP4).

1D-SAX [269] expands SAX by introducing information about the slope of each segment into the quantization step. Therefore, for each segment, a regression line is determined and a segment is represented by the tuple $(\bar{q}_i, s_i)$, where $\bar{q}_i$ is the mean value of a segment and s_i is the slope of the line. To get the $\bar{q}_i$s, Piecewise Aggregate Approximation (PAA) is used. PAA transfers a series of length n into one of length w using $\bar{q}_i = \frac{w}{n}\sum_{j=\frac{n}{w}(i-1)+1}^{\frac{n}{w}i} q_j$. qf $= \frac{n}{w}$ is called the quantization factor. A tuple $(\bar{q}_i, s_i)$ is mapped to a symbol from alphabet A of length $N = N_A * N_S$. Thus, the time series Q is converted to the 1D-SAX-quantified sequence $\hat{Q}$. A user can increase qf to further accelerate the computation with an eventual drawback regarding the accuracy. Further parameters are the allowed warping path represented by R and the sizes N_A and N_S of the alphabet A.

The calculation of the similarity value between two quantified time series $\hat{Q}$ relies on a novel similarity function between two 1D-SAX symbols. Therefore, Step (4) is the design of this novel function $s_{\text{1D-SAX}}(i, j)$ using ACP5 and CP1. This function uses an approximate data type as input and the calculations are performed on demand (CP1).



Candidate sequence represented in 1d-SAX
query sequence represented in 1d-SAX
SC0
SC1
SCn
SQ0
SQ1
SQn
loc max
glob max
Warping Window r
[Parameters 1d-SAX (qf, N_a, N_s)]

Figure 5.19: Schematic for the calculation of the SW matrix between a candidate and a query sequence represented in 1D-SAX. The approximation removes dependencies shown in red. Hence, a global value only depends on values from the lower triangle matrix. Therefore, the measure uses a local value, i.e., the maximum element of the matrix. Elements that are not in the warping window, i.e., between the red lines are ignored. The implementation can exploit to calculate elements in the green block concurrently. The element, where a arrow points to, depends on the origin element of the arrow.

Additionally, the operations are performed in fixed-point arithmetic (ACP5). $s_{\text{1D-SAX}}(i,j)$ represents a novel similarity function between two 1D-SAX symbols. This function is defined as follows:

$$s_{\text{1D-SAX}}(i,j) = \frac{1}{1+\text{hamming}(\hat{q}_i,\hat{c}_j)}, \tag{5.2}$$

where hamming() is the hamming distance between two symbols. Furthermore, $\hat{q}_i$ is an element of the quantized query sequence $\hat{Q}$ and $\hat{c}_j$ of the candidate sequence $\hat{C}$.

Definition of a new similarity measure. The resulting measure is an approximative variant of the SW algorithm using 1D-SAX-quantified time series as input. The resulting matrix recurrence is

$$\begin{aligned}
\text{SWA}_{i,0} &= 0, \text{for } 0 \le i \le w \\
\text{SWA}_{0,j} &= 0, \text{for } 0 \le j \le w \\
\text{SWA}_{i,j} &= \max \begin{cases} \text{SWA}_{i-1,j-1} + s_{\text{1D-SAX}}(i,j) & \text{if } (dis)similarity \\ \max_{1\le k\le i}\{\text{SWA}_{i-k,j} + W(k)\} & \text{if } deletion \\ 0 & \text{else} \end{cases},
\end{aligned} \tag{5.3}$$

where w is the sequence length of the 1D-SAX-quantified time series. The maximum value of the SWA matrix represents the similarity score between two time series. $W()$ is a cost function for deletion of symbols.

FPGA-based Design for the ACP-based Measure. A FPGA-based implementation of Equation (5.3) is designed (see Figure 5.20). To efficiently use the integrated memory resources of an FPGA, one or more query sequences $\hat{Q}$s are transferred to these memories. Since for distance-based mining tasks, distances between the query and all databases sequences $\hat{C}$s have to be calculated, transferring the query multiple times is avoided trough the above approach. Internally, a `query memory` stores query time series. Candidate sequences $\hat{C}_i$s are streamed to the FPGA. Local off-chip memories store these $\hat{C}_i$s and include a part of a database or an entire database, which depends if the FPGAs should calculate distances for one or more databases. Each FPGA performs the similarity measurements for a certain query, but different FPGAs can use other queries as input.

A host CPU starts the FPGA-based execution, while delivering the required parameters, sequence length, warping window R and the number of $\hat{C}_i$s. Each of the m similarity measurement units (`SMUs`) integrated into an FPGA perform a similarity calculation. p internal Smith-Waterman-Stream units (`SW-Stream`) calculate a SWA matrix concurrently. A `SW-Stream` calculates certain lines of SWA sequentially (see bottom right in Figure 5.20). The similarity function (Equation (5.2)) is evaluated on the FPGA dynamically (CP1). A following unit collects all elements of SWA and determines the maximum element, which represents the similarity score. Each FPGA transfers the calculated similarity scores to the host for further processing steps. In order to reduce the resource consumption, fixed-point arithmetic is used for the FPGA design (ACP5). Moreover, the SMU supports the RK-Band as warping constraint but the execution time is independent from this band in the current version. The design is implemented using the high-level language MaxJ on the Maxeler Workstation. According to a manual design space exploration, the best implementable design for a data flow engine of the workstation has 8 SMUs and 8 SW-Streams. The frequency of the design is 100 MHz.

Improve Classification Accuracy. Machine learning (ML) methods can be used in order to improve the classification result. Kate et al. [270] proposed a feature-based representation of a time series by using all calculated similarities between the query sequence Q and all entries in a database C_i as features. This approach down votes noisy training examples that would lead to wrong classifications using a 1NN classification. Moreover, this approach can be extended with more features just by combining all features into a vector. The ML method tries to find a more complex classification hypothesis to improve the classification accuracy. The most time-consuming part of this feature-based approach is the calculation of features as also mentioned by Kate. To drastically reduce this issue, my novel FPGA-based measure is used to create a feature vector Feature-SW-1DSAX=(SW-1DSAX($\hat{Q}$,$\hat{C}_0$), ..., SW-1DSAX($\hat{Q}$,$\hat{C}_{l-1}$)) for a given $\hat{Q}$, where l is the number of entries in the database. I consider *logistic regression*, *decision trees* and *SVMs* with different kernels as ML method.

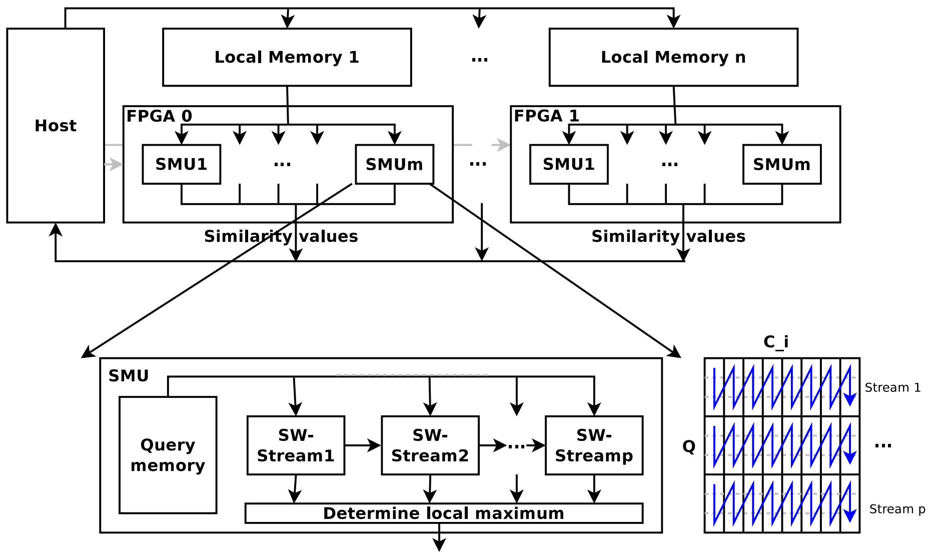


Figure 5.20: FPGA-based architecture design for SW-1DSAX calculations.

Table 5.7: Average kernel execution times over entire uniprot20 database.

sequential		**SSE**		**Convey HC-1**	
original	adapted	original	adapted	1^{st} approach	2^{nd} approach
512.7 µs	607.2 µs	16.1 µs	18.9 µs	31.8 µs	23.1 µs

5.5 Evaluation and Results

In this section, I compare the achieved accuracy and performance of the ACP-based FPGA designs with the original implementations running on a x86 processor. These original implementations may use several cores or exploit the vector extension unit. I omit a comparison against an exact FPGA implementation of the different algorithms, since often such an implementation is not possible or requires a complex redesign of the algorithm in order to get an efficient and optimized FPGA design.

5.5.1 HHblits

For heterogeneous architectures, it is important to know which parts of an application can efficiently be executed on the different resources. Efficient execution is not only influenced by the algorithm itself, but also depends on data size. A resource can be faster for smaller databases than other resources, while for larger databases it is contrary. Therefore, an important aspect is to analyze for which sequence lengths the hardware-based `ungapped score calculation unit` is faster than the SSE-based implementation. Note that using the novel designed and implemented ungapped score calculation unit does neither influence the final QoR of the application nor the result of the first prefilter step. Hence, I do not present accuracy measurements for the ungapped score calculation unit and HHblits.

Measurement of performance for an ungapped score calculation unit. I measure the average kernel execution time for calculating an ungapped score with different implementations (see Table 5.7). For the coprocessor-supported implementation, I have to adapt the original source code of the tool HHblits. The focus is on accelerating the adapted version, but I also compare the results with the original implementation. I measure the execution time over 30 runs for a sequential and an SSE version calculating the ungapped score on the dual-core Intel Xeon 5138 of the Convey HC-1. The first approach for a unit calculating an ungapped score loads the required query profile line dynamically and directly from coprocessor memory. The second approach prefetches the whole query profile into profile memory to calculate all ungapped scores. With the second approach, I achieve speedup of $512.7/23.1 = 22.2$ with an AE including 16 PUs

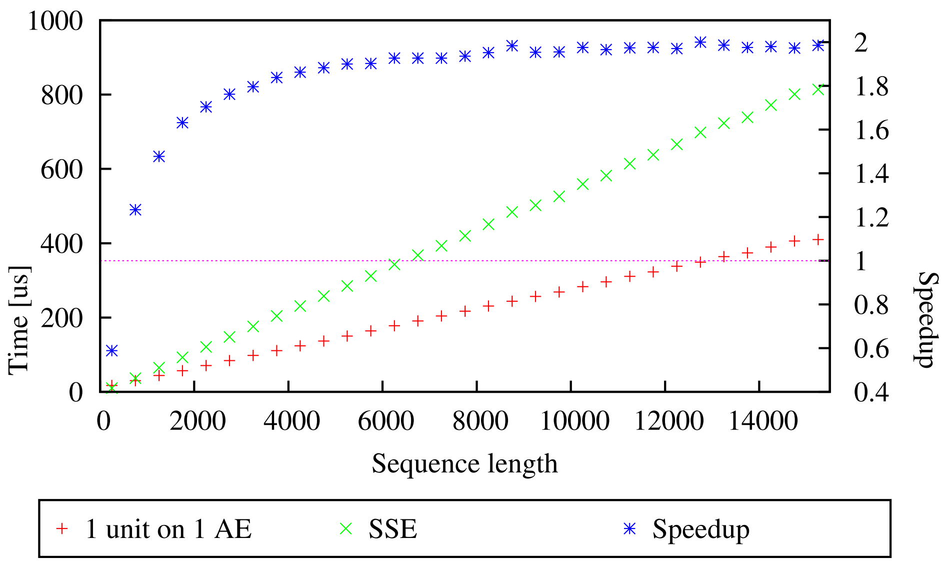


Figure 5.21: Kernel execution time and obtained speedup of single-threaded SSE execution and coprocessor with 1 unit on 1 AE for varying sequence lengths.

for calculating an ungapped score against a sequential implementation using the original profile. The second approach outperforms the first approach. Therefore, I continue with the second approach to calculate the ungapped scores in the final design. After that, I look at how computation time depends on sequence length. To find out precisely for which data size speedup can be obtained, I split the sequences into 31 buckets of length 500 each as in Figure 5.4. I present the kernel execution time of the second approach in relation to single-threaded SSE-supported execution in Figure 5.21. Speedup by means of coprocessor execution can be gained from sequences larger than 500 Byte. For small sequences, the overhead for starting up coprocessor execution has the major influence of the execution time. This influence is negligible for larger sequences. So, for these sequences coprocessor execution is up to factor two faster than on the SSE unit of the Intel Xeon 5138 processor. Considering Figure 5.4, most approximated sequences in the uniprot20 database used for prefiltering are smaller than 500 Byte. Therefore, the average kernel execution time is faster on the SSE unit.

Measuring execution time for calculating all ungapped scores. To evaluate the benefit of the coprocessor design, I measure the execution time of the loop calculating the ungapped scores for all sequences. Both sequential, unaccelerated execution and SSE-supported single- and dual-threaded execution on the dual-core Intel Xeon 5138 of the Convey HC-1 and coprocessor-accelerated execution are measured. As can be seen from Table 5.8, SSE support is really necessary to obtain reasonable execution time. According to Remmert et. al. [249], the SSE version scales nicely for up to eight cores.

The necessary increase of data padding has minor negative impact on the remaining functions of the coprocessor versions. Still, the coprocessor design running with four AEs and four ungapped-score-caclulation units on each AE accelerates the sequential first prefilter step more than 150 times and is even faster than dual-threaded, SSE-supported execution by a factor of $\frac{25.1}{10.6} = 2.37$ for the first prefilter step.

As normally 4*4 sequence comparisons are issued concurrently to the coprocessor, the execution time depends on the longest of the 16 sequences. Using the uniprot20 database the design scales not very well (2.2×) from one to four AEs. Therefore, sequences in the database have to be ordered first to account for their lengths so that only similar data sizes are taken so that concurrent sequence calculations are balanced with regard to their lengths. With such an ordered database, the design including four units is $\frac{50.0}{6.3} = 7.94$ faster than single-threaded SSE-supported execution and $\frac{25.1}{6.3} = 3.98$ faster than dual-threaded execution for the ungapped score calculations. However, it currently incurs additional overhead at runtime for the tool HHblits. Sorting should therefore be done in a preprocessing step upon release of a new database version. Using an ordered database, our design scales slightly better (2.62×) from one AE to four AEs. The bottleneck is hence the sequential comparison against a threshold for one sequence after the other. An often used measure to compare SW implementations is to use giga cell updates per second (GCUPS). The implementation achieves 157 GCUPS.

Currently, the design uses only 4 of the available 16 memory ports of each AE. So, each AE uses $4\ ports\ * 8\ Byte * 150MHz = 4.8GB/s$ of the available coprocessor memory bandwidth of $19.2GB/s$. Therefore, using an FPGA of the newest generation offering more available resources on the chip would make an integration of more ungapped score calculation units possible. Due to the remaining unused bandwidth, 12 additional units could be provided with data. Additionally, according to the internal calculation, a new element of a sequence is needed every fourth cycle. This would theoretically lead to integrating $4 * 16 = 64$ units. By a wise data striping, eight times more units could be integrated because an element of the sequence is only 1 Byte large. This shows that the memory bandwidth poses no limiting factor to the design.

Table 5.8: Average execution time over 30 runs of first prefilter step for entire uniprot20 database.

Hardware	**CPU**	**CPU**		**Convey HC-1**			
Impl.	**seq.**	**SSE**		**3 units**		**4 units**	
Cores/AEs	1	1	2	1	4	1	4
Unsorted database	1601.9 s	50.0 s	25.1 s	28.6 s	13.1 s	23.3 s	10.6 s
Sorted database	1601.9 s	50.0 s	25.1 s	21.7 s	8.3 s	16.5 s	**6.3 s**

Performance evaluation for HHblits. After looking at the execution time for the first prefilter step, I measure the execution time for the entire tool HHblits. I again measure a sequential, unaccelerated version and SSE-supported single- and dual-threaded execution and the coprocessor-accelerated execution using 4 ungapped score units on each AE (see Table 5.9). For the measurements, I use the original uniprot20 database and a database sorted by sequence length. At the moment, this ordering is done at runtime by an adapted quicksort algorithm. Therefore, I measure the execution time for sorting and subtract it from the entire execution time of the tool HHblits. For the sorted and unsorted database, I measure time with sequential coprocessor-accelerated prefiltering as well as with task-parallel, overlapped prefiltering. The sequential version begins with calculating the first prefilter stage by the coprocessor, while the host processor is idle. Once the coprocessor has finished calculations, the host processor calculates the second prefilter stage. A second worker thread for all coprocessor-supported implementations is enabled in HHblits to process the Viterbi algorithm in a task-parallel fashion.

The design including four ungapped score units and using all four application engines is $\frac{1626.2}{33.9} = 48\times$ faster than an unaccelerated, sequential version on a sorted database and $\frac{44.5}{74.1} = 2.19\times$ faster than the original single-threaded SSE-supported version of HHblits. Comparisons between the coprocessor-supported execution and the original one is not completely fair and authentic due to different compilers (see Figure 5.15). However, through a wisely workload-distributed task-parallel design for prefiltering the database, a significant performance gain is achieved.

5.5.2 DP-based Algorithm for Stereo Vision

Two FPGA designs were implemented for stereo vision. The following paragraphs evaluate the designs regarding achieved accuracy, performance, and resource consumption.

Convey HC-1. I compare the proposed design implemented for the Convey HC-1 to methods from StereoMatcher (SM) framework. Namely, the sum of absolute differences

Table 5.9: Average execution times over 30 runs of the tool HHblits for entire uniprot20 database.

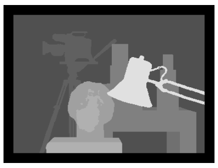
Hardware		**CPU**	**CPU**		**Convey HC-1**	
Impl.		**seq.**	**SSE**		**4 units**	
Cores/AEs		1	1	2	1	4
Unsorted database	Sequential	1626.2 s	74.1 s	44.5 s	51.4 s	38.9 s
	Task parallel	1626.2 s	74.1 s	44.5 s	48.3 s	35.7 s
Sorted database	Sequential	1626.2 s	74.1 s	44.5 s	46.6 s	37.0 s
	Task parallel	1626.2 s	74.1 s	44.5 s	44.0 s	**33.9 s**

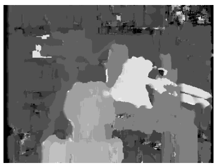
(SM-SAD), simulated annealing (SM-SA), scan-line optimization (SM-SO), dynamic-programming (SM-DP) and graph cut (SM-GC) algorithm. I use the default parameters of the example scripts for each algorithm in the SM framework. Additionally, I compare the accuracy against the top ranked CSM [271] and the middle ranked DTAggr-P [272] algorithm. Figure 5.22 shows the depth maps of the algorithms under consideration.

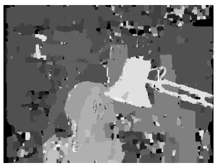
The ground truth visualizes the correct disparities of the objects in the scene, hence it is the correct depth map. According to the human visual perception, the CSM method delivers the best result because of its similarity to the ground truth. Depth maps created by global approaches (CSM, GC) are very smooth and accurate. Since the decision of finding the optimal disparities is based on the local information of a column, the edges of the different objects are not smooth in the depth map calculated by my approach. Objects in the front are calculated more accurately.

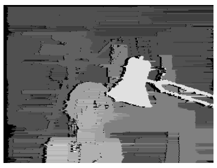
To also get a quantitative value for the accuracy, I use the root mean square (RMSE) error $\sqrt{\frac{\sum_{i=1}^{n}(x_i-t_i)^2}{n}}$, where n is the number of pixels, x_i the calculated values of the depth map, and t_i the ground truth values. The RMSEs of the different approaches are shown in Table 5.10. The DP and local approaches produce slightly less accurate results than the global approaches. The quality of the FPGA-based approximation approach is comparable to algorithms that are ranked in the lower middle range of the Middlebury Stereo Evaluation. However, there is a difference between the results published in the Middlebury Stereo Evaluation and those collected by the execution of the methods in the

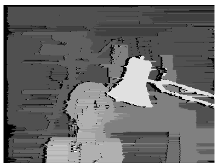
(a) Original Image.

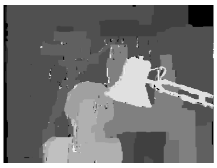

(b) Ground truth.

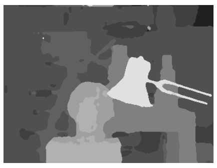

(c) SM-SAD.

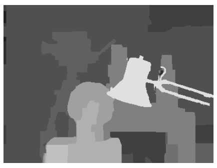

(d) SM-SA.


(e) SM-SO.


(f) SM-DP.


(g) SM-GC.


(h) DTAggr-P.


(i) CSM.

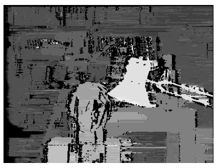

(j) ACP-based design.

Figure 5.22: Depth maps calculated by different stereo correspondence algorithms.

SM framework. This should be caused by different parameters used for the algorithms.

Besides the accuracy, the execution time of the algorithm is important. The execution time of the different methods is measured on a 2.13 GHz Intel Xeon 5138 processor (see Table 5.11). I also present the execution times required for three calculation steps, matching cost calculation (*Costs*), cost aggregation (*Aggr/Transfer*), disparity calculation and optimization (*Opt*). I do not apply the fourth step, disparity map refinement. As shown in Table 5.10 the global algorithms (GC, CSM) offer a good accuracy of the results but the consumed execution time for the GC is between $\frac{159.32}{0.11} = 1448$ and $\frac{684.24}{0.2} = 3421$ times higher than the local approaches (SAD). Increasing the maximum allowed disparity value leads to an enormous increase of the execution time for global algorithms. Algorithms with such a large execution time are not applicable to processing a huge amount of data. So, by using an ACP-based approach, we can handle such a high amount of data. The FPGA-based approximation unit allows us to compute the disparity values $\frac{0,14}{0.000381} = 367\times$ faster than the original DP-based algorithm for the tsukuba image (384×288 pixel). The ACP-based unit computes large images up to 4096×4096 pixels or more in an adequate time (see Table 5.12). The FPGA-based design processes $\frac{1}{3.74 \cdot 10^4 \mu s} = 26$ frames per second (fps). The most time-consuming parts of the FPGA-based approach are the data transfer and the computation of the cost matrix. The latter can easily be integrated into the FPGA design itself. Exactly, this is done for the OpenCL ACP-based variant in the following.

SoCrates II Board. The novel OpenCL ACP-based DP FPGA design is compared with several local matchers, which are implemented on the same SoC. This includes SAD and SHD matchers as well as different preprocessing methods (sobel filter and census transformation). I apply different optimization approaches for local matchers such as

Table 5.10: Root mean square error (RMSE) between depth maps created by different approaches and the ground truth tsukuba image. For the *nonocc* case, only the non-occluded pixels are considered for the error calculation, where and every pixel is used for *all*. *rank* specifies the rank of the algorithm in the Middlebury evaluation.

	kind	**nonocc [RMSE]**	**rank**	**all [RMSE]**	**rank**
CSM	global	0.82	1	1.2	3
SM-GC	global	1.08		1.40	
SM-DP	DP	1.54		1.78	
SM-SAD	local	1.61		1.78	
SM-SSD	local	1.67		1.84	
SM-SO	DP	1.67		1.88	
DTAggr-P [272]	local	1.75	76	2.10	61
SM-SA	global	1.84		2.03	
FPGA-AC	approx.	3.24		3.34	

Table 5.11: Average execution time for different algorithms calculating a depth map for tsukuba.

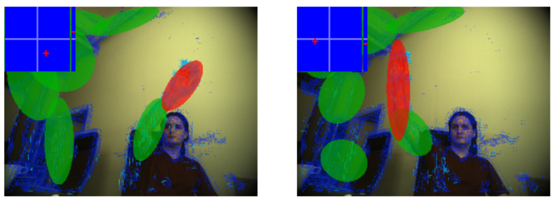
	Execution time [s]							
	Number of disparities: 16				**Number of disparities: 32**			
	Costs	**Aggr/Transfer**	**Opt**	**Total**	**Costs**	**Aggr/Transfer**	**Opt**	**Total**
SM-GC	0.07	-	159.31	159.32	0.13	-	683.99	684.24
SM-DP	0.07	-	0.14	0.22	0.13	-	0.31	0.45
SM-SAD	0.06	0.03	0.01	0.11	0.11	0.06	0.01	0.20
SM-SO	0.07	-	0.14	0.21	0.13	-	0.38	0.52
SM-SA	0.07	-	258.51	258.59	0.14	-	503.19	503.33
FPGA-AC	0.05	3.41 ms	**381 µs**	0.06	0.11	5.99 ms	**603 µs**	0.12

Table 5.12: Average execution time [µs] of the optimizing step for different algorithms.

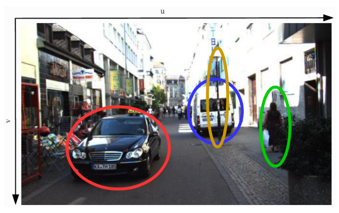
	Execution time [µs]		
Image size	**1024× 1024**	**2048× 2048**	**4096× 4096**
Local method	$5.30 * 10^4$	$1.76 * 10^5$	$8.28 * 10^5$
Global method	$2,54 * 10^9$	$1,04 * 10^{10}$	$4.31 * 10^{10}$
DP method	$1.33 * 10^6$	$5.89 * 10^6$	$2.13 * 10^7$
FPGA-AC	$2.46 * 10^3$	$9.43 * 10^3$	$3.74 * 10^4$

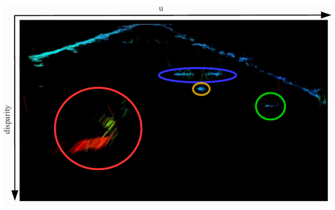
reducing the internal data width, remove operations (full, sparse), a novel cost function and vary the considered neighboring pixels (window) A_q. Optimization *3 bit* means that only the leading 3 bits of a data type are used and *novel* indicates that I use my novel cost function for SHD. The execution time only depends on the size of the image for all FPGA designs. Processing images of size 680×480 requires 3.6 ms (277 fps) and images of size 1360×1024 need 13.8 ms (72 fps). The resource consumption between the same matcher varying in supported image size is roughly the same, only the novel ACP-based DP algorithm requires more logic for larger images because the internal data width depends on the size of an image row. Post-processing is not considered, which further can improve the accuracy or visibility of depth maps. However, it is used for approaches discussed in Section 5.7. Hence, a direct comparison is not fair. Detailed information about my implementation for local approaches can be found in [BES+17].

Moreover, I successfully use one of these local approaches exploiting AC for 6D vision [BSE+17]. This leads to a gesture control system and to the ability of extracting valuable information from a traffic scene. The goal of the gesture control system is to find a dominant moving object in the scene as well as determining direction and speed (see Figure 5.23a). Therefore, I calculate 6D points using the system and cluster them on the CPU. The cluster with the highest movement is considered as the dominant object, for instance, the hand. Moreover, a qualitative analysis shows that the approach



(a)





(b) (c)

Figure 5.23: Detecting moving hand (a). A real-world traffic scene(b [253]+c).

is useful for providing information about a traffic scene (see Figures 5.23b and 5.23c). Objects can be found in the 6D field by using a clustering of similar 6D vectors. A line connects points in the current frame with the same point seen at the previous frame and the color presents the distance, hence the velocity. Therefore, the relatively fast movement of the black car (red circle) is seen as red lines. The 6D vision approach is able to find the critical objects in the scene.

The different stereo vision designs are evaluated regarding accuracy of the calculated depth map. A quantitative evaluation is performed by using 200 images from the KITTI benchmark [253] and four from the Middlebury framework, see Table 5.13. According to the KITTI benchmark, a depth value is correct if the difference to the ground truth value is less than 3. I use the same quality function. Since the ground truths of the KITTI benchmark do not provide a depth value for all objects in the scene, checking the accuracy and density for those pixels is not possible. Hence, I cannot verify the incorrectness of found depth values in these regions.

The resource consumption of SAD approaches is high and optimization approaches like reducing data width results in a poor accuracy. The only exception in terms of accuracy is the sparse approach. In terms of supported disparity values, SHD approaches outweigh the others. The SHD sparse approach significantly increases the supported disparities and has a reasonable accuracy. Census-based matchers offer a good trade-off between accuracy of the results and supported disparities. Additionally, they achieve a high accuracy even with small A_q.

The novel ACP-based algorithm has the highest accuracy on the KITTI images and provides a high number of supported disparities. For the KITTI camera setup, the algorithm calculates real world distances between 4.3 m and 432 m. Census approaches

Table 5.13: Design space exploration and evaluation of the accuracy.

Preprocessing	Local Method	Optimization	Resource Cons. Max. Disp.	Resource Cons. Cons. Logic	Middlebury accuracy	KITTI accuracy
Sobel	SAD	A_q=5x5	76	90%	83.53%	90.96%
Sobel	SAD	A_q=5x5, 3 bit	52	37%	29.39%	81.68%
Sobel	SAD	A_q=7x7	42	89%	90.19%	89.56%
Sobel	SAD	A_q=9x9	26	89%	88.67%	86.35%
Sobel	SAD	A_q=11x11, sparse	54	87%	90.63%	91.38%
Sobel	SHD	A_q=7x7, *novel*, 3 bit	180	97%	74.52%	87.84%
Sobel	SHD	A_q=9x9, *novel*, 3 bit	120	99%	84.44%	89.99%
Sobel	SHD	A_q=11x11, *novel*,3 bit	74	88%	89.25%	91.24%
Sobel	SHD	A_q=15x15,*novel*, sparse, 3 bit	140	95%	86.71%	89.34%
Census16	SHD	A_q=5x1	156	91%	79.30%	89.42%
Census16	SHD	A_q=9x1	90	93%	85.79%	91.13%
Census16	SHD	A_q=3x3	90	94%	85.13%	90.59%
Census16	SHD	A_q=5x5	30	85%	90.65%	87.08%
Census24	SHD	A_q=3x1	190	99%	81.60%	88.17%
Census24	SHD	A_q=5x1	62	61%	86.21%	89.14%
Census24	SHD	A_q=3x3	62	96%	88.20%	88.85%
novel ACP-based DP		-	100	94%	83.91%	94.26%
DP (orginal)	original SW version		-	-	92.91%	83.85%

can even detect distances that are closer to the camera (up to 2,3 m). Interestingly, the original DP algorithm provides the highest accuracy on the images that are part of the Middleburry stereo matcher framework, but badly performs for the KITTI benchmark. On the other side, our novel algorithm badly performs on scenes that include overlapped objects or only single object like rectangles, where striking is a major issue (Middleburry images). However, in many applications like self-driving cars the front object is more important, and our algorithm calculates the distance of these objects very accurate as seen by the KITTI image results.

In order to make a qualitative evaluation of the different approaches, a visual comparison is applied. Thus, an already existing stereo vision system[5] is used. This system additionally provides a laser projection to enable texture on all areas, which improves the density of local methods [273]. The recorded image size of both cameras is 1360×1024, and the overlap of both images is 1024×1024. The used static scene and the ground truth are shown in Figure 5.24. The texture on the foreground board is projected by the infrared laser. A correct depth map for the scene has a color gradient from red (objects in the front) to blue (background). The front object requires that the matcher finds disparity values greater than 130, which cannot be achieved by each implemented matcher. The results of different matchers are shown in Figure 5.25.

Since no post-processing filters are applied, invalid values are perceived as noise. SAD-5x5 returns a relatively dense depth map, but only 76 disparities can be found, hence it cannot find objects close to the camera like the panel in the upper part of the image. Reducing the data width enables reducing the used logic, but less accurate

[5]6D vision technology by MYESTRO Interactive GmbH

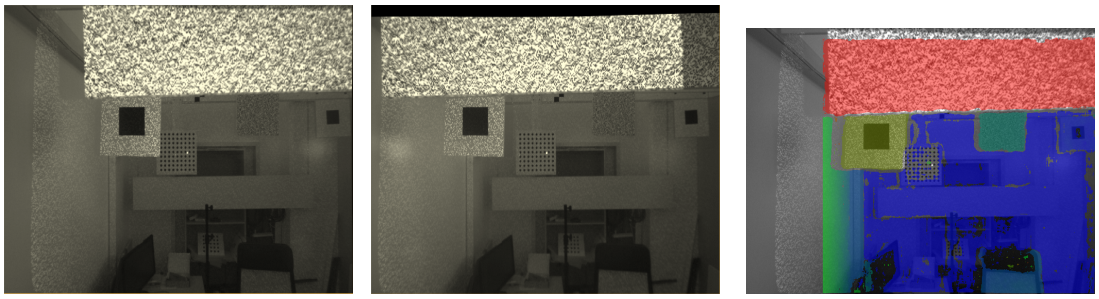

(a) Left camera (b) Right camera (c) Ground truth depth map.

Figure 5.24: Left and right image recorded by the stereo vision system.

values are found (see SAD-5x5-3bit). Larger window sizes as used for SAD-7x7 and SAD-9x9 lead to a more dense depth map, but the increased resource usage drastically reduces d_{max}. Therefore, the distance to close objects can not be calculated. Using a larger window size in combination with the sparse approach as used in SAD-11x11-sparse can slightly increase the number of considered disparities. It results in a dense depth map but objects in the front are enlarged.

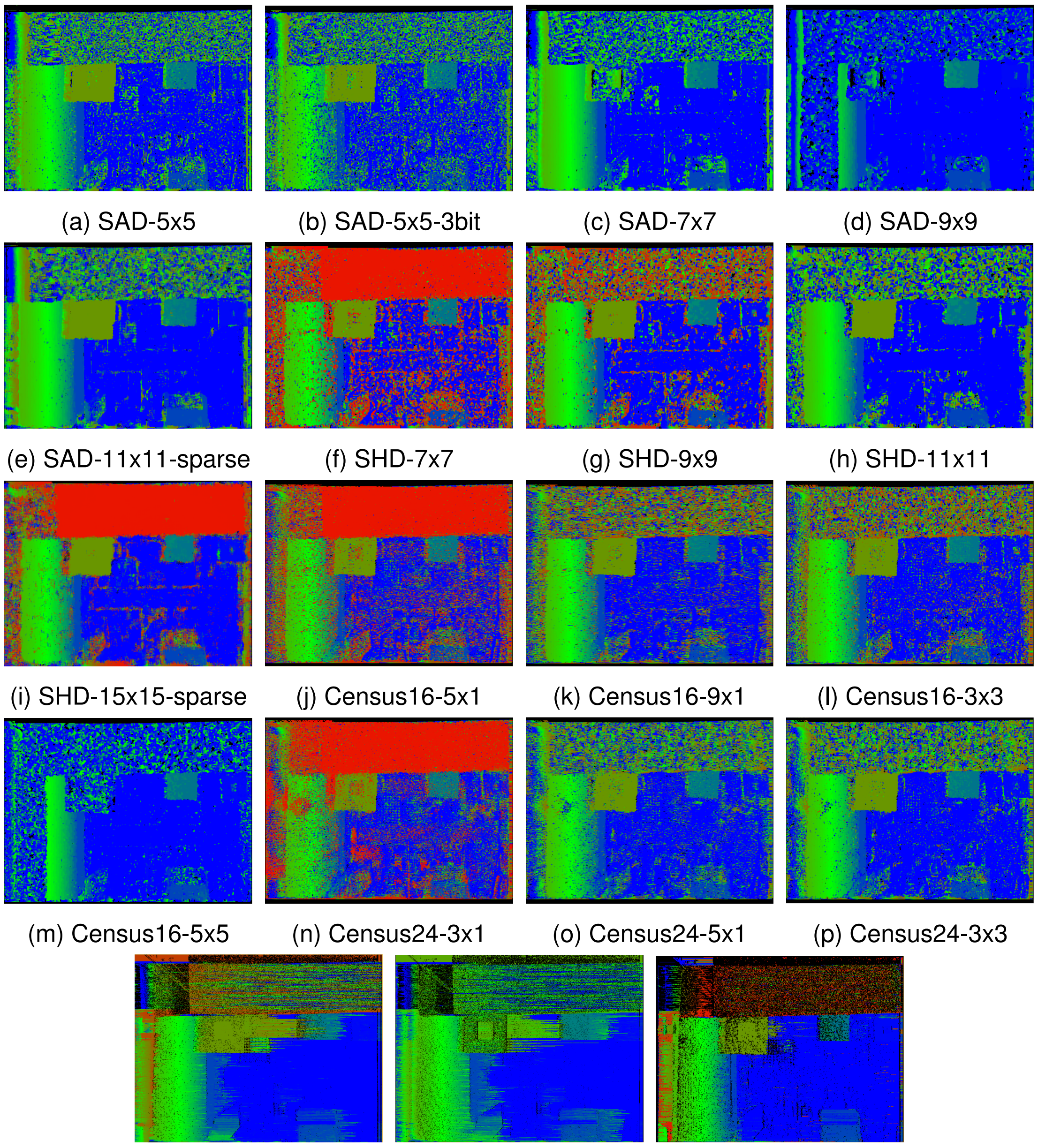
Since using the hamming distance drastically reduces the amount of used logic, SHD-7x7 enables $d_{max} = 186$. Therefore, accurate distances are calculated for objects in the front. Wrong distances tend to be large as can be seen by the red noise. Similar to SAD, larger window sizes lead to more dense depth maps (SHD-9x9, SHD-11x11). SHD-15x15-sparse is comparable to SAD-11x11-sparse, but more than doubles d_{max}.

Census-based approaches enable the use of so-called line matchers, which only consider values of a line. Compared to SAD and SHD, this is feasible because information about adjoining image rows is given by the census transformation. Census16-5x1 enables that d_{max} is very high, but increasing the window size (Census16-9x1) results in a more dense disparity map. But again, increasing the window size decreases d_{max}. The same behavior holds for Census16-3x3 and Census16-5x5. Going to Census24 methods increases the logic consumption, hence d_{max} decreases but no benefit arises for accuracy and/or density.

The approximated DP matcher (DP-SM) creates a very dense depth map. But similar to the original DP approach smearing occurs at object boundaries, hence it is not guaranteed that the disparity value is valid. The combination of the left-right matching combined with the reset approach yields a more accurate depth map but unnecessary resets introduce incorrect disparities (adapted ACP-based algorithm).

Next, I consider the importance of the laser texture. For this, some methods are applied to a real world traffic scene, where a pedestrian is moving, see Figure 5.26. The trend in this real-world scenario is that the behavior of each method is similar to the experimental setup. Hence, the laser texture is not required in real-world scenarios.

I apply all methods to an object detection framework (see Figure 5.27). Using this



(a) SAD-5x5 (b) SAD-5x5-3bit (c) SAD-7x7 (d) SAD-9x9

(e) SAD-11x11-sparse (f) SHD-7x7 (g) SHD-9x9 (h) SHD-11x11

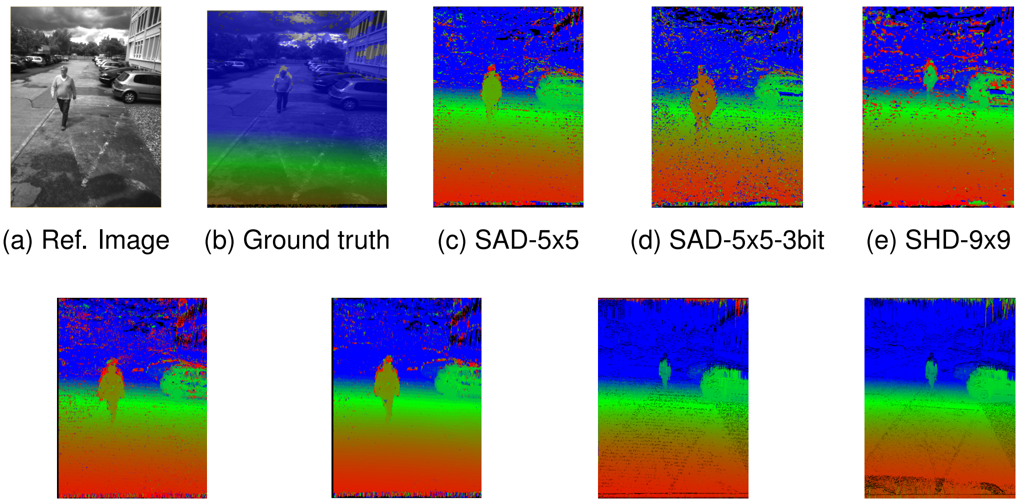
(i) SHD-15x15-sparse (j) Census16-5x1 (k) Census16-9x1 (l) Census16-3x3

(m) Census16-5x5 (n) Census24-3x1 (o) Census24-5x1 (p) Census24-3x3

(q) DP-SM(680x512) (r) DP-SM(1360x1024) (s) Adapted ACP-based DP algorithm.

Figure 5.25: Depth maps returned by the different methods aligned to the left image.

framework, pedestrians can be detected in a real world traffic scene. A depth map is used as input. Such a system has potential to detect collisions of a subject with objects.



(a) Ref. Image (b) Ground truth (c) SAD-5x5 (d) SAD-5x5-3bit (e) SHD-9x9

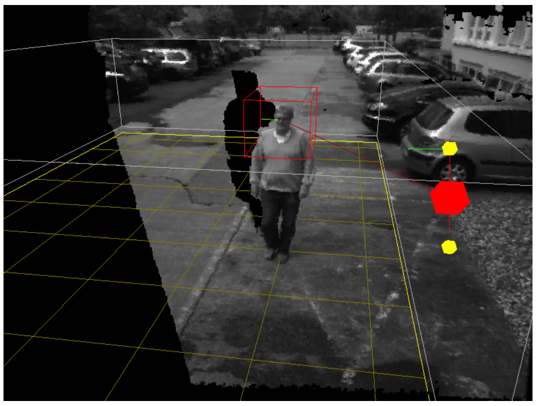
(f) Census16-5x1 (g) Census16-9x1 (h) DP-SM (680x512) (i) DP-SM (1360x1024)

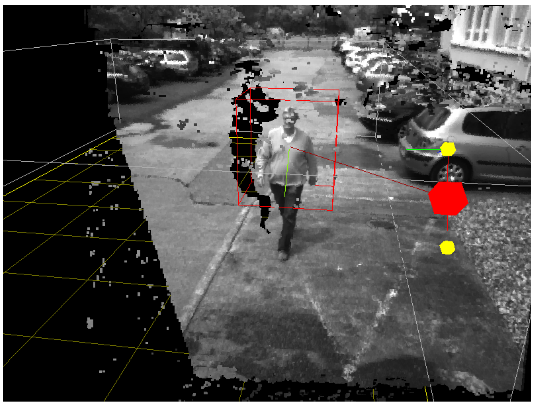
Figure 5.26: Depth maps returned by the different methods aligned to the left image.

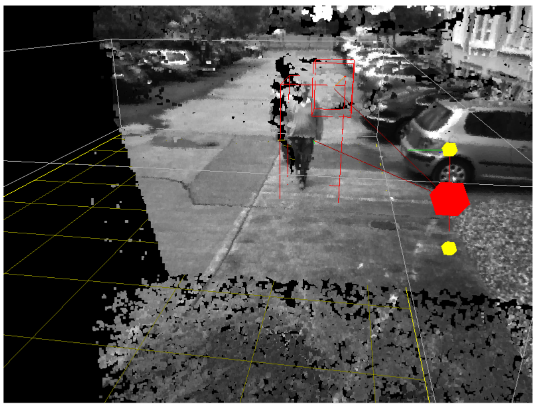
As ground truth, I use an object detection application provided by a stereo vision system[6]. The white rectangle shows the detection range, the yellow ones represent the distances. The stereo camera is shown as yellow points. The right point shows that there is a potential collision with a pedestrian. Often, the methods found incorrect objects, i.e., they find objects where no real object is, but most of them were able to detect the pedestrian, hence a post-processing step should help avoiding a false detection. The novel ACP-based DP algorithm is the only approach that is able to find the correct objects in the real world scenario (the red bounding box surrounds only the pedestrian).

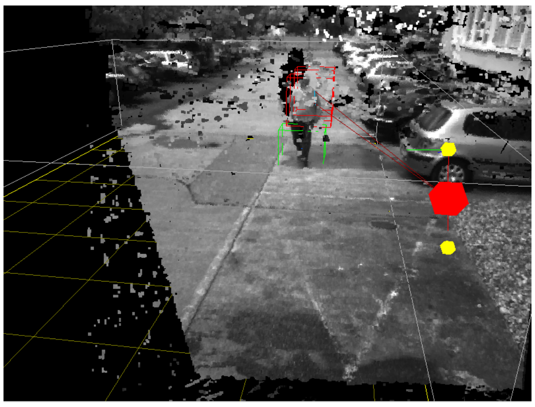
To sum up, SAD-based designs have a high resource consumption. SHD needs a larger window size to provide similar accuracy but still requires less resources, hence offers more supported disparity values. Going from 16 bit to 24 bit Census is not useful. Census16-based approaches are similar in terms of accuracy and resource consumption to SHD approaches using our novel cost function. Our novel DP algorithm outperforms all other approaches for typical traffic scenes in terms of accuracy and supports a high number of disparities. Hence, the most promising candidate for an integration into an object detection system for self-driving cars of the considered approaches is the novel ACP-based DP FPGA design.

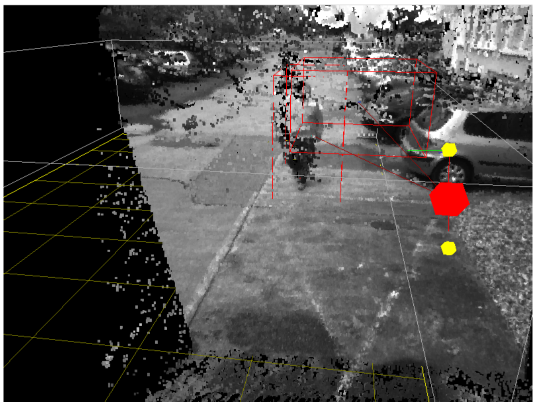
[6]I use an already existing stereo vision system (6D vision technology by MYESTRO Interactive GmbH).


(a) Ground truth


(b) SAD-11x11-sparse


(c) Census16-5x5


(d) SHD-11x11


(e) Census24-5x1

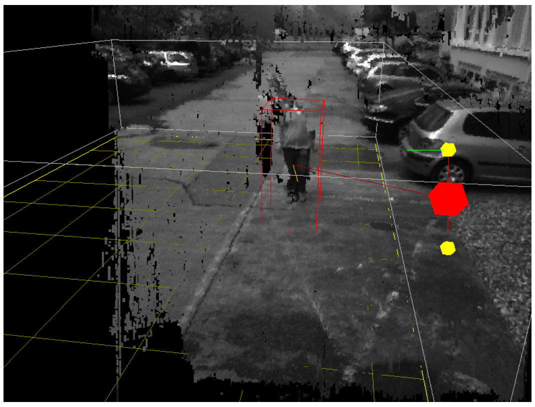

(f) Novel ACP-based DP algorithm

Figure 5.27: Detection of a pedestrian using different implemented stereo matcher

5.5.3 Time Series Analysis

As evaluation data, I use all databases from the UCR archive [274], which includes time series data from completely different domains. The current version of the UCR archive includes 85 different time series databases, where the smallest database has 16 entries and the largest one 8926. The length of the time series varies between 24 and 2709 and the number of classes between 2 and 60. For each database, there is a number of queries varying between 20 and 8236. The total zip compressed archive has about 350 MB. In all cases, the 1D-SAX quantization is performed as preprocessing step.

I evaluate the influence of different parameters to the ACP-based measure and compare it to ED and DTW in terms of a 1NN classification, where the classification accuracy ($= \frac{\text{correct classifications}}{\text{number of classifications}}$) and the execution time represent the quality metric. Additionally, I check the occurrence of the top DTW candidate in the top ten/twenty time series returned by a query by content search using SW-1DSAX. Afterwards, I investigate if I can determine ahead of time when SW-1DSAX outperforms ED in terms of classification accuracy. Then, I evaluate the feature-based approach in terms of classification accuracy.

Influence of relaxed data dependency (ACP1). I apply a 1NN classification using the ACP-based measure. The result is that removing the insertion case has only minor influence on the classification accuracy (cf. Figure 5.28). The average classification error increases by 1.1% to 28.6%. This is slightly better than using ED. The novel method is better on 47 databases.

Selection of Parameters. The ACP-based measure has four parameters and $W() = -1$, see Equation (5.3). Firstly, I consider the influence of the warping window R on the classification error for different databases. A leave-one-out cross-validation is used on the training data to find the optimum $R \in \{0\%, 100\%\}$ on each database, where qf$= 2$, $N_A = N_S = 8$. Using the extracted optimal R per database, the average classification error on the training data is 0.267. I calculate the classification errors of the test sets for a 1NN using SW-1DSAX5, SW-1DSAXopt, SW-1DSAX100, DTWopt and ED (see Table 5.14), where `opt` means that for each database, the warping window is used which is found optimal on the training set. Additionally, classification error comparisons between all used methods are shown. The comparison is presented as win/loss/tie between two methods on the UCR archive, where the column method is the base line. The influence of the optimal warping window is less for SW-1DSAX compared to DTW, but small R enables a faster execution of the ACP-based similarity measure. Finding the optimal R for SW-1DSAX requires a suitable amount of available training data. Especially, for small training sets one should use $r = 5\%$ or $r = 100\%$, respectively. SW-1Dsax performs better than ED on more than half of the databases. It also outperforms DTW on one third of the databases.

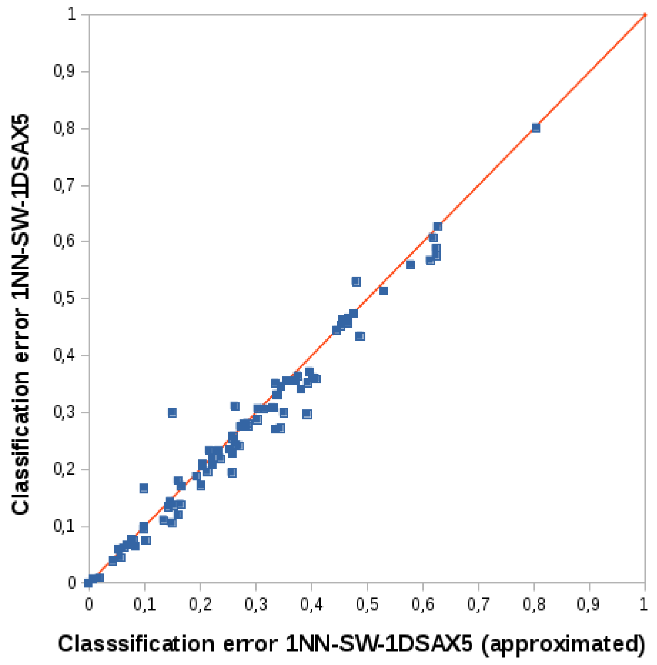


Figure 5.28: Classification error rate of SW-1DSAX compared with approximated SW-1DSAX with 5% SC Band. For databases above the line approximated SW-1DSAX is better, below line SW-1DSAX is better.

Table 5.14: Comparison of classification error between different (dis)similarity measures on the entire UCR archive.

	SW-1DSAX5	**SW-1DSAXoptr**	**SW-1DSAX100**	**DTWopt**	**ED**
class. error	0.286	0.288	0.282	0,237	0.288
SW-1DSAXopt	35/37/13	-	-	-	-
SW-1DSAX100	28/34/23	24/32/29	-	-	-
DTWopt	21/63/1	24/59/2	26/58/1	-	-
ED	47/37/1	44/40/1	47/36/2	54/8/23	-

Secondly, I determine the best values for all four parameters using leave one out cross-validation. For such a grid search, the parameters space is $R \in \{0, 1, ..., 100\}$, $L \in \{2, 4, 8, 10, 15, 20, 25, 30, 35, 40\}$, $N_s \in \{2, 4, 8, 16\}$, and $N_a \in \{2, 4, 8, 16, 32$ $, 64\}$. The product $N = N_a * N_S$ is restricted to a maximum of 256, which is equivalent to the original 1DSAX setup. The total amount of combinations is 21 for the alphabet lengths, hence the total number is 21210. Applying this grid search to the complete UCR archive would be very time-consuming, therefore I restrict R to $[optr - 2, ..., optr + 2]$. The number of combination decreases to 1050, but this still requires 2.7 days on a CPU-based system. However, this approach does not guarantee to find the best parameters.

For example, on the CBF database the optimal parameters are $R = 13, qf = 8, N_a = 8, N_s = 2$ with trainings classification error 0.033. But with restricted R we find $R = 5$, $qf = 15$, $N_a = 4$, $N_s = 2$ with a training error of 0.066. Therefore, it is worth it to perform a full grid search for a dedicated domain.

Moreover, such a grid-based parameter search requires a certain amount of training data in order to avoid over-fitting of the parameters. According to the results on the test data, it turns out that databases with more than 300 training examples determine parameters that perform better than the standard parameters. Databases with a smaller amount of training data are not reliable in terms of parameter determination. Using the grid based parameters for databases with more than 300 training elements (see Table 5.15) results in a 0.012 better performance in terms of classification error on average, 0.229 in the best case and -0.037 in the worst case compared to SW-1DSAXoptr.

FPGA-based Execution. As already mentioned, the execution time of the FPGA design is independent of the parameter R. The FPGA design is restricted to time series with a minimum length of 80 after quantization. The design outperforms the single-threaded SW-1DSAX5 version running on the Maxeler host CPU by a factor of 2.9 on average in terms of execution time, where the single-threaded SW-1DSAX5 outperforms single-threaded DTW by a factor of 2.4. On two databases, LargeKitchenAppliances and StarLightCurves, the FPGA design is about two times faster than an eight-threaded SW-1DSAXoptr version running on the host CPU. Since Maxeler offers a cloud-based FPGA cluster, I theoretically calculate the amount of FPGAs required to outperform the 40-threaded versions of DTW, SW-1DSAX-5, and SW-1DSAX-100, where both approximated SW measures use an AVX implementation (see Section 5.6). 20 FPGAs would outperform the 40-threaded version on most of the databases. For some databases one or two FPGAs outperform the DTW execution time on the server system. The FPGA execution is suitable for large time series and often in combination with a required high warping window.

Besides 1NN classification, I also consider kNN classification. Given a query, the task is to find the most similar time series using a certain similarity measure. The top similar candidate sequence is found per test query for each database using DTWoptr

Table 5.15: Comparison of classification error between different (dis)similarity measures on databases from the UCR archive that have more than 300 training elements.

	SW-1DSAXopt	**DTW**	**ED**
class. error	0.279	0.229	0.291
DTWopt	8/26/0	-	-
ED	22/12/0	23/4/7	-

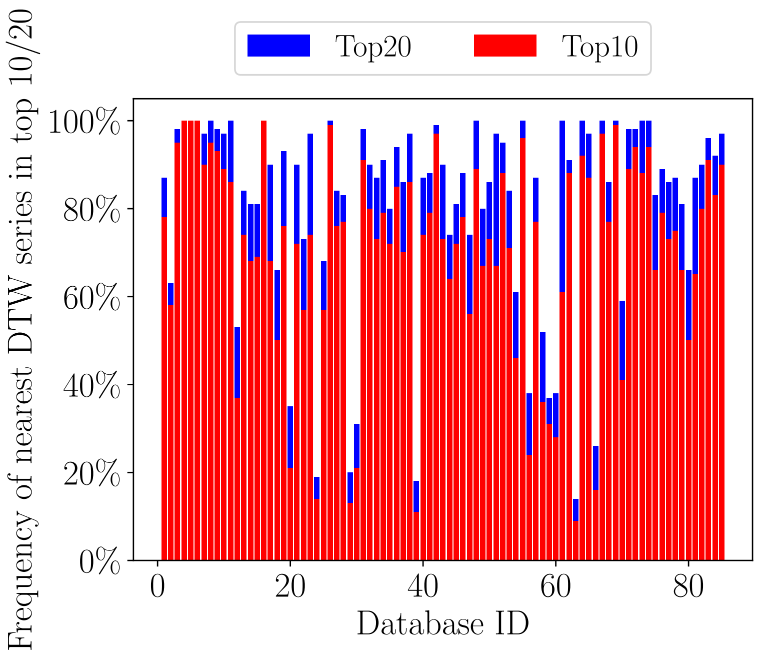


Figure 5.29: Percentage of the occurrence of the top DTW candidate in top of 1D-SAX.

and a check if this sequence is in the top 10 or 20 of 1NN-1DSAX[7]. The results per database are shown in Figure 5.29. In roughly 81% of the cases the top DTW candidate for each query is part of the top 20 of 1D-SAX and roughly 70% are in the top 10. For 33 databases the occurrence is more than 90%, for 63 more than 80% and only 10 databases are below 50%. Hence, on most of the databases we can accelerate the query by content while the approximated results are still valuable.

No distance measure can outperform all other measures in terms of accuracy, hence it is important to know beforehand which measure shall be used. Therefore, it is crucial to predict for which problems the ACP-based method is more accurate than ED or DTW. A simple way is to perform a cross-validation on the training set in order to select the measure for classifying unknown objects [275]. Using leave-one-out cross-validation and an 1NN classification, the expected accuracy gain is

$$gain_{\text{exp}} = \frac{accuracy_{\text{SW-1DSAX}}}{accuracy_{\text{ED}}}. \tag{5.4}$$

If the expected gain is greater than one, it is expected that the method will outperform ED in terms of classification accuracy on the given database. The classification error is determined on the test set for SW-1DSAX and ED. The actual accuracy gain is calculated

[7]Recapitulate that I use SW-1DSAX5 in case the database has less than 300 items, else I use SW-1DSAXopt.

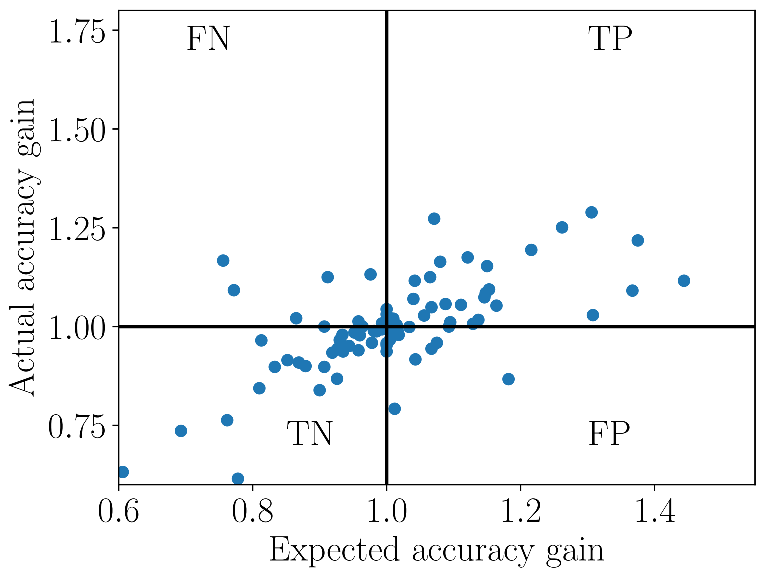


Figure 5.30: Expected and actual classification accuracy gained regarding ED.

by dividing these two numbers. The results are shown in a Texas Sharpshooter Plot (see Figure 5.30).

According to Batista et al. [275], the plot is divided into four regions. The true positive (TP) region includes all databases shown as points in Figure 5.30 for which an accuracy gain is expected and this assumption is true. The true negative (TN) region includes all databases for which I can correctly estimate to be worse than ED. All databases for which I predict that my method is worser than ED, but the accuracy actually increases lies in the false negative (FN) region. Unfortunately, the false negative (FN) region indicates that we lose the chance to improve the accuracy. The only bad region for my approach in terms of accuracy is the false positive (FP) region which includes all databases for which I wrongly expect an accuracy gain. Points, i.e., databases that are close to (1,1) only have a minimal magnitude of accuracy improvement or decrease. 13 databases fall into the FP region, where three have an accuracy decrease of more than 10%. Predicting in which cases my method outperforms ED is especially difficult for small training databases. For a considerable number of databases, it can be determined ahead of time when the ACP-based method outperforms ED.

Feature-based approach. As discussed in Section 5.4.3, I apply the calculated similarities of SW-1DSAX5 as features for different machine learning approaches. I exploit the python library scikit. I use the similarities between the training data elements as features to learn the different classifiers. The aforementioned ML methods are compared to each other in Table 5.16. The LR classifier-based method outperforms ED on 61 databases.

Table 5.16: Comparison of different ML methods to improve the classification accuracy.

	SVM linear	**SVM poly**	**LR**	**RFC**
class. error	0.272	0.250	0.248	0,275
SW-1DSAX5	52/32/1	53/31/1	61/24/0	45/39/1
SW-1DSAX100	51/32/2	53/31/1	59/24/2	45/38/2
DTWopt	36/49/0	39/46/0	42/43/0	29/56/0
ED	56/28/1	56/28/1	61/23/1	54/29/2

Notice that on 21 databases ED is the ideal measure. Better results are achievable by using DTWr distances as features [270], but this results in a higher execution time. I am able to outperform DTWoptr on about 50% of the databases in terms of accuracy and execution time using LR.

5.6 ACP-based Vector Extension Unit Design

I develop an implementation that uses SIMD instructions to perform the ACP-based similarity measure (see Algorithm 5.2). The algorithm uses SIMD instructions as much as possible, while reducing the scalar operations to a minimum. The only critical part that hampers the performance is the shifting step which cannot be omitted due to necessary data dependencies. The implementation uses a query sequence $\hat{Q}$, a candidate sequence $\hat{C}_i$, and a warping constraint R as input. N represents the number of lanes, i.e., the amount of parallel processing elements in the VEU. Therefore, N depends on the certain VEU and also on the used data type. The latter means that the number of processing elements varies between different data types for the same VEU. The algorithm returns the maximum local similarity score in the DP matrix. I evaluate the ACP-based measure and run it on a NUMA system. This NUMA system has an Intel Xeon E5-2670 v2 on each of the two sockets. The base frequency is 2.5 GHz and each processor has 10 cores with 2 HW-threads each. The E5-2670 has 20 MB smart cache and AVX as instruction extension. I consistently use gcc in Version 5 for compiling my source code and OpenMP to enable a parallel execution of several similarity calculations for the same query. The measured execution times are shown in Tables 5.17 and 5.18.

The ACP-based 1DSAX5 is on average 3.2× faster than DTWopt, but it mainly depends on the warping window. Small R together even with larger time series length results in a slow-down of the execution on the AVX units for example on the HAM database. However, in these cases, I am still faster by using the sequential variant of my method. In the case of the LargeKitchenAppliances database, which needs a high warping window, I achieve a speedup of 29.4. However, the most interesting databases are those for which we run significantly faster (speedup > 2) as DTWopt and have a higher accuracy

Algorithm 5.2: SIMD-based implementation for VEUs.

```
Input: $\hat{Q}$ // query sequence
Input: $\hat{C}_i$ // candidate sequence
Input: $\hat{R}$ // warping constraint
1  num_blocks = $(|\hat{C}_i| + N - 1)/N$;
2  bitmap=calc_bitmap(r,$|\hat{Q}|$,$|\hat{C}_i|$); // Array for invaldating matrix elements
3  initialize other variables;
   // Loop over DP matrix columns
4  for j=1 to $|\hat{C}_i|$ do
5      SWAP(curr,prev);
6      $i_{\text{init}}$ = $\max$(0, (j-1-r)/N);
7      if $i_{init} == 0$ then
8          shift_val_vec[0] = 0;
9      else
10         shift_val_vec[0]=prev[$i_{\text{init}}$-1][N-1];
11     end
       // Loop blockwise over DP rows
12     for i=$i_{init}$ to $\min$((j-1+r)/N, num_blocks-1) do
13         sim_score_vec = calc_sim_hamming(); // calculates N similarity score
              values
14         prev_shift_vec = SLL(prev[i],1);
15         prev_diag_vec = prev_shift_vec | shift_val_vec;
16         diag_vec = prev_diag_vec + sim_score_vec;
17         diag_vec = diag_vec | bitmap[(j-1)*num_blocks+i];// Invalidate matrix entries
18         best_so_far_vec = $\max$(diag_vec,best_so_far_vec);
19         left_vec = prev[i] + W_vec;
20         left_vec = left_vec | bitmap[(j-1)*num_blocks+i];// Invalidate matrix entries
21         curr[i] = $\max$(diag_vec, left_vec) ;
22         shift_val_vec[0]=prev[i][N-1];
23     end
24 end
25 return $\max$(best_so_far_vec);
```

than ED. This holds for 19 databases for SW-1DSAXoptr and 39 for SW-1DSAX5 and I also outperform DTWopt in terms of accuracy on 26 databases.

Besides accelerating DTW by using a warping constraint, there exists lower bounding. Lower bounding is a technique to calculate a lower bound of the distance between Q and C_i in order to prune off sequences. According to Rakthanmanon et al. [276], I use a sequence of lower bounds to accelerate a 1NN classification[8]. As first lower bound, I use a variant of LB_kim [277] described in [276], which is a $O(1)$ lower bound. The second lower bound is LB_keogh [278], which has complexity $O(N)$, but early abandoning enables it to stop the calculation once the distance is greater than the current best distance.

[8]I slightly adapt the source code from `http://www.cs.ucr.edu/~eamonn/UCRsuite.html`

Table 5.17: Comparison of execution time between different (dis)similarity measures on the entire UCR archive.

	SW-1DSAX5	**SW-1DSAXoptr**	**SW-1DSAX100**	**DTWopt**	**ED**
exec. time	81.87 s	252.38 s	624.37 s	1123.49 s	21.89 s

Table 5.18: Comparison of execution time between different (dis)similarity measures on databases from the UCR archive that have more than 300 training elements.

	SW-1DSAXopt	**DTW**	**ED**
exec. time	226.1 s	1040.7 s	21.1 s

Table 5.19: Comparison of DTW lower bounding in combination with our measure.

	SW-1DSAX5	**SW-1DSAXoptr**	**SW-1DSAX100**
class. error	0.295	0.291	0.293
exec. time	41,3 s	37.6 s	173,08 s
DTWopt	19/65/1	17/68/0	21/64/0
ED	41/44/0	36/49/0	41/44/0

I use the approach from Lemire to calculate the required envelopes for LB_keogh [279].

Moreover, the proposed reordering step is also used to enable abandoning as soon as possible. Therefore, I order the query sequence according to the values in a time series. Early abandoning enables stopping the calculation once the similarity value is less than the best so far value. Hence, it drastically reduces the execution time of a 1NN classification. I apply above approach together with SW-1DSAX as measure instead of DTW. LB_kim and LB_keogh are two lower bounds for DTW, which are exact. However, applying a different similarity function lead to different results. Therefore, I need to evaluate this factor.

I use a single core of the CPU-based server to execute the different versions. The results are shown in Table 5.19. Executing the entire UCR archive using DTWoptr with LB_kim and LB_keogh requires 3588,09 s. Hence, my method is significantly faster than DTW with lower bounding, but have a slightly worser classification error compared to ED in general. However, it is important for my method to be faster than DTW and more accurate than ED on a significant number of databases. This is true for 28 databases for SW-1DSAXoptr, 25 for SW-1DSax100, and 31 for SW-1DSax5. Using SW-1DSAX5, my method is $2579\times$ faster than DTW with lower bounding on these 31 databases with a maximum of 471 for the RefigerationDevices database. In the case of SW-1DSAX100, the total speedup on the 28 databases is 3143 with a maximum of 766 for Phoneme. This is very interesting because Phoneme is normally a very huge database, but only a subset is part of the UCR Archive.

5.7 Related Work

Design patterns are a well-known concept in software engineering and help to solve reoccurring problems [280]. A pattern is described in such a way that it is independent of a certain context. It provides a template which shows how to solve the accompanying problem. Thus, patterns represent best practices. However, the introduction of design patterns in the field of FPGAs is low. De Hon et al. [262] provide a classification of design patterns for reconfigurable computing. They group patterns amongst others in patterns for area-time trade-offs, for extracting parallelism, for common case optimization such as memoization, for using hardware efficiently, and for number representations. To reduce the design effort, parallel patterns implemented in a functional language, such as Scala, can be exploited to generate configurable hardware in an automatic way [281]. A systolic array is an often used pattern to exploit parallelism on anti diagonals, called wavefront parallelization [260]. The aforementioned approaches do not consider the high potential of approximation.

Gribbon et al. [246] discuss which design patterns for reconfigurable computing can be used for image processing FPGA designs and group them in patterns for timing, bandwidth, and resources. Their investigation includes approximation patterns such as `downsampling` or `functional approximation`. GRATER [12] is an automated design workflow for FPGA accelerators that tunes the precision of data types within an OpenCL kernel. Additionally, there exist approaches for certain domains like artificial neural networks [132], which use memory access skipping, precision scaling, and approximate arithmetic blocks. SAGE [11] is an AC self-tuning approach dedicated to GPUs, and uses the following AC patterns: skipping atomic instructions, data packing, and skipping computations of inactive threads. In contrast, my ACP patterns have their focus on higher abstraction layers to introduce AC, while focusing on 2D DP algorithms.

Smith Waterman Algorithm. As the Smith Waterman (SW) algorithm is the most accurate method for comparing biological sequences, there exists a huge research effort to accelerate the underlying computation. A recent survey considers SW algorithm implementations on different hardware architectures dedicated to protein sequences [282]. On the CPU level, the implementations of Wozniak [283], Rognes and Seeberg [266] and Farrar [250] are state-of-the-art and use intra-parallelism. To make things clear for the reader, intra-parallelism use the available parallelism within a sequence comparison, while inter-parallelism performs several comparisons concurrently. Exploiting AVX2 and inter-parallelism, Daily [284] achieves 291.5 giga cell updates per second (GCUPS) on two Intel Xeon E5-2670 2.3 GHz processors. CUDASSW++ 3.0 [285] is the de-factor standard tool for GPUs but uses the inter parallelism approach on the CPU and GPU. CUDASW++ 3.0 reached 119 GCUPS on a quad-core Intel i7 2700k 3.5Ghz processor and an NVIDIA GeForce GTX 680 for the Swiss-Prot database.

The main focus towards FPGA approaches lies on DNA sequences due to its simplicity. These approaches are based on systolic arrays [286] and reach up to 98 GCUPS for synthetic data. Similar to my ACP-based approach, Rucci et al. [287] exploit inter parallelism and get 58.5 GCUPS using the FPGA mode and the NR database. By combining two Intel Xeon E5-2620 v2 2.0Ghz and two Intel Xeon Phi 7110P, Lan et al. [288] achieve s 220 GCUPs for a merged database. The ACP-based design running on the Convey HC-1, which is a relatively old technology compared to the mentioned ones, leads to 157 GCUPs for the adapted uniprot database. My design exploits inter- and intra-parallelism in combination with approximation techniques, which is not used by the other ones.

Stereo Vision. The related work regarding local approaches is huge, therefore I refer to a current survey [255]. The execution time of FPGA-based local approaches only depends on the number of pixels, in case the resources are sufficient for the current window size. My ACP patterns transfer a SGM algorithm in such a way that the execution time also only depends on the number of pixels, hence they lead to a streaming architecture. Regarding SGM approaches, learning a matching cost function using deep neural networks based on ground truth data improves the accuracy of SGM, but is very computationally intensive and requires high-performance hardware [289]. Other SGM approaches aim at minimizing the memory consumption using tiling, architectural optimizations, for instance by storing only a certain part of the intermediate data [290], or reducing the considered paths [291].

A recent GPU implementation on a Tigra X1 [291] achieves 42 fps considering 128 disparities on VGA images (640$\times$480) and has an accuracy of 93.36% on the entire KITTI benchmark [253]. To support a large disparity range, Gehrig et al. [292] adapt the disparity step size to a maximum of 4 for large disparities. It achieves a performance of 22 fps considering up to 256 disparities for images of size 1024$\times$512 and has an accuracy of 93% on KITTI. Hoffmann et al. [293] propose a scalable SGM design for low-end as well as high-end FPGAs achieving an accuracy of 91.6% on Middlebury and providing 33 fps on VGA images on the Zed board.

A solution on a high-end FPGA achieves 42 fps for 128 disparity levels on images of size 1600$\times$1200 achieving an accuracy of 94,5% on Middlebury [294]. Recently, a tree-based global algorithm was proposed on a costly FPGA and achieves 30 fps for 60 disparities on images of size (1920$\times$1680) and has an accuracy of 93% on Middlebury [295]. I addressed the point which techniques overcome the issues regarding memory and resource consumption for a SGM approach with a tolerable impact on the accuracy. Therefore, I systematically apply the novel and general ACP patterns, which can also be applied on a high-level programming language like OpenCL. This results in efficient ACP-based FPGA designs, which are streaming architectures and achieve high frame rates with useful accuracy. In contrast, other approaches rely on a hand-optimized low-level implementation.

Time series analysis. There exist few works that exploit heterogeneous architectures for time series databases. Some approaches port DP algorithms like SW and DTW to special hardware using systolic arrays [296, 297]. Moreover, different implementations for DTW exist on GPUs [298, 299] and many-cores [300]. A comparison between GPUs and FPGAs is done in [301], where FPGA versions of DTW outperforms GPU and sequential CPU. All special hardware approaches do not consider the aspect of approximation to improve the performance, which I do using the ACP patterns.

5.8 Summary

Dynamic programming-based algorithms play an important role in different domains such as computational biology, image processing, or time series analysis. Such algorithms often have a high computational complexity and require a large memory space. However, the application fields of these algorithms make it necessary to process huge amount of data or to perform the task very fast. While reconfigurable architectures can provide a sufficient hardware platform to achieve these requirements, porting the algorithms into an FPGA poses a high design effort.

Approximation techniques reduce this design effort and often enable satisfying the requirements as shown for different applications. Therefore, I have identified approximate computing design patterns, which help a hardware designer to port 2D DP algorithms into an FPGA efficiently. These patterns increase the exploitable parallelism, reduce the required memory space and bandwidth, and decrease the required hardware resources. As this chapter showed, these patterns are not only usable for a low-level implementation, they can also be used for higher hardware programming languages. Moreover, these patterns are applicable to other hardware architectures.

The evaluation in this chapter gives valuable feedback for hardware designers to see on which kind of problems it is beneficial to use the ACPs. However, the influence regarding the QoR strongly depends on the algorithm, hence an evaluation has to be performed to get the resulted impact. Such an evaluation can be performed according to a CPU implementation which significantly reduces the effort. Moreover, the hardware designer gets an early feedback if certain ACPs can be applied.

An implementation based on the ACPs poses an accuracy-aware method on the architecture layer and thus can also be used by the configuration layer. Furthermore, this implementation can also provide accuracy-aware knobs that can be tuned by the configuration layer. For instance, the quantization factor and the warping window pose such a knob for the ACP-based measure.

CHAPTER
SIX

INNOVATIVE SOFTWARE APPROACHES FOR ACCURACY-AWARE COMPUTING

Accuracy-aware computing methods on the software layer have high potential to improve different design values. Therefore, I introduce innovative and general accuracy-aware software methods in this chapter.

The novel concept of applying *contract-based algorithms* in the domain of accuracy-aware computing is proposed. Furthermore, I address the challenge how to control the budget consumption of these algorithms and to realize contract-based tasks in an efficient way. This includes the capability to adapt the execution of a contract-based task per input to reach a desired quality of result (QoR).

Moreover, I introduce *performance profiles* for accuracy-aware computing. Performance profiles act as a behavior description for contract-based tasks and present a relation between budget consumption and QoR. This description also leads to a general way how to describe the behavior of other accuracy-aware methods that can be located in any system layer. Moreover, exploiting this description leads to considering multiple objectives and different hardware parameters concurrently. Besides contract-based tasks and performance profiles, I introduce a novel approach for *fuzzy memoization*.

6.1 Introduction

Applying accuracy-aware methods on the software layer poses huge opportunity to increase the efficiency of the execution for an application regarding different design values[1]. Using approximation on a software level is not a new approach. The main difference of accuracy-aware methods compared to already existing approximation methods

[1]Cf. Prof. Anand Raghunatan's talk at the center of computational brain research (CCBR) workshop at the Indian institute of technology madras 2017 `https://ccbr.iitmadras.in/workshops-previous`.

is that allowing a certain degree of approximation is a further desired design parameter and not a compelling necessity to solve certain problems such as NP-hard ones.

There exist different accuracy-aware methods on the software layer [302]. Domain-specific approximations [131] and function-specific approximate implementations [17] have the drawback that they are not applicable on a broad spectrum. Furthermore, for a fine-grained adaptation of the approximation degree, several approximate versions have to be available.

The same holds for neural network-based approximate computing (AC) methods [15, 182], however, they present an automatic way to create an approximate version for a certain code snippet. While the resulting neural network could be inferred using software, a performance gain is only achieved by exploiting a neural processing unit. The above method relies on the aspect that the number of inputs and outputs is fixed for a function and thus different input sizes require a different neural network.

More general software-only AC methods are loop perforation [13] and loop tiling (approximate loop memoization) [14]. Note that I use loop tiling in this section to indicate the method of approximate loop memoization from Samadi et al. Researchers showed that loop perforation performs worse in general compared to a neural network-based method exploiting a neural processing unit [303]. Furthermore, loop perforation is also not generally useful as backed up by the findings in this chapter (cf. Section 6.5.2). The applicability of loop tiling strongly depends on the algorithm and works well for image processing. Moreover, the performance gain by increasing the perforation rate or tiling size is often not clear and therefore requires two models that describe the behavior of the performance and the quality for different settings in order to control the approximation degree.

Therefore, this chapter proposes novel accuracy-aware software methods that enable a general and fine-grained adaptation of the approximation degree. This includes a novel way of realizing *fuzzy* or *approximate memoization*. Instead of executing a function, fuzzy memoization looks in a table whether a result is stored that is based on calling the function with a similar input to the current one. This avoids the actual execution of the function. The proposed novel memoization method is based on locality-sensitive hashing (LSH). LSH creates an input-aware address to the memoization table and thus increases the hit rate in the table, while providing useful quality of results (QoRs). The only restriction of this method is that the function has to be *referential transparency*. This means that given the same inputs the function returns the same results, hence is deterministic. Such functions are also called pure functions. However, the overhead of memoization methods is too high for typical approximation-tolerant applications and therefore, they do not present the needed solution. Therefore, I propose a general and innovative solution to realize an accuracy-aware software method that exploits the concept of *contract algorithms* [140]. These algorithms improve the QoR with increasing budget consumption and hence correlate the budget with the QoR.

This fundamentally differs from existing AC methods that provide knobs for the approximation degree and cannot inherently restrict one objective dimension, while maximizing the other one. Furthermore, I introduce an input-aware approach that controls the approximation degree for each input individually. This comes with almost no overhead compared to the original function. Moreover, my innovative approach does not require to differ between exact and approximate implementations of a function, since providing a sufficient budget to the contract-based task leads to the exact result. I also introduce an ordering scheme for processing input data items that improves the performance of contract-based tasks.

The behavior of contract-based tasks is described with *performance profiles*. A performance profile correlates the required budget of a contract-based task to the achieved QoR according to a Pareto-optimal front. The proposed approach can use the execution time, the consumed energy, or a combination of both as budget. The combined budget leads to considering multiple objectives concurrently. The energy delay product metric [304] makes it possible to combine the execution time and the energy consumption for the budget metric. Performance profiles can be used to describe other accuracy-aware methods on different layers and allows the approach to take non-AC hardware parameters (conventional methods) such as the number of threads or dynamic voltage and frequency scaling into account. Therefore, I introduce the novel concept of exploiting performance profiles as an intermediate layer between accuracy-aware methods and approaches that control these methods and other hardware parameters. The usage of

To sum up, this chapter presents the following novel contributions.

- Introducing the concept of contract-based tasks as general and fine-grained accuracy-aware software method.
- Exploiting performance profiles as intermediate layer between accuracy-aware methods and accuracy-aware control approaches. This represents a transparent view on accuracy-aware methods.
- Considering multiple objectives by using an energy delay product metric for performance profiles.
- Input-aware adaptation of contract-based tasks according to a desired QoR.
- Improving the performance of contract-based tasks by adapting the sampling ordering of the input data.
- Introduction of LSH-based memoization in combination with the demonstration that software memoization has a non-tolerable overhead for typical AC benchmarks.

6.2 Fundamentals

Since I consider two different approaches to realize novel ways for introducing AC on the software level, I give a brief description about the fundamentals regarding theses two approaches. Firstly, I describe the concept of memoization and discuss about hash functions, which are required for my fuzzy memoization approach. Secondly, I introduce the concept and background about *anytime algorithms*.

6.2.1 Memoization

Memoization is an approach for program optimization and trades memory space for performance. Note that the term optimization is rarely used to indicate a truly optimal program, since the required effort is not reasonable. The idea behind memoization is to cache results of software functions in order to reuse this result when the function is called with the same parameters again. Hence, the computational effort for performing the function is avoided.

The results are stored into an associative array that can be a list, a tree structure, or more often a hash table. On average, the computational complexity of a lookup or an element insertion into a hash table is $O(1)$ and therefore it is often the favorable choice. Theoretically, the binary representation of the input data, which are scalars, vectors, or matrices, can be used as index to the hash table. This results in a big hash value and thus applying a hash function is useful.

As the range of the hash value is limited, we have to deal with collisions. Two well-known approaches to address collisions are *separate chaining* and *open addressing*. Separate chaining only stores an address to an association list that stores the elements in the hash table. Open addressing uses empty slots in the table to store new values.

The required probing of slots is determined by a probing sequence such as linear or quadratic probing. Even then, it is possible that the memoization table has no free slot left. Hence, elements have to be replaced or the table is increased.

In contrast to memoization, lookup table-based approaches completely cover a certain domain for the function. The number of lookup table entries depends on the discretization of the required range of the domain. Such lookup table approaches replace the actual function, for instance, a sine function. The input value acts as index to the table and often this approach is only used when the dimension of the input is low. Otherwise, the required space is often too large to store efficiently. Look-up tables are generated during the implementation or compile time, whereas memoization tables are generated during initialization or runtime.

6.2.2 Hash functions

The main task of a hash function is to map a large value range to limited amount of so-called hash values. There exists a wide variety of hash functions that map an input $x \in \mathcal{I}$ to a hash value, where $\mathcal{I} = 0, \cdots, N-1$ is a set of 1D elements. Simple hash functions are $h(x) := x \mod n$ or $h(x) := x \cdot n/N$, where 0 to $n-1$ is the range of the hash value. Ideally, n is a prime number. The efficiency of these simple functions depends on the distribution of the input data. Moreover, certain hash function can deal with variable length of inputs. According to [305], a good hash function has the following properties. Good hash functions

- evenly distribute the input data (collision probability is independent from the input).
- spread the data (similar inputs get different hash values).
- are correct (work for all possible inputs).
- have a clearly limited range of the hash value.
- have sufficient performance.

Especially, the last point is important in my case, since the hash function is often the most time-consuming part of the memoization approach. There exist different classes of hash functions. Universal hash functions are randomized algorithms and allow us to specify the probability of a collision. Checksum algorithms like CRC32 can be exploited to perform hashing. Moreover, multiplicative hashing is an efficient and simple method, where Murmur3 and xxHash are representatives of this class. Furthermore, cryptographic hash functions play an important role in different applications such as password verification or checking the authenticity of files or messages. Additional properties of cryptographic hash functions like SHA are deterministic behavior and the non-existence of an inverse function. The latter impedes finding the message that has generated a given hash value. Small changes in the message should also cause a huge difference in the hash value. However, these properties are purchased by high computational effort compared to other hash functions and therefore not useful for memoization.

In contrast to other hash functions, locality-sensitive hashing (LSH) maximizes the probability of a collision for similar inputs. Such methods are based on certain distance metrics that specify the similarity of inputs. A main field of these hash functions is finding the nearest neighbor in a database for a given query. A formal definition of LSH is proposed by Indyk et al. [306]. Being D a distance function in space X, then a metric space $\mathcal{M}(X, D)$ is given. An LSH family $\mathcal{H}$ is defined as $\mathcal{H} = \{h : \mathcal{M} \to B\}$, where $b \in B$ is a bucket. A family is called (r_1, r_2, p_1, p_2)-sensitive if the following conditions are satisfied for any two points $\vec{x}, \vec{y} \in \mathcal{M}$:

- if $D(\vec{x}, \vec{y}) \leq r_1$, then $P_{\mathcal{H}}[h(\vec{x}) = h(\vec{y})] \geq p1$

- if $D(\vec{x},\vec{y}) > r_2$, then $P_{\mathcal{H}}[h(\vec{x}) = h(\vec{y})] \leq p2$,

where p_i is a probability, r_i a distance radius, and $r_1 < r_2$. A family $\mathcal{H}$ is useful if and only if $p_1 > p_2$. A hash function h is chosen randomly from $\mathcal{H}$.

It is possible to generate another LSH family $\mathcal{G} = \{g_i\}, 0 \leq i \leq L$ by using an AND- or OR-construction of several $h_j \in \mathcal{H}, 0 \leq j \leq k$. This approach, we call it *multi-tabling*, reduces the probability to create *bad* hash values, i.e., a hash value that is equal to a value generated by an input that is not similar to the current one. To create a hash function $g_i \in \mathcal{G}$, we choose k random functions from $\mathcal{H}$ and combine them. MinHash and SimHash [307] are noticeable approaches to generate LSH families. An extension of SimHash is based on a p-stable distribution and called E^{2}LSH [308]. For E^{2}LSH,

$$\mathcal{H} = \left\{ h_{\vec{a},b} = \left\lfloor \frac{\vec{a}\vec{x} + b}{w} \right\rfloor \right\},$$

where w is a quantization factor. A hash function $h \in \mathcal{H}$ is parametrized with a random projection vector $\vec{a}$ chosen from a normal distribution and a scalar b, which is uniformly taken from the interval $[0, w]$.

Since $g_i \in \mathcal{G}$ represents a vector with integers, we are not able to use this as an index for a bucket in the hash table H. Hence, another hash function

$$h_1 := \mathcal{Z}^k \rightarrow \{0, \ldots, \mathsf{H}_{\mathsf{size}}\}$$

is used and returns an index to the table H. For instance,

$$h_1(g_i(\vec{x})) = h_1(h_{a_1,b_1}(\vec{x}), \ldots, h_{a_k,b_k}(\vec{x})) = \left(\left(\sum_{i=1}^{k} r_i^{'} h_{a_i,b_i} \right) \bmod \mathsf{prime} \right) \bmod \mathsf{H}_{\mathsf{size}},$$

where $r_i^{'}$ is a random integer and modprime is a prime number. However, we can also use other hash functions such as xxHash.

To further improve the probability to find the *best* table entry, i.e., most similar input in the hash table, we can use *multi-probing*, which is a probing approach [308]. *Multi-probe* LSH creates so-called perturbation vectors $\vec{\delta_h}$, which represent buckets that most likely include the searched entry. These vectors are ordered according to the probability that their addressed bucket includes the wanted entry. A preprocessing step can calculate these vectors based on expectation values. Details can be found in the E^{2}LSH user manual [308].

6.2.3 Anytime algorithms

Anytime algorithms pose a special type of algorithms and normally improve the quality of their results over their execution time. Compared to other algorithms, which return their results after they finished the computation, anytime algorithms can return results in between any time. Hence, such algorithms easily allows us to trade off quality for performance. Early works consider such algorithms in the domain of real-time systems [309, 310, 311], especially for artificial intelligence, time-dependent planning, decision making. Anytime algorithms can be grouped into *contract algorithms* and *interruptible algorithms*. *Contract algorithms* require to specify a budget, for instance, execution time beforehand. Then, the algorithm returns the best QoR that can be returned using the given budget. In contrast, *interruptible algorithms* are interrupted by an external function that can immediately get the current result. Contract algorithms are often easier to implement and maintain [312]. Researchers have shown that any *interruptible algorithm* can be realized as *contract algorithm* [313, 312].

Performance profiles describe the behavior of anytime algorithms regarding the QoR and budget trade-off [314]. Simulation-based approaches or an analytical strategy lead to the generation of performance profiles. Having performance profiles for different tasks that pose an application, it is possible to combine these performance profiles into a performance profile for the application. This procedure is called the *compilation* of anytime algorithms. However, the existing works mainly include theoretical considerations and make assumptions that are not true for approximation-tolerant applications [315, 316]. For instance, for *probabilistic performance profiles* that occur for these applications, only a linear relationship is possible for a *compilation* of anytime algorithms. Such a relationship is often not given. A probabilistic performance profile takes into account that a certain budget leads to different QoRs for various inputs. To apply anytime algorithms in approximation-tolerant applications, it is important to deal with many low-level inputs.

Exactly, due to this probabilistic behavior, monitoring is a useful approach to estimate the current QoR of an anytime algorithm [317, 318, 319]. The overhead of monitoring negatively impacts the QoR improvement and thus evaluating when to monitor can significantly reduce this overhead [320]. However, compared to planning algorithms which have a utility function, finding good monitoring functions poses a tough challenge for AC.

Furthermore, there exists an approach that determines a scheduling based on anytime and non-anytime alternatives in order to determine a complex solution plan for a given task in the domain of real-time AI problems [321]. In contrast, this chapter provides an approach to exploit anytime algorithms to realize AC-based applications and additionally it takes care that the introduced overhead is as low as possible. Furthermore, it is a general method to combine several AC approaches. There exist basic works how to automate steps for creating anytime-based applications [322].

6.3 Local-sensitive Hashing-based Memoization

In this section, I introduce a novel software concept for fuzzy memoization. Fuzzy memoization extends the memoization approach in a way that not only exact matches lead to a hit in the memoization table. Instead, having a result in the table produced by a similar input, the result from the table is used.

Therefore, such an approach wants to improve the hit rate and hence it is more often avoided to calculate the actual function. First of all, I introduce a theoretical relation that shows, when it is likely that fuzzy memoization is beneficial for an application in terms of performance. Then, I present the design of a generic library that can be used for different applications to introduce fuzzy memoization.

6.3.1 Theoretical Considerations

In the context of databases, it is important that LSH-based approaches find the correct nearest neighbor with a high certainty. However, in the field of approximate or fuzzy memoization, we require a sufficient result as soon as possible. Thus, if the lookup consumes too much time, it is wise to perform the actual function instead.

How much time we are allowed to spend for the lookup depends on several factors and therefore I present a relation for that. This relation corresponds to a standard cache miss model. The average time T_{avg} for a function call using memoization is

$$T_{\text{avg}} = p_{\text{hit}} T_{\text{hit}} + p_{\text{miss}} (T_{\text{miss}} + T_{\text{fcall}}),$$

where p_{hit} is the probability of a hit in the memoization table and $p_{\text{miss}} = 1 - p_{\text{hit}}$ the probability of a miss. T_{hit} represents the consumed time for getting a result from the memoization table including the time to find the right bucket using an LSH approach. T_{miss} is the time until the search in the memoization table is aborted which also depends on *multi-tabling* and *multi-probing*. The time for executing the function is T_{fcall}. If we relatively specify T_{hit} and T_{miss} to T_{fcall}

$$\begin{aligned} T_{\text{hit}} &= h_{\text{rel}} \cdot T_{\text{fcall}} \\ T_{\text{miss}} &= m_{\text{rel}} \cdot T_{\text{fcall}} \end{aligned} \quad ,$$

we get the following relation

$$T_{\text{avg}} = T_{\text{fcall}} (p_{\text{hit}} (h_{\text{rel}} - m_{\text{rel}} - 1) + m_{\text{rel}} + 1),$$

where h_{rel} and m_{rel} are the relative factors for a hit and miss, respectively. Setting the memoization time factor as

$$f_{\text{mem}} = (p_{\text{hit}} (h_{\text{rel}} - m_{\text{rel}} - 1) + m_{\text{rel}} + 1), \tag{6.1}$$

the average time $T_{\text{avg}} = f_{\text{mem}} T_{\text{fcall}}$. We can clearly see that when $f_{\text{mem}} < 1$, we get a performance gain using memoization. However, this relation only considers the performance aspect of approximate memoization.

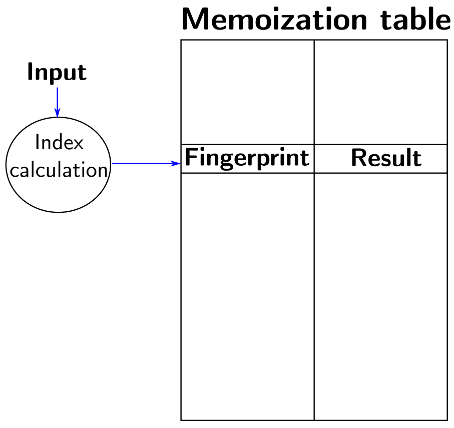


Figure 6.1: Structure of the memoization table.

According to the Equation (6.1), we can estimate the potential performance gain of adaptations regarding the approximate memoization approach. Such an adaption is useful if $f_{\text{mem}_{\text{old}}} > f_{\text{mem}_{\text{new}}}$. For instance, applying *multi-probing* may increase the hit probability $p_{\text{hit}_{\text{new}}} = p_{\text{hit}_{\text{old}}} + \Delta_{\text{hit}}$, but it requires more time in case of a hit and a miss for a lookup and thus $h_{\text{rel}_{\text{new}}} = h_{\text{rel}_{\text{old}}} + \Delta_h$ and $m_{\text{rel}_{\text{new}}} = m_{\text{rel}_{\text{old}}} + \Delta_m$. Such an adaption is useful when the following inequality is true

$$p_{\text{hit}_{\text{old}}}(\Delta_m - \Delta_h) > \Delta_{\text{hit}}(h_{\text{rel}_{\text{old}}} + \Delta_h - m_{\text{rel}_{\text{old}}} - \Delta_m - 1).$$

General assumptions about the accuracy influence of approximate memoization are not possible and so the influence has to be evaluated for a certain application. But the used method described in more detail in the next section allows a programmer to set parameters for adapting the accuracy, but likely reduce p_{hit}.

6.3.2 Design of a Software Library for Approximate Memoization

In order to evaluate LSH-based approximate memoization for different applications, a general library is designed. The library deals with different data types and input sizes by using a template-based approach. The library can be used to create the content of the memoization table in a pre-processing step or during the execution of an application.

Besides LSH, the library supports exact memoization and the quantization-based approach from Brandalero et al. [323]. Moreover, the library provides the usage of different hash functions such as SHA256, xxHash, and Murmur3, which are used to perform the actual bucket hashing. Bucket hashing is the required step to determine an index to the memoization table. Furthermore, fingerprints are used to create a label for the input that creates the entry. This is important to deal with unwanted hash collisions. Figure 6.1 shows the structure of the memoization table.

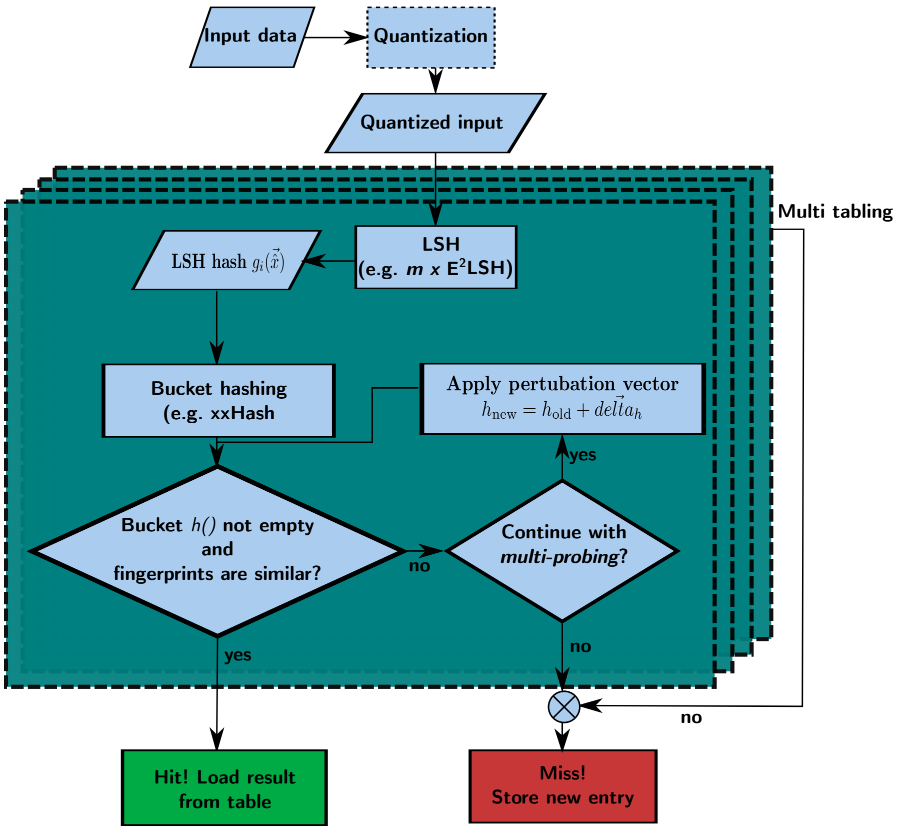

Input data
Quantization
Quantized input
Multi tabling
LSH hash $g_i(\vec{x})$
LSH (e.g. *m x* E²LSH)
Bucket hashing (e.g. xxHash
Apply pertubation vector $h_{\text{new}} = h_{\text{old}} + \vec{delta}_h$
yes
Bucket *h()* not empty and fingerprints are similar?
no
Continue with *multi-probing*?
yes
no
no
Hit! Load result from table
Miss! Store new entry

Figure 6.2: Flowchart for checking bucket entries in the memoization table.

The library has three main interfaces: creating a memoization table, checking if a similar result exists in table, and inserting new elements in table. Additionally, the library can provide useful information to a programmer to improve the parameters of the memoization table using the interface. The flowchart in Figure 6.2 shows the steps that are performed for finding similar results in the table. The input data is considered as an n-dimensional vector and therefore all parameters of a function are included in this vector. An optional step quantizes the elements of this vector. The next optional step performs one or more LSH computations (*multi-tabling*).

Each LSH computation results in an m-dimensional LSH hash $g_i(\vec{x})$, for instance, determined by m E^{2}LSH. Afterwards, for each g_i, a hash function such as $h_1()$ or xxHash is applied. This function leads to the index $h()$ of a bucket in a certain table.

The library checks (`check_bucket`) if the bucket is filled and the fingerprint of the current input is similar to the one that is in the bucket. When this check is successful, we have a hit in the memoization table and can use the stored result as result for the current function call. For the fingerprint, the upper part of the bucket hash is used. This allows it to deal with unwanted bucket collisions. A fingerprint is considered as similar if the hamming distance between the fingerprints is less than a certain threshold.

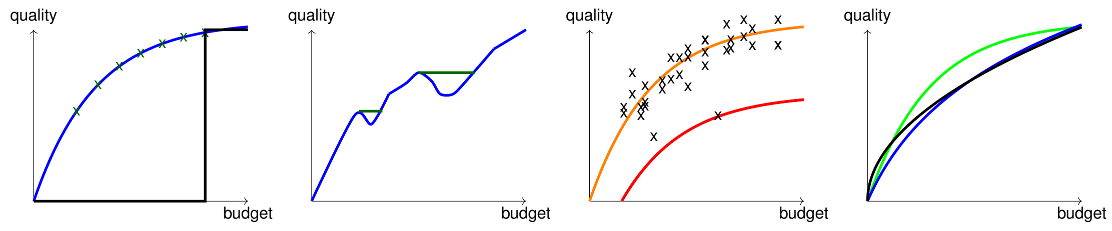
If the check is not successful, *multi-probing* can occur. *Multi-probing* is applied, when it is specified by a programmer and the final search depth is not reached. In this case, the current bucket hash is adapted with a perturbation vector $\vec{\delta}_h$ and `check_bucket` is performed again. If the *multi-probing* is stopped and all other tables also returns a miss, we have not been able to find a useful result in the table. Hence, the original function has to perform the calculation in order to get the wanted result. According to a policy, the library can store the new result together with the fingerprint in the memoization table. This can also requiring replace a result in a table.

6.4 Anytime Algorithms for Approximate Computing

This chapter does not only introduce contract algorithms as a general concept for realizing AC on the software level, it also claims that the usage of performance profiles (PPs) is a way to abstract different AC methods that target various levels. Such an abstract view allows us to exploit PPs as an intermediate representation in the domain of AC (more details in Section 6.4.3). Thus, we can separate the tasks of realizing approximation methods and of finding ways for controlling these methods (see Part III). PPs are an essential part of contract algorithms. As contract algorithms improve their QoR, while consuming more budget, a PP represents their performance. Hence, PPs make the comparison of different anytime realizations for a certain task possible. Furthermore, contract-based functions make it possible to improve a QoR if required by exploiting a concept similar to application checkpointing.

I represent the budget as the allowed execution time of the algorithm, for instance, a firm real-time constraint or the allowed energy consumption. The latter is a novel approach compared to the original invention of anytime algorithms. Later, I also describe how it is possible to consider multiple objectives. The quality of a PP is represented in an absolute metric like Signal-to-Noise-Ratio (SNR) or using a relative metric such as relative error (RE) or mean absolute percentage error (MAPE). The quality function is given by a domain expert in case the result of the contract algorithm is the output of an application. Internally, the PP only acts as information for controlling the internal QoRs of the contract algorithms (more details see Part III) and thus any quality metric does the job. Furthermore, I also introduce a multi-objective consideration of the budget by relying on a so-called energy delay metric (see Section 6.4.1).

A sketchy illustration of the comparison between a PP of a contract algorithm (blue line) and a traditional algorithm (black line) is shown in Figure 6.3a. In this case the PP of traditional algorithms is a step function (the maximum QoR is reached after a certain time, black line), contract algorithms can return results in between with different QoRs (blue line). However, using contract algorithms for proper tasks, discrete points (green crosses) are often considered only, see Figure 6.3a.



(a) Comparison. (b) Non-monotonic PP. (c) QoR variance. (d) Different PPs.

Figure 6.3: As an illustration, Figure 6.3a compares an illustrative performance profile (PP) (blue) to a conventional algorithm (black). A PP can be a non-monotonically increasing function for contract algorithms performing real computations, see Figure 6.3b. An issue with PPs is the variance of the QoR for different inputs (see Figure 6.3c). For a certain task, there exist different realizations of anytime algorithms having different PPs (see Figure 6.3d).

Internally, a PP can be represented as a closed and analytical formula or as a table of discrete points. I use the latter. Furthermore, the assumption that the PP is always a monotonically increasing function is not true. There exist cases, where spending more budget leads to poorer outcomes (see Figure 6.3b). Therefore, it is useful to store the best so far result (dark green line). However, identifying these results is not easy and thus this thesis provides solutions to it in the remainder (see Section 6.4.1 and part III).

Besides the variance of QoRs that occurs due to an indeterministic system behavior[2], the QoR depends on the current input data for a fixed budget. Therefore, I present two input-aware methods that deal with the QoR variance: monitoring and budget estimation (see Section 6.4.2). As a PP is required for controlling aspects, the PP for a contract algorithm with input-dependent QoR behavior can be modeled in different ways. One way is just to use the average QoR for a certain budget, see orange curve in Figure 6.3c. Another one is to represent the PP as the lower bound of the various QoRs (red curve). Furthermore, we can use the expectation value or the probability of getting a certain expectation value for varying budgets to generate a PP. The approach presented in this chapter makes no assumptions about the used category of a PP.

A PP depends on the distinct design and implementation of the contract algorithm. PPs are created by a simulation-based approach, where representative data is used to create points in the QoR-budget space. It also depends on the order in which the input data is processed. A different order can lead to a completely different PP as shown in Figure 6.3d. The green curve provides the best QoR in the mid-range of the consumed budget, whereas the black one offers a slightly better QoR improvement at the beginning. My proposed solution is task-specific ordering of data items of the input and output data

[2]External and internal circumstances in a computer system can lead to an influence on the contract algorithm and thus result in a different QoR even for the same input.

for contract-based tasks in order to extract the best PP for a given contract algorithm.

Finally, the usage of PPs enables taking non-AC hardware parameters into account such as using dynamic voltage and frequency scaling or exploiting parallelization. Making use of the so-called *merger* concept for anytime algorithms, a PP is generated for getting information in order to control the execution, see Section 6.4.1.

6.4.1 Design, Implementation, and Controlling of Contract-based Approximate Computing Tasks

In general, each task or function that enables us to estimate the final result based on the current knowledge within the function is suitable for an anytime-based design. Algorithms that already calculate a sequence of intermediate results make it possible to return these intermediate results, which pose an approximation of the result, before the completion of the algorithm [313]. For instance, search functions like RTA*, randomized algorithms, or iterative methods such as Newton's method or the conjugate gradient method. Such methods change the QoR over time, hence a contract-based implementation is straightforward but potentially requires to suppress newer results with lower QoR.

A naïve way to realize an anytime algorithm is to start the same function with different approximate parameter settings sequentially as suggested by San Miguel et al. [144]. This introduces essential overhead during runtime. Furthermore, the authors solely execute a single anytime algorithm on a single core and hence waste resources. Sampling of the input and output data is a further approach to realize an anytime algorithm.

Besides the actual computation within a contract-based function, an initial step and a post-processing step can improve the QoR behavior. To enable fast initial results, some tasks allow us to use the input data as a first useful approximation. For instance, an image convolution using a Gaussian filter can use the input image for such an initial approximation. In such cases, the result buffer is filled with the input data. Moreover, we can adapt the approximated output before returning the results to improve the QoR, which can be performed in a post-processing step[3]. This includes scaling and simple interpolation of the output results and causes negligible additional overhead.

Determining a good visiting order of samples. Sampling per element, as done by Miguel et al. [144], can violate data locality and thus I apply sampling at least on cache line granularity to fully exploit the potential of caches. The adaption of certain fixed sampling approaches, for instance, exploiting linear-feedback shift register or N-dimensional bit-reverse, to a certain input data size can be cumbersome. Therefore, I present the novel concept of *iterative loop perforation* (see Algorithm 6.1). For iterative loop perforation, a single loop is manually transferred into a nested loop, where loop perforation is

[3]Which is part of the function `determineResult()` used in Algorithm 6.5.

exploited in the inner loop and the outer loop adapts the starting point of the inner loop. This leads to fast traversal of the entire input space but allows that probably all elements are considered depending on the provided budget.

Algorithm 6.1: Iterative loop perforation.

```
for i→ 0 to step-1, i++ do
    for j→ i to n-1, j+=step do
        do_work(j);
    end
end
```

In contrast to fixed sampling approaches (sequential, tree, or pseudo sampling presented in [144] or iterative loop perforation), I introduce task-specific ways to perform a sampling in order to get better PPs, i.e., a QoR improvement at the beginning of the execution. A wise visiting order of the input data enables fast improvement of the QoR. Algorithm 6.2 illustrates the usage of a static task-specific order that is determined for a certain contract-based task during design time using Algorithm 6.3. The result is an ordered set O, which specifies the visiting order of items in the input data. This set is delivered to the contract-based function.

As input, Algorithm 6.3 gets a Task T (contract-based function) and a set of representative data D. D includes different input items d_i that represent an input for the task T. Furthermore, d_i can be divided into N different sample items, where a certain item is denoted as d_{i_j}. Then, the algorithm determines the error caused by only executing items with index i_j and indexes in the already determined ordering set O in comparison to using the entire d_i. The error is calculated by error function E, which can be any error function, for instance, mean square error or the peak to noise ratio. The list O corresponds to the order in which the sample points shall be visited to get a good performance profile, i.e., a fast decreasing error at the beginning. Hence, the QoR enhancement of a contract-based function is improved.

If a suitable order is input-dependent, then it is beneficial to spend some time to calculate a dynamic task-specific order of input items. In this case, the first step of a contract-based function is to determine a wise visiting order for its current input data using the function *determineOrdering()*, see Line 9 in Algorithm 6.4. Afterwards, the calculation is performed according to this visiting order L_p and the function parameters P. The input of the function *determineOrdering()* is the current input data I of the function, the number N of sample points in I, a score function V and the information about different priority levels (number of priority levels NUM$_P$ and the interval size b of score values that fall into a certain priority level). The priority levels specify in which order the samples shall be processed. The algorithm puts sample indexes where their respective score value lies in a certain interval of size b to the same list. Each priority level p is represented by a list L_p. The function processes the input samples according to these priority lists starting with the list that contains the indexes with the highest priority. Such a strategy

Algorithm 6.2: Usage of task-specific static ordering.

```
// $I$ specifies the input data and $P$ the function parameters.
// Static ordering $O$ was determined during design time using Algorithm 6.3.
1  Function contractFunction($I$,$P$,$O$):
2      do_work;// according to $P$ and $O$
```

Algorithm 6.3: Determining a static task-specific ordering during design time.

```
   Input: Task $T$, number of input samples $N$, set of representative data $D$, error metric $E$
   Output: Ordering of input sampling $O$.
1  $O = \emptyset$;
2  for $i \to 0$ to $N-1$ do
3      for each input sample index $j$ in $(0, N-1)$ do
4          error = 0.0; error$_{\max} = \inf$;
5          best$_j$ = -1;
6          for each input data $d_i$ in $D$ do
7              exact = $T(i)$; // uses all items in $d_i$
8              approx = $T(i_j \cup O)$; // uses the $j$th and $i_k$th items in $d_i$, $k \in O$
9              error += $E$(exact, approx);
10         end
11         if error<error$_{\max}$ then
12             error$_{\max}$ = error; best$_j$ = $j$;
13         end
14     end
15     $O$.append(best$_j$);
16 end
```

Algorithm 6.4: Task-specific dynamic ordering.

```
   // Input:  Input data $I$ distributed into $N$ sets, score function $V$, number
      of priorities NUM$_P$, list size $b$
   // Output:  Ordering of the $N$ sets (NUM$_P$ lists $L_p$).
1  Function determineOrdering($I$,$V$,NUM$_P$,$b$):
2      Initialize NUM$_P$ lists $L_p$;
3      for each set $S$ in $d$ do
4          score = $V(S)$;
5          index = $\lfloor$score$/b\rfloor$;
6          $L_{\text{index}}$.append(Index(S));
7      end
   // $I$ specifies the input data and $P$ the function parameters.
8  Function contractFunction($I$,$P$):
       // $V$ is a score function, NUM$_P$ the number of priority levels, and $b$
          indicates the maximum elements per level.
9      $L_p$ = determineOrdering($I$,$V$,NUM$_P$,$b$);
10     do_work;// according to $P$ and $L_p$
```

Algorithm 6.5: Integration of checking the budget within a contract-based function.

```
// Check budget after each compute step, first option
Function function1($I$,$P$):
    ···
    step1;
    ···
    if budgetConsumed then
        return determineResult();
    end
    ···
    stepM;
    ···
    if budgetConsumed then
        return determineResult();
    end
    ···
    stepLast;
    return correctResult;
// Check budget after each iteration, second option
Function function2($I$,$P$):
    ···
    for i$\to$ 0 to N-1 do
        doIteration;
        if budgetConsumed then
            return determineResult();
        end
    end
    return correctResult;
```

is, for instance, beneficial for an image convolution using a Gaussian kernel, where the variance between image pixels in a certain area is used to define a score function as shown in the evaluation of this chapter.

Implementation aspects for getting the consumed part of the granted budget. Specifying a maximum number of allowed iterations or compute steps to a contract-based function can result in high variation regarding the execution time and energy consumption between different runs. Therefore, I use the maximum allowed budget as an input parameter to a contract-based function. The function has to frequently check whether the assigned budget is consumed (see Algorithm 6.5). If so, the task returns the current result.

I rely on lightweight methods to measure time or energy consumption in order to introduce minimal overhead. This is of particular interest in this case because the progress of the execution relates to the QoR. Thus, I exploit hardware counters.

The time stamp counter (TSC) provided by x86 architectures is used to get the consumed execution time [324, Volume 3, Chapter 17.17]. Note that I translate the allowed execution time into cycles and check in the contract-based function whether the number of allowed cycles is consumed. Modern Intel CPUs tick the TSC not by using the actual frequency, but the nominal frequency, and thus it does not indicate core ticks. Therefore, it matches wall clock time. The further issues[4] of TSC are acceptable for measuring the consumed budget. We can call an intrinsic function called `__rdtsc()` to get the content of the TSC. One of the test systems has a time resolution of 160 ns using TSC, while consuming 7.9 ns compared to 40 ns using a kernel function.

Intel's Running Average Power Limit (RAPL) interface enables getting the energy consumption [324, Volume 3, Chapter 14.9]. Since the Sandy Bridge architecture, Intel supports an interface to get information about the consumed energy of processors and memory. This can be controlled using model-specific registers (MSRs). I use the publicly available tool `likwid`[5] [325] to get the energy information using a C/C++ API.

Checkpointing for improving QoR using a contract-based function. Checkpointing is a well-known and important technique for providing fault tolerance in a computing system. The idea behind it is to store snapshots of an application or a task state and restart from that point in case of a failure. The same concept can also be used for contract-based functions, where the benefit is not providing fault tolerance but enabling a lightweight method to improve the QoR, if required. Therefore, the internal state is stored when the provided budget is consumed and before the result is returned. If now a controller or user decides that the resulting QoR is not sufficient, the contract-based function is continued from the stored checkpoint. It depends on the actual contract-based function, which internal buffers, control and data variables have to be stored inside a checkpoint in order to continue execution.

Having a 2D convolution implemented as contract-based function, we have to store the visiting order list (cf. Algorithm 6.4) and the current position in that list for a checkpoint. For a k-nearest neighbor algorithm, we also require the order list and additionally the so-far maximum similarity between the query and an entry in the database to corectly continue the execution of the contract algorithm.

Dealing with non-monotonically increasing performance profiles. An issue that arises with contract-based functions are non-monotonically increasing PPs. This means that higher QoRs can be reached by spending less budget and spending more budget will locally decrease the QoR, or in other words, the PP has local maxima (cf. Figure 6.3b). Depending on the actual algorithm, lightweight error checks pose a possible solution [202] to identify local maxima. Such checks allow us to get the current QoR of a

[4]`http://oliveryang.net/2015/09/pitfalls-of-TSC-usage/`, last visited on 01/24/2018

[5]I used the version 4.3.1 of `likwid`. Furthermore, I use direct access to the performance counter.

solution without executing the exact version. For instance, having nearly solved a system of linear equations, we can insert the solution vector into the system and the deviation between the left and right side is a useful indicator for the solution quality, also known as the residual. This check is much less computationally expensive than solving the system with an exact algorithm and comparing the exact solution vector with the one determined by an approximation. For searching an element in a database using a similarity metric, we can keep the best so far solution and thus the current solution is the one with the highest QoR so far.

In case the variability regarding the QoRs for various inputs is low[6], we can exploit the information from the generated PP to avoid spending too high budgets. This will result in reaching the local maxima of the QoR (this will be described in Part III). However, it is more likely that we have high variability for QoRs. Therefore, my solution is to gather runtime information of the input. This information is used to adapt the provided budget. More details are given in Section 6.4.2.

Taking hardware parameters into account for performance profiles. Current computer systems provide different technical opportunities that influence the execution of an application and can even be controlled by an application. An opportunity is to exploit heterogeneous hardware units of the system, as discussed in Chapter 5. Furthermore, modern CPUs provide the adaptability that the software or the processor itself can change the frequency called *dynamic frequency scaling* (DFS). Often DFS is combined with dynamic voltage scaling and then called *dynamic voltage and frequency scaling* (DVFS). The main motivation for integrating DVFS is that the dynamic power

$$P_{\mathsf{dyn}} \approx aCV^2 f,$$

where a is a switching factor and C the capacitance of the CPU, depends on the supply voltage V and the frequency f [32]. DVFS adapts V and f in order to get a reduction of the consumed energy for a certain task. Especially, this has low influence on the performance for non-CPU-bound tasks. The dynamic power influences the temperature of the CPU and thus also influences the static power consumption. Two commercial examples of DVFS are Intel's SpeedStep [326] and AMD's Pure Power [327]. Moreover, modern CPUs are often multi-core architectures and therefore an application can exploit several cores. This can be seen as a further *hardware parameter*.

In general, the above mentioned parameters can influence execution time or energy consumption of a task or function or even both. In the context of contract-based functions, this can influence the relation between QoR and consumed budget. For instance, increasing the frequency of a core or exploiting more cores can lead to reaching a certain QoR earlier in terms of execution time, but impacts the energy consumption.

[6]Providing the same budget for various inputs roughly leads to the same QoR.

In the remainder of this chapter, I show for DVFS and internal parallelization how to employ PPs to control a contract-based function that can use hardware parameters. To change the internal parallelization degree of a certain function, I use OpenMP. OpenMP allows a programmer to specify the number of running threads $\text{num}_{\text{threads}}$ using the function `omp_set_num_threads()` or an environment variable.

Similarly to specifying the number of threads within an application, a programmer can assign a desired frequency to a core of the CPU. However, modern CPUs, for instance, from Intel, do not guarantee that the user-specified frequency is used. The hardware can change the frequency of the core at any time regardless from the frequency assigned by the operating system or a programmer.

Instead of the default Linux driver `intel_pstate`[7], I use the driver `acpi-cpufreq` as CPU frequency driver. This driver enables using the so-called `Userspace` governor. This governor deactivates DVFS for the operating system and makes it possible to specify a frequency from the user-space. For that, the Linux kernel provides the library `libcpufreq`. The library offers an access to `acpi-cpufreq` driver. Using the library, we can get the current frequency of a core and can set the desired frequency $f_{\texttt{desired}}$ of a core within an accuracy-aware application.

Both hardware parameters, $\text{num}_{\text{threads}}$ and $f_{\texttt{desired}}$, individually influence execution time and energy consumption, and hence also lead to different PPs. Generating a PP for each value of a parameter, we get several PPs for a single task (similar to Figure 6.3d). Following the principle of a Pareto-optimal front, I combine all PPs of a task and a hardware parameter into a single PP by employing the concept of a *merger* [313]. This concept combines several PPs by finding the best PP for each budget, cf. Figure 6.5a. As a PP is stored in a discrete way, each element of a PP is equipped with the actual value of the hardware parameter that leads to this point. This is important in order to control the hardware parameter.

So far, I only considered a single hardware parameter. However, according to the approach we could create individual PPs for all value combinations of both hardware parameters. Assuming we have a CPU allowing 48 hardware threads and 10 DVFS steps, this requires us to evaluate and create 480 PPs. If this brute force or exhaustive method is too time consuming, a possible solution is to sample this space and to not test all parameter values. Another method is to only combine the parameter values that are part of a respective *merger*. As the time consumption for the exhaustive search is tolerable for the functions that I consider, finding a suitable method to reduce the effort is left for future work.

How to consider multiple objectives? Until now, I have looked at the budget as a single objective execution time or energy consumption and thus a PP shows the context in the style of a Pareto-optimal front for a bi-objective problem. Considering both at the

[7]This driver is default, since the Linux kernel version 3.9 for Intel Sandy Bridge and newer Intel CPUs.

same time leads to a third-order objective problem spanned by the conflicting objectives, accuracy, execution time, and energy consumption. Finding good configuration points in that space is a difficult task and would probably require a 3D PP in my case.

Solely optimizing for energy consumption is known as optimizing the power delay product (PDP) in computer architecture. The PDP represents the energy $E = P_{\text{avg}} t_{\text{exec}}$, where P_{avg} is the average power and t_{exec} the execution time. Exploiting the aforementioned hardware parameters allows us to influence P_{avg} and t_{exec} to reach a certain QoR. However, the PDP does not explicitly address the performance aspects and thus can consider a parameter setting as good which leads to low energy and low performance. To avoid such a problem, I use a concept similar to scalarization [328]. Scalarization transfers a multiple objective problem into a single objective one, where the optimal solutions correspond to the Pareto-optimal solutions of the multiple objective problem. More precisely, in my case, the energy consumption as optimization goal can be weighted with the execution time and thus the execution time is a parameter for the scalarization.

Therefore, I can reduce the design goals, energy consumption and execution time, to a single optimization goal. Such an approach is also known from computer architecture and low power design and summarized as the ED^n metric [304], where E is the energy and D the delay of a circuit. In the present case, $D = t_{\text{exec}}$. While for $n = 0$, the metric reduces to the PDP, $n = 1$ represents the energy delay product (EDP) [329] and $n = 2$ the energy delay squared product [330]. Generally speaking, any $\text{ED}^{\text{n}}\text{P}$ metric is an indicator for the energy usage per computation and therefore a lower value shows that the energy is more efficiently used [331]. The value n represents that it is tolerable to spend n percent more energy for one percent reduction in execution time. Hence, n is a parameter indicating how high the emphasis on the performance over the energy consumption is.

Since I am already able to get a measured value for the execution time and the energy consumption (see Section 6.4.1), I can easily combine them using any $\text{ED}^{\text{n}}\text{P}$ metric and generate a PP for it. This PP shows the QoR over the used $\text{ED}^{\text{n}}\text{P}$ metric. Moreover, this makes it possible to constrain the $\text{ED}^{\text{n}}\text{P}$ value, but also to constrain the energy consumption or execution time separately. This is achievable, since the values for the energy consumption and execution time are separately available and thus we can specify these metrics to use for the `budgetConsumed` condition in Algorithm 6.5. Therefore, besides an $\text{ED}^{\text{n}}\text{P}$ constraint, we can additionally use a constraint for the energy or the execution time.

6.4.2 Determining Required Budget and Monitoring the Progress

The QoR often has a high variance for a certain contract-based function and different inputs even if the budget is the same. One way to deal with such a variance is to use models that map the desired QoR to the required budget. I call this model the `budget` `model`. The model uses input-specific parameters and the input itself in addition to the desired QoR to determine such a budget. There exist different machine learning algorithms like rule-based methods, neuronal networks, or regression approaches to learn such a model [332]. However, it depends on the actual contract-based function what overhead is tolerable at the beginning of the actual execution. I suggest as a suitable method the usage of M5 models [333], which are tree-based models. Such models divide the input space into subspaces and apply a linear regression for each subspace. By specifying the amount of subspaces, we can influence the complexity of the model, hence minimize the required overhead. Currently, the input features have to be defined by the system designer. But such M5 models can be generated automatically using Cubist [334]. For the function under consideration, I use the number of columns and rows and the mean and variance of the input data as features for the model.

Another method to enable input-specific behavior of contract-based tasks is to use a monitoring approach (`monitor`). A monitor observes the progress of the function regarding the QoR. If the monitor observes that the desired QoR is reached, the task returns the current estimated result. A limited number of specific contract functions can use lightweight checks to determine the current QoR [199]. It is not possible to directly determine the current QoR in general, because it would require to know the exact result, which is often not achievable, or a tight lower bound of the QoR.

If this not the case, I exploit algorithm-specific features that correlate with the QoR. In the absence of general features that always works well, a system designer has to define useful features. Algorithms that iteratively improve the result allow us to use the difference between the previous and current result as a useful feature for the current QoR. Knowing such features, we can apply a linear regression between the features f_i and the current QoR q. Assuming we have a single feature f_0, we get the following relation using linear regression: $q = a * f_0 + b$, where a is the slope and b a constant term. Then, we reach a given QoR q, if the following equation is true: $f_0 \geq \frac{q-b}{a}$. Multiple features are combined by multiplying them in oder to still use linear regression or by switching to multiple linear regression. The quality of the monitor strongly depends on the actual contract-based function and the used features. As features, I use, for instance, the ratio between found hull points and pixels left to visit. Alternatively, I multiply the percentage of not-visited pixels with the variation of the final result.

Figure 6.4 shows the integration of both approaches, `budget model` and `monitor`, for a contract-based task. The `budget model` gets the desired QoR, the input parameters and derived features from the input data features and outputs an estimation for the required budget. The features for the input data are derived using a pre-processing step that calculates the needed features. Inside the contract-based task, the `monitor` derives features that are derived during the progress of the calculation which are related to the current QoR. If the features indicate that the desired QoR is reached, the contract-based task is completed and the current result returned. Alternatively, the task is completed,

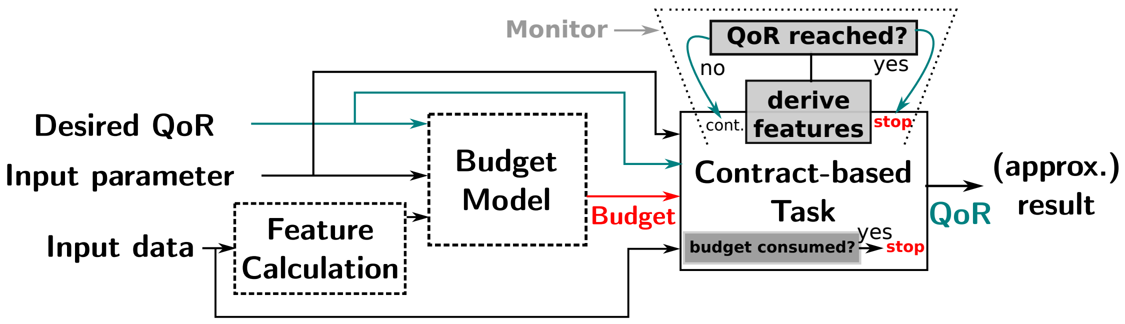


Figure 6.4: Integration of the budget model and the monitor for a contract-based task. Both methods are optional.

when the budget is consumed. Each method, `budget model` or `monitor`, is optional for a contract-based task within a system.

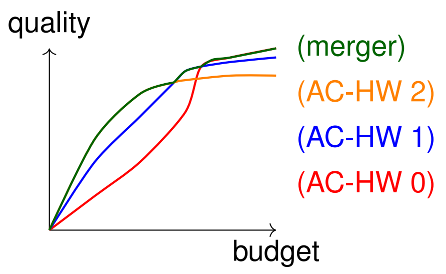
6.4.3 Exploiting Performance Profiles as Transparent View On Accuracy-aware Methods

The proposed approach of exploiting PPs to have an abstract or transparent view on the resulting approximation degree allows us to cover existing accuracy-aware method as already discussed for loop perforation (cf. Section 6.4.1). I categorize existing approximation methods into two different cases and therefor present different solutions to generate a PP.

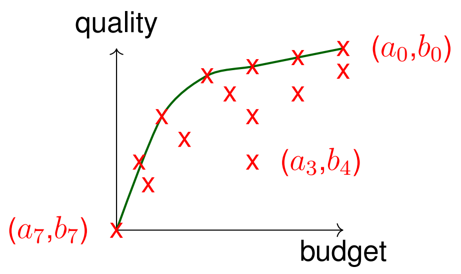
The first category includes methods that influence the hardware, where a task or function is actually executed. This is somehow similar to hardware parameters as discussed in Section 6.4.1, but these methods also have impact on the QoR. For instance, selecting approximate function units, using dynamic voltage accuracy scaling (DVAS) [9], or applying my method described in Chapter 4 pose such methods.

The second category poses methods that I consider as black boxes and thus there is no way to internally adapt the execution or measure the progress of the execution using internal features. In general, such methods create different implementation versions of a same task [17, 11, 15] or provide knobs in order to select the approximation degree. Typical examples are special hardware accelerators, for instance, based on neural processing units or DP-based AC FPGA designs presented in Chapter 5, or approximate memoization approaches (see Section 6.3.2).

However, the memoization approaches suffer from the fact that there can be a huge variance between the execution time in case of a hit or a miss. Hence, the resulting PP for approximate memoization can also be seen as an average indicator which QoR and budget trade-off can be achieved for a certain parameterization. But using fuzzy memoization can reduce the initial time of a contract-based function to produce a first



(a) A PP for AC hardware methods influencing the executing hardware are generated by the *merger* concept.



(b) Several knob settings create different points in the QoR-budget space for black box AC methods. In accordance with a Pareto-optimal front, a PP is generated.

Figure 6.5: A performance profile provides an abstract view on the resulting QoR degradation and hence can be used to control AC methods.

useful output result. Methods that do not fall into one of the above categories can be seen as white boxes and hence using the scheme as shown in Algorithm 6.5 is possible.

For the first category, I apply the same procedure as for hardware parameters. As normally the amount of configurations is low, a PP is generated for each configuration using representative data and finally combined using the *merger* concept. A configuration has a value for each available knob. Figure 6.5a shows this concept using an illustrative example. Let us assume we have three AC hardware knob values. Knob value *AC-HW 2* applies a more aggressive internal approximation and therefore has a better QoR improvement, but cannot reach the highest QoR . Using knob value *AC-HW 0* leads to the correct solution, but offers a worser QoR for lower budgets. Creating a *merger* for the PPs, we get a single PP that offers the best QoR for each budget. The information about knob value that is used to get a certain QoR is memorized in the *merger*. Depending on the desired QoR, we select the knob value in accordance with the *merger*.

In the case of a black box AC method (category 2), we cannot measure a budget and stop the execution, since we would not have a useful output result. The reason is that these methods are not made in a way that they provide a result in between the execution. Potentially, we could use the concept of starting the same method with different knob values and use the newest result that is returned as output and abort the remaining executions. However, this comes with high overhead and, moreover, we would still need some information about the behavior of the method regarding QoR and consumed budget for different configurations. Therefore, I suggest to build a model that describes this behavior as a PP.

Using representative data and a certain configuration lead to a point in the QoR versus budget space. The budget is not measured inside the method, it is rather measured for an entire call. Having two knobs for an approximation method applied to a task as shown in Figure 6.5b, then different settings (a_i, b_i) result in various points. I assume that a higher value for a knob results in a more aggressive usage of approximation. In accordance with a Pareto-optimal front, we can identify those points where a single objective has the best value compared to all other points, see Figure 6.5b. More formally, for a point $p_1 = (t_l, q_l)$, where t_l is the budget value and q_l a QoR value, there exists no point p_2 for which $b_j < b_l$ and $q_j \geq q_l$.

Having a list of different points p, generating the PP is straightforward. First, the list is sorted in ascending order regarding the required budget. Furthermore, points that have the same budget value are sorted in descending order according to the QoR value. Then, the first point in the list is added to the PP. Further points are only added if the QoR value is larger compared to the one of the previously added point. For each point in the PP, the required knob setting is also stored. This is necessary for tuning aspects (more details in Part III).

6.5 Evaluation and Results

In this section, I present experimental results for the approaches presented in this chapter. This includes an evaluation of the LSH-based fuzzy memoization and of the contract-based task approach. Furthermore, I investigate PPs not only as behavior description for contract based-tasks but also for hardware parameters and other AC methods.

6.5.1 Fuzzy Memoization

The evaluation setup for the LSH-based memoization is described in this subsection. Furthermore, I present and discuss performance and accuracy results achieved by the novel fuzzy memoization approach.

Evaluation setup. For the evaluation, I use the benchmarks from AxBench [303], which are often used in the domain of AC.

- `blackscholes` is a financial applications, which values options in the financial market. Therefore, several differential equations have to be solved.
- `inversek2j` calculates the inverse kinematic, which is a reverse transformation to calculate the joint angles for a given position in world coordinates.
- `jmeint` checks if a triangle intersects with another triangle in a 3D space.
- `jpeg` is a simple JPEG encoder. I consider the discrete cosine transformation (DCT) together with the following quantization step for the memoization.

- `kmeans` finds clusters in an image. It distributes the image into six clusters according to the pixel colors.
- `sobel` is an image filter that is applicable for edge detection. In the output images, edges are marked with white pixels.

As input data, I use the available data of AxBench.

The developed fuzzy memoization framework is appropriately integrated into each benchmark. Each benchmark is compiled with g++ in version 7 together with the options `-O2` and `-march=native`. The latter is important to exploit AVX for vector operations. The evaluation system has an Intel Core i5-7200 U with 3 MB L3 cache. The Linux governor is set to “performance”.

Performance evaluation. In the related work, often an evaluation of the achieved performance gain is missing or just the number of saved arithmetic operations is reported. In contrast, the present evaluation includes an investigation of fuzzy memoization by considering the entire application for the performance tests. Furthermore, the introduced fuzzy memoization framework is a software-only approach and thus no adaptation of the hardware is needed. For the test, I use a memoization table size of 2^{16} entries. The quantization levels are adapted for both fuzzy memoization approaches such that a hit rate of 50% shall be reached. The LSH approach uses 8 random projections. To determine the QoR, I use the normalized and averaged root mean square error between the exact and the approximated images and the available error calculation within AxBench for the remaining benchmarks.

Table 6.1 shows the execution times for different memoization approaches and benchmarks. The baseline represents the average time consumed by the function that is the memoization candidate. For the exact memoization, I show the results for using xxHash. The hit rate is independent from the general purpose hash, but xxHash provides the best performance of all considered hash functions. Quantization and the LSH-based approach work as described in Section 6.3.2. T_{miss} is the average time consumed for detecting a miss and T_{hit} for a hit, respectively. T_{avg} is the average time required for the memoized function, hence it includes the lookup in case of a hit and the actual computation for a miss. Therefore, if $T_{\text{avg}} < T_{\text{baseline}}$, then the memoization approach is valuable in terms of performance. As we can see, there is no memoization approach that is beneficial in terms of execution time. The only exception is the exact memoization approach for `blackscholes`, however, this is due to the input data set. For this set, the inputs repeat after 1000 entries and thus all data can be stored in the table. Note that the time for an execution of the function often increases, since the memoization code influences the execution time, for instance, due to cache replacements and additional jumps. The quantization approach leads to a reduction of T_{avg} compared to an exact memoization approach for two benchmarks, inversek2j and sobel. The increased hit rate is responsible for that. However, the overhead of the LSH-based approach to calculate the table entry index revokes that benefit.

Accuracy evaluation. A main benefit of the LSH-based approach is that even if the quantitative error is the same, the LSH-based approach provides a better subjective quality compared to the quantization memoization, see Figure 6.6.

Furthermore, the distribution of the erroneous pixels is different as shown in Figure 6.7. The lower part of the figure shows two error maps, which indicates the position and the magnitude of the errors. Especially, we can see that the LSH-based approach preserves more detail in homogeneous regions, for instance, the clouds in the background. Moreover, the random distribution of the error leads to a better visual quality. Note that the quantization approach produces a higher number of errors, where the magnitude of the error is larger than 10 (deviation of the pixel intensities between the exact and approximated value).

A further benefit of the LSH-based approach is the adaptation capability. While the quantization approach only achieves a hit rate of 22% for jpeg, the LSH-based approach can achieve a hit rate of 96% introducing an error of 37% by tuning the available knobs. However, this leads to even higher execution times than showed in Table 6.1. Further tests (not shown here) reveal that the usage of neither *multi-probing* nor *multi-tabling* is beneficial for fuzzy memoization.

Table 6.1: Execution time of different fuzzy memoization approaches for different benchmarks.

		blackscholes	**inversek2j**	**jmeint**	**jpeg**	**kmeans**	**sobel**
Baseline function	T_{baseline}	125.5 ns	97.9 ns	71.4 ns	155.2 ns	25.3 ns	39.0 ns
	T_{miss}	95.8 ns	125.5 ns	171.2 ns	221.6 ns	147.6 ns	182.0 ns
xxHash	T_{hit}	65.3 ns	-	-	110.1 ns	81.3 ns	106.3 ns
(exact memoization)	T_{avg}	116.1 ns	274.3 ns	291.3 ns	417.5 ns	177.6 ns	267.8 ns
	hit rate	≈99%	0%	0%	≈1%	≈47%	≈4%
	T_{miss}	108.3 ns	131.1 ns	212.8 ns	333.8 ns	164.2 ns	196.1 ns
	T_{hit}	77.8 ns	121.5 ns	253.9 ns	235.7 ns	98.5 ns	106.4 ns
Quantization (fuzzy memoization)	T_{avg}	128.9 ns	225.8 ns	326.3 ns	425.0 ns	187.0 ns	217.4 ns
	hit rate	≈99%	≈52%	≈22%	≈45%	≈56%	≈52%
	error	0%	≈40%	≈7%	≈6%	≈1%	≈4%
	T_{miss}	146.6 ns	174.9 ns	253.6 ns	527.4 ns	226.0 ns	707.1 ns
LSH	T_{hit}	111.4 ns	238.1 ns	299.0 ns	471.9 ns	141.0 ns	609.8 ns
(my fuzzy	T_{avg}	163.3 ns	325.4 ns	415.9 ns	650.4 ns	235.2 ns	721.7 ns
memoization approach)	hit rate	≈99%	≈59%	≈21%	≈48%	≈56%	≈53%
	error	2%	≈44%	≈8%	≈3%	≈0%	≈5%

6.5.2 Contract-based Tasks

The concept of contract-based tasks leads to a generally applicable and innovative software accuracy-aware method. In this subsection, I present PPs for variously implemented contract-based tasks and compare them to other general accuracy-aware software methods. This does not mean that the usage of these other methods is always valuable. Then, I show the benefit of determining a visiting order per task in detail, both in a static and dynamic manner. By exploiting an ED^{n}P, the consideration of multiple objectives, energy consumption and execution time, is just a matter of exchanging the definition of the budget metric for the PP. This results in a different PP curve. The generation of this PP is trivial. The PP is realized as a discrete function. We can measure the corresponding values of the objectives within the function, see Section 6.4.1. We use these values to calculate the desired ED^{n}P metric.

Moreover, I present how to integrate the checkpoint approach and investigate the introduced overhead of it. As the QoR of a contract-based function often depends besides

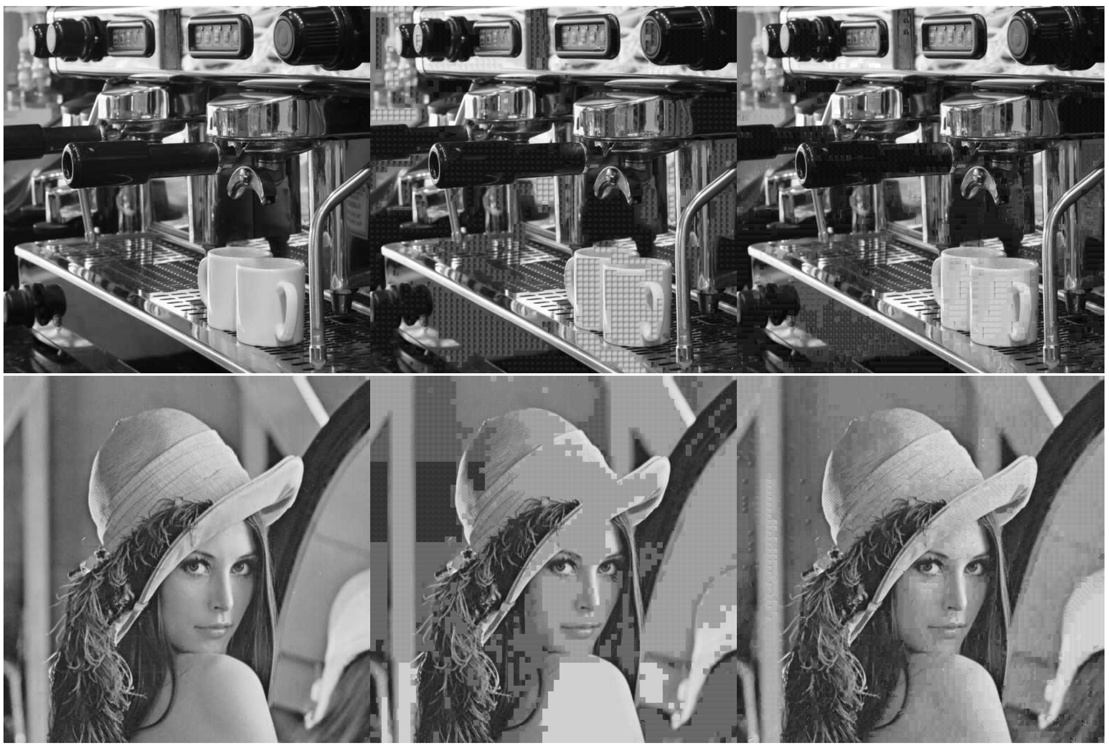
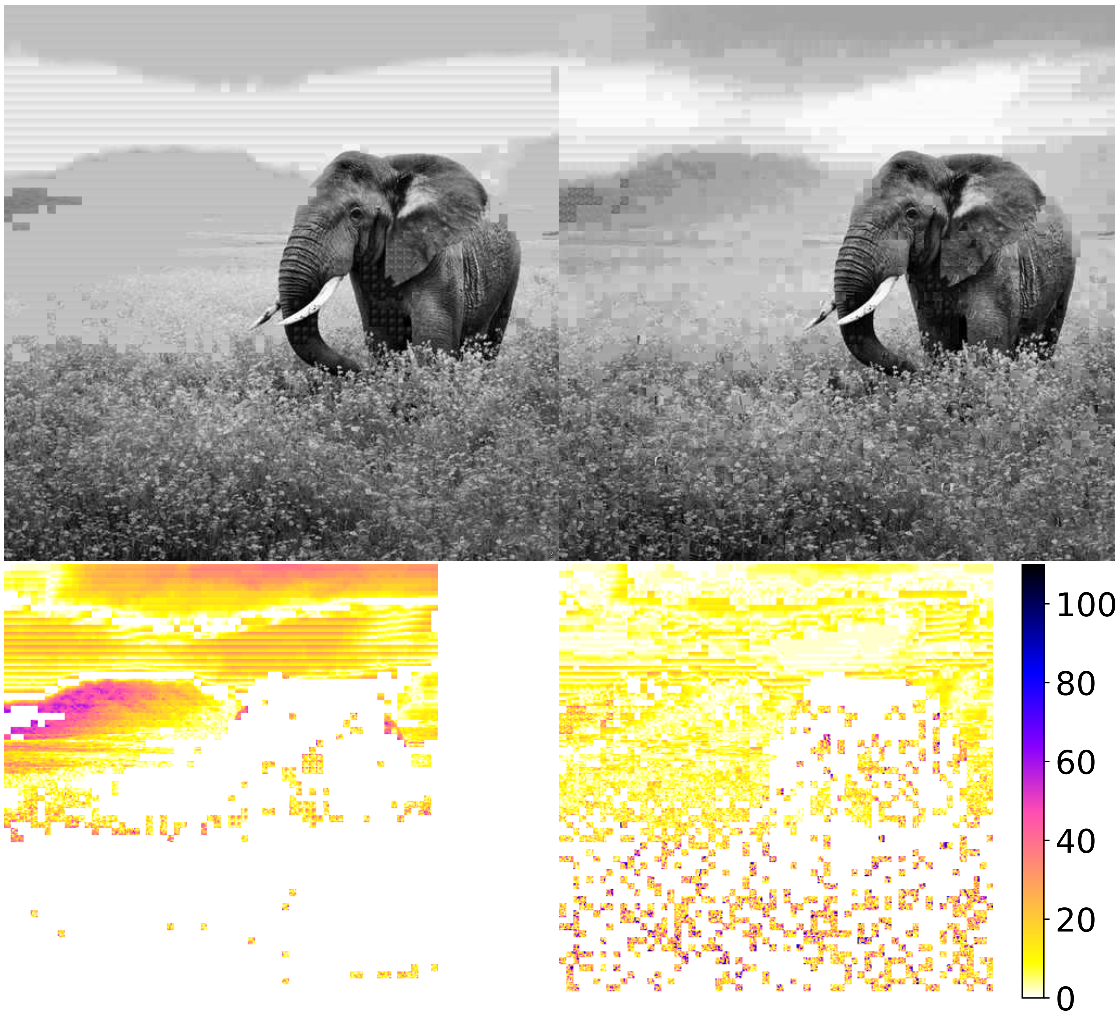

(a) Original algorithm. (b) Quant.-based memoization. (c) LSH-based memoization.

Figure 6.6: Visual comparison of the JPEG encoder benchmark between the quantization approach and the LSH approach, while the hit rate is similar (Quantization: 44%, LSH: 46%).

on the budget on the actual input, I show real data to emphasize this issue. Afterwards, I measure the overhead and accuracy of my suggested methods to deal with the aforementioned issue and present how they reduce the QoR variance.

As evaluation systems, I use three different systems. Test system A includes an Intel Core i7-4550U CPU running at 2 GHz with 8 GB DDR3 RAM. The L2 cache provides 256 kB and the L3 cache 4 MB. System B has two Intel Xeon CPU E5-2650 v4 running



(a) Quantization-based memoization.

(b) LSH-based memoization.

Figure 6.7: Comparison of memoization methods according to JPEG encoder benchmark with a set QoR of 95 %. (a) Quantization-based memoization (42 % hit rate, QoR=26.2532 dB PSNR). (b): LSH-based memoization (48 % hit rate, QoR=26.1723 dB PSNR). Colormaps visualize the deviation in pixel intensity between the exact and approximate version. The error bar shows the magnitude of the pixel error.

at 2.2 GHz with 128 GB RAM. The total number of cores is 24 and therefore 48 hardware threads are possible. Each processor has 30 MB SmartCache[8] (last level cache). Test system C is configured in a way that it is possible to change the frequency of a core from user space. The system comprises an Intel Core i5-6500 CPU running at 3.2 GHz. The processor has 6 MB SmartCache and 8 GB of RAM attached.

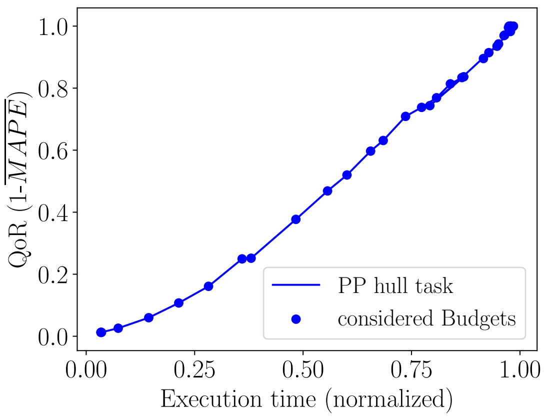
First examples of real performance profiles. To get an impression for PPs based on real data, I present several PPs. These tasks determine different object features that are used to classify an object within a binary image. The task `Calculate Hull` calculates the hull of the object and represents the hull as points within the image. To determine the area, `Calculate Area` counts the "white" pixels in the binary image. `Calculate Diameter` exploits the hull points in oder to calculate the diameter of the objects. The diameter is the maximum Euclidean distance between two hull points.

To realize a contract-based implementation of the above algorithms, I use the procedure explained in Section 6.4.1. In case of `Calculate Hull` and `Calculate Diameter`, I integrate an exit condition into a for-loop body (according to Algorithm 6.5) and measure if the time budget is consumed. Therefore, these contract-based task process the input data items sequentially. For `Calculate Area`, I apply the proposed approach of *nested loop perforation*. Furthermore, the area is scaled according to the currently calculated area and the so-far considered rows. Thus, a post-processing step is applied. The contract-based algorithms are implemented in C++-11. g++ in version 6 and the compiler option `-O3` are used.

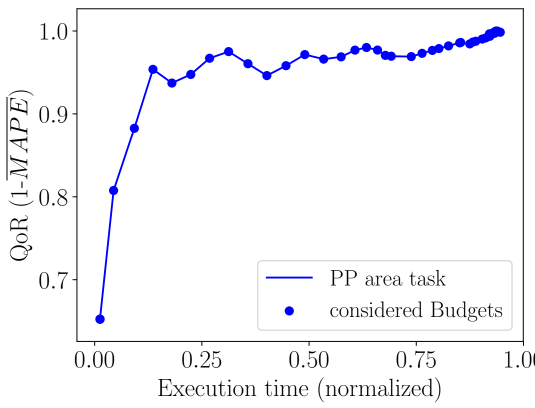
To generate an individual PP, I use more than 1760 binary images showing one of three different object types (sphere, cylinder, square) as representative input data. For `Calculate Diameter`, the corresponding result of `Calculate Hull` is used. I configure `Calculate Hull` in such a way that it returns the exact hull. For the PP generation, I use different budgets for the execution time (shown as points) and determine the resulting QoR. $QoR = 1 - \overline{\text{MAPE}}$ and the average MAPE is $\overline{\text{MAPE}} = \frac{\sum^{n} \text{MAPE}}{n}$. MAPE represents the mean absolute percentage error and is defined as MAPE $= \frac{|\text{res}_{\text{exact}} - \text{res}_{\text{approx}}|}{\text{res}_{\text{exact}}}$, where $\text{res}_{\text{exact}}$ is the result of the exact algorithm and $\text{res}_{\text{approx}}$ of the approximated. For `Calculate Hull`, I use the number of found hull points, since even for the approximate version a found point is always correct. Between measured points in the QoR and budget space, interpolation is used to generate a visual line for the generated figures. The PPs shown in Figure 6.8 are based on measurements performed on Test System A. The contract-based tasks are executed on a single core.

All shown PPs have an almost monotonically increasing trend of their QoR over an increase of the budget. In Figure 6.8b, the PP for `Calculate Area` has a high QoR im-

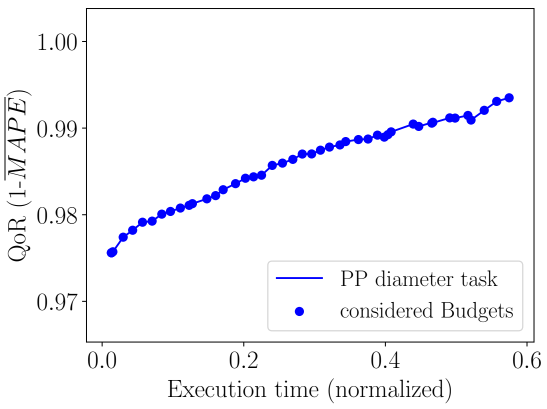
[8]SmartCache is L2 or L3 cache method developed by Intel. It shares the cache memory between multiple cores.



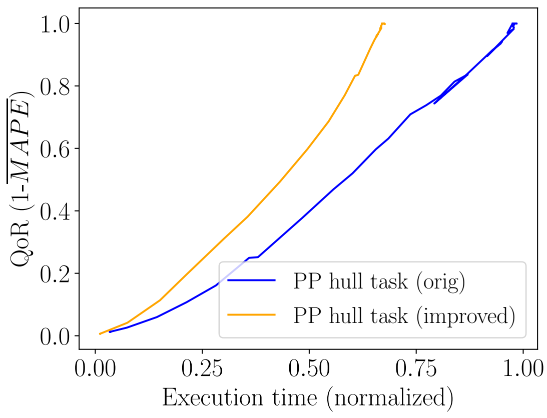
(a) `Calculate Hull`.



(b) `Calculate Area`.



(c) `Calculate Diameter`.



(d) Improved `Calculate Hull`.

Figure 6.8: Performance profiles for different contract-based AC tasks.

provement in the beginning. For `Calculate Diameter`, we see that already providing a low budget is sufficient to get a high QoR (see Figure 6.8c). For each contract algorithm, the order of processing the input data is sequential, except for `Calculate Area`, where I use the nested loop perforation approach. However, to emphasize that changing the processing order is very beneficial, I implemented an improved version of `Calculate Hull` regarding the QoR improvement highlighted in Figure 6.8d.

For the improved version of `Calculate Hull`, I changed the algorithm itself such that a binary input image is not sequentially processed. In an alternating way, pixels are considered from the left, top, right, and lower part of the image until all pixels are processed. This accelerates the process of finding the hull points. Such an improved implementation requires a priori algorithm knowledge. Therefore, I have proposed the general solution of determining a task-specific ordering using Algorithms 6.3 and 6.4.

Comparison with existing accuracy-aware software methods. In this paragraph, I compare the contract-based task approach against existing approximation techniques at software level. I consider loop tiling (LT) [14] and loop perforation (LP) [13]. I use these methods for the outer loop of the kernels. Additionally, I compare it against the methods presented in [335] (named `automaton`). The `sequential` method processes input data items in a sequential order. `Automaton (LFSR random)` uses a pseudo-random order using a linear-feedback shift register (LFSR). `Automaton (tree)` exploits tree permutation sampling.

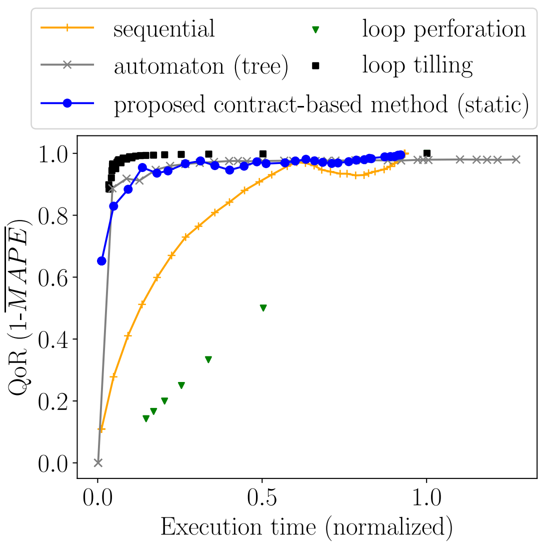
For the comparison, I use three different kernels from the PERFECT benchmark suite (*2D convolution*, *histogram calculation*, *debayering*), which are compute kernels in the domain of embedded computing [336]. For these benchmarks, I measure the QoR in terms of signal-to-noise-ratio (SNR), where SNR $= 20\log_{10}\left(\frac{\text{signal}}{\text{RMSE noise}}\right)$ and hence the SNR is expressed in decibels (db). RMSE is the root mean square error. For images, I calculate the SNR using the following equation SNR $= 20\log_{10}(I_{\text{max}}) - 10\log_{10}(\text{MSE})$, where I_{max} is the maximum possible pixel intensity. The mean square error

$$\text{MSE} = \frac{\sum_{i=1}^{n}(I_{\text{ref}}(i) - I_{\text{test}}(i))^2}{n}$$

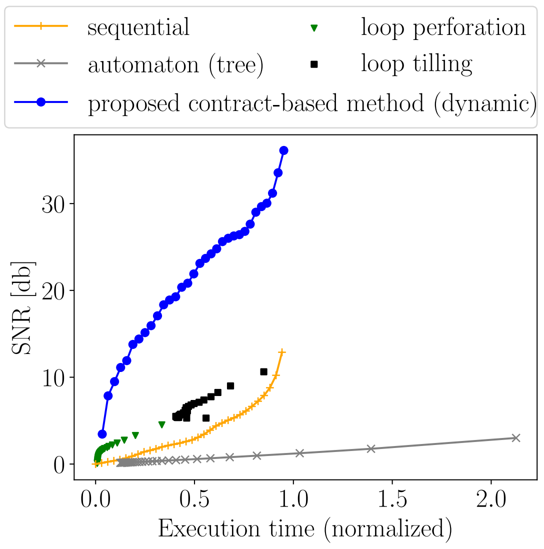
where $I_{\text{ref}}(i)$ is the intensity of the ith pixel in the exact image and $I_{\text{test}}(i)$ the ith pixel in the approximate image. Both images consist of n pixels. I exploit histogram equalization to get a SNR value for the histogram calculation kernel. It is noteworthy that it is not clear how non-computed output pixels are treated before calculating SNR by the authors of [335]. Therefore, I assume that these pixels are zero. Moreover, I compare the existing approximation methods with my proposed contract-based approach for `Calculate Area`. I also consider a *matrix-vector multiplication* using input matrices of size 100×100, where the elements are defined by a probability distribution. I use the *exponential*, *dweibull*, and *chi2* probability distribution.

I apply the static task-specific approach (`static`), where the sampling ordering is determined during design time, and the dynamic task-specific approach (`dynamic`), where the ordering is determined at runtime and thus the PP includes the overhead for it. To calculate the contract-based histogram, iterative loop perforation adapted by the detected static ordering in combination with post-processing scaling is used. In case of contract-based *debayering*, I use a post-processing step.

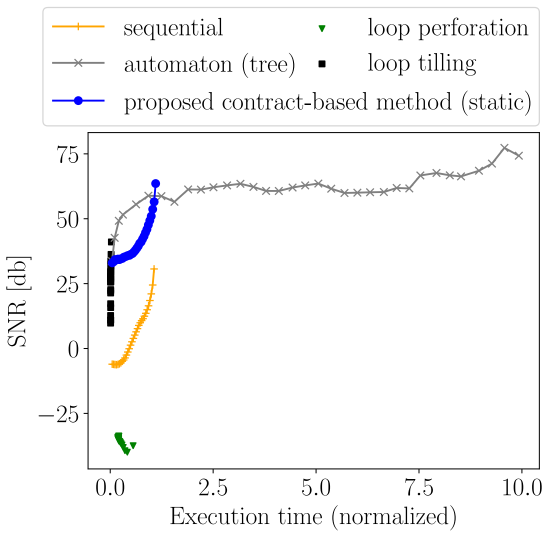
The figures shown in Figures 6.9 and 6.10a are generated running the same kernel multiple times and halting it at different times for the existing methods. For the contract-based tasks, I vary the time budget. The overhead of my proposed contract-based method (blue line) is almost negligible compared to the original benchmarks to get the exact solution. This can be seen, since the highest QoR is reached by my method at around 100% (1.0) of the normalized execution time. The only exception exists for the *2D convolution* benchmark. This higher overhead is mainly caused by calculating the



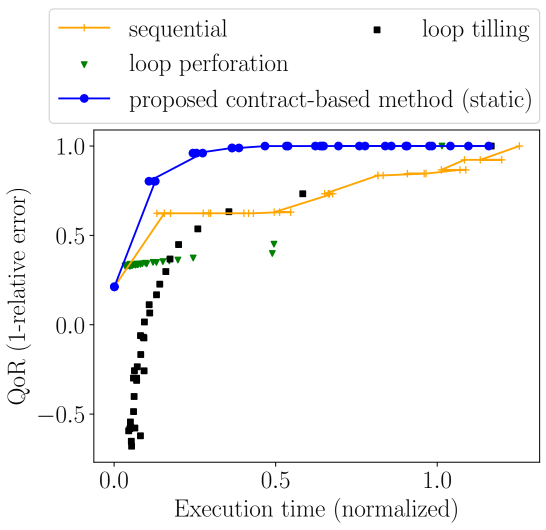
(a) `Calculate Area`.



(b) Debayering.



(c) Histogram equalization.



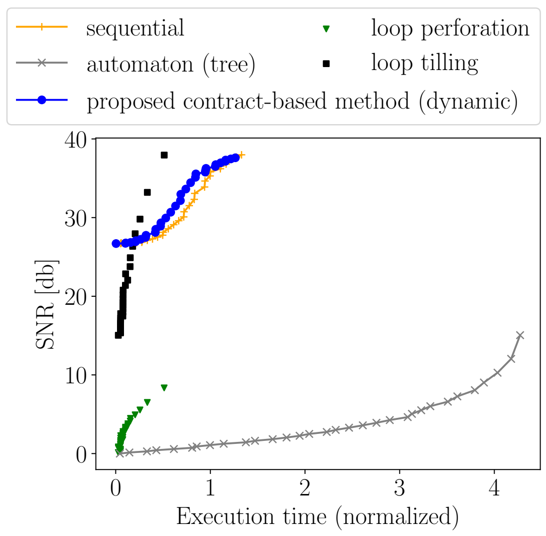
(d) Matrix-vector multiplication.

Figure 6.9: Comparisons of my proposed contract-based method (static or dynamic, blue line) with existing AC software methods for different benchmarks.

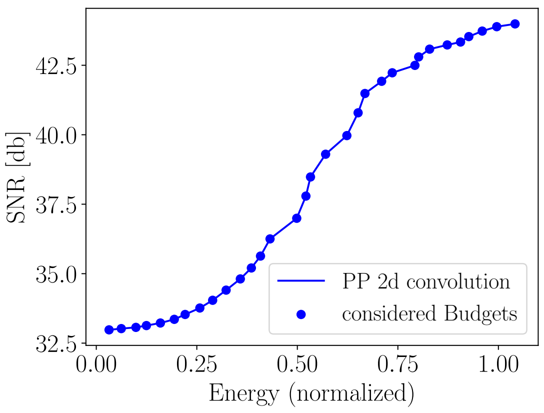
dynamic ordering. Note that my approach has a significant lower overhead than the existing `automaton` approach (gray line). This overhead is especially significant for the 2D convolution benchmark, where the `automaton` approach requires for times more to reach a QoR of 1 compared the original benchmark. The reason is that my contract-based tasks exploit the cache better. That is a main issue of `automaton` as mentioned by the authors.

For `Caculate Area` (Figure 6.9a), my proposed approach (blue line) has a high improvement in the beginning and only a slow improvement later similar to `automaton` (gray line). In the case of *debayering* (Figure 6.9b), my approach (blue line) reaches acceptable output quality (20 db) after 40% and excellent image quality (32 db) after 85% of the baseline execution time. A high QoR for the *histogram equalization* (Figure 6.9c) is reached by my approach and `automaton` quite early. Furthermore, my proposed method significantly outperforms all other existing methods on the *matrix-vector multiplication* benchmark (Figure 6.9d). As we can see for *2D convolution* (Figure 6.10a), the dynamic ordering (blue line) enables a higher improvement of the QoR between 30% and 100% of the original execution time compared to the sequential method (orange line). Loop tiling (black line) has an even higher QoR improvement, but offers only a few points that can be reached and are below the performance of my method (above blue line). Thus, a fine-tuning is not possible.

The task-specific ordering of my method (blue line) always outperforms the sequential method (orange line). The sequential method processes input data items sequentially. LP (green line) shows the worst behavior on all compute kernels in general. Skipping each second iteration (smallest possible approximation knob value) already leads to a high QoR loss. LT (black line) is quite reasonable, especially, as it reaches good QoR early. The main drawback of LT is that fine-tuning is often not possible, since the distance



(a) 2D convolution.



(b) Energy-based PP for 2D convolution.

Figure 6.10: Comparison of my approach (static, blue line) with existing AC software methods for 2D convolution (a). Furthermore, PP is shown generated with the consumed energy as budget (b).

between reachable QoR points increases with higher quality. Additionally, LT shows different behavior for the benchmarks. For instance, it performs worse on the matrix-vector product benchmark. LT would require re-executing all tasks to increase the QoR of the output, where my proposed checkpointing method for contract-based tasks makes this unnecessary. However, I could use LT to reach a fast and reasonable initial QoR for the proposed contract-based tasks.

In total, my method provides a high QoR improvement over time combined with negligible overhead compared to the original version as shown on completely different kernels. Hence the introduced method is a **general** software method that enables **fine-tuning** of the approximation degree. Besides contracting the time, my method allows us to constrain the energy consumption of the contract-based task. Using the consumed energy as budget makes this possible. For instance, we can generate a PP for the contract-based 2D convolution using the consumed energy (see Figure 6.10b). However, this comes with additional overhead, since *likwid* is used to get an access on the hardware counters. Direct access via *likwid* causes an overhead ranging from 2 µs to 7 µs depending on the set frequency on the Test System B.

The shape of the resulting PP for the 2D convolution budgeting the energy looks similar to the one using the time. Which PP is the desired one depends on the actual purpose and thus it depends on which single objective is more important. Later, I will show how we can consider both objectives simultaneously.

Influence of the visiting order of samples. The above results show that task-specific ordering is very beneficial for contract-based tasks. So far, I consider either the static or the dynamic ordering method for an application. Now, I consider both for the same algorithm. In time series analysis, an important task is to classify an unknown time series by means of a database. For this task, I use an one-nearest neighbor algorithm (1NN) that uses the dynamic time warping (DTW) as distance metric. The usage of DTW results in a reasonable accuracy of the classification [337]. The class of the nearest neighbor is returned as class for the unknown query. I use time series databases from the UCR archive for the evaluation and use the Sakoe-Chiba (SC) band [258], which restricts the allowed warping path for DTW (more details see Section 5.2.5)

For the distance metric required for the 1NN, I use a contract-based variant of DTW that uses a *static* task-specific ordering determined using Algorithm 6.3. The contract-based version of 1NN uses a *dynamic* task-specific ordering to get a per-input (unknown time series) ordering by using Algorithm 6.4. The ordering specifies the order in which the time series of the database are considered as possible nearest neighbor. Hence, those time series are considered first that are more likely the searched one. This enables a fast improvement of the QoR for the contract-based 1NN. The score function V in Algorithm 6.4 is realized by using the Euclidean distance between the query time series and the current time series in the database. Note that using Euclidean distance as metric for the 1NN normally results in a lower classification accuracy, however it provides a good rough estimation for the dynamic ordering of database time series.

The QoR is defined as QoR $= 1 - (\overline{c}_{\text{error}}^{\text{approx}} - \overline{c}_{\text{error}}^{\text{exact}})$, where $\overline{c}_{\text{error}}^{\text{exact}}$ is the average classification error of the original method and $\overline{c}_{\text{error}}^{\text{approx}}$ of the contract-based version. The average classification error $\overline{c}_{\text{error}}$ is calculated using available class labels of the provided test time series in the UCR archive [274]. It can occur that the QoR is larger than 1, since it is possible that $\overline{c}_{\text{error}}^{\text{approx}} < \overline{c}_{\text{error}}^{\text{exact}}$. The rationale is that even when we use DTW as metric, we do not get a correct classification in each case.

Figure 6.11a shows the PPs for the proposed *static* task-specific ordering version (solid lines) and a *sequential* version of the contract-based DTW method (dashed lines) applied to time series data of different UCR archive databases. The *sequential* method processes the elements of the time series in a sequential order. The proposed *static* method achieves an accuracy close to using the original DTW after roughly 33% of the DTW-based execution time. The *sequential* method requires a budget of more than roughly 80% or 90% to reach an acceptable QoR. Hence, considering the right points of a time series at the beginning leads to significant benefits. This is exactly achieved by the proposed static task-specific ordering.

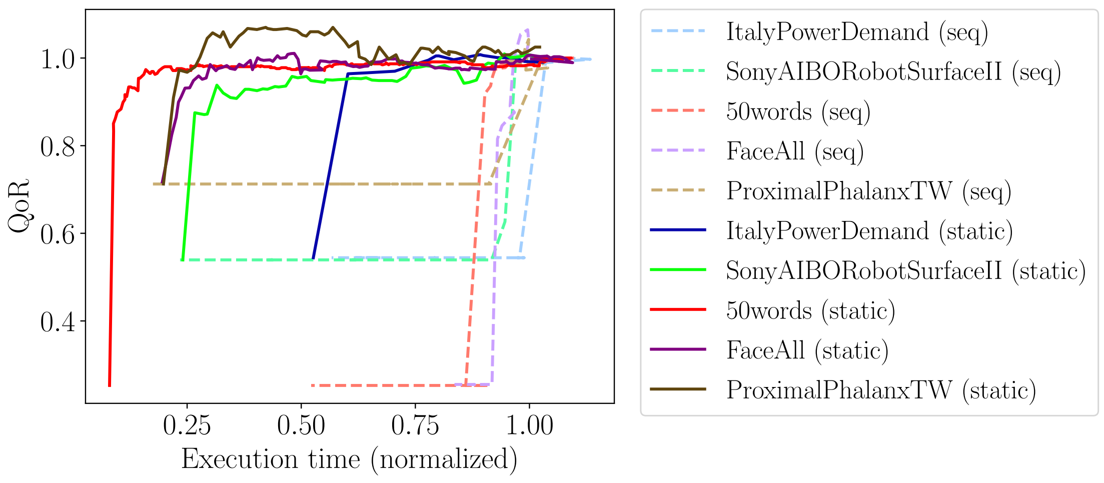
In the next experiment, I compare a *sequential* version (dashed lines) with a *dynamic* task-specific ordering-based version (solid lines) of the contract-based 1NN. As we can see in Figure 6.11b, the proposed *dynamic* ordering for visiting the database time series inside the contract-based 1NN is important to reach a high QoR at the beginning of the contract-based 1NN execution. Again, the task-specific method outperforms the sequential-based method (dashed lines). In contrast to the static method, the dynamic ordering method is independent from the database, hence it does not have to be adapted to a certain time series domain. It is possible to use the *static* contract-based DTW version within the *dynamic* contract-based version of 1NN. This combination of both is considered in Part III.

To see further benefits of a task-specific ordering, I employ the contract-based 2D convolution inside the Richardson-Lucy deconvolution (RLD). The RLD is an iterative method (see Section 4.2.1). In each iteration, this deconvolution performs two convolutions, a point-wise multiplication and division. In general, the best improvement regarding the QoR is achieved after 10 iterations [338]. Therefore, I set the iteration count to 10 for the evaluation. According to [20], I distribute the execution into several phases (iteration 1, iteration 2 to 5, and iteration 6 to 10). The significance of each phase is almost similar except for the first phase of the second convolution and therefore I avoid an approximation of the first phase. As QoR, I use the *peak to signal noise ratio* (PSNR)[9].

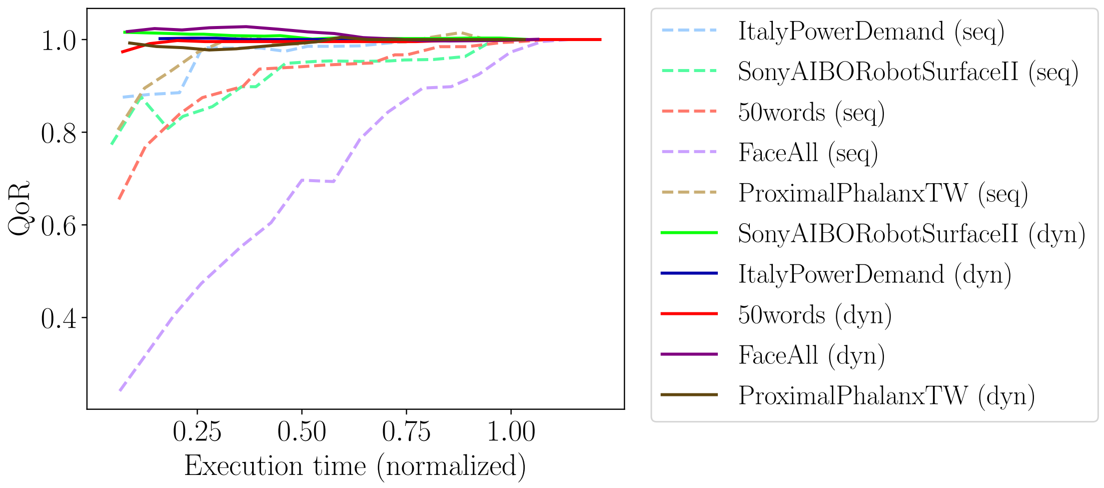
For a given microscopy image (see Figure 6.12a), the algorithm mainly improves the

[9]PSNR is often used as performance metric for deconvolution in the domain of microscopy (see `http://bigwww.epfl.ch/deconvolution/challenge/index.html`)

blurring effect around an object[10]. My method enables that approximations unlikely occur close to the object as indicated by the white area in Figure 6.12b, while reducing the execution time by 10%. The dynamic task-specific ordering allows us to find the relevant parts in the image and start the RLD calculation there.



(a) Comparison of the *static* task-specific ordering version (solid lines) with a *sequential* version (dashed lines) of the contract-based DTW metric.



(b) Comparison of the *dynamic* task-specific ordering for the contract-based 1NN with a *sequential* version.

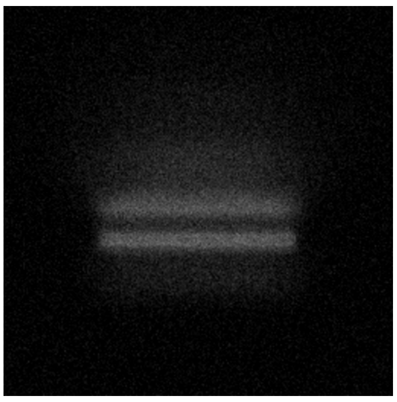
Figure 6.11: Performance investigations of contract-based versions of tasks for time series analysis.

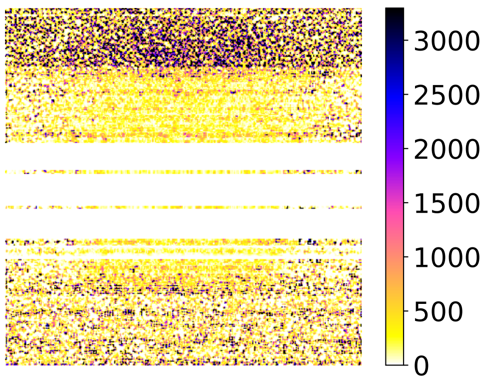
[10]The image in Figure 6.12b shows the change in the intensity level of each pixel.

Considering checkpointing for contract-based tasks and its overhead. The motivation behind using checkpointing is that we can continue the execution of a contract-based task in order to improve its QoR. This requires less overhead than restarting the entire task with a different budget that presumably allows reaching the desired QoR. Figure 6.13 shows a progress of the QoR for different tasks and the introduced overhead for checkpointing.

In case of 2D convolution, the task works until the budget is reached. This results in a SNR of roughly 32 decibel. Afterwards, additional computations, control methods or a quality check by the user can be performed. I simulate these computations with a method that essentially ensures that the data residing in the cache for the contract-based task is replaced. Therefore, I request a huge memory space and apply computations that work on this space. Restarting the execution from the checkpoint only comes with negligible overhead of 2% compared to a direct execution resulting in the same final QoR.

However, the overhead is more critical for other contract-based tasks that have different memory footprints, for instance, the matrix-vector product, where the contract-based version is based on a blocking scheme. For such tasks, the overhead is up to 70%. This likely indicates that the missing data in the cache plays an important role regarding the restarting overhead. Even in this case, the checkpoint method is faster than restarting the contract-based task. In another experiment, I also consider the overhead for the contract-based version of the 1NN. The overhead is around 2% to 10% depending on the specified frequency of the executing core.


(a) Input image.



(b) Error map shown as deviated intensities.

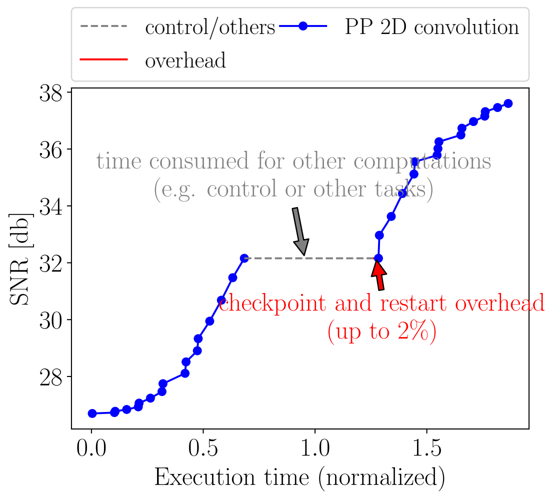
Figure 6.12: Richardson-Lucy Deconvolution. Figure 6.12b shows the deviation caused by approximation for the input image (see Figure 6.12a). The bars specify colors that indicate the magnitude of the error starting from white (no error).

Dealing with input-dependent QoR variance. As already mentioned, a static assignment of budgets is only optimal if the variance regarding the QoR is small [317] even for different inputs, as discussed in Section 6.4.2. To emphasize this aspect, I collect the difference in QoR for contract-based algorithms and inputs as shown in Figure 6.14.

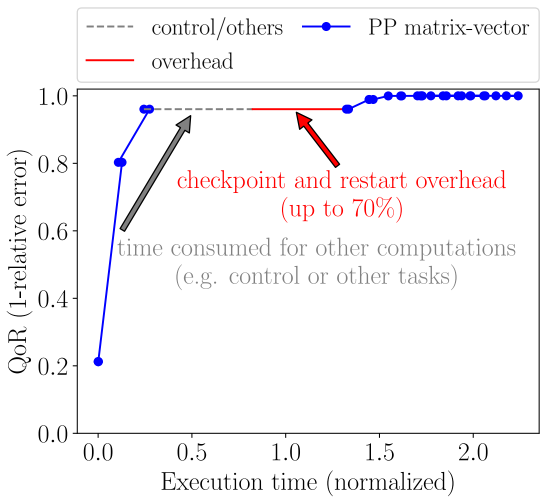
For the RLD, I use different microscopy images as input. The QoR is defined as the number of erroneous pixels (relative error occurrence). As we can see in the box plot shown in Figure 6.14a, granting a certain budget for different inputs leads to different QoRs indicated by the size of the boxes. The size of each box is an indicator for the QoR variance for a certain budget. Especially, the variance is quite high for low budgets.

The box plot for the contract-based version of `Calculate Hull` is shown in Figure 6.14b. I generate the data by using a large set of binary images and providing different budgets. The QoR is defined as $1 -$ MAPE. Especially, the boxes are large for medium size budgets and thus indicate a high QoR variance for different inputs. The histogram in Figure 6.14c shows the resulting QoRs for setting the budget to 18 time units for `Calculate Hull` and different inputs. While most inputs reach a QoR of 1, there are some that have really low QoRs. The number of bars in the histogram is an indicator of the QoR variance. The achieved average QoR is 0.83 for this experiment.

According to the results, we are facing with the following challenge: How can we adapt the budget in order to reach a desired QoR? Hence, it is important to identify the required budget for each input individually. The goal is now to achieve that for all inputs the individual QoR is the same as the desired one, for instance, 0.83. The rationale is manifold: Firstly, there should be almost no inputs where the resulting QoR is lower than

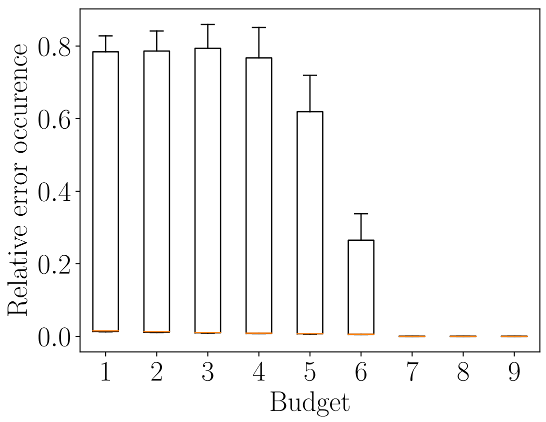


(a) 2D convolution.

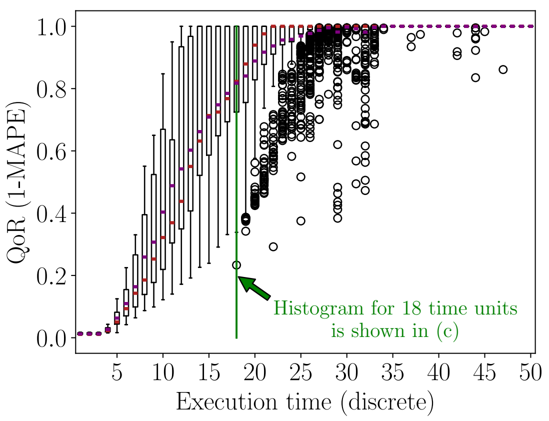


(b) Matrix-vector product.

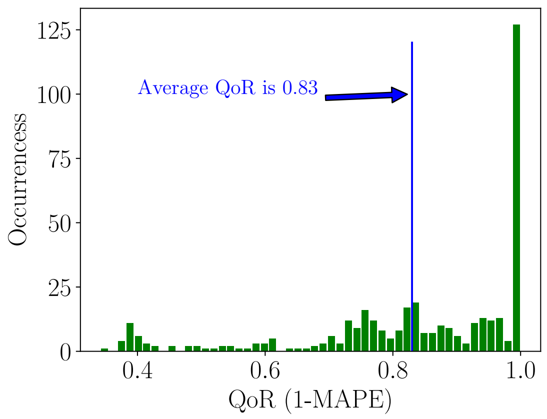
Figure 6.13: Resulting PP when exploiting the checkpoint approach and considering the involved overhead.



(a) Box plot for the Richardson-Lucy deconvolution and different input images.



(b) Box plot for `Calculate Hull` and different input images.

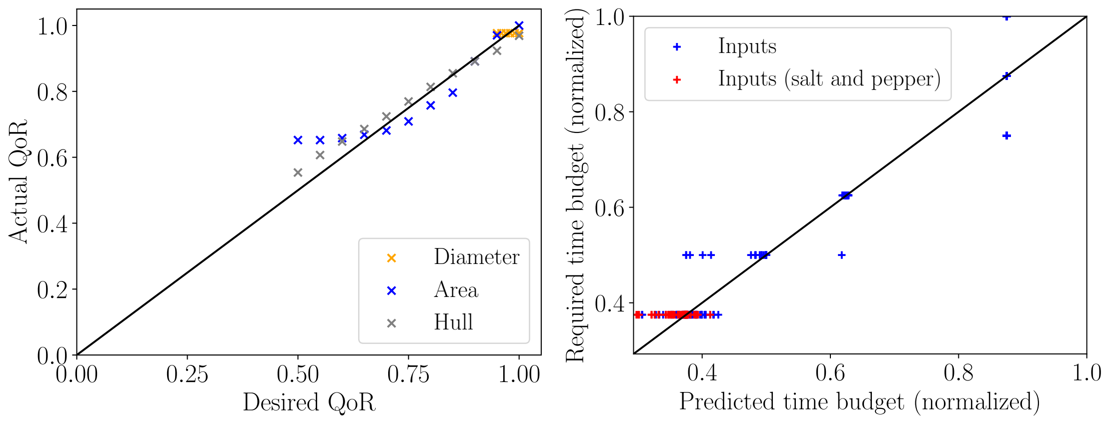


(c) Histogram of the resulting QoRs for different input images processed by `Calculate Hull`. The budget is set to 18 time units.

Figure 6.14: Resulting QoRs of different contract-based tasks for different inputs and budgets.

desired. Secondly, a higher QoR than the desired one likely indicates that we can reduce the computational effort, while still meeting the desired QoR. This can be exploited to improve the value of other objectives. Finally, controlling the contract-based task from a global point of view is easier. This is important for controlling several of these tasks within an application or part of different applications. However, there is always a trade-off between the achieved accuracy of a method regarding the determination of the required budget and the computational effort introduced by this method. A high effort rules out the benefits of the proposed contract-based method.

To achieve this goal, I presented a solution in Section 6.4.2. The *budget model* esti-



(a) Different image feature contract-based tasks.
(b) Richardson-Lucy deconvolution.

Figure 6.15: Accuracy investigation for the *budget model* designed for different contract-based tasks .

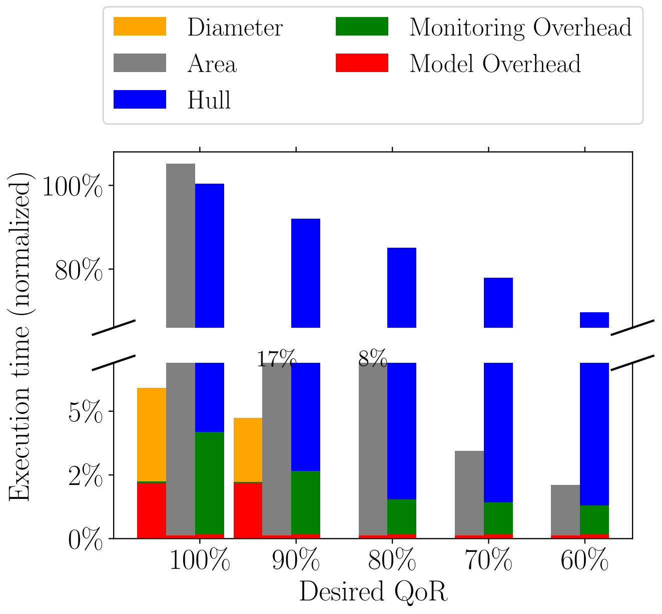


Figure 6.16: Overhead of the *budget model* and the `monitor`.

mates the required budget to reach a desired QoR. I investigate the accuracy of *budget model* by comparing the desired QoR with the one that was achieved by the estimated budget for different contract-based tasks and inputs, see Figure 6.15. In Figure 6.15a, I consider the accuracy of the *budget model* for contract-based tasks that calculate image features (diameter, area, and hull). I also consider this for the RLD, seeFigure 6.15b.

As we can see, the *budget model* works quite well for the image feature contract-based tasks. This can be seen, since almost all points fall onto the (black) line. A point on the line indicates a perfect estimation. Similarly, this is the case for the *budget model* of RLD and high budgets (high QoRs). For lower budgets, the estimation is not very accurate, however, an accurate estimation is pointless, since low budgets are not sufficient for a reasonable QoR. Furthermore, for gray images that have a salt and pepper noise in it, the estimation is harder.

Besides the accuracy of the model, the introduced overhead is an important point. Figure 6.16 shows this overhead for the *budget model* (red). Additionally, I also consider the overhead of the proposed *monitoring* method (green). The overhead is represented as the relatively consumed execution time based on the respective original task. For `Calculate Area`, the overhead is negligible. The overhead for `Calculate Hull` varies between 1% and 4%, especially, the *monitoring* has a higher impact. Since `Calculate Diameter` gets a reasonable QoR quite early (cf. Figure 6.8c), finding a controlling mechanism is difficult. Moreover, the overhead is more significant because the execution time for getting a high QoR is low.

Next, we consider how the proposed methods lead to a better solution compared to assigning a fix budget. Let us assume that we want to achieve a QoR of 0.83 for each input processed by `Calculate Hull`. Applying the *budget model*, the number of inputs that have a larger QoR than 0.83 is reduced, see Figure 6.17a and cf. Figure 6.14c. This reduction leads to a decrease of the QoR variance from 0.031 to 0.012 as seen by the compression of the histogram compared to the histogram in Figure 6.14c. Moreover, the average time budget is reduced to 17 time units instead of 18, cf. Figure 6.17b. There are some inputs where the resulting QoR is too bad (< 0.83), see Figure 6.17a. If it is crucial to avoid larger errors, approaches from literature can be used [24, 203]. Such approaches use a guard band for the QoR, for instance, by adding a constant to the needed QoR. Building more accurate models would require more knowledge about the inputs, and therefore more complex models. This is only applicable if the additional overhead does not rule out the savings achieved by the approximation.

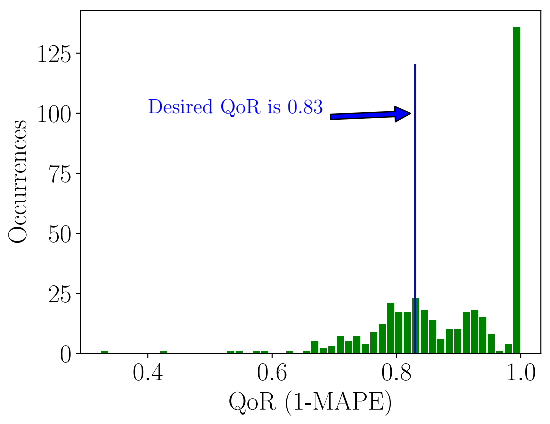
On the other side, the resulting average QoR is increased to 0.88, while we want to have a QoR of 0.83. Hence, we miss potential to gain performance, because most inputs have a QoR of 1. Using the *monitoring* and *budget model* together, we achieve to get an average QoR of 0.83, see Figure 6.17c. Additionally, we can see that bars of the histogram are more located at the desired QoR and thus the variance is reduced to 0.009. Hence, this is also an indicator that the computational effort is decreased.

6.5.3 Exploiting Performance Profiles for System Tuning

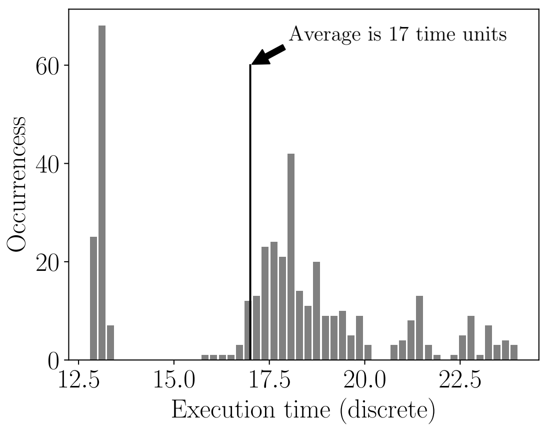
Generalizing the behavior description of AC tasks using the concept of PPs enables a unified method to control the approximation degree of different tasks, even when the underlying approximation methods are different. Therefore, PPs play an important role for my thesis approach to control the approximation degree within a system.

Here, I present results for exploiting PPs as a behavior description of further approximation methods. Moreover, PPs easily allows us to consider multiple objectives and take further hardware parameters into account.

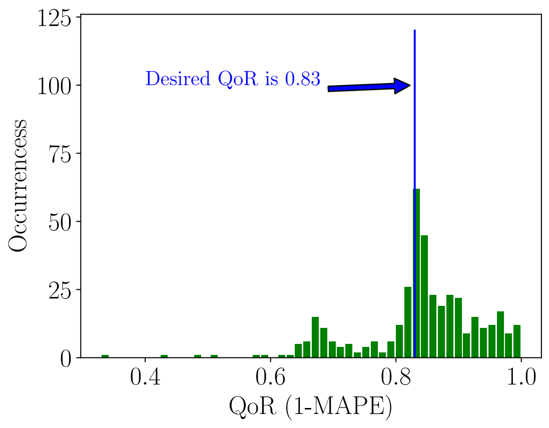
Dealing with multiple objectives. Instead of considering the objectives, performance and energy consumption, individually, my approach also allows us to consider both at the same time. Therefore, different energy delay product metrics can be exploited. This is related to just changing the definition of the budget for the PPs. Since the values for both objectives, execution time and consumed energy, are already available, we can combine



(a) Applying the *budget model*.



(b) Histogram about the estimated budgets.



(c) Applying the budget model and the monitor.

Figure 6.17: QoR variance improvement compared to a fix budget (18 time units) by applying the *budget model* and the *monitoring*.

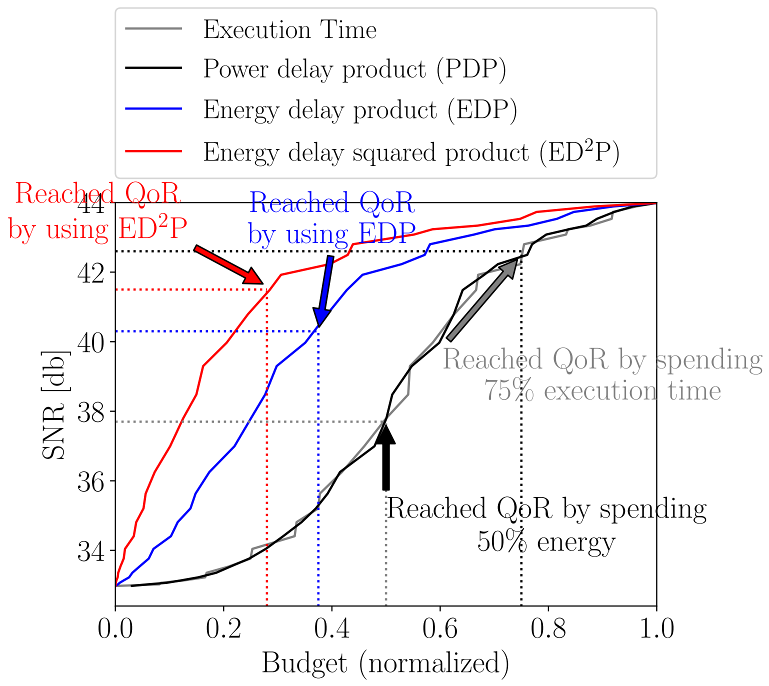


Figure 6.18: Showing different performance profiles for the 2D convolution, while varying the metric of the budget. This leads to a multiple objective consideration of performance profiles. It is important to emphasize that each curve has a different meaning of the budget (x-axis) as indicated by the label.

them to generate a budget metric and thus a new PP.

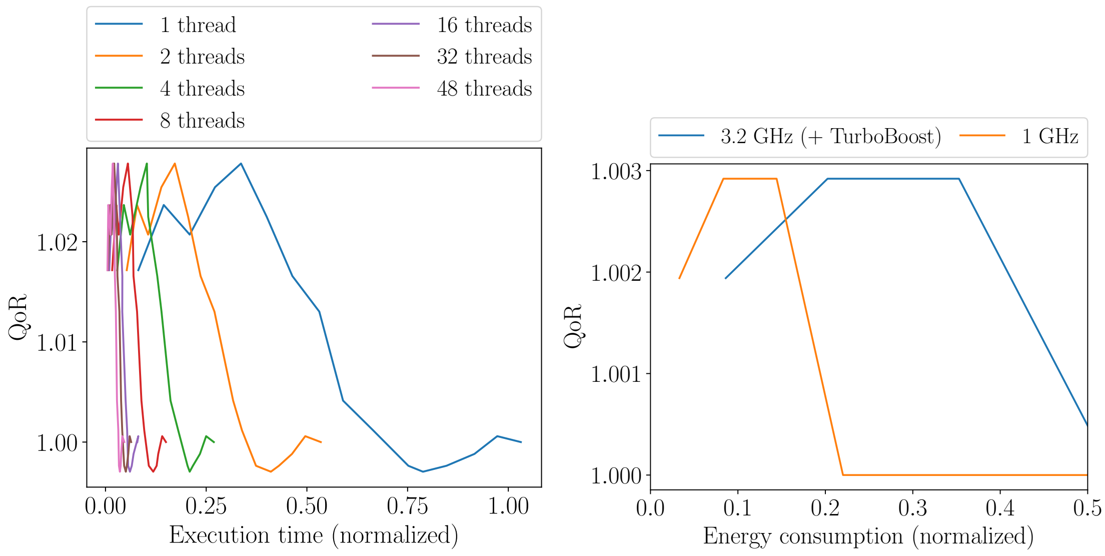
Figure 6.18 shows the PPs for the different used budget metrics. It is important to note, that the x-axis is scaled between 0 and 1 for each PP individually and therefore a direct comparison is not possible. The main purpose is to show how a changing budget definition will lead to different selected configuration points.

Let us consider that our objective is to not spend more than 50% of energy and 75% of execution time, while maximizing the QoR. Considering the execution time alone, we would reach a SNR of roughly 43 decibel. However, since this would also require more than 50% of energy, we can only reach a QoR of roughly 38 db, when satisfying this goal.

Considering both objectives at the same time and thus finding a good compromise between them is possible for my approach by relying on the concept of exploiting ED^{n}P metrics. Having a PP generated by using an ED^{n}P, the respective budget is calculated using the given objectives. For instance, for the EDP the budget is $50\% * 75\% = 37.5\%$ and hence the best achievable QoR is roughly 40 db according to the PP. Putting more emphasis onto the performance objective, the ED^{2}P can be used. This results in a budget of $50\% * 75\% * 75\% = 28.125\%$ and the best achievable QoR is roughly 41.5

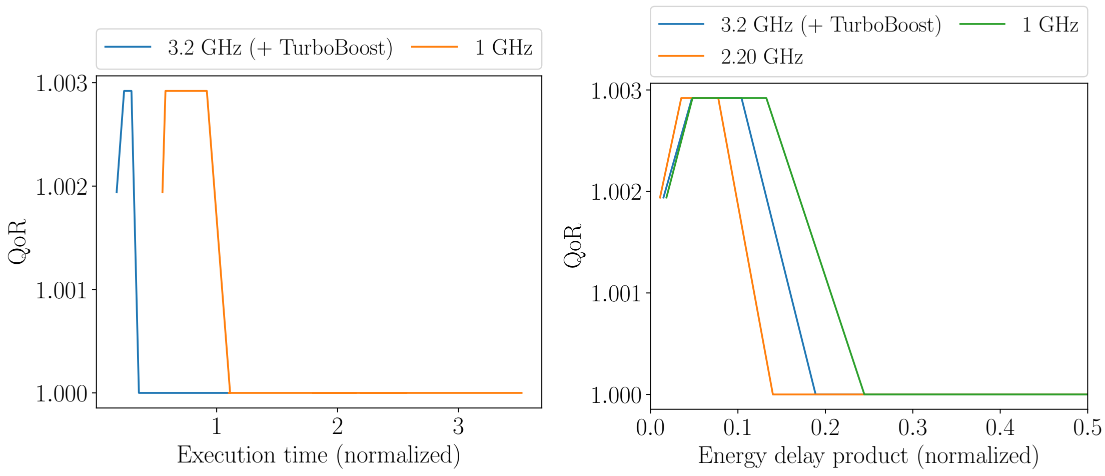
db. This clearly shows that the willingness of spending more time as done by using the ED^{2}P metric reduces the influence of considering the energy objective.

Considering non-AC hardware parameters for generating PPs. As hardware parameter, I use the number of used cores and the frequency of a core. The usage of



(a) Budget: execution time, Parameter: number of cores.

(b) Budget: PDP (consumed energy), Parameter: DVFS..



(c) Budget: execution time, Parameter: DVFS.

(d) Budget: EDP, Parameter: DVFS..

Figure 6.19: Influence of different conventional hardware parameters (non AC) on PPs for the contract-based 1NN task. The budget metric is also varied.

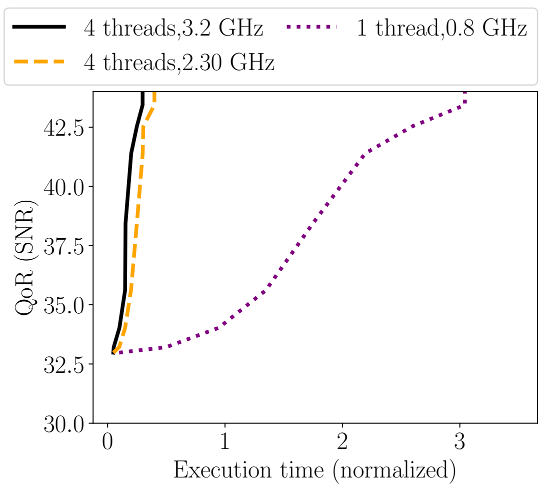
OpenMP allows a programmer to change the number of used cores, while the overhead for forking and joining a parallel section is within the introduced *pragma*. On the other hand, changing the frequency requires a library call. This call requires 51 µs for changing from the highest to the lowest frequency and 62 µs for the other direction on the Test System B. Getting the current frequency ranges from 13.5 µs to 17.5 µs.

Figure 6.19 shows the influence of two non-AC hardware parameter on the PPs generated for the contract-based 1NN classification using DTW as metric. I use the Test System C, which enables up to 48 hardware threads, to evaluate how exploiting parallelization effects the shape of the PP and show it in Figure 6.11b. As we can see in Figure 6.19a, increasing the number of hardware threads for the contract-based function results in a compression of the PP regarding the x-axis which represents the execution time. However, the ratio of the compression decreases with an increasing number of used cores and thus there is almost no further benefit for going from 16 to 32 cores.

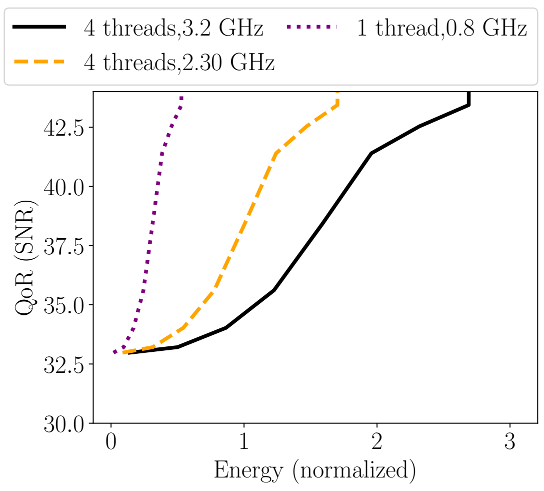
The purpose of Figures 6.19b to 6.19d is to show how the budget metric influences the decision about the parameter value that will be used during runtime. I exploit as hardware parameter DVFS, while running the contract-based 1NN on Test System B. This system offers the possibility to change the frequency for the core within the application. In the present case, the parameter is the current frequency of the core. Using the PDP metric (energy), we would decide to set the frequency of the core to 1 GHz for executing the contract-based function. Considering the execution time as budget, we would set the frequency to the highest possible value (3.2 GHz). By changing the budget metric to the EDP metric, we can consider the energy and the execution time together, while keeping the notation of PPs. For the EDP, the best configuration is 2.2 GHz.

So far, I only have considered a single hardware parameter and therefore a combination of hardware parameters is investigated now together with different budget definitions. As benchmark, I use the contract-based 2D convolution and run it on Test System B. The used conventional methods (non-AC hardware parameters) are DVFS (15 different settings) and the number of used cores (3 different settings). In total, the configuration space consists of 45 different knob settings. Depending on the budget metric, a certain configuration outperforms the other ones and therefore the entire PP of this configuration also poses the *merger*. In Figure 6.20, I only compare settings that pose the best one regarding one of the four budget metrics.

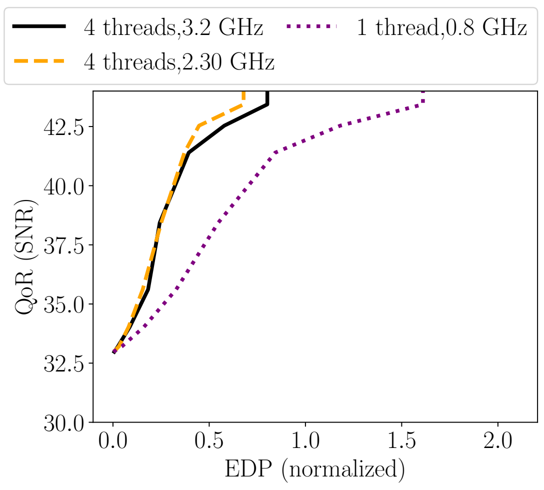
In case of the execution time, the best configuration is the one using four cores (4 threads) and the highest frequency (3.2 GHz). In terms of PDP, the opposite is the best configuration (1 thread + 0.8 GHz). The configuration with four hardware threads and a 2.3 GHz core frequency is the best configuration for the multiple objective case based on the EDP. Applying the ED^{2}P metric, there are two configurations equivalently good (4 threads + 3.2 GHz and 4 threads + 2.3 GHz). The results reveal that a high emphasis on the performance leads to a configuration with a high value per knob. On the other side, a configuration leading to a low energy consumption uses a small value per knob.



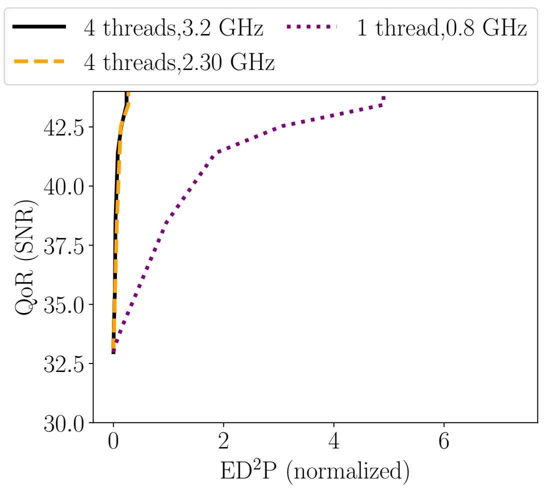
(a) Budget: Execution time.



(b) Budget: PDP.



(c) Budget: EDP.



(d) Budget: ED^{2}P.

Figure 6.20: Considering a combination of non-AC parameters for the contract-based 2D convolution.

Performance profiles represent a transparent view on approximation methods. As already mentioned, the hardware AC methods can be used by contract-based algorithms equally to non AC hardware parameters. I show this aspect according to the Richardson-Lucy deconvolution. The Richardson-Lucy deconvolution, especially the 2D convolutions, tolerates small approximations of floating-point operations. Therefore, I consider two different possibilities to approximate these operations. The first one is the conversion method presented in Chapter 4. The second method considers approximate main memories based on DRAM.

The conversion unit reduces the data that has to be transferred from a floating-point unit to the memory by using the conversion method as described in Chapter 4. As the conversion unit works on single data types, it is a representative for other approximate computing approaches working on single data type operands, such as processing units (multipliers, adders, etc.). The difference is that approximate processing units do not necessarily truncate the least significant bits. They also approximate the computation itself. There exist different designs for integer operations that vary in the average relative error, in the absolute error, or in the likelihood of an error. Furthermore, there exists a publicly available library that emulates these processing units [339]. However, the work regarding approximate floating-point units is limited and no real measurement of the performance is possible without a re-implementation of available methods [340, 341]. Therefore, I do not present results for approximate floating-point units explicitly.

An approximate DRAM is achievable by adapting the number of cycles between a refresh of the DRAM. This can lead to reading wrong data. I consider different probabilities for reading out wrong data, see Table 6.2. For the following test, I make a pessimistic assumption that every operation inside the *2D convolution*s that requires a pixel from the input image needs an access to an approximate main memory[11]. Note that a DRAM approximation method does not influence the bandwidth of the memory in general. Moreover, as the number of iterations is set, this method also does not influence the execution time. Approximate DRAM can lead to a significant reduction of the power required for refresh [8].

Figure 6.21 shows the PPs exploiting accuracy-aware parameters. As the Richardson-Lucy deconvolution mainly consists of *2D convolution*s, I assume that my conversion unit (CU) achieves a speedup of 3.3 (cf. Chapter 4). As QoR metric, I use the peak signal to noise ratio (Figure 6.21a) and one minus the error rate (Figure 6.21b). As we can see, the PP representation is able to capture the behavior of different accuracy-aware hardware methods. Thus, this representation easily allows us to compare different methods. Moreover, building a `merger` is important to adapt the approximation degree using the PP representation during runtime. In case of the Richardson-Lucy deconvolution, the CU-based design builds the `merger` (blue line). However, the results also reveal that it is

Table 6.2: Potential power savings by increasing the cycles between a refresh of the DRAM. The data is taken from [8].

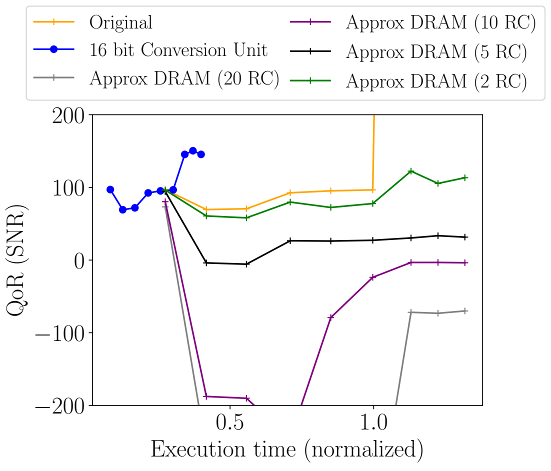
Refresh cycle(s) [$\times$]	**Error rate**	**Power savings**
2	2.6×10^{-7}	~23%
5	3.8×10^{-6}	~24%
10	2.0×10^{-5}	~24.3%
20	1.3×10^{-4}	~24.6%

[11] Approaches for realizing approximate main memory are already discussed in Sections 2.2 and 4.5.

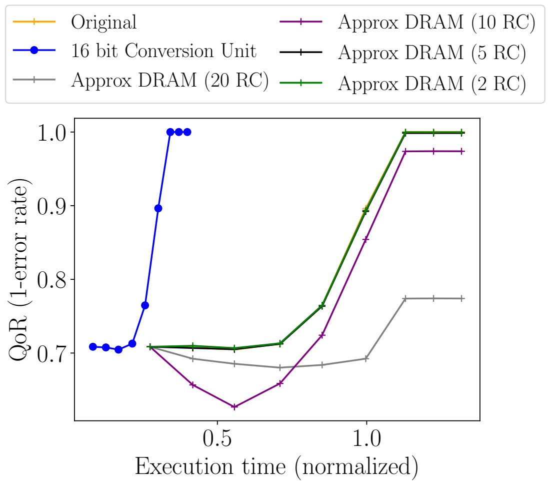
beneficial to exploit a slight (up to 2 RC, green line) approximation setting of the DRAM in order to reduce the energy consumption for refreshing the DRAM content.

For black box AC methods, I investigate loop perforation, memoization, and neural networks. The PP of such a method consists of few points in the QoR budget space. Results for loop perforation are already shown in Figures 6.9 and 6.10a. According to AxBench [303], the neural network AC approach outperforms loop perforation. The neural network based approach relies on hardware support for inferring the network in order to achieve an acceleration [15]. In the absence of such an accelerator, I use publicly available results for it [303, 15, 182]. However, the achieved QoR of different neural network topologies for certain applications can be determined by the available AxBench source code[12].

Figure 6.22 presents the resulting PPs of the AC black box methods. Results are presented for jpeg (see Figure 6.22a) and the jmeint (see Figure 6.22b). The results for fuzzy memoization are based on my LSH-based memoization method. The LSH-based versions differ in the number of used random projections (4,8,12,16). As we can see, the LSH-based method achieves reasonable QoRs for different configurations, however, the performance it too low to be beneficial. For the neural network approximation, only the result of a single configuration is published in the literature. This neural network configuration presents the best network regarding the achievable QoR. Therefore, I assume two further configurations to show what a PP would look like if we could choose between



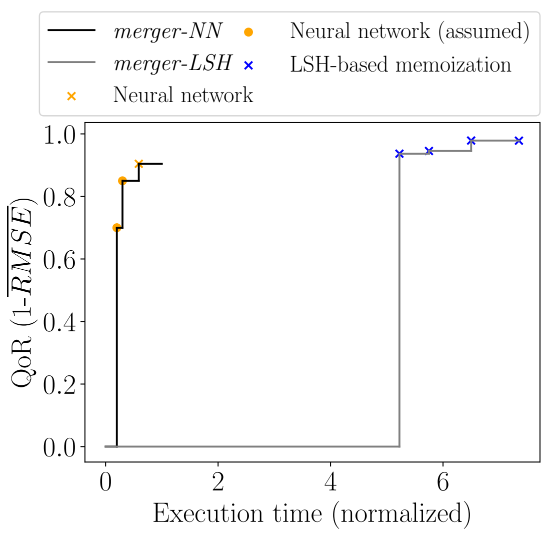
(a) QoR metric: SNR.



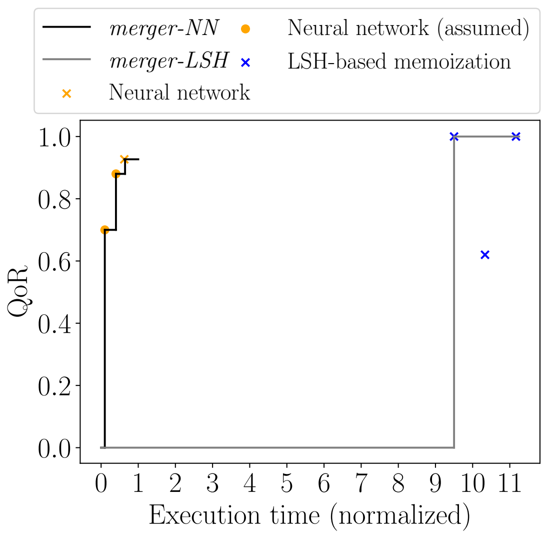
(b) QoR metric: 1-error rate.

Figure 6.21: Performance profiles for the Richardson-Lucy deconvolution using different QoR metrics and accuracy-aware methods (approximate DRAM and conversion unit). RC represents the number of cycles between a refresh.

[12]The code is available on `http://axbench.org/`.



(a) jpeg benchmark (QoR: average RMSE).



(b) jmeint benchmark (QoR metric: misclassification rate).

Figure 6.22: PPs for black box accuaracy-aware methods and different benchmarks.

multiple configurations. Furthermore, we can see for jmeint, how we filter out configurations that do not lie on the Pareto-optimal front. This is done according to the procedure presented in Section 6.4.3.

In each case, the PP for black box models presents a step function that represents the *merger*. This means that we can reach a certain QoR until we reach the point for another configuration. This knowledge can be exploited during runtime to control the AC black box methods.

6.6 Related Work

Memoization approaches have been considered at software level [342, 343, 14, 323] and hardware level [344, 178, 177, 345, 179, 171, 172, 346, 181]. Hardware approaches rely on special hardware extensions that are not given in common computing systems.

My focus is on software-only memoization approaches. There exist automatic ways to generate lookup tables during compile-time by varying the levels of quantization to reduce the required memory space [347] or by exploiting piecewise approximation [348]. These approaches only work for functions having up to two inputs.

Exact memoization approaches often used hash tables as memoization table. They can be used by an application without adapting the source core [342, 343]. This is achieved by a compiler-based approach or by adapting dynamically linked math functions. Compared to exact approaches, fuzzy memoization can increase the hit rate. Therefore, Paraprox uses a non-uniform quantization of inputs and concatenates the remaining bits of the inputs to generate an index to the memoization table [14]. This solution poses issues when the number of bits used per input or the amount of inputs increase and a quantification significantly leads to getting wrong results from the table. Then, the Paraprox approach would require a large table, which is inefficient, especially, if (most parts of) the table cannot be stored in the cache.

In order to compress the table, while also increasing the hit rate, Brandalero et al. apply hashing after a quantization [323]. The main drawback of this work is that they only show the potential of fuzzy memoization regarding the potential performance improvement. They mention that SHA1 could be a useful hash function, however, as my investigation reveals SHA1 is not a useful candidate. The introduced overhead of SHA1 is higher than using xxHash. Moreover, my results show that at software level such an approach is not beneficial. The authors say that accessing to the hash table shall require no more than 200 processor cycles but no solution is presented for that.

The concept of anytime algorithms was applied on image processing methods such as smoothing or edge detection [141]. As already mentioned, anytime algorithms can be divided into contract-based and interruptible algorithms. Compared to Miguel et al. [335], I argue that contract algorithms are more suitable than interruptible algorithms for AC.

Firstly, contract algorithms are often easier to implement and maintain [312]. Secondly, the approach of Miguel et al. requires that each interruptible task be assigned to an exclusive core to get an suitable solution. This hampers full utilization of a single core. Thirdly, keeping the cache hierarchy in mind, my contract-based approach has almost negligible overhead compared to [335]. Finally, since there exist kernels and algorithms that cannot be transferred into an anytime algorithm and the quality improvement is different for each anytime task, controlling interruptible tasks is difficult. I further elaborate on this point in Part III.

6.7 Summary

In this chapter, I introduced novel accuracy-aware software methods. Methods on the software layer have high potential to increase the efficiency of the execution. While the introduced LSH-based fuzzy memoization provides useful QoRs, the overhead for introducing it rules out any savings regarding execution time. However, the method can still be useful for functions that have higher execution time than the time required for finding a result in the memoization table.

In contrast to fuzzy memoization, the proposed innovative accuracy-aware method that exploits contract algorithms poses a general solution for exploiting AC at software layer. I presented how tasks can be transferred into contract-based tasks and how to control them according to a certain budget metric. Determining a task-specific ordering for processing input and output data items leads to a significant improvement of the performance profiles for different tasks.

Performance profiles describe the behavior of contract-based tasks in the QoR and budget space. Therefore, such profiles pose an intermediate layer between approximation methods and control methods as shown in this chapter. They represent a transparent view. This also enables us to consider a combination of different accuracy-aware methods and hardware parameters that influence the execution of contract-based and non-contract based tasks. This aspect of exploiting performance profiles as intermediate layer is more elaborated in Part III.

Finally, multiple objectives can be concurrently considered using performance profiles. This is achieved by changing the definition of the budget metric for a single objective to a metric that considers several objectives such as the energy delay product metrics. Using these metrics, we can consider three different objectives, QoR, execution time, and energy consumption for contract-based tasks.

CHAPTER
SEVEN

EVALUATION OF HIGH PRECISION ARITHMETIC

The three previous chapters consider innovative accuracy-aware methods at various layers focusing on approximation techniques. In this chapter, I consider a further aspect at the software layer. Instead of reducing the QoR, I present a framework that allows programmers to investigate different high precision arithmetic (HPA) libraries for their algorithms and applications. Hence, the content of this chapter tackles the problem of computational difficulties that can occur during an execution of an algorithm. Especially, algorithms used in scientific computing can suffer from computational errors.

Using the proposed framework, I evaluate the different libraries regarding the computational overhead and the improvement regarding the computational errors. Finally, the chapter concludes the extracted insights and discusses further aspects.

The main conclusion is that by using HPA the introduced overhead to the computation is not tolerable. Algorithmic-specific adaptations are required in order to improve the numerical stability and to reduce the usage for HPA. Hence, it is not suitable to consider the HPA framework together with the innovative *adaptive accuracy-aware approach* presented in Part III, since an algorithmic-specific solution is inevitable.

7.1 Introduction

The algorithms and applications considered in this thesis offer the potential to reduce the computational effort by exploiting approximation at different layers. In contrast, there exists a wide range of applications that require a certain degree of precision for internal computations in order to reach desired accuracy. IEEE-754 32 bit or 64 bit floating-point arithmetic is most of the time sufficient to satisfy these requirements. However, floating-point operations can pose computational difficulties such as cancellation or round-off errors. Researchers argue that analyzing the code regarding numerical stability is the most useful way to avoid numerical issues. This includes implementing the algorithm in the most stable way [208]. Therefore, programmers need a thorough understanding of numerical analysis. However, this is not the case for scientific programmers in general. On that account, extracting a more stable algorithm typically takes a lot of time and may cause other problems, for instance, with respect to parallelization.

Other researchers propose the usage of HPA. Especially, for the large growing body of applications where the precision of arithmetic operations provided by modern processors is not enough [207, 349]. These applications include numerical reproducibility, ill-conditioned linear systems, large summations, long-time simulations, probabilistic timing analysis [350] or optimization solvers [351]. The incorrect usage of data types or an inexpertly handling of floating-point operations can lead to catastrophic situations [352, 353]. Premature or delayed algorithm termination can also be a resulting issue.

Some compilers enable the usage of 128 bit floating-point operations or of double-double data types (~31 digit precision). There exists a wide range of HPA libraries that allow operator overloading to easily integrate HPA into existing source code [354, 355, 356]. Applying HPA data types only to relevant operations within an application reduces the overhead for HPA. This can be done automatically [16, 245]. There also exist coprocessor-based designs that provide HPA [214, 212, 215]. However, these designs are limited regarding supported precision and integration into existing source codes. Furthermore, the performance over software implementations is not necessarily higher.

In this chapter, I consider the influence of data type precision on an algorithm used in scientific computing, namely the Lanczos algorithm. This includes an evaluation of the statement that issues regarding computational errors are solvable by HPA [207]. Moreover, I show a comparison of different HPA libraries in this chapter. Evaluating these HPA libraries fills a missing point in the related literature, since there exists only a work considering interval data types [357]. Additionally, I evaluate an approach that is a candidate for reducing the overhead of HPA by exploiting the work of Kulisch et al. [217]. This includes a fine-grained coprocessor-based design. The contributions of the chapter are:

- A framework to integrate different HPA libraries into one algorithm.
- Evaluating the influence of data type precision, matrix dimension, and eigenvalue distribution on the Lanczos algorithm. This is important to see, whether HPA bypass the computational errors of standard floating-point operations.
- Comparison of HPA libraries regarding accuracy, efficiency, and execution time.
- The investigation of an exact scalar product for the Lanczos algorithm.
- Consideration of a fine-grained OpenCL-based exact scalar product.

7.2 Fundamentals and Mathematical Background

In this section, I present the mathematical backgrounds of the used algorithms. As I investigate the influence of data type precision with a simulation-based approach, I choose the Lanczos algorithm as a benchmark. This algorithm is known to be sensitive against numerical errors [208, 358]. It is important to differentiate between precision and accuracy. Here, precision means the level of exactness for a single value stored into a data type. Normally, the precision is represented according to the number of bits. The accuracy is a measure for quality of result. It provides a quantity of the deviation to the actual exact result.

7.2.1 Numerical Issues and Cause of Computational Errors

The usage of floating-point operations has different issues. First of all, not each real number can be exactly represented by the IEEE-754 standard. For example, this is the case for the number $\frac{1}{3}$. Hence, there can be a difference between the real number x and the representation of this number $\tilde{x}$ in the IEEE-754 double format. This round-off error can be indicated as absolute error E_{abs} or as relative error E_{rel}.

$$E_{abs}(\tilde{x}) = |x - \tilde{x}|, E_{rel}(\tilde{x}) = \frac{|x - \tilde{x}|}{|x|}$$

These equations can be generalized for vectors and matrices. Moreover, they are also true for results of an algorithm.

In general, there are four sources of errors introduced by floating-point arithmetic: data errors, representation errors, round-off errors, and truncation errors. Data errors are due to an inexact measurement or input values. Representation errors are already discussed above. Round-off errors are not only an issue for the representation of a real number, they also occur after floating-point operations, such as multiplication. Truncation errors occur, for instance, for infinite mathematical series, since at a certain point we have to approximate this series in the computer. These errors can accumulate after each operation, when applied to a series of operations or to an algorithm. The so-called cancellation error occurs, for instance, for subtraction. Assuming two floating-point numbers that are approximately equal and the last significant bits of the mantissa are generated due to rounding. Then, a subtraction of these numbers removes the most significant bits and only the result of the rounded, presumably incorrect bits remains. Hence, the cancellation phenomena likely removes correct bits in the mantissa, while keeping bits that can be affected by an accumulation of rounding errors. Another issue with floating-point arithmetic is absorption. Absorption describes the fact that adding or subtracting a small number with a huge number has no effect. For instance, the result of the formula $2^{32} + 2 - 2^{32} = 0$ instead of 2. Moreover, the associative and the distribution law are not valid for floating-point arithmetic as seen in the example above. Calculating $2^{32} - 2^{32} + 2$ yields to the correct result.

The aforementioned issues do not necessarily lead to critical errors of the final result [208]. Therefore, a numerical analysis is important. A numerical analysis consists of three important methods: considering the condition number, the numerical stability, and the consistency. The condition number represents the sensitivity of a mathematical problem to the inexactness of the input data. A condition number close to one represents a well-defined problem, where a huge number specifies an ill-conditioned problem. In contrast, the numerical stability is a matter of the implemented algorithm. Hence, the stability depends on the influence of the round-off errors to an algorithm. The consistency of an algorithm indicates, whether the given algorithm solves more or less the actual problem. More on numerical analysis can be found in [208].

7.2.2 Number Formats

This section briefly summarizes number formats for real or rationale numbers. Floating-point numbers are usually defined according to $\pm m \cdot b^{\pm e}$, where m is the significand, b the base, and e the exponent. Normally, the base is implicitly defined and is two. According to the IEEE-754 standard, a floating-point number consists of three parts: a sign bit, a biased exponent, and a fraction. The number of bits per part depends on the actual format, for instance, single or double precision. Combining floating-point values increases the precision and still uses the available hardware floating-point units. For instance, the double-double format uses a double for the MSBs and one for the LSBs. According to the IEEE-754 standard, we can specify number formats with higher biased exponent and fraction.

Fixed-point formats offer an alternative to floating-point arithmetic, in case that the data range is comparatively small. The makes it possible to build more efficient hardware units. Fixed-point formats can be specified using QX.Y convention, where X represents the integer part and Y the fractional part. As a benefit of fixed-points formats one can think of that the absolute error magnitude is equal for the entire representable data range. This is not the case for floating-point arithmetic. However, exactly multiplying two fixed-point numbers representing the entire data range of a double precision floating-point value requires more than 4000 bits [217].

An approach that takes rounding errors into account is the concept of interval arithmetic. Here, a number is represented as an interval $[a, b]$. Interval arithmetic ensures that the correct result of an operation is inside the returned interval, even if the result cannot be represented in the used data format. Hence, the result lies in the interval that is specified by two representable data values. A problem of interval arithmetic is that intervals can be growing significantly large. This contradicts useful results.

Further examples of number formats are continued fraction-based formats [359], logarithmic number systems [360] or residue number systems [361]. Though all of them have benefits in certain areas, they are not superior regarding precision compared to floating-point formats. Therefore, a detailed description is not given.

The universal number (*unum*) format is an approach to overcome round-off and representation errors caused by floating-point arithmetic [362]. It also prevents under- and overflows. There exist three *unum* formats: Type I, Type II, and Type III (*posit*). These formats are significantly different from each other. Type I is a variable precision data type, where the significand and exponent are variable. Two additional fields store the size of these parts. A bit specifies, whether the number is exactly represented or part of an open interval. This variable format does not store zeros at the end of the significand for exact numbers and only stores exact bits for inexact numbers. This shall avoid accumulation of rounding errors. The exponent can be adapted to the required bit length. However, this format needs a suitable memory management and hardware. This is important to keep the benefit of the reduced amount of data and thus leading to improvements regarding performance and energy consumption.

The focus of Type II is on custom number formats for applications that require ultra-low accuracy [218]. This Type shall reduce the effort for hardware designers and programmers. Type II maps signed integers to the *projective real number line*. Gustavson et al. argue that Type II leads to a software-defined number system. Since operations are performed by exploiting lookup tables, this approach only works for small precision (up to 20 bits). A lookup table theoretically makes it possible to perform an operation in a cycle. This could also be applied to half precision floating-point arithmetic.

Type I and Type II are not very hardware-friendly, thus Gustafson et al. propose Type III, which shall be more hardware-friendly [363]. Type III exploits the concept of tapered accuracy [364]. A Type III value has a fixed size and a fixed number of exponent bits. So-called regime bits realize the tapered accuracy. This enables values with small exponents to have more precision, while very small or high values have less precision. The regime together with the exponent forms a 2^k scale vector. The remaining bits form the significand or fraction, hence $x = (-1)^{sign} \cdot 2^k \cdot (1 + fraction)$. The further concept of *quires* (cf. Kulisch et. al [217]) performs dot products, sums, and other operations in a way that a rounding error can only occur after the operation.

All *unum* types are more or less just concepts and almost no hardware designs exist that evaluate these concepts. Especially, the integration into a processor is missing in the literature. Hence, considerations regarding performance, area, or energy consumption are just theoretical estimations. Software libraries are available that allows a programmer to use a simulation of *unum* types. There is also an ongoing discussion about the suitability of *unum*[1]. Therefore, according to the conceptual state and the lack of hardware, I do not further consider *unum* in this chapter.

[1]cf. the Great Debate at ARITH23

7.2.3 Lanczos Algorithm

The Lanczos algorithm is an iterative method that, for instance, supports the calculation of eigenvalues for a given squared matrix. This method is highly sensitive against numerical rounding errors and cancellations. Instead of the Lanczos algorithm, one can use the Householder or Givens method, but the computational effort is significantly higher for sparse matrices. Nowadays, the Lanczos algorithm is mostly seen as an efficient iterative method, while the other two approaches are direct reduction methods to calculate the eigenvalues [365]. Given a large sparse squared matrix, the Lanczos algorithm is an efficient method to support the calculation of a subset of the eigenvalues [366], i.e. determining some of the largest or smallest eigenvalues. Applications like PageRank [367], latent semantic indexing [368], or graph bisection using spectral methods [369] require the calculation of eigenvalues. Even the domain of condensed matter physics requires such information for certain applications [370].

The Lanczos algorithm creates a tri-diagonal matrix T_m that has up to m eigenvalues of a matrix $A \in \mathbb{R}^{n \times n}$, where $m \leq n$. The main diagonal is represented by the vector $\vec{\alpha}$ and the upper and lower diagonal by the vector $\vec{\beta}$.

$$T_m = \begin{pmatrix} \alpha_0 & \beta_0 & & & 0 \\ \beta_0 & \alpha_1 & \beta_1 & & \\ & \beta_1 & \ddots & \ddots & \\ & & \ddots & \ddots & \beta_{m-2} \\ 0 & & & \beta_{m-2} & \alpha_{m-1} \end{pmatrix}$$

Algorithm 7.1 shows the unmodified Lanczos algorithm in its most numerically stable variant [366]. The computational complexity of the unmodified Lanczos algorithm is $O(n^2m)$. The eigenvalues of T_m can be calculated by using QL or QR algorithm. The QL or QR algorithm are iterative methods that apply a QL or QR decomposition in each step, respectively. This leads to a convergence of the diagonal elements of the triangular matrix, L or R, to the eigenvalues of T_m. Since T_m is a tri-diagonal matrix, the computational complexity of the QL algorithm is reduced to $O(n)$ instead of $O(n^3)$. The critical point regarding computational errors is the orthogonality step (line 6). Here, standard floating-point arithmetic leads to numerical instability. To improve these issues, the modified variant shown in Algorithm 7.2 adapts the orthogonalization step. It uses all of the calculated vectors q_i instead of using the last two q_i (see line 7) [358]. This introduces an additional overhead, because significantly more dot products have to be performed per iteration. The amount of dot products increases over the iterations, since the number of q_i increases. Line 7 represents the modified Gram-Schmidt process. The modified process is more numerically stable.

Algorithm 7.1: Unmodified symmetric Lanczos algorithm.

```
Input: $m$, Matrix $A$, random start vector $\boldsymbol{x}$, where $\boldsymbol{x}$ is not Eigenvector of $A$
Output: Vectors $\vec{\alpha}$ and $\vec{\beta}$
1  $\beta_{-1} \leftarrow 0$;
2  $\boldsymbol{q_{-1}} \leftarrow 0$;
3  $\boldsymbol{q_0} \leftarrow \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|}$;
4  for $i \leftarrow 0$ to $m-2$ do
5      $\alpha_i \leftarrow \langle A\boldsymbol{q_i}, \boldsymbol{q_i}\rangle$;
6      $\boldsymbol{r_i} \leftarrow A\boldsymbol{q_i} - \alpha_i\boldsymbol{q_i} - \beta_{i-1}\boldsymbol{q_{i-1}}$;   // Orthogonalization against $\boldsymbol{q_i}$, $\boldsymbol{q_{i-1}}$
7      $\beta_i \leftarrow \|\boldsymbol{r_i}\|$;
8      if $\|\boldsymbol{r_i}\| < \epsilon \cdot \|A\boldsymbol{q_i}\|$ then
9          break;   // Break, if orthogonalization is violated
10     end
11     $\boldsymbol{q_{i+1}} \leftarrow \frac{\boldsymbol{r_i}}{\beta_i}$;   // Normalization
12 end
13 $\alpha_{m-1} \leftarrow \langle A\boldsymbol{q_{m-1}}, \boldsymbol{q_{m-1}}\rangle$;
14 return $(\vec{\alpha}, \vec{\beta})$;
```

Algorithm 7.2: Modified symmetric Lanczos algorithm.

```
Input: $m$, Matrix $A$, random start vector $\boldsymbol{x}$, where $\boldsymbol{x}$ is not Eigenvector of $A$
Output: Vectors $\vec{\alpha}$ and $\vec{\beta}$
1  $\beta_{-1} \leftarrow 0$;
2  $\boldsymbol{q_{-1}} \leftarrow 0$;
3  $\boldsymbol{q_0} \leftarrow \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|}$;
4  $\alpha_0 \leftarrow \langle A\boldsymbol{q_0}, \boldsymbol{q_0}\rangle$;   // Different to unmodified version
5  $\beta_0 \leftarrow 0$;   // Different to unmodified version
6  for $i \leftarrow 0$ to $m-2$ do
7      $\boldsymbol{r_i} \leftarrow A\boldsymbol{q_i} - \sum_{j=0}^{i-1}\langle A\boldsymbol{q_i}, \boldsymbol{q_j}\rangle \cdot \boldsymbol{q_j}$;   // Orthogonalization against $\boldsymbol{q_1}$,..., $\boldsymbol{q_i}$,
         replaces lines 7 to 9 of unmodified version
8      if $\|\boldsymbol{r_i}\| < \epsilon \cdot \|A\boldsymbol{q_i}\|$ then
9          break;   // Break, if orthogonalization is violated
10     end
11     $\boldsymbol{q_{i+1}} \leftarrow \frac{\boldsymbol{r_i}}{\|\boldsymbol{r_i}\|}$;   // Normalization, replaces line 11 of unmodified version
12     $\alpha_{i+1} \leftarrow \langle A\boldsymbol{q_{i+1}}, \boldsymbol{q_{i+1}}\rangle$;   // Different to unmodified version
13     $\beta_{i+1} \leftarrow \langle A\boldsymbol{q_{i+1}}, \boldsymbol{q_i}\rangle$;   // Different to unmodified version
14 end
15 $\alpha_{m-1} \leftarrow \langle A\boldsymbol{q_{m-1}}, \boldsymbol{q_{m-1}}\rangle$;
16 return $(\vec{\alpha}, \vec{\beta})$;
```

7.2.4 Synthetic Input Data

The stability of the Lanczos algorithm depends amongst others on the input data, hence the squared matrix A. The eigenvalues of A have a big influence. Especially, the distribution of eigenvalues has a high impact. In general, the algorithm behaves more stable for exponential distributions. Unfortunately, the distribution of eigenvalues is most of the time not known beforehand. Therefore, the matrix A is artificially constructed for the benchmark to investigate different distributions. A matrix A is constructed in a way that one can select the distribution of the eigenvalues [358]. Moreover, as we know the correct eigenvalues, we can compare the calculated eigenvalues with the correct values.

For this purpose, the eigenvalues are calculated through the following equation

$$\lambda_i = c_1 \cdot e^{-c_2 \cdot (i-1)^{\alpha}} \tag{7.1}$$

with $c_1, c_2 > 0, \alpha \in (0, 1]$. c_1 is a scale factor of the eigenvalues, but has no effect on the distribution. c_2 influences the distance between the eigenvalues of A and α changes the distance and the distribution and hence the convergence behavior of the Lanczos algorithm. Note that these eigenvalues are all positive and real numbers, we construct a square $A \in \mathbb{R}^{n\times n}$ with the help of singular value decomposition. This decomposition is defined by

$$A = V \cdot D \cdot U,$$

where $V \in \mathbb{R}^{n\times n}$ is an orthogonal normalized matrix and represents the eigenvectors of A. $U \in \mathbb{R}^{n\times n}$ is a transpose of an orthogonal matrix. $D \in \mathbb{R}^{n\times n}$ is a diagonal matrix that includes the eigenvalues λ_i of A in case of positive eigenvalues. Choosing $U = V^T$ simplifies the computation. To construct the needed orthogonal normalized matrix V, we first assemble an orthogonal matrix $T_n \in \mathbb{R}^{n\times n}$ with the help of Tschebyscheff nodes and polynomials, because A is just defined over the dimension n. These Tschebyscheff nodes and polynomials are a system of orthogonal functions $i, j \in \{0, 1, \ldots, n-1\}$ defined by

$$t_i(x_j) = \cos(i \cdot \arccos(x_j)), \qquad x_j = \cos\left(\frac{j+\frac{1}{2}}{n} \cdot \pi\right).$$

T_n is now assembled by $(T_n)_{ij} = t_i(x_j)$. To normalize this matrix, we can define a diagonal matrix Δ_n that includes the squares of the reciprocal of the product $T_n^T \cdot T_n$. Then,

$$A = V \cdot D \cdot V^T = (T_n \cdot \Delta_n) \cdot D \cdot (T_n \cdot \Delta_n)^T.$$

Algorithm 7.3 shows the synthetic construction of the input matrix A according to the scheme described above.

7.3 Overview of High Precision Arithmetic Libraries

This section presents the different HPA libraries used for the evaluation. Besides these libraries, I consider double (53 bit significand) and long double (64 bit significand) as hardware-supported data types according to the IEEE 754-2008 standard [371]. I set de-normalized numbers to zero by using Intel intrinsics. I use HPA data types provided by `MPFR` [356], `MPIR` [372], `ARPREC` [354], and `C-XSC` [355].

Algorithm 7.3: Create squared matrix $A \in \mathbb{R}^{n\times n}$ with eigenvalues $\lambda_i = c_1 \cdot e^{-c_2\cdot(k-1)^\alpha}$

```
Input: Dimension $n$ of output matrix $A$, Inputs $c_1$,$c_2$, and $\alpha$ for Equation (7.1)
Output: Matrix $A$
for $i \leftarrow 0$ to $n-1$ do
    $\lambda_i \leftarrow c_1 \cdot e^{-c_2\cdot(k-1)^\alpha}$;
    for $j \leftarrow 0$ to $n-1$ do
        $(T_n)_{ij} \leftarrow \cos\left(j \cdot \frac{i+\frac{1}{2}}{n} \cdot \pi\right)$;
        if $i \neq j$ then
            $(\Delta_n)_{ij} \leftarrow 0$, $(D)_{ij} \leftarrow 0$;
        else
            $(D)_{ij} \leftarrow \lambda_i$;
            if $i = 0$ then
                $(\Delta_n)_{ij} \leftarrow \sqrt{\frac{1}{n}}$;
            else
                $(\Delta_n)_{ij} \leftarrow \sqrt{\frac{2}{n}}$;
    end
end
$V \leftarrow T_n \cdot \Delta_n$;
$A \leftarrow V \cdot D \cdot V^T$;
return $(A, \lambda)$;
```

MPFR is based on GNU Multiple Precision (GMP) [373] and supports mathematical standard functions of the C programming language standard C99. The maximum precision depends on the available memory space and data types are stored in floating-point representation. It supports IEEE-754 rounding and special cases such as not-a-number and infinity, but does not provide de-normalized numbers. Internally, numbers are treated as a floating-point values, where the size of the exponent and significand can be specified by the programmer. Performance critical parts are written in assembly. For the evaluation, I use a C++ interface for MPFR [374][2]. MPIR is similar to MPFR, but it is optimized for manycore architectures and graphical processor units[3]. ARPREC[4] represents a number A as an array a of doubles, where a_0 represents the number of elements in the array, a_1 the sign (0 or 1), a_2 the number of elements representing the mantissa, a_3 the exponent, and $a_4\ to\ a_{n+4}$ are the n blocks of the mantissa. Each block has a length of 48 bits. Hence, a represents the number

$$A = (-1)^{a_1} \cdot 2^{48\cdot a_3} \cdot \sum_{k=0}^{n-1} a_{k+4} \cdot 2^{-48\cdot k}, n = a_2.$$

[2]The MPFR version 3.1.3, the GMP version 6.0.0a, and the interface version 3.6.2 are used.

[3]The MPIR version 2.7.0 is used.

[4]The ARPREC version 2.2.17 is used.

ARPREC has optimized routines for precisions larger than 1000 bit, for instance, a FFT-based multiplication. Internally, ARPREC exploits fused and multiply add operations.

C-XSC[5] is a library for scientific applications and provides two high precision arithmetic data types, *dot precision* and *l_real*. It provides algorithms such as FFT, solving linear equations, or zero calculation of a function. *Dot precision* is a fixed-point format supposed for an exact product of double types required by an exact scalar product. This data type is based on the approach of Kulisch et al. [217]. The data type *l_real* represents a number as summation of doubles (staggered arithmetic). The programmer can choose the amount of doubles.

I have considered more libraries, but it was not possible to integrate them into the framework. For instance, interval arithmetic data types pose problems regarding dividing by zero. The reason is that the result of a re-orthogonalization can be close to zero. The used interval data type of the C-XSC library includes zero for the norm in case the boundaries are specified by 64-bit floating-point values. A presumable solution would be to exploit another library or data type.

7.4 Integration of High Precision Arithmetic Libraries

To investigate the influence of data type precision, I consider Algorithms 7.1 and 7.2. These algorithms are well suited due to their sensitivity against computational errors. It is important for such an investigation that the input data itself introduces no errors, for instance, caused by a representation error. The same applies to the output data, since the outputs is used to determined the achieved accuracy of different libraries. This comparison should not be subjected to any errors. Therefore, I use higher precision for the input data and the remaining steps than inside the Lanczos algorithm.

For a fair comparison of libraries, I remove the exit condition of Algorithm 7.1 (lines 8 and 9). Hence, the algorithm executes n iterations. The reason for this change is that the usage of the exit condition can significantly vary between different HPA data types thus thus makes the comparison very difficult. This does not hold for Algorithm 7.2. Since the Lanczos algorithm only creates a matrix that has m eigenvalues identical to the input matrix, I use the QL method to determine the eigenvalues on the resulting tri-diagonal matrix. As discussed above, the QL algorithm uses the same precision than the input data, hence higher precision compared to the Lanczos algorithm.

Many HPA libraries are available for C/C++, especially, the ones described in Section 7.3. Therefore, a C++-based framework using templates is implemented that easily integrates different HPA libraries. This template-based method allows a programmer to easily select and adapt the HPA data type during design time. On that account, the programmer can select the data type of a HPA library and the desired size. The frame-

[5]The C-XSC version 2.5.4 is used.

work includes algorithm 7.3 to generate a input matrix A. Moreover, it contains an algorithm that determines the accuracy of the results. This algorithm compares the resulting eigenvalues with the correct ones. The algorithm gets as input correct eigenvalues λ_k and calculated μ_i eigenvalues. A parameter ϵ represents the relative allowed deviation of eigenvalues. If the actual relative deviation of a correct eigenvalue and a calculated eigenvalue is smaller than ϵ, the calculated eigenvalue is considered as corresponding eigenvalue. To this end, the evaluation algorithm returns the largest absolute error Δ_{max} between two eigenvalues, the largest relative error δ_{max}, the number of calculated eigenvalues that are correct η, and the number of collisions κ. κ counts the number of calculated eigenvalues that occur several times. The algorithm exploits the fact that according to Algorithm 7.3 each eigenvalue only occurs once. For instance, $\lambda_0 = 1.0$, $\mu_0 = 0.998$, $\mu_1 = 0.999$, and $\epsilon = 0.001$, then the algorithm considers μ_1 as correct eigenvalue. Everything within the framework can easily be adapted in order to plug in other algorithms and perform an evaluation for it.

7.5 Evaluating the Influence of Data Type Precision

In this section, I investigate the influence of data type precision on the Lanczos algorithm. I describe the setup and discuss different evaluation features.

7.5.1 Evaluation Setup

The test system includes an AMD Opteron 2400 MHz and 16 GB main memory. The processor does not support fused multiply and add operations. To generate the input matrix A according to Algorithm 7.3, we assume $c_1 = c_2 = 1.0$ for Equation (7.1) and all tests. A starting vector $\vec{x}$ is randomly generated and tested whether it is an eigenvector of A. The vector $\vec{x}$ is constant during all benchmark tests for a certain matrix A. The Eigen library [375] is deployed for matrix and vector operations. This library performs all operations with HPA. The desired HPA data type is specified by a template instantiation. An eigenvalue is considered as correct if the relative deviation ϵ is less than 0.001.

7.5.2 Influence of the Precision

As a first test, I investigate the influence of the data type precision on the results of the modified and unmodified Lanczos algorithm (Algorithms 7.1 and 7.2). This allows us to draw conclusions about the stability of the both methods. Therefore, the precision of a MPFR data type is varied inside the Lanczos algorithm from 64 bit to 512 bit. The number of bits essentially specifies the size of significand. Figures 7.1 and 7.2 show the number of correctly calculated eigenvalues for different αs and thus for different eigenvalue distributions.

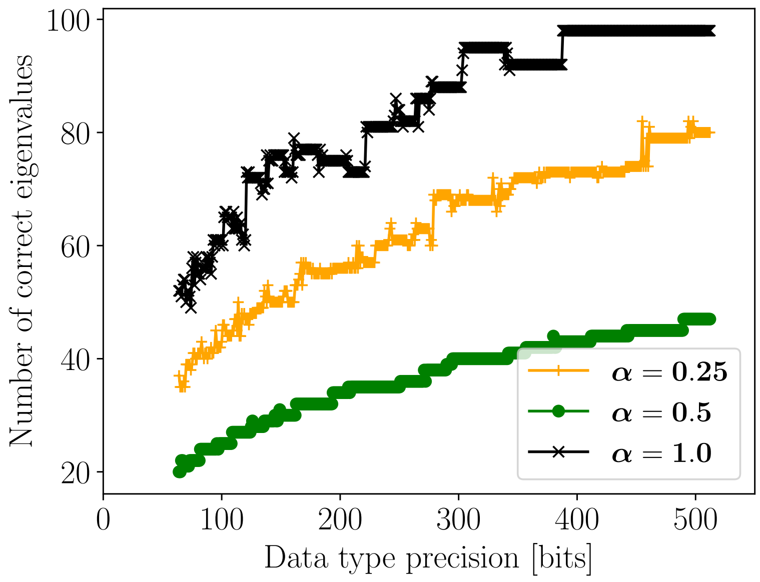


Figure 7.1: Influence of the internal data type precision regarding number of calculated eigenvalues using the unmodified (Algorithm 7.1) Lanczos algorithm.

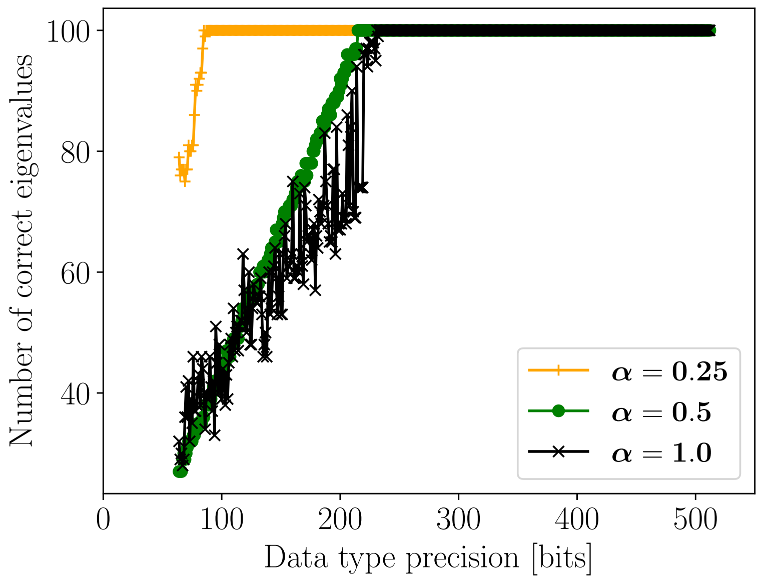


Figure 7.2: Influence of the internal data type precision regarding number of calculated eigenvalues using the modified (Algorithm 7.2) Lanczos algorithm.

For the Algorithm 7.1, the curves of $\alpha = 0.25$ and $\alpha = 0.5$ have local variations, but the trend is that HPA leads to more correct eigenvalues and fewer variations. These variations are higher for less exponential distributions of eigenvalues (smaller α) and likely caused by rounding errors. The influence of cancellations increases for larger exponential distributions. This leads to calculating the same eigenvalue several times (more collisions). On that account, significantly less eigenvalues are correctly calculated by the Algorithm 7.1. In addition, higher precision has less influence for larger α and so slowly improves the stability. At a certain precision point, all 100 eigenvalues can be correctly calculated. For instance, the $\alpha = 0.25$ curve is stable at 400 bits of data type precision. However, this would introduce a large computational overhead for larger α.

In case of the modified algorithm, we can see variations for all α, but they are quite high for $\alpha = 0.25$ (cf. Figure 7.2). The general trend is that the computation is more stable on less precision compared to the unmodified version, hence it leads to correctly getting all eigenvalues. Interestingly, this point is reached fastest for $\alpha = 0.5$ at roughly 128 bits. This is presumably due to the fact that the combination of convergence and influence of cancellation is best. Unfortunately, it is not known beforehand, where this point lies. This knowledge would help to adapt the HPA in the right way in advance of the computation. Theoretically, the convergence of $\alpha = 1.0$ is the best. To sum up, the modified version significantly benefits from HPA even for small enlargements of the number of precision bits.

7.5.3 Influence of the Matrix Dimension

Next, I investigate the influence of the matrix dimension in combination with HPA. I set $\alpha = 0.5$ and use MPFR data types. Figure 7.3 shows the number of eigenvalues collisions that occur for different data types and matrix dimensions n. Such collisions only occur for Algorithm 7.1 as discussed. The algorithm does not suffer from collisions for small matrices up to a dimension of roughly 30. From that point on, drastic computation errors occur that negatively impact the stability of the algorithm. To overcome such errors, HPA is beneficial, but only with a significant number of precision bits.

Then, I consider the number of correctly calculated eigenvalues, see Figure 7.4. Each of the modified versions (dashed lines) correctly calculate all eigenvalues. This agrees with the result of the previous section. Note that all lines (dashed lines) overlap in Figure 7.4 and therefore cannot be seen. In contrast, the unmodified algorithm (solid lines) significantly benefit from a higher precision using HPA.

Besides the accuracy, I measure the required iterations and the execution time per correct eigenvalue. The required iterations and the required execution time per correct eigenvalues are calculated by dividing the number of iterations or the execution time through the number of correct eigenvalues, respectively. Booth metrics can be seen as an efficiency measure. The number of iterations per eigenvalue shown in Figure 7.5 is always one for the modified version (dashed lines), since 128-bit is already sufficient to avoid computation errors that impact the result. We can see that the required number

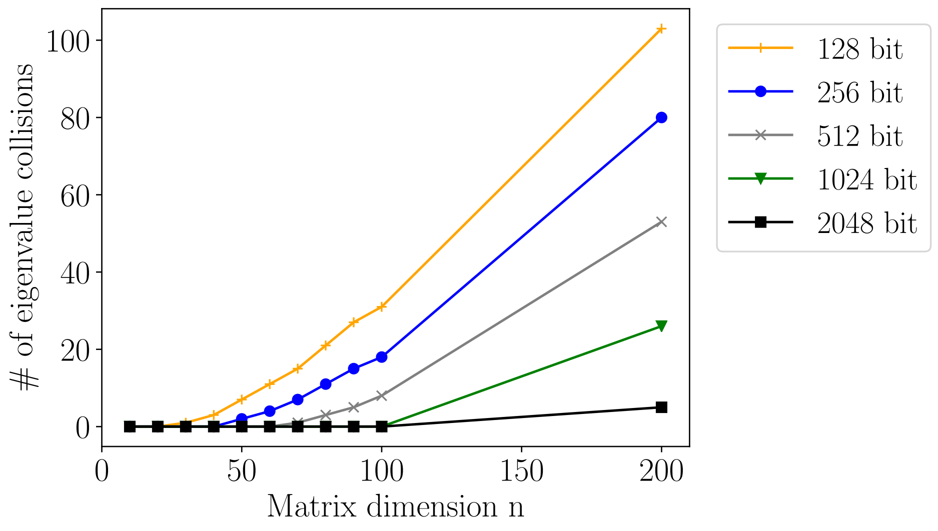


Figure 7.3: Number of eigenvalue collisions depending on the matrix dimension.

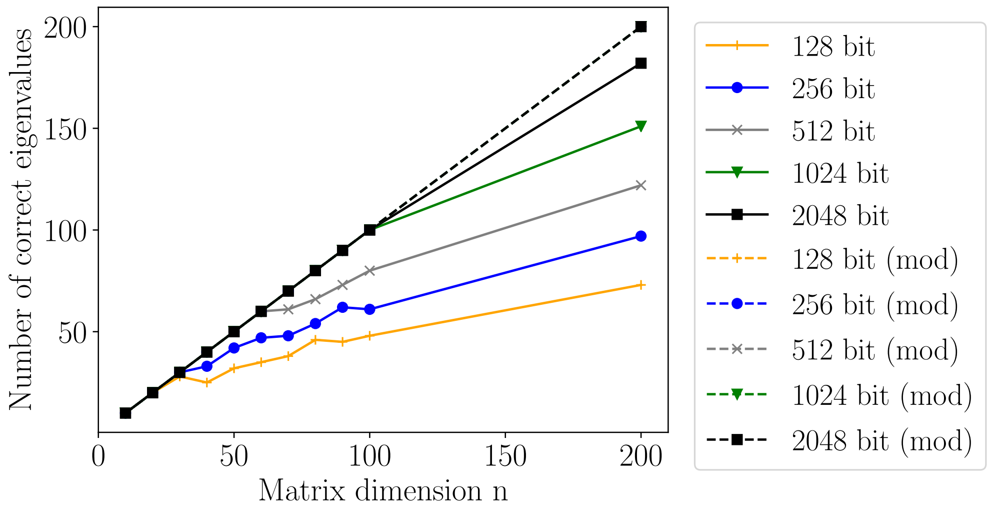


Figure 7.4: Number of correct eigenvalues for different dimensions and data types.

of iterations to get a correct eigenvalue decreases by exploiting higher precision for the unmodified version (black solid line). As we can see in Figure 7.6 the highest efficiency regarding the execution time and larger matrix dimension is provided by the modified version using a 128 bit data type (orange dashed line). In general, a higher precision is less efficient, especially, for the modified version. The unmodified algorithm using 512 bit data types performs best (gray solid line) for a matrix dimension of 100. The raw execution time for the modified version is roughly twice as high compared to unmodified

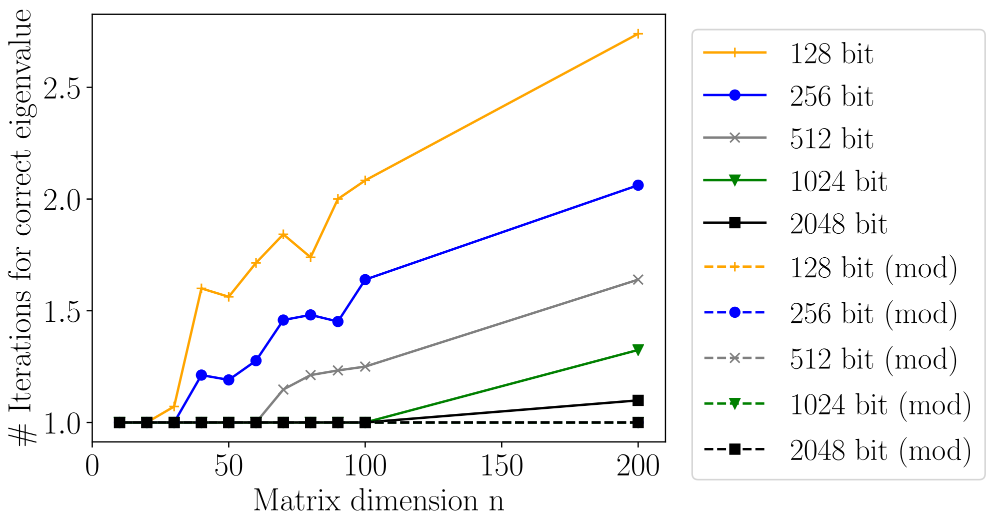


Figure 7.5: Number of required iterations per correct eigenvalue.

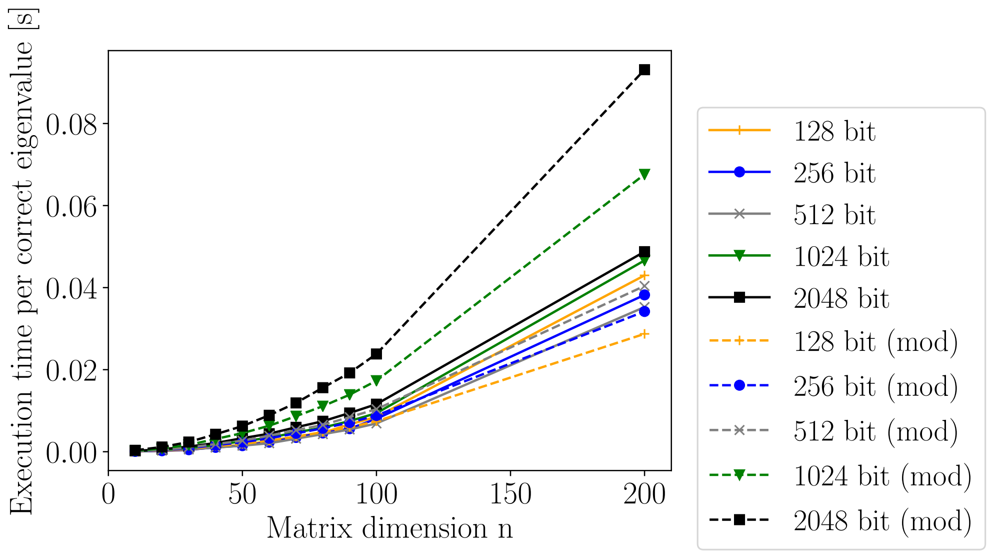


Figure 7.6: Execution time consumed for a correct eigenvalue.

version using the same data type precision (cf. Figure 7.7, compare the solid and the corresponding dashed line per color). However, the unmodified version often calculates wrong results as we have investigated above. Therefore, one can draw the conclusion that the modified version is more efficient in general, especially, for $\alpha = 0.5$. Other eigenvalue distributions could possibly benefit from the unmodified version in cases, where the modified version requires more precision to reach a stable point.

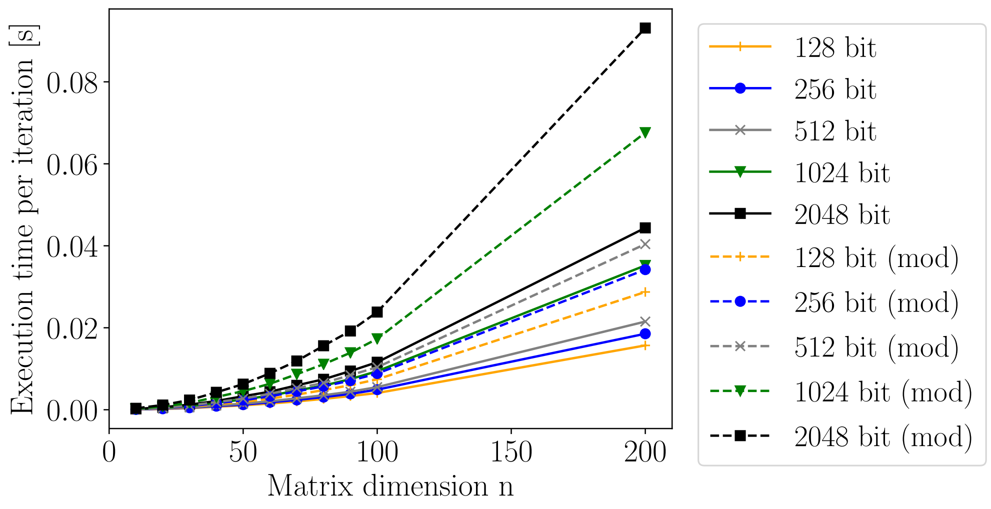


Figure 7.7: Entire execution time of the algorithms.

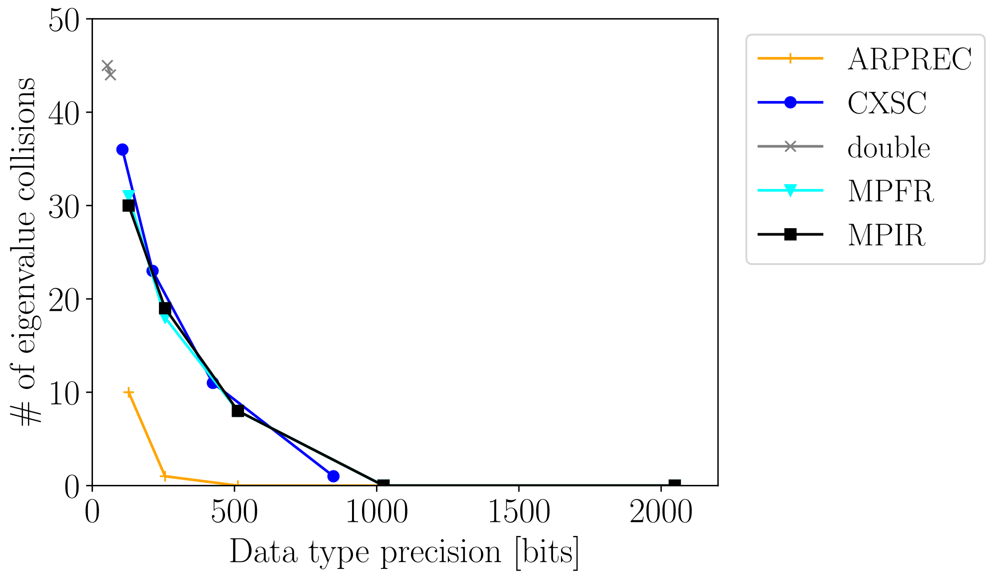


Figure 7.8: Number of eigenvalue collisions. Input matrix is generated using $\alpha = 0.5$.

7.5.4 Comparison of Different Libraries

I compare HPA libraries using input matrices of size 100×100, while $\alpha = 0.25$ or $\alpha = 0.5$, respectively. Note that the generated matrix has 100 different eigenvalues. Figures 7.8 and 7.9 show the number of collisions. The Figures 7.10 and 7.11 show the number of correctly calculated eigenvalues.

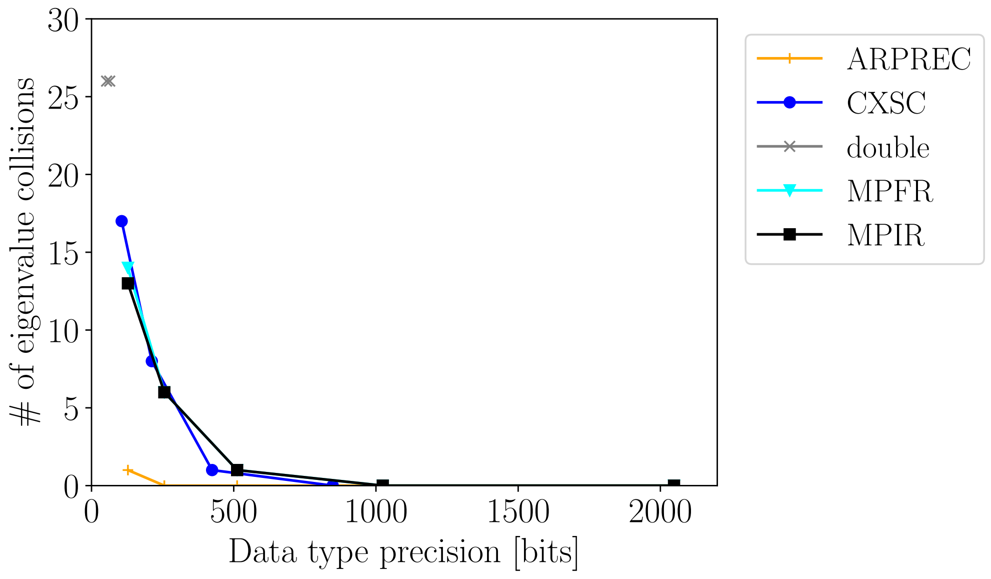


Figure 7.9: Number of eigenvalue collisions. Input matrix is generated using $\alpha = 0.25$.

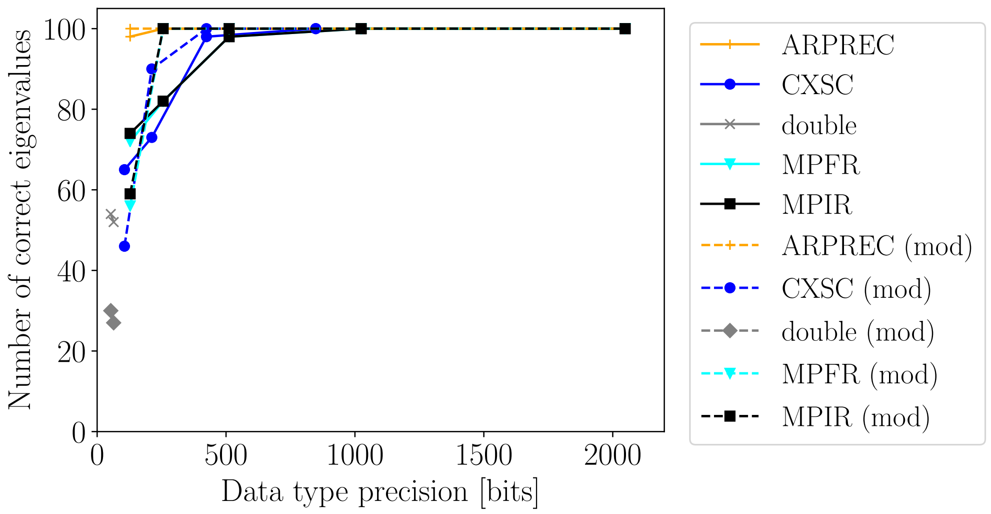


Figure 7.10: Number of correct eigenvalues. Input matrix is generated using $\alpha = 0.25$.

All libraries show a similar behavior, except ARPREC (orange lines). Please remember that the modified version correctly calculates all eigenvalues when the precision is above 128 bit. ARPREC offers a better accuracy for smaller bit widths compared to the other libraries. Therefore, ARPREC shows less collisions. The number of collisions decreases with smaller α. In addition, we can see that double or long double is not sufficient to calculate all eigenvalues (gray lines). This is also the case for the modified version.

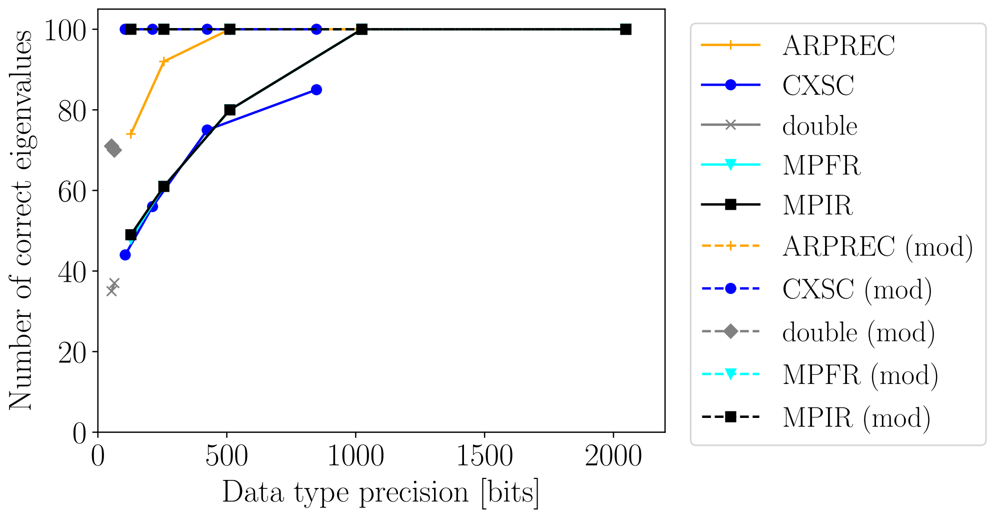


Figure 7.11: Number of correct eigenvalues. Input matrix is generated using $\alpha = 0.5$.

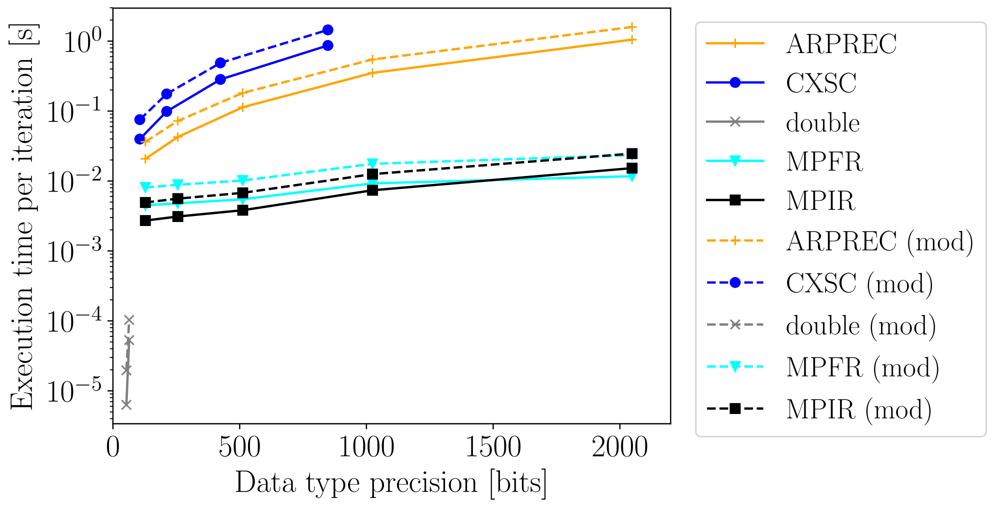


Figure 7.12: Average execution time per iteration. Input matrix is generated using $\alpha = 0.5$

The unmodified algorithm earlier reaches the optimal point (0 collisions or 100 correct eigenvalues) for $\alpha = 0.25$ than for $\alpha = 0.5$. The modified versions (dashed lines) require a minimal precision of roughly 220 bits to reach the optimum for $\alpha = 0.25$. ARPREC (orange line) performs better than the other libraries regarding the accuracy. In contrast, 128 bit is enough for all libraries and $\alpha = 0.5$ using the modified version. ARPREC reaches the optimum for the unmodified and modified version quite early.

Figures 7.12 and 7.13 present the execution time (in seconds) of a single iteration of

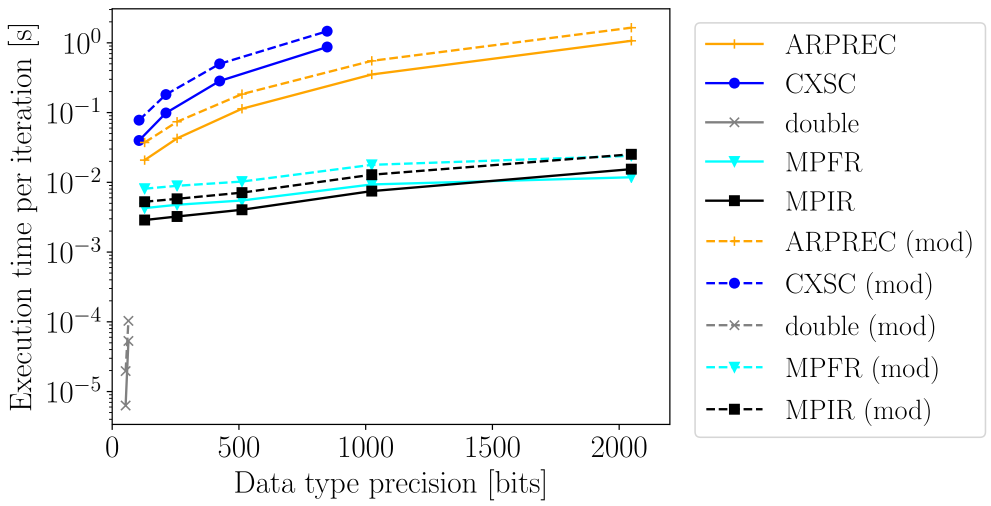


Figure 7.13: Average execution time per iteration. Input matrix is generated using $\alpha = 0.25$.

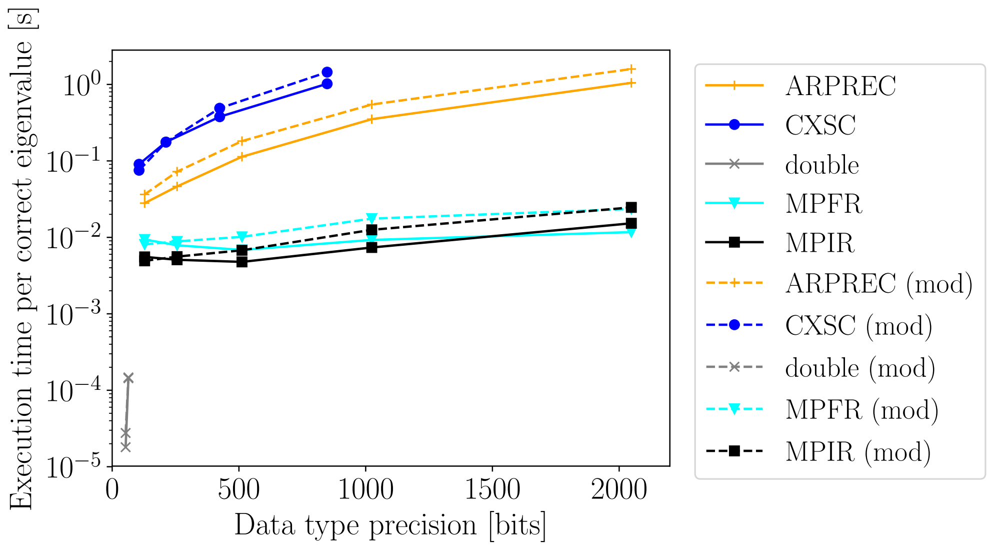


Figure 7.14: Average execution time per correct eigenvalue. Input matrix is generated using $\alpha = 0.5$.

the Lanczos algorithm using different HPA libraries and eigenvalue distributions. Note that the y-axes have logarithmic scales. We can see in the figures that the modified version always requires twice as long as the unmodified version (compare the dashed line with the corresponding solid line in the same color).

The overhead of `MPFR` and `MPIR` is significantly lower compared to the other two HPA

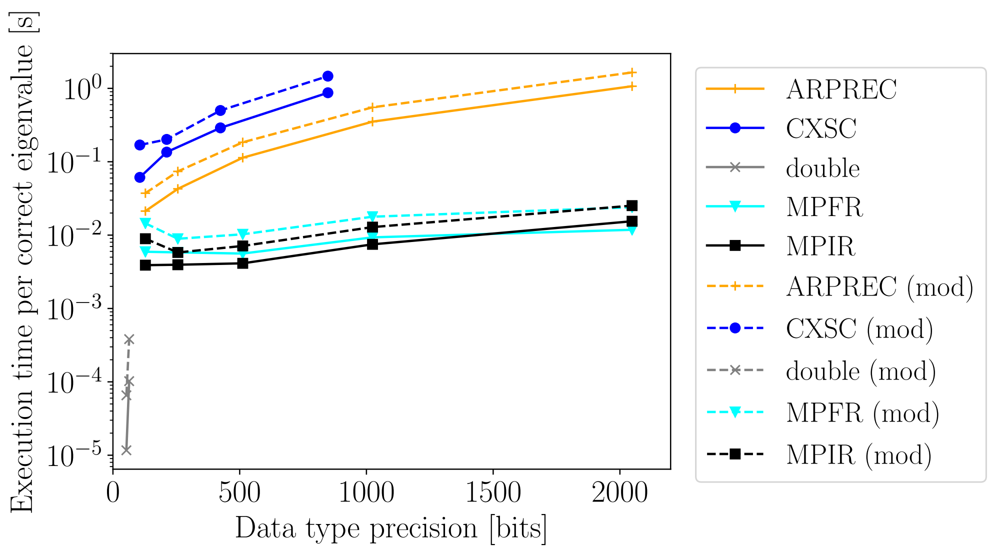


Figure 7.15: Average execution time per correct eigenvalue. Input matrix is generated using $\alpha = 0.25$.

libraries. ARPREC is three to ten times slower than MPFR and CSXC is about three to four times slower than ARPREC. The overhead of MPFR and MPIR does not significantly increase with number of precision bits. ARPREC and CSXC have an enormous increase regarding the overhead for higher bit widths. MPIR is two times faster until 1024 bit compared to MPFR. However, this consideration does not include the accuracy of eigenvalues. Therefore, Figures 7.14 and 7.15 shows the efficiency of the executions.

Please remember that the efficiency is the entire execution time divided by the number of correctly calculated eigenvalues. Despite the intention that the modified version is less efficient than the unmodified version, this is not really the case. Since the modified version requires less precision to calculate all eigenvalues of the matrix, we can see that for 128 bits the modified version using MPIR is the most efficient library. Higher precision just increases the overhead for the modified version, but this higher precision is not necessary.

7.5.5 Findings

One of the initial question is, whether HPA, in combination with an unstable algorithm, can outperform a modified algorithm using standard floating-point operations (double). Looking at the accuracy of the results, HPA leads to more accurate results. The modified algorithm using double calculates 30 and 71 correct eigenvalues for $\alpha = 0.25$ and $\alpha = 0.25$, respectively. HPA in combination with the unmodified version calculates more correct eigenvalues using 128 and 512 bit, respectively. However, HPA introduces an overhead of 142$\times$ and 194$\times$ in terms of execution time to achieve the higher accuracy.

The results indicate that even the modified algorithm requires a higher precision than supported by current processors. Considering the raw execution time, MPIR is until 1024 bits roughly twice as fast compared to MPFR. The raw performance is almost the same above 1024 bit. In general, the modified variant is twice as slow as the unmodified algorithm, but has the higher efficiency. The benefit of HPA is significantly higher for the unmodified algorithm. The rationale is the low numerical stability. In total, it is clear that the modified algorithm is more efficient than the unmodified algorithm.

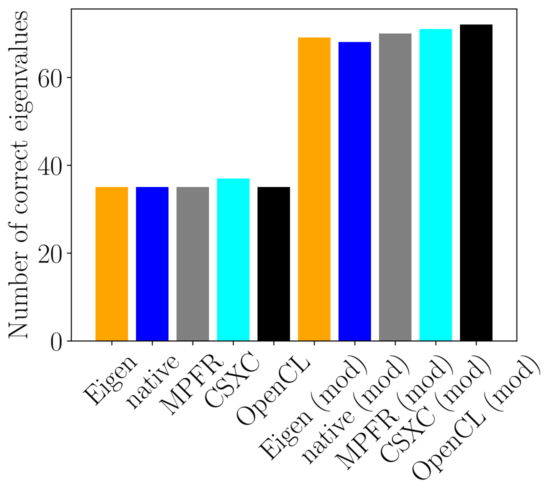
The important outcome is that neither for the modified nor for the unmodified algorithm hardware-supported data types are sufficient to calculate exact results. Therefore, even the numerically more stable algorithm requires higher internal precision depending on the size of the input matrix. While HPA improves the stability of a certain algorithm, it introduces a significant overhead that is often not tolerable. Hence, a numerical analysis in combination with an algorithmic improvement is still required.

7.6 What about the Exact Dot Product?

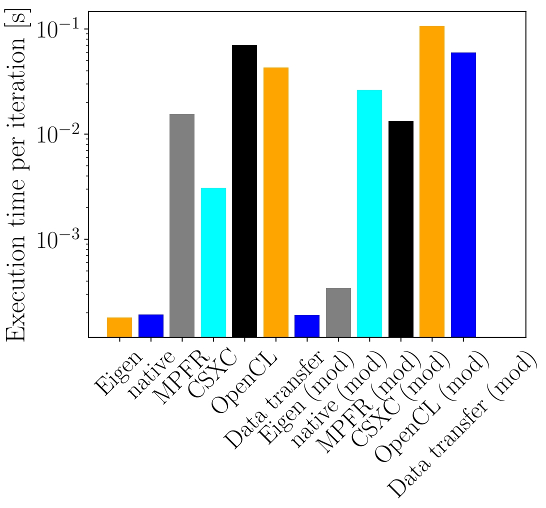
According to the findings in Section 7.5, HPA is inevitable for certain algorithms, but causes significant overhead regarding execution time and memory footprint. To avoid high overhead, applying HPA to less operations can be a solution. Kulisch et al. argue that an exact dot product is crucial for HPA [217]. Therefore, I consider an exact dot product for all dot products in Algorithms 7.1 and 7.2. Each matrix vector multiplication $\vec{t} = A \cdot \vec{q_i}$ is replaced by $t_i = \langle \vec{a_i} \cdot \vec{q_i} \rangle = \vec{a_i}^T \cdot \vec{q_i}$, where t_i represents an element of $\vec{t}$ and $\vec{a_i}$ is the ith column of Matrix A. Norm operations are substituted with $||\vec{v}|| = \sqrt{|\langle v, v \rangle|}$. Matrix vector multiplications and norm calculations are also performed by using an exact dot product. All remaining operations are executed in double precision.

For the exact dot product, I consider MPFR, C-XSC, and an OpenCL implementation based on [217]. Additionally, I compare these approaches with a native double implementation and an implementation based on the Eigen library. A MPFR data type requires at least a precision that can exactly perform a multiplication with two doubles for an exact dot product. A fixed-point representation of a double needs 2098 bits. The amount of bits has to be doubled after a multiplication to make sure that the value is still stored exactly. Since MPFR is based on 64 bit blocks, the required data type has 66 blocks and thus represents 4224 bits of precision. For C-XSC, dotprecision is used. The OpenCL variant also provides a precision of 4224 bits. The conversion into a double value is done on the host processor. The reason is that this would require synchronizations and data transports between compute units on the GPU.

Again, the AMD Opteron system is used. This system includes a NVIDIA GeForce GTX 960. The proposed OpenCL kernel is executed on this GTX 960. To create an input matrix, the parameters $n = 100$, $c_1 = c_2 = 1.0$, and $\alpha = 0.5$ are used.



(a) Number of correct eigenvalues



(b) Execution time per iteration

Figure 7.16: Comparison of various exact dot products within the two Lanczos methods. Note that the execution time has a logarithmic scale.

Figure 7.16a shows the number of correctly calculated eigenvalues for different used exact dot products. We can see that the influence of the exact dot product is low on the accuracy of the Lanczos algorithm. In case of the unmodified version, the `CXSC` approach calculates two eigenvalues more, but this is not significant. This is presumably caused by the variations shown in Figures 7.1 and 7.2. The same holds for the `OpenCL` variant that calculates one eigenvalue more using the modified algorithm.

Although the benefit regarding accuracy is negligible, the introduced overheads of the exact dot products are enormous, see Figure 7.16b. The fastest method, `CXSC`, is ten times slower than the double variant. The `OpenCL` method is 320× slower, while `MPFR` is 8× faster than the `OpenCL` method. As expected, the execution time of a single dot product on a PCI-attached GPU is mainly consumed by data transfers (`Data transfer`). These transfers pose more than 50% of the time. The intention of a OpenCL-based implementation is that a GPU integrated on the same chip can act as a HPA coprocessor. However, an exact dot product has no benefit for the algorithm. According to the results, we can conclude that the critical part of the algorithm is the re-orthogonalization step. This step involves subtractions that can cause cancellations.

7.7 Summary

Knowing the correct eigenvalues of an input matrix is an important point to compare the different HPA libraries. Therefore, the approach discussed in Section 7.2.4 was used.

This avoids the usage of another method to calculate eigenvalues for a given matrix. While the used approach is very useful to analyze the influence of HPA and to compare different libraries, the practical relevance is limited. Determining the required precision for data types in advance is an important aspect for real-world applications.

Setting the precision to a fixed value is not sufficient, since an unnecessary high precision introduces an avoidable overhead and a too low value leads to wrong results. This is backed up by the findings in Section 7.5. The required precision depends amongst others on the distribution of the eigenvalues. The knowledge about this distribution would be beneficial. However, the estimation of the range of the eigenvalues using known methods, for instance, greshgorin circles, does not help. The rationale is that the estimation is too coarse grained. To that end, it is an open question how to set the internal precision for a given input matrix.

The evaluation by means of investigating the influence of data type precision on the Lanczos algorithm reveals that HPA is inevitable to calculate correct results. While there exist several software solutions to easily integrate HPA into existing applications, they all introduce intolerable, additional overhead compared to hardware-supported data types. General applicable coprocessor designs that could reduce the overhead of HPA are still not available. Therefore, an entire re-implementation of the application has to be done for porting it to a coprocessor. Applying HPA to only critical parts of an application is a solution, but this requires a deep knowledge of an algorithm. For instance, using an exact dot product is not sufficient for the Lanczos method. A numerical analysis would presumably turn out that the re-orthogonalization step is sill critical, even when the modified version significantly improves it.

Another outcome is that HPA is not sufficient alone to overcome numerical stability issues. While HPA improves the stability, it introduces an enormous overhead. A numerical analysis during the implementation can lead to improvements for the efficiency of an algorithm. The usage of HPA for the critical operations reduces the overhead.

The core idea of this thesis is to have different accuracy-aware methods that allow a user or a controller to vary the QoR. However, adapting the QoR by varying the degree of HPA would not be the best way regarding the considered design goals in that thesis. This chapter has revealed that domain expertise is required for efficient implementations. On the basis of the outcome of this chapter, I do not further consider HPA for the next part of the thesis. This part proposes an approach for tuning accuracy-aware systems.

Part III

Tuning of Accuracy-aware Systems

CHAPTER
EIGHT

PREPARATIONS DURING DESIGN TIME

In this chapter, a novel accuracy-aware tuning approach is proposed. This approach tunes accuracy-aware applications during design time. The novel tuning algorithm finds suitable parameter settings for the accuracy-aware methods within the applications. This includes methods that are presented in Part II. Hence, this tuning algorithm is an import part of the *configuration layer*. The extracted configurations of the tuning algorithm can be exploited to tune the system during runtime. This results in a novel runtime control approach presented in this and the following chapter. In the following, the design-time part is presented. Furthermore, accuracy-aware computing is applied to a new domain to highlight, why it is beneficial to tune the system in a *horizontal* and a *vertical view*.

8.1 Introduction

Accuracy-aware methods provide knobs that allow designers to control the approximation degree, as discussed in Part II. This makes it possible to configure the accuracy in order to trade off quality of result (QoR) for other design goals. A knob specifies a parameter of an accuracy-aware method, selects between different approximate function versions, or represents an existing parameter of an application that affects the QoR. These knobs can be controlled using various methods that differ amongst others in the point of time, when the knobs are set. In this chapter, the focus is on approaches applied during design time. However, these approaches can also extract important knowledge required for a tuning during runtime.

The first part of this chapter presents, why it is important to exploit and tune accuracy-aware methods in a *horizontal* and a *vertical* view. An application in the domain of scientific computing is selected for this purpose. Scientific computing poses a difficult challenge for people from different domains, especially in order to find a suitable trade-off between desired solution quality and computational effort. Even the high parallel capabilities of todays hardware and novel parallel algorithms do not lead to a significant reduction of these challenges because of the increasing dimensions of current problems. Hence, we must rely on new ways to overcome the aforementioned issues. Such a new way is accuracy-aware computing.

Normally, high accuracy is often a prerequisite for scientific computing. Thus, at first glance, it seems counterproductive to marry approximate computing (AC) with scientific computing. There already exists successful work that introduces AC into scientific computing [376, 147, 377, 378, 379, 380]. However, these approaches mostly analyze the influence of data type precision on the accuracy only. Asynchronous parallelization methods are well-known in numerics and show a high efficiency on GPUs [381]. However, a systematic evaluation of AC on different layers inside a scientific application is missing in the literature. Therefore, this chapter presents a first step to apply a holistic evaluation of AC on a widely used algorithm in scientific computing.

This gives us the knowledge, where it is possible to apply AC and how we can combine orthogonal accuracy-aware methods. *Horizontal* and *vertical* AC methods are used for this evaluation. Previous work shows that the consideration of various layers and different AC methods results in an enormous benefit [26]. However, this orthogonal view is missing for scientific applications. Firstly, representative input data is assembled for the evaluation (see Section 8.2). Then, suitable and promising AC approaches are selected for the evaluation in Section 8.3.1. The systematic evaluation compares the different accuracy-aware methods regarding the execution times and the relative error as described in Section 8.3.1. This evaluation aims to answer the following questions: How big is the influence of well-known AC methods on the accuracy of a scientific algorithm? Is it possible to combine AC methods to improve other design parameters while keeping an acceptable accuracy?

Based on the outcome of the experiments, the following findings can be drawn:

- **Finding 1**: There exist further AC approaches besides precision scaling which are useful for scientific computing. Loop tiling and loop truncation enable a programmer to trade off accuracy for performance for the synchronous and parallelized Jacobi algorithm. Additionally, an approximation parameter that specifies the degree of relaxed synchronization poses an opportunity to find an optimal configuration point for accuracy and performance.

- **Finding 2**: Combining orthogonal AC methods leads to configuration points that cannot not be reached by a single method. Hence, this combination outperforms single methods regarding accuracy and performance. The combination of up to five AC methods is possible for the Jacobi method.

- **Finding 3**: Using a simple greedy-based tuning algorithm [11], suitable parameter values for the orthogonal AC methods are found. A user can state a desired relative error that is tolerable for the solution of the Jacobi method. Then, the algorithm finds the best possible performance for that given error by tuning the AC parameter.

While greedy tree-based tuning algorithms significantly reduce the computational effort compared to an exhaustive search, they do not exploit application knowledge to further reduce the effort. They only return limited information that can be exploited during runtime (more details in Chapter 9).

Model-based tuning approaches [206, 17, 20] create a model of the applications behavior regarding the QoR and performance. This can further reduce the tuning effort. However, these approaches require to learn separate multi-dimensional models describing the QoR and performance behavior and solve an optimization problem.

All of the existing approaches do not consider multiple objectives. Additionally, they do not take care about other accuracy-aware applications within the system. Conventional tuning opportunities are not part of the tuning. In summary, no static tuning approach is sufficient in the domain of accuracy-aware computing. Therefore, a novel static tuning approach is introduced in this chapter.

This approach considers accuracy-aware methods on a *horizontal* and *vertical* view. The tuning approach takes conventional methods and multiple objectives into account. The consideration of multiple objectives is achieved by exploiting an energy delay metric as budget metric.

A hierarchical approach is used for the tuning. Assuming that an application is composed of tasks, local *merger*s (performance profile, PP) per task are generated according to Chapter 6. This applied local view significantly reduces the number of global configurations that have to be considered. The rationale is that useless local configuration points are filtered out by the *merger*. The integration of a task into another application does not require to build a new *merger* for it. The information within the local *merger*s, especially, the gradients are used by the proposed adaptive global tuning approach. This approach only has to consider promising global configurations and thus it reduces the computational effort.

In sum, the contributions of the present chapter are the following:

- An evaluation study that shows the importance of considering accuracy-aware methods on a *horizontal* and *vertical* view.
- A detailed evaluation of accuracy-aware computing on a novel domain.
- The presentation of drawbacks that existing accuracy-aware tuning approaches have.
- The introduction of a novel static tuning algorithm that significantly reduces the computational effort. The tuning algorithm takes conventional tuning methods and multiple objectives into account.

Note that the information generated by the global static tuning approach is important to extract knowledge for the proposed runtime approach discussed in Chapter 9.

8.2 Fundamentals

A common task within scientific computing is numerically solving partial differential equations (PDEs). This is typically done by transforming the basic problem into a large scaled system of (linear) equations [382]. The finite element method, for example, directly transfers a weak formulation of the PDE into a system of linear equations by using arbitrary test-functions:

$$Ax = b, \tag{8.1}$$

where x_i are the coefficients of the linear combination of the basis functions of an appropriate function space. This function space approximates the solution of the PDE. Depending on the set of basis functions, the original problem, and the given approximation of the observed area, A has different characteristics including high dimensionality. Wisely selecting the basis functions leads to a sparse A. Hence, Krylow subspace methods are ideal candidates for solving the problem (8.1) [383].

Lowering the conditional number of A results in a higher convergence for those methods. This is accomplished by multiplying a suitable matrix B with A [384]. A method to find a suitable B, the so-called preconditioning matrix, is a factorization of A based on its characteristics. A widely usable factorization is the incomplete LU-factorization [385]:

$$A \approx LU = B^{-1},$$

where L and U are lower and upper triangle matrices, respectively. As for performance reasons B is embedded within the Krylow subspace method by multiplying it with a basis vector v_m of the actual Krylow subspace V_m. This leads to two further systems of equations that have to be computed:

$$Bv_m = y \quad \Leftrightarrow \quad LUy = v_m \quad \Leftrightarrow \quad L\tilde{y} = v_m, \quad Uy = \tilde{y}. \tag{8.2}$$

Because L and U are sparse but triangle matrices, typically solvers based on splitting methods like the Jacobi method are used to solve the inner systems [385].

The main challenge now is to solve these inner systems (Equation (8.2)) very efficiently to keep the performance benefit due to less iterations of the Krylow subspace method. An important fact to note is that the accuracy of the solution of the inner systems only affects the convergence rate, hence it does not affect the solution of the outer method. Note that there are important mathematical properties for solvers and preconditioning methods. First of all, the preconditioning operator B has to be invariable over the whole iteration process for most Krylow subspace methods [383]. Manipulating the updating process of the inner solver may change the operator from one iteration to another. However, methods such as FGMRES allow the adaptation of preconditioners per iteration [383].

The second problem is the convergence of the inner solver. Having spectral radius ρ of L and U smaller than unity results in a secured convergence [386]. Although this might be not the case for all matrices assembled from discretization of PDEs and incomplete factorization, there are large and relevant classes of problems with resulting triangular matrices. These matrices can be solved by matrix splitting based solving methods.

Now, we take a look at the generation of the test data. The basic problem is an inhomogeneous Poisson's problem with homogeneous boundary conditions on the unit square. The discretization is done with a five-point-stencil and the finite difference method. The resulting system of equations is diagonally dominant, irreducible, and can be easily scaled to any useful dimension. A is sparse, symmetric, and positive definite.

For the evaluation study, the Jacobi method is used. This method acts as the inner solver as discussed. The right side v_m of Equation (8.2) is a set of vectors that are created as residuals within a performed CG method. Not that only the influence of AC on the Jacobi method is evaluated. The evaluation shows that exploiting *vertical* and *horizontal* AC methods together is wise.

8.3 The Potential of Exploiting Horizontal and Vertical AC Methods

In part II, accuracy-aware methods are considered in isolation and compared on a *horizontal* view. In this chapter, an evaluation study is performed to investigate the usefulness of combining different accuracy-aware methods also on a *vertical* view. Furthermore, requirements are identified that are important for controlling the collaboration of different methods during design time and runtime. These points represent the requirements for the *configuration layer*.

8.3.1 Evaluation Study

For the evaluation study, a benchmark from numerical mathematics is chosen. The detailed description of the benchmark is given in Section 8.2.

Considered approximation methods

The selection of the accuracy-aware methods is inspired by two things. Firstly, I want to evaluate orthogonal methods, which are applicable concurrently. Secondly, I decide to

Table 8.1: Overview about the considered accuracy-aware methods for the Jacobi method.

Layer	Approximation target	Approaches		
Problem layer	**Input**	Input data approximation		
Software layer	**Parallelization**	Relaxed synchronization		
	Sampling (data items)	Loop perforation	Loop tiling	Loop skipping
Hardware layer	**Data operations**	Precision scaling	Approximate memory	

use methods that seem promising and have a high standing in the AC domain. Moreover, each of them has shown great success on different applications. The selection of methods is shown in Table 8.1. Each of these methods offers different parameters that influence the trade-off between different design goals such as QoR and performance. In the following, I describe the meaning of each parameter and present the used approximation parameters.

Problem layer: An input data approximation directly targets the problem description itself and thus approximates the input to the approximate task for solving the linear equation system. More precisely, it adapts the system of linear equations in a way that less work is required for solving the system.

In the test case, this can be done by taking influence to the ILU factorization as this specifies the resulting system of equations. This resulting system represents the input data of the Jacobi method. The usage of a sparsity pattern makes it possible to specify entries of L or U that are set to zero. This reduces the operations within the Jacobi method. The challenge is to decide which entries have the least impact on the accuracy of the Jacobi method. These entries are most likely the best entries to remove.

Taking a look at the updating process of the Jacobi method, it is obvious that there exist two possibilities to determine the best element of L or U to remove. The restriction is not to remove the diagonals of the matrices. The first removes the entry matching to one of y from Equation (8.2), which is close to zero. The second is the one which is closest to zero itself.

As y is unknown during the computation of the ILU factorization, the latter method is the one of choice. To keep the original structure of the matrices as long as possible, I additionally decide to give removing priority to the leftmost (respectively rightmost) element of a row. This results in removing these elements first. The number of removed entries poses the approximation parameter.

Software layer: Relaxed synchronization is a way to reduce the synchronization overhead introduced for a parallel execution [19]. It means that some synchronization points are intentionally violated to improve performance. However, relaxed synchronization can hamper the accuracy of the result. Hence, programmers have to take care where relaxed synchronization is viable. Barriers or synchronizations that assure to read the most recent data are good points to introduce relaxation.

For the evaluation, I use an algorithmic-specific relaxation, which is often called an asynchronous method in numerics. This relaxation is based on a work of Anzt et al. [381]. Normally, a given start vector is updated within each step of the Jacobi method. The update process can be done in parallel, but needs synchronization at the end of an iteration. The motivation behind the relaxation is to subdivide entries of the vector in groups of a given size. All members of the same group are synchronized at the end of the iteration step but synchronization between two different groups is relaxed. Anzt et al. showed that this relaxation may lead to great speedups on GPUs. Additionally, convergence is proven for the asynchronous Jacobi method [387]. The number of groups presents the approximation parameter.

Software layer: Sampling approaches can be realized by adapting the behavior of a loop. They select items of the input data that are used for the computation only. I also count approaches to this level that earlier stops the execution of an iterative algorithm. Figure 8.1 shows the schematic of these approaches.

Loop perforation (see Figure 8.1a) is a well-known technique of AC on the software level [13]. The motivation is to reduce the execution time of a loop by skipping iterations in between. Depending on the actual loop, this essentially results in sampling the input or output. The perforation rate is the approximation parameter (steps).

```
for $i \leftarrow 0$ to $n-1$, i+=steps do
    result = do_work();
end
```

(a) Loop perforation.

```
for $i \leftarrow 0$ to $n-1$, i+=steps do
    result[i] = do_work(input[i]);
    for $i \leftarrow 1$ to steps-1, j++ do
        result[i+j] = result[i];
    end
end
```

(b) Loop tiling.

```
for $i \leftarrow 0$ to (n-steps), i++ do
    result = do_work();
end
```

(c) Loop truncation.

Figure 8.1: Used approximation methods on the data level (sampling approaches).

Loop tiling (see Figure 8.1b) assumes that nearly located elements of an input have similar values [14]. Hence, it only calculates some iterations of the loop and assigns nearby outputs to an already calculated value. This actual forms a tile structure of the output. Loop tiling is a special form of memoization. The tile size presents the approximation parameter.

Loop truncation (see Figure 8.1c) is a method that drops the last iterations of a loop. Here, the approximation parameter specifies the number of dropped iterations. Such an approach is especially useful for iterative methods. Iterative methods are commonly used in numerical mathematics. They perform a computation in a way that it calculates a sequence of approximate solutions. This sequence ideally converges to the exact solution. The number of truncated iterations represents the approximation parameter.

Hardware level: Data operations are typically floating-point operations for numerical algorithms. Many approaches in AC present designs that deal with arithmetic units. This also includes floating-point units. These approaches can be roughly grouped into two general approaches. One deals with the precision of the operations itself. This is achieved by precision scaling or by redesigning a processing unit in an approximate way (cf. Chapter 4). These methods lead to more efficient hardware designs regarding power consumption, latency, or area. The other approaches deal with approximate memory which may affect the accuracy of involved operands [8]. In general, approximate memories can lead to indeterministic stored data and thus getting wrong data, while loading data from the memory.

To include these approaches in the evaluation, floating-point operations are adapted within the algorithm. The first group is simulated by truncating bits of the significand (called `precision scaling`). The approximation parameter represents the number of truncated bits. For the second group, random bits are used for those less significant bits. However, this means that each memory accesses would be affected. Therefore, another experiments is performed. For this experiment, errors are introduced according to realistic error rates [8].

Evaluation setup

The described approximation methods are applied to the iterative Jacobi method individually. As input, the described system of linear equations is used according to Section 8.2. In a next step, a combination of several AC methods is considered. All the experiments are executed on an AMD Opteron 6128 processor providing 64 GB of main memory. A synchronous and parallel version of the Jacobi solver executed using 32 threads is the base line. The parallelization is done over matrix rows.

The parallel algorithm requires $130.1\,\mathrm{ms}$ for a matrix A with a dimension of 1024^2 and $631.2\,\mathrm{ms}$ for a dimension of 2048^2. If not otherwise mentioned, the iteration count is set to 10. Stopping the iterative method after 10 iterations results in a relative error of roughly 10^{-4} compared to the exact solution independent from the matrix dimension d.

Evaluation metrics

For the accuracy, the relative error is used

$$E_{rel} = \frac{||\vec{x} - \vec{\tilde{x}}||_2}{||\vec{x}||_2},$$

where $\vec{x}$ is the solution vector of the base line and $\vec{\tilde{x}}$ the solution of the approximate version. Moreover, the execution time is measured, when possible. In other cases, realistic numbers are used from the literature.

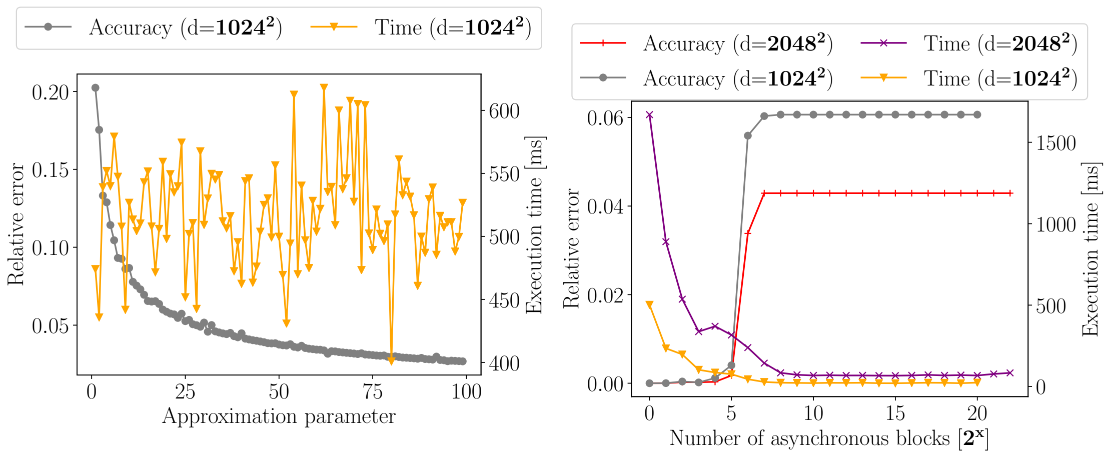
Evaluation result

Next, I present the results of the cross-layer evaluation study.

Problem layer: Input approximation adapts the input data and thus the problem statement. Therefore, certain inputs are removed. The approximation parameter represents an offset, which specifies the affected rows of the input matrix. For instance, 20 means that each 20th row is influenced. In general, affecting fewer rows leads to a reduction of the error. Until a parameter value of 20, this reduction is exponential (see Figure 8.2a). Afterwards, the error decreases slowly.

However, we cannot see that removing certain inputs have a clear influence on the execution time. There are strong variations in the execution time. This means that they are independent from the approximation parameter. According to these results, I draw the conclusion that input approximation is not useful for our test case.

Software layer: Relaxed synchronization is investigated for the following experiment (see Figure 8.2b). A higher number of blocks means that more synchronizations are relaxed during the execution. The relaxation method introduces a small error until the number of blocks is larger than the number of available cores, in this case $2^5 = 32$. At this point, we can see a high increase of the relative error. In contrast, the optimal point regarding performance is reached when the number of blocks is roughly eight times the number of cores. The curves show similar behavior for different matrix dimensions, but the relative error is smaller for the larger dimensions. The performance gain is more significant for larger matrix dimensions.



(a) Input approximation[1].

(b) Relaxed synchronization.

Figure 8.2: Consideration of input approximation and relaxed synchronization on the Jacobi method.

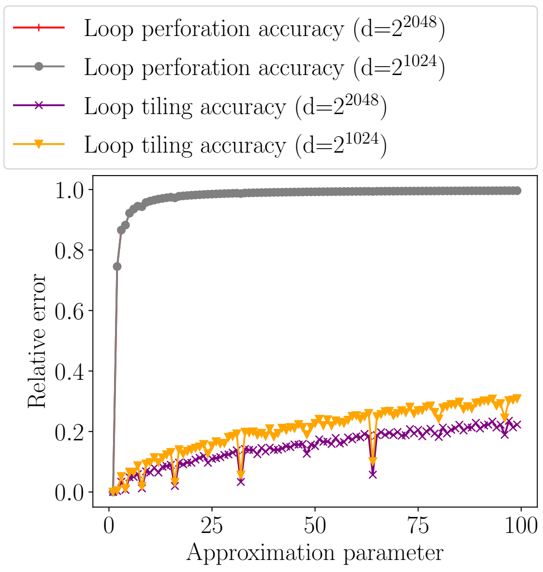
Software layer: Sampling strategies are common method in AC. It also adapts the execution of iterations for a loop. This essentially leads to skipping iterations or a sampling scheme on the input data. Figure 8.3 shows the impact of loop perforation and loop tiling for different approximation parameters (called `steps` in Figure 8.1).

The method `loop perforation` is not applicable at all for the considered algorithm, since the error exponentially increases with the approximation parameter. In contrast, `loop tiling` works quite well. Especially, small values for the approximation parameter still lead to small errors. We can see an influence of the dimension on the accuracy for `loop tiling`. A smaller dimension shows a higher error behavior.

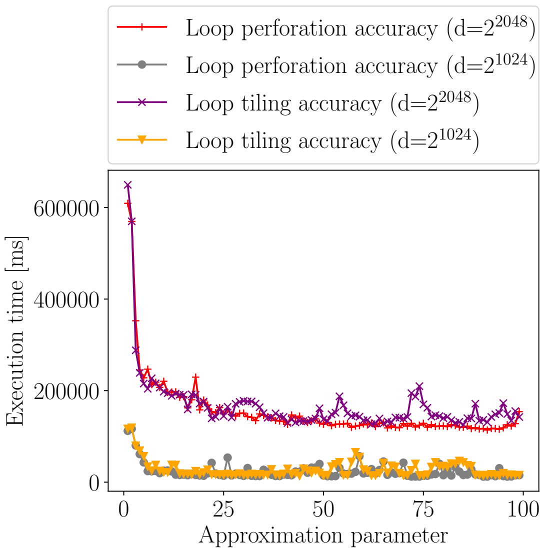
Fortunately, the execution time significantly decreases for small parameter values. Larger values have no further considerable benefit regarding the execution time. The rationale behind is that at a certain point the synchronization overhead of the parallelization and other parts of the algorithm, where the AC methods have no effect, have the main impact on the execution time.

`loop truncation` is a natural way to approximate iterative methods. It just stops the iterative method before it converges. Figure 8.4 shows the accuracy and execution time for different stop points. A stop point specifies the number of allowed iterations. Again the relative error is almost independent from the matrix dimension. The error exponentially decreases with the iterations at the beginning and then requires some time to converge. The execution time for large dimensions scales roughly linearly with the number of iterations. For small dimensions, the synchronization overhead is quite high.

[1] I am aware of the strange time measurements but unfortunately it is unclear where the oscillation comes from. However, they are reproducible.



(a) Accuracy[2].



(b) Execution time.

Figure 8.3: Influence of loop perforation and loop tiling.

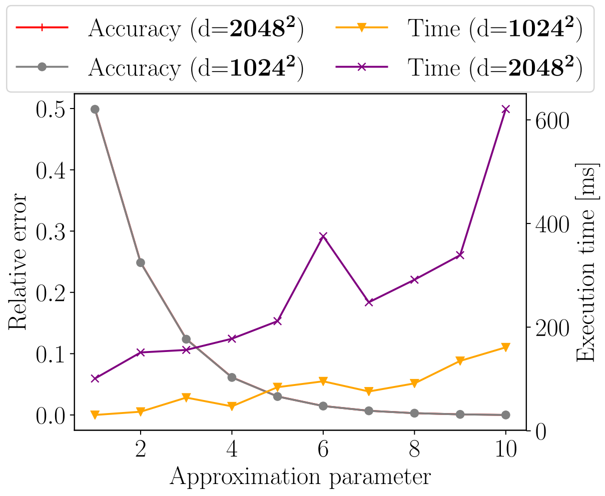


Figure 8.4: Influence of loop truncation regarding accuracy and performance[2].

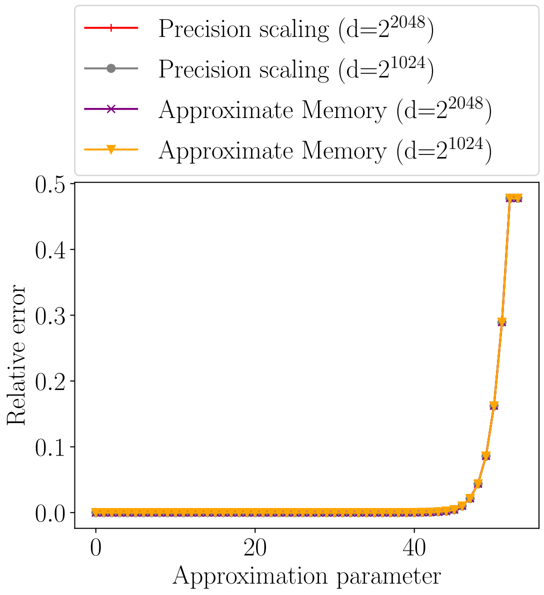
To sum up, `loop perforation` is not a useful approach for the Jacobi method. Regarding the error and performance, `loop truncation` provides the best solution in general. However, `loop tiling` can be a useful method for larger allowed relative errors.

[2]The accuracy measurements are overlapping for loop perforation (red and gray curve).

Hardware level: Influence of AC on the data type level is investigated regarding the impact on the accuracy of the solution vector. Since I cannot perform these experiments on current hardware, I use an emulation scheme. The reason is that current hardware does not provide other floating-point data types apart from float or double in general. I consider two well-known AC methods: precision scaling and approximate memory. Figure 8.5a shows the impact of these methods on the relative error. I vary the number of influenced precision bits of the significands from 53 to 0.

We can see that for the given system of linear equations, the most of the least significant bits of the significand play a minor role for the accuracy. Moreover, the results are more or less independent from the matrix dimension d and the way how the data type precision is influenced. 13 bits are enough to have almost no additional error compared to the base line. Having less than roughly 8 correct bits leads to an exponential increase in the relative error.

However, according to literature it is not very likely that all memory reads are affected by approximation. It actually depends on how this approximation method is implemented. A common way is to increase the refresh cycle time of a DDR memory bank, which can significantly save energy. Depending on this increase, the error rate of getting wrong results from the memory also raises. For some realistic values, I consider how this error rate impacts the accuracy of the Jacobi solver, see Figure 8.5b. Even if we have relatively high error rates, for instance 1.3×10^{-4}, the influence on the accuracy is not drastic.



(a) Precision scaling.

	Relative error	
Error rate	$d = 1024^2$	$d = 2048^2$
1.3×10^{-4}	0.00234583	0.0026825
2.0×10^{-5}	0.00221096	0.0010068
3.8×10^{-6}	0	0
2.6×10^{-7}	0	0

(b) Approximate Memory.

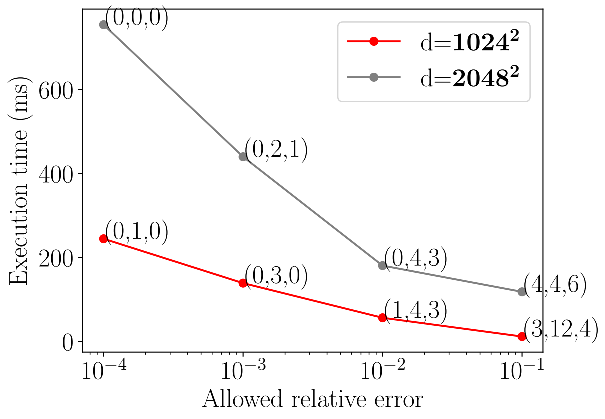
Figure 8.5: Influence of the data type precision on the accuracy.

Such an approximate memory approach decreases the power required for refresh up to 25% having an error rate of 1.3×10^{-4} [8]. Getting the actual performance or energy gain is very difficult, since it would require to build such a hardware and to evaluate the wanted metrics. Here, I show the potential of the reduction in precision bits.

Putting everything together: A cross-layer consideration of multiple and orthogonal AC methods is presented in the following. According to the results so far, I include *loop truncation*, *loop tiling*, *relaxed synchronization* and *precision scaling*. All of them have an approximation parameter that is tunable. I set these approximation parameter values according to a given relative error, which represents the QoR constraint. To find a good configuration of parameter values that satisfies the constraint, I exploit a known greedy algorithm [11] based on steepest ascent hill climbing.

For the first test, I exclude precision scaling, since performance measurements are not possible for this method without the actual AC-based hardware. I adapt approximation parameters in a way that higher values present a more aggressive approximation level. The results are shown in Figure 8.6a for different error constraints.

As we can see, the greedy algorithm tunes the parameter of all three orthogonal methods. Hence, the combination of methods is beneficial to reach good performance points for different error constraints. Allowing a relative error of 1%, we get a performance improvement of roughly 300% compared to the 32 threaded basis version. Moreover, a 10% allowed error leads to almost a speed up of 600%.



(a) Three approximation methods.

	Parameter set	
max $\boldsymbol{E_{rel}}$	$d = 1024^2$	$d = 2048^2$
1×10^{-4}	$(0 \mid 2^1 \mid 0 \mid 32)$	$(0 \mid 2^1 \mid 0 \mid 52)$
1×10^{-3}	$(0 \mid 2^2 \mid 1 \mid 35)$	$(0 \mid 2^3 \mid 0 \mid 52)$
1×10^{-2}	$(0 \mid 2^4 \mid 3 \mid 40)$	$(1 \mid 2^4 \mid 3 \mid 52)$
1×10^{-1}	$(4 \mid 2^4 \mid 6 \mid 52)$	$(2 \mid 2^{12} \mid 2 \mid 52)$

(b) Four approximation methods.

Figure 8.6: Considering multiple orthogonal approximation methods for the Jacobi method. Parameter set $(TI|RS|TR|PS)$ TI: Loop tiling, RS: Relaxed synchronization, TR: Loop truncation, PS: Precision scaling

For the found configuration points, I further consider the potential of applying precision scaling, see Figure 8.6b. This consideration reveals that all found configurations further enable the usage of AC on the data level. This allows a hardware designer to approximate hardware arithmetic units for the algorithm under test. Additionally, another possibility is to exploit approximate DRAM as *fifth parameter*.

Summary

Taking a look on the results, we see that not only a single accuracy-aware method is useful. A combination of strategies is very beneficial. The results of the combined accuracy-aware method show that remarkable speed ups can be gained with careful QoR reductions. Moreover, the results reveal that even in the domain of scientific computing accuracy-aware computing is beneficial.

However, the important point of combining different accuracy-aware methods is how to find suitable configurations. For the evaluation, I exploited a well-known greedy algorithm which is based on steepest ascent hill climbing according to [388]. This algorithm reduces the effort for finding a good configuration that satisfies (user) constraints compared to an exhaustive search. I elaborate on this point in detail in the following.

8.3.2 Requirement Analysis and Drawbacks of Current Approaches

An accuracy-aware method provides a knob that varies the degree of approximation. The meaning of a knob is different. A knob allows a designer to select between different approximate versions, for instance, a different hardware unit or neural network representation. A knob can represent a parameter that influences the execution behavior. Examples are the perforation rate or the point in time for a DRAM refresh. In case of the proposed contract-based tasks, a knob represents the granted budget to a task.

An issue of finding suitable values for these different knobs is that each value for a knob can have a completely different meaning. Such a mix of discrete and continuous parameters makes solving an optimization problem very cumbersome.

The goal of the tuning is to find a (near) optimal setting for the different parameter values. This setting satisfies all of the existing constraints and achieves the best values for the remaining design goals. For instance, having a QoR constraint, the goal is to find the best configuration that satisfies this constraint and provides the best achievable performance. Since we cannot make any assumptions about the properties of the optimization function, an exhaustive search only guarantees to find an optimal solution.

Such an exhaustive search requires to consider all possible configuration points. The number of configurations $|C|$ depends on the number of tasks within the application $|T|$, the number of different tasks orders $|O|$ that also indicates the used tasks, and the number of approximation configurations $|\text{AC}|$. Thus, $|C| = |O||\text{AC}|^{|T|}$. The used tasks depend on the application parameters. In case that we only use a single approximation method per task, $|\mathsf{AC}| = |\mathsf{AM}||\mathsf{K}|$, where $|\mathsf{AM}|$ is the number of approximation methods and $|\mathsf{K}|$ is the number of knob values per approximation method.

Considering a combination of approximation methods per task, which is important for the *horizontal* and *vertical* view, the number of possible combinations is $2^{|\mathsf{AM}|}$. The total number of different knob settings per combination depends on the number of combined approximation methods and thus

$$|\mathsf{AC}| = \sum_{k=1}^{|\mathsf{AM}|} |\mathsf{K}|^k = \sum_{k=0}^{|\mathsf{AM}|} |\mathsf{K}|^k - 1 = \frac{|\mathsf{K}|^{|\mathsf{AM}|+1} - 1}{|\mathsf{K}| - 1} - 1.$$

Let us consider examples. Having three different tasks ($|T| = 3$), two different approximation methods ($|\mathsf{AM}| = 2$), two knob settings per method ($|\mathsf{K}| = 2$) and a single task ordering ($|O| = 1$), then the number of possible configurations is $|C| = \left(\frac{2^{2+1}-1}{2-1} - 1\right)^{|3|} = 216$. Now, let us also consider $|T| = 3$, $|\mathsf{AM}| = 5$, and $|\mathsf{K}| = 4$, then we have $|C| = 2.54 \cdot 10^9$ configurations. For the cross-layer evaluation study, $|T| = 1$, $|\mathsf{AM}| = 7$ and $|\mathsf{K}| = 45$, which is an average value for the possible knob settings. Thus, the number of configurations $|C| = 188.7 \cdot 10^6$.

Since there is no analytical function to evaluate a configuration, we have to empirically determine the wanted values of the design goals. Therefore, a typical procedure in AC is to execute the application according to the current configuration for each item of representative data. Afterwards, the average QoR and the wanted design goal value such as consumed energy or execution time is determined.

The configuration that provides the best value for the considered design goal and satisfies the constraints is the desired configuration, which is used during runtime. The required time to find this configuration using an exhaustive search is $t = |C||D|t_{\mathsf{app}}$, where $|D|$ is the number of representative data items and t_{app} the average time required to execute the application.

Considering a further approximation method or changing something within the application leads to a re-execution of the exhaustive search. If a certain knob represents a continuous value such as a budget for contract-based tasks, $|\mathsf{K}|$ depends on a suitable discretization factor. We can discretize the parameter into 0.5% or 1% steps according to the baseline execution time leading to $|\mathsf{K}| = 100$ and $|\mathsf{K}| = 200$, respectively.

An often used method to drastically reduce the number of considered configurations is the usage of a greedy-based tree algorithm. This algorithm is based on steepest ascent hill climbing [11, 25]. Such an algorithm was used in the cross-layer evaluation study.

This process called tuning wants to find the configuration, which satisfies a given QoR constraint and provides the best performance. All approximation methods applied to an application are considered at once. Their respective knobs span an n-dimensional space $(K_1, K_1, \cdots, K_n)$, where the best configuration shall be found. Tuning is applied during design time. This procedure assumes that a lower knob value represents a less aggressive approximation and thus $(0, 0, \cdots, 0)$ represents the exact version. The root node of the tree represents the exact version and thus all knobs are set to zero. The tuning algorithm starts from the root. Then, it applies an incrementally more aggressive value for each knob, which also represent the child nodes. The child node offering the best performance, while satisfying the QoR constraint, is selected for the next step. In case that no such child node exist, no approximation is possible.

The selected child node is used as starting point for the next iteration. Again, an incrementally more aggressive value for each knob is applied. This procedure is continued until no further valid child node can be found. The last valid child node is returned and used for the actual execution. Figure 8.7 shows an example run of the greedy-based tuning algorithm. The number of considered configurations $|C_{\text{greedy}}| = nd$, where d is the number of tuning iterations. For the example in Figure 8.6a, the greedy algorithm reduces the number of considered configurations by

$$\frac{|C|}{|C_{\text{greedy}}|} = \frac{127,550}{60} \approx 2125.$$

But considering all seven possible accuracy-aware methods and assuming 70 iterations of the greedy algorithm, we have 490 configurations to consider. To get a stable measured value for the execution time, it is common to execute a task multiple times, for instance, 30 times. Additionally, if we would consider further hardware parameters, the time for tuning can be significantly high. These parameters are not considered by current

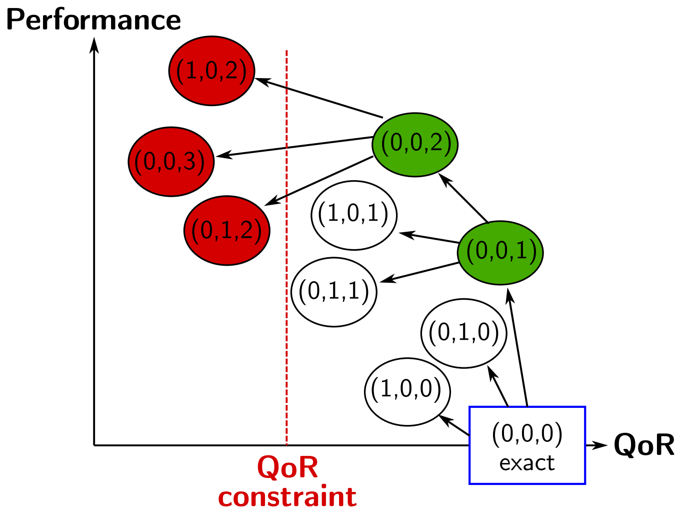


Figure 8.7: Example run of the existing greedy tree-based tuning algorithm [11, 25]. The algorithm starts with the exact version, which also represents the QoR baseline. The algorithm stops, when during an iteration no valid configuration is found (shown in red). Then, it returns the last valid (green) configuration.

approaches. Let us assume two hardware parameters, then we have to consider 9800 configurations and hence have to execute the task $2 \cdot 490 \cdot 30 = 29,400$ times. Assuming an execution requires a second, the tuning requires more than 8 hours.

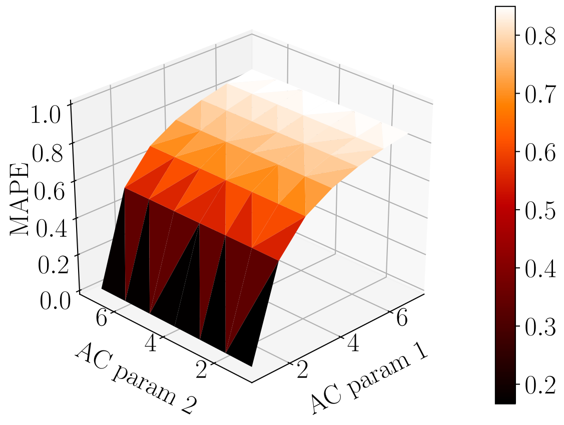
The existing tuning algorithm has further significant drawbacks. Firstly, it does not consider multiple objectives. Secondly, it does not take hardware parameters into account that do not influence the QoR as already mentioned. Furthermore, changing the QoR constraint or integrating a further AC method requires a re-execution of the tuning algorithm. It does not allow us to take care about a further constraint such as the allowed execution time. Moreover, it does not take into account that further tasks or applications can also have an influence on the QoR. The rationale is that these applications compete about the same resources. Finally, the existing tuning algorithm can get stuck in local minima of the configuration space.

I have identified three opportunities that can significantly reduce the effort for the tuning. Firstly, filtering out useless accuracy-aware methods is a suitable method. For instance, this is the case for loop perforation for the evaluation study. Secondly, the greedy algorithm does not exploit any knowledge about the meaning and type of a knob. Hence, it also has no idea about the presumable effect caused by increasing the knob setting. Such a knowledge leads to an adaptive increase of the step size for adjusting a knob and thus significantly reduces the number of considered configurations. Thirdly, this knowledge avoids to consider configurations that most likely pose a worser configuration than we already have.

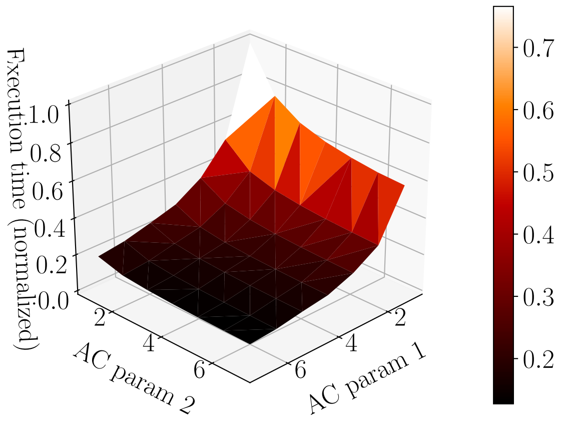
Instead of an explorative search, other tuning approaches rely on two behavior models of the application [17, 20, 206]. In general, these approaches build a model for the behavior of the QoR and the execution time depending on the knobs. The approaches differ in how they build these models and find the configurations. The approach of Sui et al. [206] generates global models, which consider all knobs together. Other approaches generate local models for different accuracy-aware methods applied to certain tasks. The combination of the local models generates the global configuration. Sui's approach is limited to already existing application parameters that influence the QoR.

Green [17] builds different models for various accuracy-aware methods using MATLAB for interpolation and curve fitting. This approach is limited to certain accuracy-aware methods or requires a huge programming effort to integrate own approximations. Furthermore, it still relies on an exhaustive search. In [20], the authors use polynomial models for local and global models. The optimal knob setting is found by solving a polynomial numerical optimization problem. The authors did not mention the used optimization algorithm.

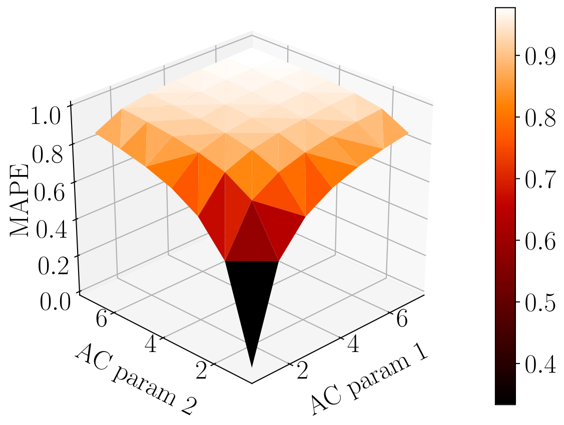
While the approach of building local models reduces the complexity compared to a global model, the number of models increases. Each accuracy-aware method and task requires two models. Figure 8.8 shows for tasks the QoR and knob space and the execution time and knob space, respectively. Both tasks consist of a two nested loop. AC



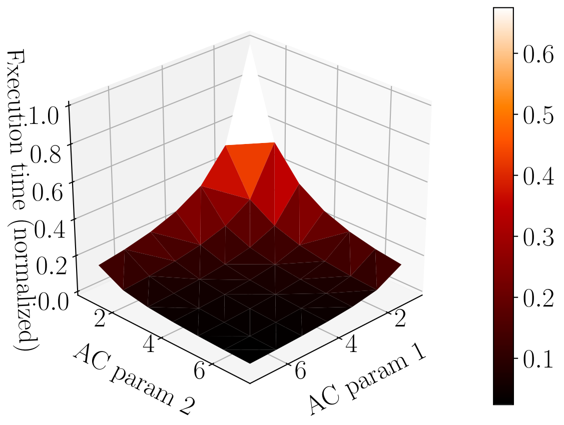
(a) `Calculate Hull`.



(b) `Calculate Hull`.



(c) `Calculate Area`.



(d) `Calculate Area`

Figure 8.8: Execution time and QoR space of the loop perforation approach for different tasks. The QoR is represented by the mean absolute percentage error (MAPE) and the execution time is normalized using the execution time of the baseline. The color bars show the magnitude of MAPE or execution time.

parameter 1 describes the perforation rate of the outer loop and AC parameter 2 of the inner loop. The two shown planes has to be modeled for the model-based approaches.

The main drawback of the model-based approaches is that they rely on two cost models, performance and QoR, to find suitable configurations given certain constraints. Hence, finding a suitable model or curve can be cumbersome. The reason is that the influence on the execution time (performance), for instance, by adapting the number of skipped iterations is not necessarily linear.

To make things even worse, increasing the number of knobs can result in more complicated configuration spaces. This makes it hard to generate correct models. Moreover, it is also difficult to find good configurations in that space, since it requires global optimization algorithms. Another issue is that the different meaning of knobs, for instance, discrete and continuous, increases the complexity to solve the optimization problem.

Still, the model-based approaches have the same drawbacks in mind than greedy-based tuning approaches. These models do not consider multiple objectives concurrently. They do not exploit the advantage of conventional methods. Equally, they do not take the competition of several applications into account.

8.4 Performance Profile-based Tuning Algorithm

In this section, I introduce a novel static tuning approach. This approach adapts knobs of accuracy-aware and conventional methods that influence different design values. For the tuning, the proposed approach considers given constraints. The core of this approach is based on the motivation to correlate the value of design goals to the QoR. The correlation is achieved by the presented approach to exploit performance profiles (PPs) in the domain of accuracy-aware computing as described in Chapter 6. This avoids building several models per task. Hence, the presented approach uses one dimension to control the other one. For instance, given a QoR constraint, finding the best available performance is easy. This also leads to a unified meaning of parameters and thus simplifies the combination of several tasks.

Building a single PP for an application is very complex, since it requires to build many global PPs and to generate a *merger* from it. Therefore, the proposed approach is distributed into three main tasks (see also in Figure 8.9):

1. Building a *merger* per task (see Chapter 6)
2. Analyzing the control flow of tasks
3. Exploiting the local PPs to find suitable global configurations for an application (results in Algorithm 8.1)

Similarly to other model-based approaches, the proposed approach generates local information to determine suitable global configurations. The introduced approach deals with multiple optimization goals and considers non-AC hardware parameters. Furthermore, it filters out useless methods for each task. The control flow can be analyzed using an approach based on a decision tree classifier as introduced in [389, 20]. This leads to different tasks flows, where each flow depends on the application parameters, for instance, the ordering of image filters. For each task flow, different global configurations are determined.

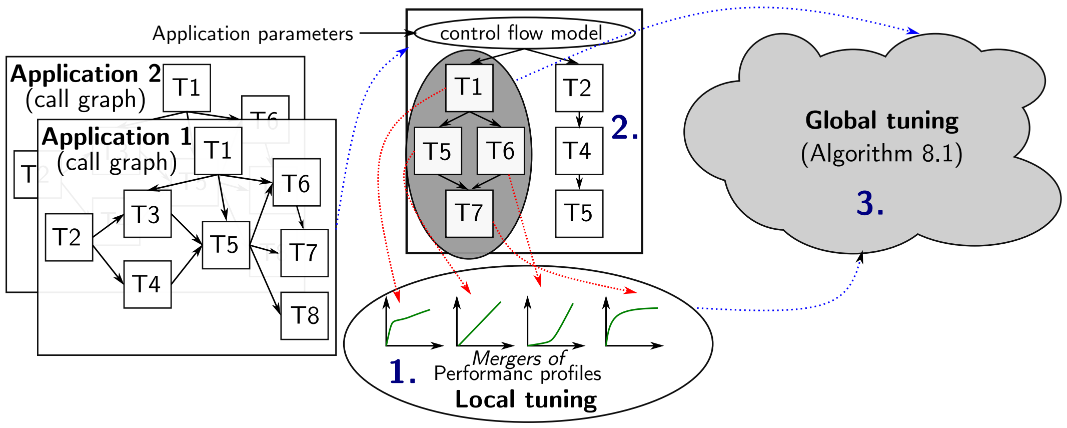


Figure 8.9: Overview of my novel tuning approach exploiting performance profiles.

8.4.1 Single Task

As already mentioned, my approach requires local information per task to determine a global configuration. In case that an application only consists of a single task, this local information also represents the global view.

For a task, it is important to generate a single PP called the *merger* (more details in Chapter 6). Each discrete point within the PP has the knowledge of the knob setting required to reach that point. Programming effort is only required, when no approximation method is already applied to a task. Possible approximation methods are thoroughly discussed in Part II. It can be needed to configure approximation methods itself, for instance, the visiting order in case of the contract-based approximation presented in this thesis. Besides the consideration of individual accuracy-aware methods for generating a *merger*, it is possible to build a *merger* for a combination of accuracy-aware and conventional method as presented in Chapter 6. Dynamic voltage and frequency scaling presents a conventional method. In Chapter 6, it is also presented in detail how a *merger* is built for a certain task. Note that the proposed tuning approach exploits accuracy-aware methods on a *horizontal* **and** a *vertical* view per task. This represents a central aspect of the present thesis. By changing the budget metric of a PP, my novel tuning method allows us to consider multiple objectives, which is not possible by other state-of-the-art tuning algorithms.

Having extracted the *merger*, finding the best configuration gets trivial. Restricting the budget, we can just use the PP point that provides the best QoR and requires less budget than allowed. The corresponding knob setting is attached to that configuration point. The same procedure can be used for restricting the QoR loss. There we use the PP point that requires the lowest budget to reach the desired QoR. Please remember, that in case of a combined budget (EDP metric), the proposed method allows us to only restrict a single design value. Figure 8.10 shows an illustration of finding the best configuration having a

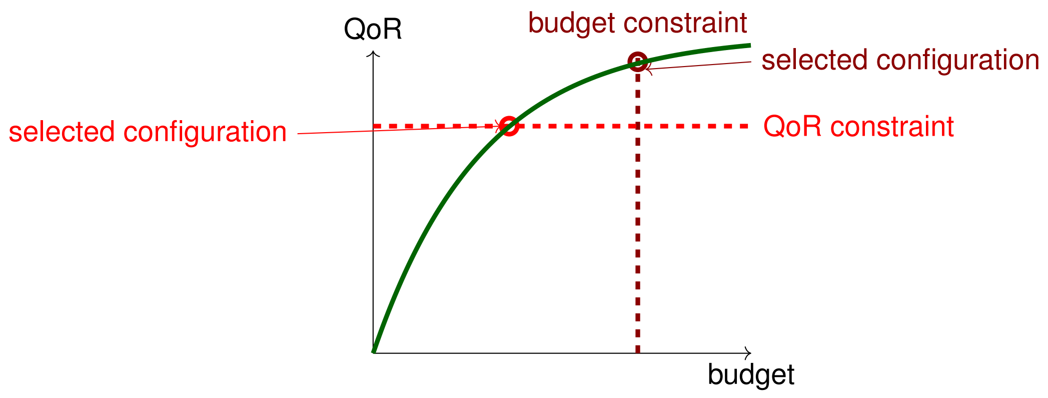


Figure 8.10: Illustration of selecting configurations from a single *merger* and for different constraints.

single *merger*.

The *merger* is represented as a list with n items. Each item has three entries: required budget, estimation of the resulting QoR, and the knob setting required to reach that point. Hence, finding the best configuration has $O(n)$ complexity in the worst case. If we know the more important constraint beforehand, we can sort the list according to this constraint. Then, we can apply binary search, which has a complexity of $O(\log(n))$.

The knowledge of a *merger* can also be exploited to determine a global configuration for an application that consists of several tasks.

8.4.2 Single Application

Before the introduction of the global tuning algorithm, I clarify the terminology used in this and the next section, see also Figure 8.11. A task represents a computation that is performed on the corresponding input data. A task that works on a certain input is called an instance of the task. This instance gets a local configuration, which is pre-set in the static case as discussed in this chapter.

An application can consist of a single task or of several tasks and thus the composition of these tasks represents the computation of the application. An instance of an application works on a single input using the tasks within the application. Therefore, each application input is processed by another instance of the application. This is important, since a global configuration is applied per instance. The global configuration consists of local configurations for each task instance. In the static case, the different knobs are set according to the global configuration during design time. The dynamic case is presented in Chapter 9.

Task instances have a direct influence on the QoR of other tasks instances within an application instance. This also poses an influence on the final QoR of the application

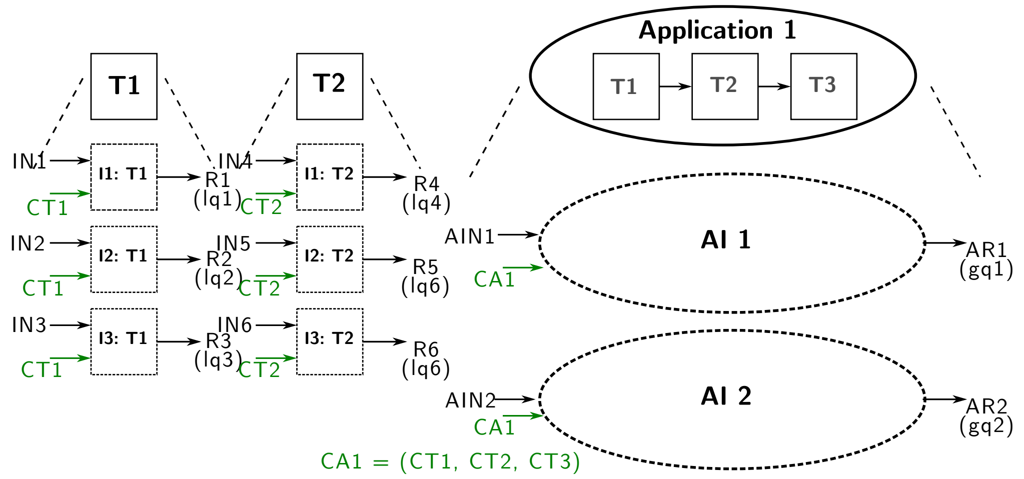


Figure 8.11: Illustration of the terminology used in this thesis. An application consist of tasks (T), which represents computation blocks. A task working on a certain task input (IN) is called instance (I) of a task. Besides the input, the task instance gets a local configuration (CT). The result (R) of a I comes with a local QoR (lq). An application instance (AI) works on a certain application input (AIN) and also gets a global configuration (CA). The result of AI (AR) has a global QoR (gq).

instance. A direct influence means that the result of a task instance is processed by a following task instance, hence its QoR impacts the QoR of the other task instance.

Since different tasks instances compete for the same compute resource, it implies an indirect influence on the QoR of other task instances, even if they are part of another application instance. This is especially the case for contract-based tasks. The rationale is that a higher execution time leads to a better QoR for a certain task instance, but may reduce the allowed budget for other tasks instances.

The QoR of a certain task instance depends on the QoR of the input generated by a previous task instance. The exact influence of the input approximation to the QoR of a task instance is not easy to determine. In the absence of a general approach, this relation is application and algorithm-dependent. Therefore, I suggest to learn good configuration points during design time. Instead of determining a single configuration that satisfy the given constraint, the proposed approach determines several Pareto-optimal configuration points. This enables a designer to deal with different constraints without re-starting the tuning algorithm as required by other approaches.

The proposed global tuning algorithm determines suitable distributions between local budgets for varying fixed global budgets. A naïve approach is to evaluate all possible local budget distributions for given global budgets. Such an exhaustive optimization algorithm is not feasible. This algorithm would have a complexity of $O(n^m)$, where n is the number of different local budgets and m the number of tasks. Since a local budget is a

Algorithm 8.1: Greedy algorithm to find global configurations that build the Pareto-optimal front between the QoR and the required global budget.

```
Input: Control flow path $A$ composed of $m$ Tasks, maximum QoR $q_{\text{max}}$, maximum iterations $i_{\text{max}}$,
       set of PPs $P$, exact result $e$, budget step size $\Delta_{\text{step}}$. quality function $Q$.
Output: Set of configurations $C$ with increasing global QoR.
Initialize each local budget $lb_T \in B$ with $\Delta_{\text{step}}$;
Initialize adaptive step size $\Delta_T \in S$ with $\Delta_{\text{step}}$;
while $i \leq i_{max}$ and $q_{best} \leq q_{max}$ do
    $\Delta_{\text{budget}} = \max\{\Delta_T | \Delta_T \in S\}$;
    for each Task $T$ in $C$ do
        Calculate gradient $g_t$ of $P_T$ between budgets $lb_T$ and $lb_T + \Delta_{\text{budget}}$;
        Determine task $T$ with maximum gradient gbest $= \max\{g_t, \text{gbest}\}$;
    end
    Increase local budget $lb_T$ by $\Delta_T$;
    result = execute $A$ with local budgets in $B$;
    $q = Q(\text{result}, e)$;
    if $q > q_{best}$ then
        $q_{\text{best}} = q$;
        C.append() ;                          // store current configuration $B$
        Initialize adaptive step size $\Delta_T \in S$ with $\Delta_{\text{step}}$;
    else
        $\Delta_T = \Delta_T + \Delta_{\text{step}}$ ;  // Increase step size to faster reach a better $q_{\text{best}}$
    end
    i++;
end
```

continuous value, many sample points have to be considered. Instead of a brute force method, my approach exploits the *merger*s of the involved tasks.

Algorithm 8.1 builds a Pareto-optimal front between the achievable QoR and the required budget. This greedy algorithm additionally determines the local budgets and local QoRs per task to reach a output QoR. The algorithm starts from zero global budget until a global budget is reached that returns the maximum desired QoR. In each step, the algorithm increases the local budget of a task that has the highest improvement of QoR according to its PP. Global configurations are only stored if the global QoR is higher than for all previous configurations. Representative data is used to calculate the QoR for different configurations.

The proposed algorithm exploits two important mechanisms to reduce the computational effort. Firstly, instead of incrementally increasing the budget, the algorithm only increases the budget for the task that has the highest gradient of its merger, when the additionally budget is granted. Hence, the algorithm exploits the knowledge of the *merger* and thus of the used tasks. State-of-the-art tuning approaches do not exploit such task knowledge. Secondly, the algorithm adapts the budget step size to fast overcome local minima of the global QoR.

Table 8.2: Fictitious example of determined global configurations for an application consisting of three tasks.

			Local configs					
Config.	Global budget	Global QoR	Task 1 local budget	Task 1 local QoR	Task 2 local budget	Task 2 local QoR	Task 3 local budget	Task 3 local QoR
0	0%	30%	0%	50%	0%	10%	0%	20%
1	8%	80%	5%	98%	20%	30%	0%	20%
2	17%	90%	10%	99%	40%	50%	0%	20%
3	25%	95%	15%	99%	60%	70%	0%	20%
4	50%	98%	20%	100%	80%	90%	50%	60%
5	73%	100%	20%	100%	100%	100%	100%	100%

Table 8.2 shows a fictitious example of an output that could be determined by Algorithm 8.1. As we can see, the algorithm returns a set of suitable configurations. The found configurations present configurations for that only the budget or the global QoR can be improved by the other found configurations. Hence, they present points according to a Pareto-optimal front. Furthermore, the global budgets and global QoRs of the configuration itself pose a global PP.

For a global and static tuning, the best configuration point within such a configuration set is selected according to given constraints during design time. Equally to the single task case, finding the configuration point requires a linear or binary search in the set. Having the global configuration, the local informations within the configuration are used to configure the task instances as described in Section 8.4.1.

8.4.3 Multiple Applications or Application Instances

The global QoR of the entire application is considered as the sum of the QoR of individual application inputs. On that account, the optimization goal is either to increase the global QoR for a fixed global budget or decrease the global budget while reaching a desired global QoR. This optimization problem is known as the multiple-choice knapsack problem (MCKP) [390]. This NP-hard MCKP has to be solved during design time for the static tuning. However, solving the MCKP during runtime enables the system to react on changing situations. I present a way to solve the MCKP during runtime in Chapter 9.

The execution of several applications within a system poses a further challenge. However, we can also setup a MCKP in this case. The difference ist that different items have a different tool set. When the entire system works under a given constraint, also conventional applications running in the system have to be considered. We rely on an estimation of the execution time or the energy consumption for these applications. The required budget of these applications minimize the budget to the accuracy-aware tasks.

Table 8.3: Determined configurations for the contract-based one nearest neighbor algorithm and different databases[3].

		50words		
			Local configurations	
Configuration	Global budget (Time norm.)	Global QoR	DTW	1NN
			Local budget	Local budget
Dyn. + static 80	1.0%	83.0%	80%	1%
Dyn. + static 99	5.3%	97.0%	99%	5%
Dyn. + static Full	7.2%	97.4%	100%	7%
		FaceAll		
			Local configurations	
Configuration	Global budget (Time norm.)	Global QoR	DTW	1NN
			Local budget	Local budget
Dyn. + static 80	2.75%	75%	80%	3%
Dyn + static 99	4.00%	98%	99%	4%
Dyn. + static Full	8.20%	102%	100%	8%

8.5 Evaluation

For single tasks, I have shown an extensive evaluation in Chapter 6. The focus here lies on finding global configurations for an entire application consisting of several tasks. I also compare the computational overhead with other tuning approaches.

8.5.1 Applying the Innovative Tuning Approach to Applications

To find good global configurations for an one-nearest neighbor classification in the domain of time series analysis, I exploit Algorithm 8.1. More details about the used benchmark can be found in Section 5.2.5. This benchmark consists of two different contract-based task, *DTW* and *1NN*, as described in Chapter 6. Some interesting configurations for two different *UCR archive* databases, *50words* and *FaceAll*, are shown in Table 8.3. Note that *DTW* task is nested in *1NN* and thus the global budget is almost similar to the local *1NN* budget.

Using the tuning approach, we achieve, for instance, a QoR of 83% for the *50words* database and require only 1% of the baseline execution time. Spending 5% of the execution time, we achieve a QoR of 97%. Similarly, we get a QoR of 98%, when spending 4% of the execution time for the *FaceAll* database. Hence, the novel contract-based tasks together with the proposed static tuning algorithm represent a novel way to accelerate the classification of time series data. The found configuration leads to impressive

[3]static XY states the required budget to reach a quality of XY% for *DTW*, while having full budget for *1NN*. Full means the full budget for DTW-CAT.

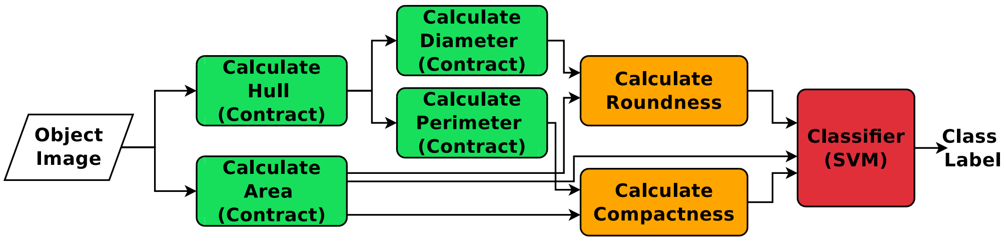

Object Image
Calculate Hull (Contract)
Calculate Area (Contract)
Calculate Diameter (Contract)
Calculate Perimeter (Contract)
Calculate Roundness
Calculate Compactness
Classifier (SVM)
Class Label

Figure 8.12: Steps to classify an object.

Table 8.4: Percentage of execution time for the different tasks within the feature-based classification pipeline.

Task	Time
Area	19.34%
Hull	12.31%
Diameter	65.98%
Perimeter	1.23%
Roundness	0.02%
Compactness	0.02%
Classifier (SVM)	0.34%

performance gains.

Next, I use the proposed tuning approach for a feature-based classification pipeline, see Figure 8.12. For instance, sensor-based sorting [391] or wheat identification [392] exploit such a pipeline. The input of the pipeline is an image containing an unknown object. Several tasks extract different features of the object. A classifier identifies the category of the object. The design goals are the classification rate and the latency of the pipeline. The classification rate is the normalized number of correct classifications and represents the QoR. In case of a firm deadline, each classification decision should require less time than the given constraint while maximizing the QoR.

Tasks with the highest execution times are perfectly suited for applying approximation in order to improve performance. However, not each of those tasks are useful to the same extent, since approximating a certain task can have a much higher impact on the QoR. Using profiling, the following tasks are the hot spots of the pipeline: `Calculate Hull`, `Calculate Area`, `Calculate Diameter` and `Calculate Perimeter`, see Table 8.4. Transferring these tasks to contract-based tasks leads to a configurable accuracy degree for each task. Furthermore, this makes it possible to apply the proposed tuning algorithm.

Figure 8.13 shows the determined configurations and thus the found budget distributions. The global budget varies from 1 to 120,000 time units. The algorithm only keeps configurations, for which the QoR is improved compared to configurations with

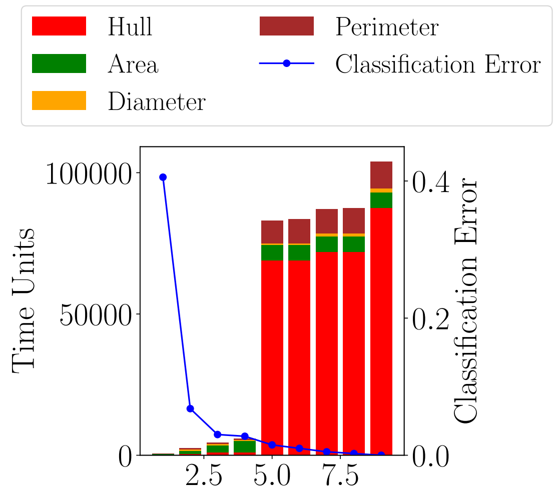


Figure 8.13: The blue points represent the average classification error (QoR) achieved by the corresponding configuration. The bars represent the time distributions of the contract algorithms to reach the corresponding error. Hence, the bars show the found suitable configurations that pose the (near) Pareto-optimal front.

smaller global budgets. Hence, each point belongs to or close to the Pareto-optimal front. Note that each configuration uses its own classifier, since this provides the best results regarding the QoR. Using the same classifier for all configurations leads to a worse classification rate. Depending on the constraints, we can select one of the configurations during design time.

8.5.2 Investigating the Tuning Effort

A central aspect for the design of the proposed tuning algorithm was to reduce the computational effort required to find suitable global configurations. Exploiting local PPs and determining all suitable configurations in a single run achieves that goal. Remember, finding m configurations with state-of-the-art tuning approaches requires m invocations.

Since the difference between some global configurations is small for the feature-based classification pipeline, see Figure 8.12, I only use the exhaustive search to find four configurations. This requires four invocations. I consider a budget for each contract-based task between 0 and the maximum budget required to reach a QoR of 100% for each representative data item. The budget per task is quantized in intervals of length 500 ns. The exhaustive search requires to investigate $|C_{exact}| = 4 \cdot 41 \cdot 65 \cdot 280 \cdot 6 \approx 17.7 \cdot 10^6$ configurations to find the configurations. Since the state-of-the-art greedy-based tree algorithm starts from the exact result, it has to investigate unnecessary configurations until it reaches the point, where the QoR begins to drop (cf. Section 6.5.2). This drop indicates the first useful global configuration. The rationale is that a QoR of 100% is also reached for the application, when we do not grant the full budget to all contract-based tasks. To reach that configuration, the state-of-the-art approach has to consider 1085 configurations. Thus, finding all four valuable configurations requires to evaluate 5600 configurations. To find these configurations, I assumed that it is already known beforehand which constraint is useful. In general, this is not necessary the case. Therefore, a significant benefit of my proposed tuning approach is that the found configurations are useful for budget and QoR constraints.

The proposed approach requires 72 iterations and thus evaluates 72 global configurations. It reduces the tuning effort about 245,000 times compared to an exhaustive search and roughly 78 times compared to the state-of-the-art greedy tree-based tuning algorithm. In total, the introduced tuning algorithm requires 272 s to find the valuable and (near) Pareto-optimal configurations.

8.6 Summary

In this chapter, I showed the benefit of combining accuracy-aware methods in a *horizontal* and *vertical* view. I presented how combined accuracy-aware methods influence the QoR and performance trade-off of a scientific computing algorithm. All methods were experimentally investigated for the Jacobi method performing on realistic data. Hence, this presents the first extensive, holistic, and schematic evaluation of accuracy-aware computing on a scientific algorithm. While single methods already can be seen as useful, a combination of them results in a much higher gain. For instance, allowing 1% relative error, we achieve an acceleration of a factor of three compared to the parallel version of Jacobi.

Furthermore, I discussed and evaluated the drawbacks of state-of-the-art static tuning approaches required to control knobs of different accuracy-aware methods. Based on this investigation, I presented an innovative tuning approach that overcomes the found drawbacks. This approach deals with multiple objectives and can exploit conventional tuning opportunities. My static tuning approach is able to significantly reduce the effort for finding global configurations. As a static approach, it cannot handle varying constraints and system states during runtime. Moreover, we already knew from Part II that the QoR behavior strongly depends on the current input. Therefore, I introduce a dynamic tuning approach in the next chapter, Chapter 9. This dynamic approach uses the information generated by the static tuning approach.

CHAPTER
NINE

DYNAMICALLY TUNING AND CONTRACTING THE APPROXIMATION DEGREE OF COMPETING TASKS

This chapter introduces two innovative approaches for a runtime tuning of accuracy-aware applications. They are located within the *configuration layer*. While the first approach is algorithm-specific, the second approach represents a general accuracy-aware tuning algorithm. The general approach selects a pre-defined configuration according to the system state and constraints. Each global configuration provides information about the required local QoR. This local QoR is used to control the approximation per task. The required control mechanism exploits the control methods presented in Chapter 6.

9.1 Introduction

Accuracy-aware methods can be used on different layers: task [19, 14, 17], algorithmic [14, 13], architecture [8, 12, 15], and hardware layer [393, 394]. Part II has introduced innovative accuracy-aware methods for the different layers. These accuracy-aware methods provide knobs that control the approximation degree and thus make it possible to trade off QoR for other design goals. A knob specifies a parameter of an accuracy-aware method, selects between different approximate function versions, or represents an existing parameter of an application that affects the QoR. These knobs can be controlled using various approaches that differ amongst others in the point of time, when they set the knobs. The previous chapter presented an innovative approach to set these knobs during design time.

9.1.1 Problem Statement

Setting the knobs during design time [15, 12, 20] can lead to undesired QoRs for certain inputs [24], for instance, for a JPEG decoder, cf. Chapter 6. The rationale is that the resulting QoR is highly input dependent. Hence, worst case assumptions have to be made during design time, which result in pessimistic gains. Thus, dynamically tuning the approximation degree is important for two reasons. Firstly, it shall be avoided that the number of inputs affected by a lower QoR than desired is intolerably high. Secondly, a higher QoR than desired provides an opportunity to reduce the computational effort and hence can be beneficial for other design objectives. Besides satisfying a QoR constraint, it is also important that a control mechanism deals with other constraints such as firm deadlines and takes multiple objectives into account. The mechanism has to consider multiple applications that can be accuracy-aware or not and run within the same system. This requires a co-controlling of applications.

9.1.2 State-of-the-Art

A hot topic in accuracy-aware computing is how to control the approximation degree during runtime. To overcome this issue for streaming applications, calibration phases are used during runtime [17, 11]. These phases statically set knobs for a certain number of following inputs according to the desired QoR and the current input. The calibration phase determines the QoR for possible knob settings by using the result of the exact algorithm and the current input. It exploits the greedy tree-based or the model-based static tuning (cf. Section 8.3.2). These approaches introduce a high overhead. The setting is used that complies with the desired QoR and provides the best value for the performance. Such control approaches assume that the behavior regarding the QoR for new inputs processed between calibrations is always similar to the one used for the calibration. Since the correctness of this assumption depends on the application, the points in time of the re-calibrations can be varied automatically [11]. But still there is no consideration of inputs processed between the re-calibrations. Approaches that exploit control theory also assume that subsequent inputs lead to a similar QoR and thus use a closed control loop for tuning knobs [22]. However, they cannot deal with QoR constraints, since it is not clear how they can measure the QoR without having the exact result.

A proactive control approach from Sui et al. leads to a dynamical adaptation of existing application parameters which have an influence on the QoR per invocation according to a current input and constraints [206]. This control approach requires to learn separate models describing the QoR and performance behavior and to solve an optimization problem. However, only the model for the performance depends on the input. In contrast, Sui et al. only use an expectation value determined off-line for the QoR model. ApproxIt is a framework that applies a dynamic approximation degree for iterative methods [147].

There exist input-aware control approaches regarding the QoR. Most of these approaches make a binary decision if an approximation shall be applied [203, 24, 395, 204, 396]. This can reduce, for instance, that bad pixels occur in an image processing applications. However, there is often some margin for an approximation rather than a full or no approximation.

On the hardware level, a pipeline of neural or special hardware accelerators provide an input awareness for specific compute patterns [185, 26]. Cascading classifiers allow an input-dependent classification [397]. Laurenzano et al. present the only complete input-aware and dynamic control method for setting knobs on the software level by using a calibration phase per input [25]. This approach uses a smaller but similar version of the input to adapt knobs. They show an almost negligible overhead for small configuration spaces. But the overhead for large configuration spaces is unclear. In contrast to controlling of the QoR, there exist control approaches that exploit accuracy-aware knobs to control the energy consumption [21] or to guarantee a hard deadline, while optimizing the energy consumption [201].

However, there exists no general approach in the literature that considers all important aspects together. A holistic approach has to deal with streaming and non-streaming applications. It shall handle iterative and non iterative methods. Additionally, it shall not only consider existing application parameters that influence the QoR. The approach shall also control *vertical* and *horizontal* accuracy-aware methods. Furthermore, it has to reduce the runtime overhead for controlling the accuracy-aware knobs as far as possible. Since the introduced innovative contract-based method is very promising, the controlling approach has to deal with continuous parameters. Besides the input awareness, it is also desirable that the control approach takes other applications running in the same system into account.

9.1.3 An Innovative Dynamic Tuning Approach

This chapter introduces a runtime approach that specifies the execution order of tasks instances running on a single core. While this approach is valuable in the theoretical sense, it highlights the importance of low overhead for accuracy-aware control approaches to achieve practical gains. Therefore, in this chapter, I propose a general and innovative tuning approach that controls accuracy-aware and conventional methods during runtime. This approach relies on performance profiles for the tuning and thus contracts the budget for the different accuracy-aware tasks within an application. During runtime, this tuning approach selects from a set of global configurations found by the proposed static tuning algorithm during design time (see Chapter 8). A global configuration is selected according to the provided constraints for the QoR or budget. As budget, the proposed approach considers the execution time, the energy consumption, or a combination of both using an energy delay product metric.

The global configurations represent (near) Pareto-optimal points between the QoR and the considered budget determined on a representative set of input data. Each global configuration specifies the allowed budget for each involved task instance working on an application input[1]. In case that a task is not a contract-based task or the PP is a *merger*, the budget is transferred to a knob setting that also includes the knobs values of the used conventional methods (see Chapter 6).

When a constraint holds for several applications and application instances, it poses a multiple choice knapsack problem. The rationale ist that application instances have an indirect influence to each other regarding the QoR, cf. Chapter 8.

To take an input awareness into account, the innovative approach performs a local tuning to reach a desired QoR. On that account, a global configuration includes a local QoR for all involved task instances that shall be reached for an input. My dynamic control approach uses the *model* and *monitoring* approach presented in Chapter 6 to tune the approximation degree per input in order to reach the desired local QoR.

In total, the innovative dynamic control approach introduces low overhead and enables a dynamically contracting of the budget per task instance. It deals with multiple objectives and co-controls several applications and application instances. Furthermore, it can react on changing constraints and system states, for instance, varying workloads.

9.1.4 Contributions

In summary, the main contributions of this chapter are

- An innovative dynamic accuracy-aware control approach that is generally usable. This approach deals with different constrains and workloads dynamically and provides an input awareness for contracting local budgets.
- The design of a low overhead algorithm to reduce the effort for solving the multiple choice knapsack problem.
- Introducing an anytime algorithm that adapts the execution order of task instances during runtime.
- Showing the importance for low overhead control approaches.
- Exploiting the innovative control approach for a *vertical* tuning of accuracy-aware methods.
- An evaluation of the proposed control approaches.

[1]A description of the used terminology for task instance and application instance is given in Section 8.4.2.

9.2 Anytime Algorithm to Determine The Execution Order of Task Instances

For several situations, it is important to have a valid result at each point of time. Let us consider an application that performs classifications, for instance, in the domain of high-throughput sensor-based sorting. The calculation of a sorting decision involves several processing steps such as filtering, extracting object regions within the image, and the final classification. The classifications are performed using decision tree-based classifiers. For each node in this tree, a feature has to be evaluated against a threshold. A leaf node represents a class. The evaluation of the tree requires to calculate several object features. The execution time can significantly vary for calculating these features and the classification itself depending on the input. Furthermore, the application exploits lazy evaluation and thus features are calculated on demand in order to minimize required execution time.

The goal of the ordering approach is to adapt the execution order in order to maximize the QoR for the next execution step. The decision about the next executed task instance on a core can be modeled as an anytime algorithm. The purpose of the anytime-based ordering approach is that the corresponding algorithm can be interrupted at anytime. After the interrupt, the approach returns classification decisions for all input objects. The proposed anytime algorithm uses the current knowledge about each object for the classification decision. Hence, this approach makes it possible to have approximate classification decisions for all inputs at each point of time during the execution of the application. This is especially important during high workloads, since the exact application is not able to comply with the firm deadline in these situations. A noteworthy strength of the anytime approach is that the system only applies approximation when the situation requires it.

A decision tree is learned using the well-known ID3 algorithm [389]. Starting at the root, the algorithm optimally splits the data set. In the present case, the split point consists of a feature and a threshold. The algorithm continues with the left and right sub-tree in a recursive manner. In a second training phase that uses distinct training data, a *node hit ratio* is determined per node. The *node hit ratio* represents the ratio of correct classifications based on the knowledge of this node.

It is possible to transfer a decision tree classifier working on several inputs into an anytime algorithm. The classification of several objects is divided into single steps during runtime. Each current step represents the current node for an object within the decision tree computation. The core idea of the novel anytime algorithm is that the next steps for all inputs are dynamically ordered according to the highest increase of information gain within the application. The ordering specifies the execution order of the steps. Hence, an additional computing step is granted to an object, for which it is likely that the correct class is not yet known. On the other side, the classification process is delayed for objects, for which the class is known with high confidence. The execution order is updated

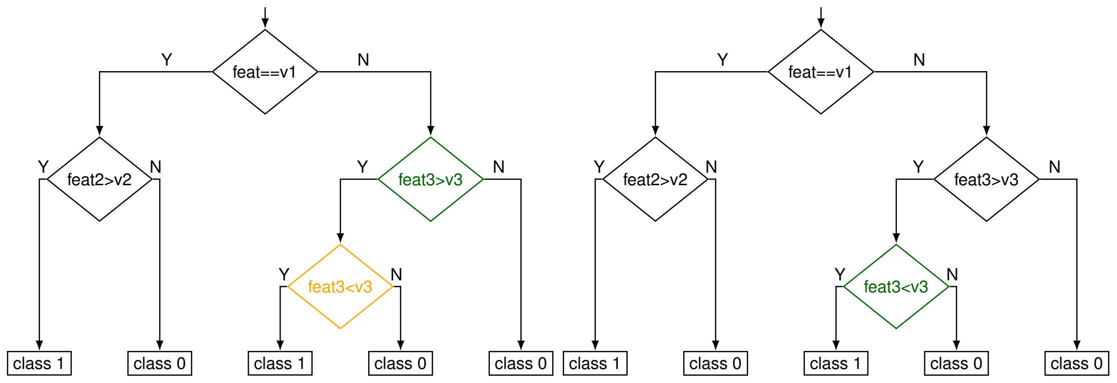


Figure 9.1: Possible states of the anytime classification algorithm for two inputs (left and right). The green nodes indicate the current step of the decision tree classifier working on an object. The orange node highlights the next step that will be executed. Assuming that the anytime classification is interrupted, the algorithm returns the current classification decisions. For the right classifier, it is the exact result, while for the left one the decision is based on the *node hit ratio*.

after each performed step. Internally, the ordered list is stored as a priority list. The priority is formulated in terms of the *node hit ratio*, where a smaller value represents a higher priority. The *node hit ratio* for each object is determined according to the current node within the classifier. Approximate classification decisions are available for each object at anytime during the execution. The anytime algorithm makes it possible to continue the execution until the deadline. When the deadline is reached, the (approximate) classification results can be queried.

An example of the anytime classification for two inputs is shown in Figure 9.1. The current visited node (shown in green) for an input represents the current step for this input within the classifier. The *node hit ratio* of this node determines the (approximate) classification. The priority list specifies the next step (shown in orange).

9.3 An Innovative Dynamic Tuning Approach for Accuracy-aware Applications

The task of a dynamic tuning approach is to tune the available knobs of an accuracy-aware application. In Section 3.1, I have identified seven important statements that are important for a dynamic tuning approach for accuracy-aware applications. The generality of the proposed tuning approach (statement IVa) is realized by relying on performance profiles (cf. Chapters 6 and 8). The usage of performance profiles also realizes a *horizontal* and *vertical* view during the tuning process (statement IVb) and a consideration of multiple objectives (statement IVg). The performance profiles also make it possible to exploit the high potential of conventional methods that influence design values (statement IVe). The integration of the input-aware methods presented in Section 6.4.2 introduces an input awareness for the tuning process (statement IVc). Representing the tuning decision as a multiple-choice knapsack problem takes the given constraints and the current system state into account (statement IVd). This representation also allows the tuning to consider competing tasks and applications (statement IVf).

Since the space of possible knob settings (configurations) is huge, the innovative dynamic tuning approach only takes configurations into account that are identified by the proposed static tuning in Section 8.4. Moreover, the influence of an approximate result to a following task is different for varying tasks. Hence, the best distributions of a global budget to local ones has to be modeled. Due to the high computational effort, this relation is modeled during design time. Additionally, for non-contract-based tasks, the behavior regarding the budget has to be known. The proposed static approach determines the required information and thus already satisfies the following statements: IVa, IVb, IVe, and IVg.

Given a QoR constraint at runtime, the innovative dynamic tuning algorithm selects a global configuration that satisfies this constraint. The same holds for applying a budget constraint. The proposed dynamic approach can deal with various budget metrics including multiple objectives, since it relies on performance profiles. As extensively discussed in Chapter 8, a certain global configuration is selected per application instance and includes a local configuration per involved task instance.

9.3.1 Dealing with Different System States, Constraints, and Multiple Applications (Instances)

Dynamically selecting a global configuration handles runtime changes regarding different constraints and system states (statement IVd). For instance, a change in the system state impacts a budget constraint, since less budget is available for the application. Changing the system from high performance to energy mode also results in a lower budget for the application instance. Moreover, a higher system load has the same effect. Also, a QoR requirement leads to a different configuration.

The proposed dynamic tuning approach has two different control modes: *QoR mode* or *budget mode*. In the *QoR mode*, an application instance shall reach their stated QoR and minimize the required budget for it. For the *budget mode*, the user or a high-level controller states a global budget, for instance, a deadline that has to be met. According to the selected mode and the current constraint, the innovative approach selects a global configuration for the application (instance).

The novel approach also enables the system to tune several application instances together (statement IVf). In this case, an universal QoR or budget constraint is shared between several application instances. It is assumed that the universal QoR is just the sum of the global QoRs achieved by the various application instances. The task is to find a global configuration per application instance. The above problem represents a classical problem in theoretical computer science. Namely, a multiple-choice knapsack problem (MCKP) [390]. This NP-hard problem has to be solved during runtime. A valid solution for the MCKP requires to exactly select one item from each class. An item consists of a weight and a profit. Items are subdivided into k classes. The k application instances represent the k classes and the items are specified by the different global configurations. The weight of item is defined according to the global budget of the global configuration. The profit is specified by the global QoR of the global configuration. The number of items per class depends on the found configurations and the corresponding control flow of the application instance. In the case that an application only has a single task control flow and no further applications are running in the system, then all classes have the same set of items. In the *budget mode*, the solution of the MCKP represents the maximum achievable universal QoR (profit) in subject to the budget constraint. For the *QoR mode*, the role of profits and weights is exchanged. There exist different exact algorithms for solving the MCKP. In this chapter, I consider the dynamic programming-based approach [390] and `mcknap`[2] for solving the MCKP. Additionally, I design and implement an approximation algorithm called `mckp_approx()` to solve a special case of the MCKP. This special case occurs, when all classes have the same items.

`mckp_approx()` divides the universal budget by the number of application instances to get a maximum global budget per instance. According to this global budget, the same global configuration is selected for all instances. In a post-processing step, the remainder of the universal budget is used to improve the approximated result. The algorithm selects between two strategies: either assigning a high budget to few tasks or a slightly higher budget to more tasks. The solution of the strategy is selected that leads to the highest universal QoR. Again, the budget and QoR is exchanged for the *QoR mode*.

Running several applications within a system poses a further challenge (statement). However, the MCKP-based approach can also be used in this case. The universal QoR of the system is the sum of the global QoRs of all application instances belonging to different applications. However, the items are different per class and thus `mckp_approx()` is not applicable. When non accuracy-ware applications have to be considered and the system has to work under a universal constraint such as the allowed total execution time, we need an estimation of the execution time of these applications. Approaches from the literature can be used [398, 399] to estimate the execution time. Having the execution times, the tuning approach subtracts it from a firm deadline. The result specifies the remaining execution time for the accuracy-aware applications. This new constraint is used as universal budget for the MCKP. To make things clear, the approximation degree is only tuned according to constraints and the system state so far. In the following, I explain how the innovative tuning approach introduces an input awareness.

[2]Source code publicly available at `http://www.diku.dk/~pisinger/mcknap.c`.

9.3.2 Enabling an Input Awareness

To realize an input awareness, the proposed dynamic tuning approach uses the information within the global configurations. They include the local QoRs that are required to reach the desired global QoR. These local QoRs are provided to the *monitoring* and *budget models* of the contract-based tasks (more details in Chapter 6). The monitoring stops the execution of a task instance, when it assumes that the stated QoR is reached. Moreover, the *budget model* may decrease or increase the assigned budget.

In case that several application instances share a universal budget, unused local budgets can be redistributed to other task instances. The unused budgets are transferred into a global budget buffer. The *budget model* is not allowed to grant more budget than available in the buffer. Unused budgets occur, when the model determines to use less budget than initially was given to the task instance. Improving the local QoRs can lead to an increase of the global QoR. Realizing the input awareness on a local level is a policy decision, since it can exploit already known information from the static tuning. However, my approach also allows system designer to introduce other policies.

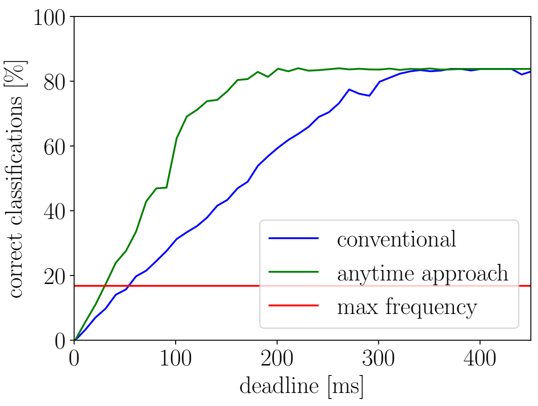
9.4 Evaluation and Results

In the following, I evaluate the anytime classifier approach discussed in Section 9.2 and my general and innovative dynamic tuning approach presented in Section 9.3. Additionally, I apply a vertical tuning.

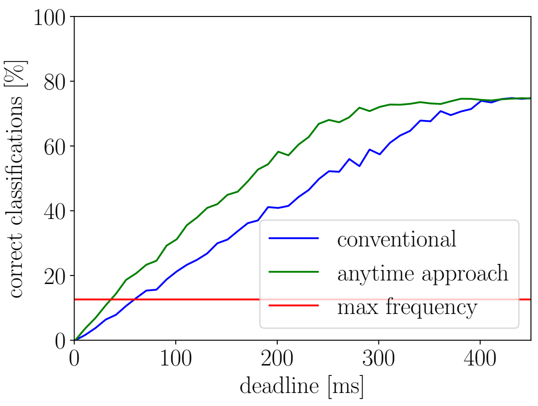
9.4.1 Anytime Classification During Runtime

The anytime classifier approach is evaluated on two different datasets. The first dataset consists of six different classes of glass shards that mainly differ in their color. The second dataset consists of eight classes of Lego bricks that mainly differ in geometric properties. For each experiment, 50 objects are randomly selected from the corresponding test set. As a reference, a conventional, non-interruptible classification based on the same decision tree is used. Objects that are not classified within a deadline are considered to be falsely classified. For the following experiments, the overhead required to calculate the execution order is not considered.

As we can see in Figure 9.2a, the proposed anytime approach significantly increases the number of correct classifications under tighter constraints (deadlines) on the glass shards dataset. The conventional approach reaches the maximum classification accuracy that is possible after $\sim$350 ms. The anytime approach reaches this QoR already at around 200 ms. Furthermore, the innovative anytime approach also performs superior compared to the conventional approach for the Lego dataset, see Figure 9.2b. The experimental results exclude the overhead for maintaining the priority list. If we take this



(a) Glas shards



(b) Lego bricks

Figure 9.2: Correct classification ratios of different classifiers for different deadlines and datasets. The *max frequency* indicates the class distribution within the corresponding data set.

overhead into account, then the novel approach performs worse than the conventional approach. There are several possibilities to reduce this overhead, however, I will not discuss them here. Instead, I investigate the general and innovative dynamic approach for tuning accuracy-aware applications.

9.4.2 Evaluation of the Novel General Dynamic Tuning Approach

For the following evaluation, I consider the feature-based classification pipeline, see Figure 8.12. I use the found global configurations from Chapter 8. As test system, I use a system that includes an Intel Core i7-4550U CPU running at 2 GHz with 8 GB DDR3 RAM attached. The L2 cache provides 256 kB and the L3 cache 4 MB.

First of all, I investigate the introduced runtime overhead caused by solving the MCKP. Only application instances of the feature-based classification pipeline are executed within the system. Hence, `mckp_approx()` is usable for solving the MCKP. The time consumed for `mcknap` is often higher than the actual execution time for all application instances. If no trivial solution exists, the overhead is between 73% and 300% for `mcknap`. The dynamic programming-based approach even has a considerably higher overhead. Compared to that, `mckp_approx()` has an overhead between 0.09% and 0.3%.

Figure 9.3 shows a comparison between `mckp_approx()` and `mcknap` regarding the quality of the approximate MCKP solution found by `mckp_approx()`. For this comparison, I use several different workloads and constraints together with the available global configurations for the feature-based classification pipeline to set up different MCKPs. The relative deviation of the profit between the found solution is $8.19 * 10^{-4}$ and hence

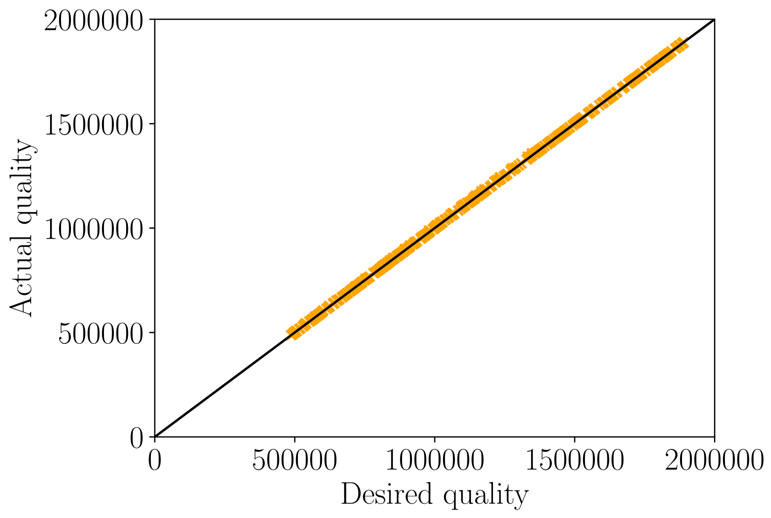
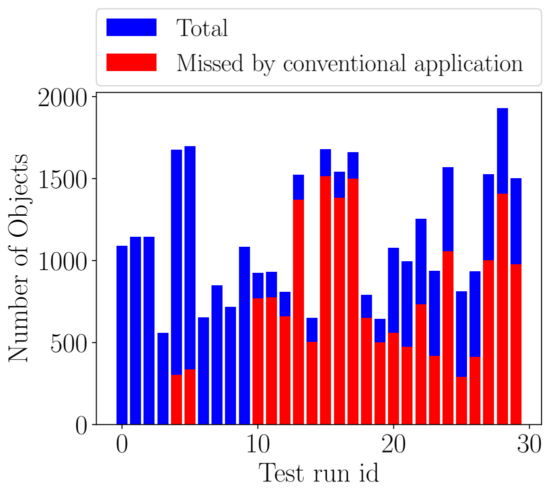


Figure 9.3: Comparisons of the exact and the approximate solution (`mckp_approx()`) for different MCKPs (orange).

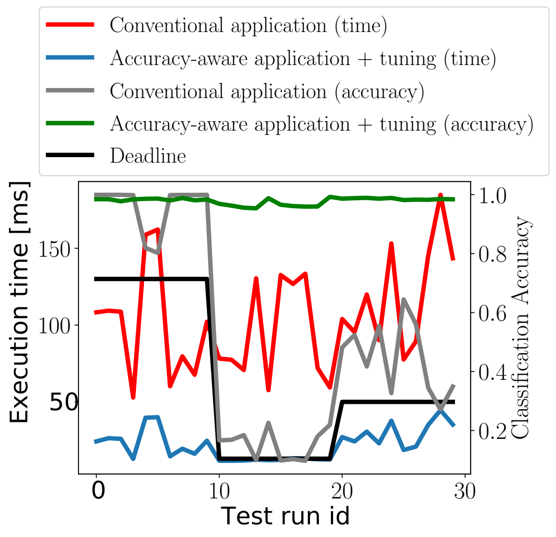
the quality loss of the approximate solution is negligible.

In the following, I evaluate the proposed runtime tuning approach. For each test run, a random number of objects is determined that shall be classified according to the classes square, cylinder, and sphere. The number of objects represent the number of application instances. All application instances of a test run are processed in a batch-wise manner. After run 9 and 17, I change the firm deadline from 130 ms to 15 ms and from 15 ms to 50 ms, respectively. Changing the number of objects and the deadline shows that my approach allows a system to react on varying constraints and system states in a proactive way. I show two different test scenarios.

The first scenario does not take input-aware approximations into account. In Figure 9.4a, the blue bars represent the number of application instances per batch in each test run. The red ones show the number of objects for which the deadline is missed using the conventional application. I assume that classifications that are not performed within the deadline are wrong. As we can see, the conventional application misses a lot of deadlines, when the deadline is tighter. Figure 9.4b shows the accuracy and execution time per test run. As the blue line is below or close to the black line, we can see that each deadline is met by my innovative runtime tuning mechanism. The conventional application (red line) only satisfies the deadline for larger deadlines and low numbers of objects. For small deadlines, 20 ms and 50 ms, there is a large drop in the classification accuracy for the conventional application (gray line). In contrast, my novel approach reacts in a proactive way to these changing situations. Besides satisfying deadlines, the QoR stays most of the time close to the highest possible QoR (around 1) (green line). Only for the small deadline and high number of objects, can we see a slight drop of the QoR.



(a) Number of objects.



(b) Accuracy and execution time.

Figure 9.4: Results of different test runs with varying amount of objects per batch and firm deadlines. (a) represents the number of objects per test run. (b) compares the conventional application with an accuracy-aware version that is tuned using the proposed innovative dynamic approach.

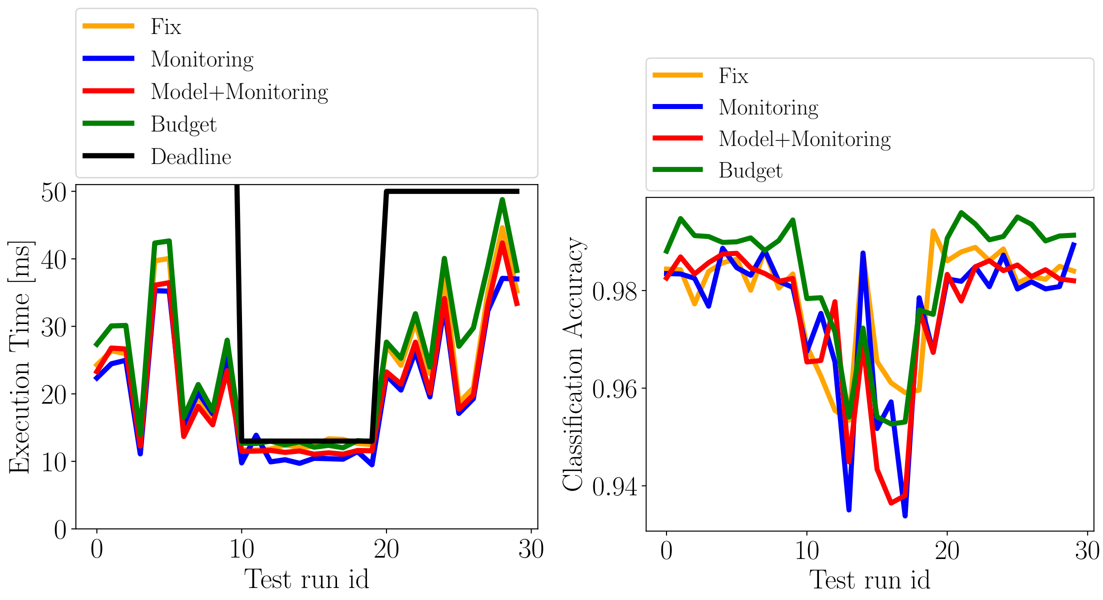
The results for the execution time and classification accuracy of the second scenario are shown in Figure 9.5. I compare a fixed budget approach (*fix*) with the *monitoring* approach (*monitoring*) and the approach that combines *monitoring* and *budget models* (*model+monitoring*). Note that a *budget model* is not allowed to spend more budget than the granted for this approach. Therefore, I consider a further approach (*budget*) that allows the proposed tuning approach to redistribute budgets.

The *budget* approach (green line) has the highest execution time but also the highest QoR in general. *Monitoring* (blue line) and *model+monitoring* (red line) reduce the execution time compared to *fix*, while *monitoring* is slightly faster due to less overhead, cf. Section 6.5.2. Regarding the accuracy, *monitoring* is close to *fix* and *model+monitoring*. However, the latter performs not as good as the other ones in extreme cases.

To sum up, in cases where the QoR is more important, *budget* is the right method. In other situations, *monitoring* reduces the execution time, while having similar accuracy.

9.4.3 Vertical Tuning of Accuracy-aware Methods

For the following experiment, I take vertical accuracy-aware methods into account to show the potential of a *vertical* view. A *vertical* essentially means a cross-layer approach. I integrate accuracy-aware methods from the *algorithmic layer* and from the *architecture*



(a) Execution time. (b) Classification accuracy.

Figure 9.5: Execution time (a) and accuracy (b) comparison of different approaches that introduce input awareness to the proposed runtime approach.

layer to the contract-based versions of the *hull* and *area* task of the feature-based classification pipeline. The selected tasks have the highest impact on the execution time and hence are suitable candidates for the vertical approach. The accuracy-aware methods are loop perforation and approximate DRAM[3].

The evaluation reveals that approximate DRAM is theoretically applicable for both tasks[4]. Loop perforation is very beneficial for the *hull* task but not for the *area* task. Figure 9.6 shows the improvement of the performance profile for the *hull* task exploiting accuracy-methods from vertical layers. We can see that the resulting *merger* (red line) is especially improved for medium budgets by exploiting the *vertical* view compared to the contract-based *hull* task (gray line).

By integrating the novel *hull* task into the feature-based classification pipeline, we are able to find better configurations using the static tuning algorithm presented in Chapter 8. These new configurations are provided to the proposed dynamic tuning approach. The results are shown in Figure 9.7 I compare the vertical view-based approach (red line) with the *monitoring* approach (blue line), since this approach offered the best solution regarding the execution time so far (cf. Figure 9.5). The merged version of the *hull* task

[3]The same procedure as described in Section 8.3 is used for simulating approximate DRAM.

[4]I have not considered a full simulation of the application running on a system with an attached approximate DRAM.

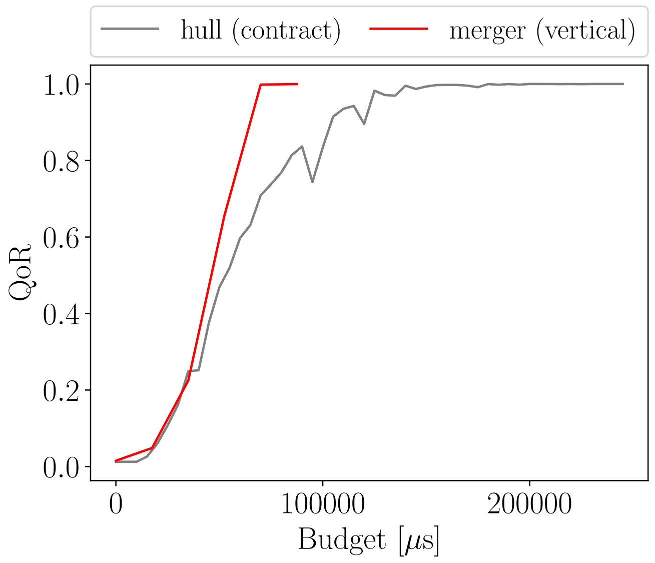
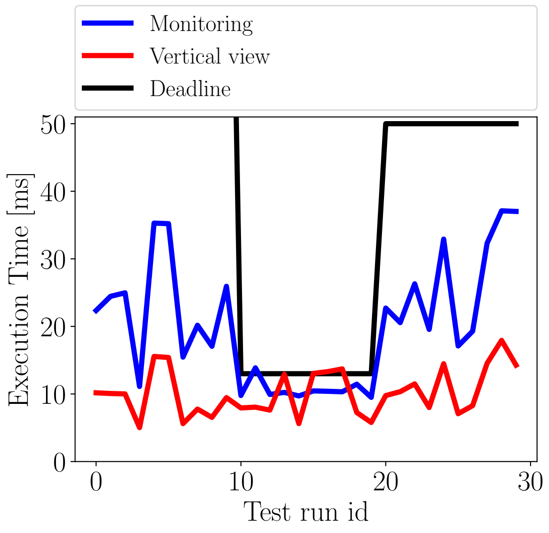
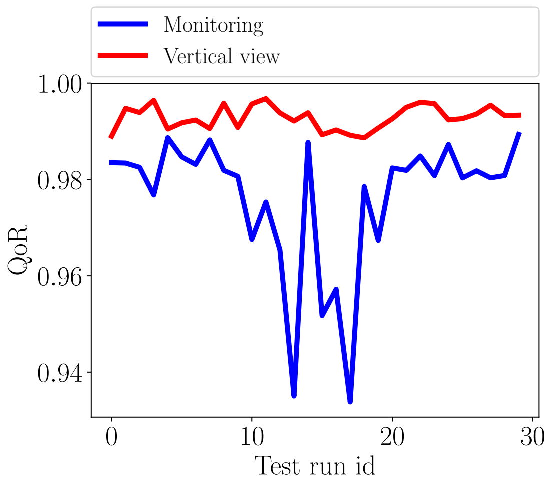


Figure 9.6: Improvement by applying vertical accuracy-aware methods to the *hull* task.



(a) Execution time.



(b) QoR.

Figure 9.7: A vertical accuracy-aware view on the feature-based classification pipeline.

leads to a significant acceleration of the entire classification pipeline and also improves the QoR. This is again a clear statement that applying a *vertical* view is very beneficial for accuracy-aware computing.

9.5 Summary

This chapter presented novel approaches to enable a dynamically contracting and tuning of accuracy-aware applications. The first approach that was proposed leads to approximated classification decisions derived by an anytime classifier. This classifier can be interrupted at anytime and returns the current classification decisions for a number of objects. The results show that such an approach can theoretically speed up classifications, while keeping classification errors low. However, the current additional overhead during runtime to realize this approach rules out the benefits. Thus, this clearly highlights, why it is important to design low-overhead control approaches for accuracy-aware computing. Therefore, I proposed a novel low-overhead dynamic tuning approach for accuracy-aware applications in this chapter.

This approach is generally usable for different kind of approximation-tolerant applications. The presented approach considers the current system state, constraints, and the actual inputs of the task instances to tune the system. To realize input awareness into the application instance, I presented a local method that individually controls the approximation degree of all task instances during runtime. The universal control problem is represented as multiple-choice knapsack problem. Solving this problem makes it possible to react on different system states and constraints during runtime. The selected pre-determined global configurations include the knowledge to control the different task instances. Furthermore, I showed that the control approach is also valuable for a vertical tuning of accuracy-aware methods.

Part IV

Summary and Outlook

CHAPTER
TEN

CONCLUSION AND OUTLOOK

This final chapter summarizes the findings of this thesis and provides a conclusion. Then, it proposes potential directions for future work in this field.

10.1 Summary and Conclusion

This thesis aims at introducing a novel methodology to realize accuracy-aware computer systems. To achieve this aim, it introduced innovative solutions to four research objectives (see Section 3.1):

Objective 1: Finding a solution for the problem of providing a runtime control approach for tuning approximation-tolerant applications;

Objective 2: Investigating approaches for tuning applications that require high-precision arithmetic;

Objective 3: Increasing the applicability scope of accuracy-aware computing; and

Objective 4: Determining innovative accuracy-aware methods that are generally applicable for approximation-tolerant applications.

In this thesis, I introduced a novel *adaptive accuracy-aware approach across system layers* to address Objective 1. This approach is an innovative way to realize accuracy-aware systems, combining accuracy-aware methods in different layers to tune a system towards conventional design goals such as performance and energy consumption. This thesis demonstrated the advantages of combining different accuracy-aware methods for a numerical task solving a system of linear equations. The algorithm used for this task was the well-known Jacobi method, and the corresponding evaluation is the first extensive consideration of accuracy-aware methods for a scientific algorithm.

By combining loop truncation, loop tiling, and relaxed synchronization, I achieved performance improvement of 300% for 99% QoR and 600% for 90% QoR. The QoR evaluation demonstrated the value of additional methods, such as approximate DRAM.

To transfer an application to an accuracy-aware application, a programmer integrates accuracy-aware methods from various layers. Integrating many of these methods into an application results in a large configuration space and using conventional methods such as parallelization and DVFS further increases that configuration space (which represents all possible combinations and respective parameter settings of the integrated methods). The configuration layer introduced in this thesis aims to determine a (near-)optimal configuration during runtime. This configuration satisfies the given constraints and provides the best design values for the remaining objectives.

The thesis introduced an innovative way to reduce the effort for finding such a (near-)optimal configuration. The proposed approach combines a hierarchical method with a design-time step and runtime mechanism. This hierarchical method considers an application as composed of tasks and determines near-Pareto-optimal configurations between QoR and budget for each task during the design-time step. This procedure essentially filters out useless configurations that would have been considered as possible global configurations during runtime, which tunes the entire application. This results in a merged performance profile showing the near-Pareto-optimal local configurations extracted for each task; the local configurations represent the useful combinations of methods and respective parameter settings for a task. As the budget metric, the thesis used execution time, energy consumption, or a combination; the latter makes it possible to consider multiple objectives (execution time, energy consumption, and QoR) for tuning. The hierarchical method has the additional benefit that when the task is plugged into another application, the same merged performance profile can still be used.

Since a suitable global configuration depends on the system state and constraints that can vary, a global configuration must be determined during runtime. To significantly reduce the set of global configurations considered during runtime, near-Pareto-optimal global configurations between the budget metric and global QoR must be extracted during design-time by the presented static tuning approach. This tuning approach assumes the highest local QoR improvement for a certain budget leads to the highest global QoR improvement, and thus it uses the information within the generated merged performance profiles. This approach significantly reduces the computational effort to extract suitable global configurations by reducing the number of global configurations considered to find a suitable one during the design-time step. Compared to an exhaustive search, it reduces effort of extracting a near-optimal configuration for an image application by a factor of 245,000; compared to state-of-the-art algorithms by a factor of 75. The quality of the global configurations found is almost the same, as shown by a study.

During runtime, a tuning mechanism uses the set of extracted global configurations to proactively adapt the system. Depending on the system state and given constraints, a configuration is chosen from this set. With multiple application instances or different accuracy-aware applications running in the system, finding a global configuration for all instances and applications represents a multiple-choice knapsack problem. Solving this problem during runtime leads to a holistic configuration of the entire system. To reduce the effort needed to solve this problem, I introduced an approximation algorithm that drastically reduces computational effort compared to exact methods while keeping the quality of the extracted solutions close to optimal when the set of possible configurations is the same for each application instance. On a local level, a granted budget can be transferred to a parameter setting for the accuracy-aware task.

Moreover, I demonstrated the importance of low-overhead solutions for input-aware approaches to avoid ruling out the benefits of accuracy-aware methods. The anytime scheduling algorithm presented provides performance improvements, which are lost when accounting for the introduced overhead to update the control data. Therefore, I proposed a low-overhead approach based on a *budget model* estimating the required budget for an accuracy-aware task and an input to reach a desired local QoR. Additionally, using an internal *monitoring* approach, a task aborts and returns the result when the desired QoR is presumably reached, which further reduces computational effort. Introducing a per-input tuning of the accuracy-aware system improves the application's global average QoR or reduces computational effort while maintaining the same QoR as the non-input-aware runtime mechanism. The policy applied for this thesis provides input awareness on the local level.

I applied my innovative methodology for realizing accuracy-aware systems to a data processing application used in sensor-based sorting. I was able to accelerate execution by a factor of 4 while providing nearly the same QoR compared to the conventional application. My approach behaves proactively and thus reacts during high-workload situations. The approach adapts the multiple-choice knapsack problem constraints to handle these situations, extracting a different holistic configuration. Whereas the conventional application has many deadline violations, my control mechanism showed none while slightly reducing the QoR for these extreme cases. Additionally, using the *vertical* view for tuning further improved the entire solution.

This thesis also considered an adaptive approach for applications requiring greater precision than offered by current hardware. Greater precision can overcome numerical instabilities caused by computational errors (Objective 2). I showed that greater precision leads to calculating more correct eigenvalues using the Lanczos method, but it introduces significant overhead to the application. It is important to use domain knowledge to implement a more stable variant of a numerical algorithm, thus requiring less additional precision and significantly reducing overhead. The required additional precision depends on the input, and thus a fixed setting for precision is sub-optimal. To that end, determining how to set the internal precision for an input matrix remains an open question. Thus, this thesis did not further consider tuning accuracy up inside a system.

To increase the applicability scope of accuracy-aware computing (Objective 3), I developed new accuracy-aware methods on various layers for different domains (Objective 4): computational biology, stereo vision, time series analysis, and sensor-based sorting.

On the hardware level, I presented a conversion unit that converts floating-point values before transferring these values to memory. Integrating such a unit into a RISC-V processor reduces energy consumption for a 2D fast Fourier transformation by a factor of 2.5 while having almost no impact on the QoR. A 2D convolution required 26% less execution time and such a conversion unit-based method provides a higher QoR than performing the application using a smaller data type.

To increase design efficiency for hardware accelerators and approximation-tolerant applications, I introduced a set of five approximation design patterns for dynamic programming algorithms; such algorithms are used for stereo vision, computational biology, and time series analysis. Exploiting these patterns resulted in a performance of 157 giga cell updates per second for an adapted version of the Smith Waterman algorithm outperforming a software version exploiting the vector extension units of an x86 processor. Furthermore, these patterns made it possible to port a dynamic programming-based stereo vision algorithm to a low resource FPGA while maintaining the same performance and having a better QoR than local stereo vision algorithms. For time series analysis, the patterns improve the performance for classifications by a factor of up to 29.4 compared to complex similarity measures while having a higher QoR than simple measures.

I designed two innovative approaches for general and useful accuracy-aware methods at the software level. While the novel fuzzy memoization technique is generally applicable, the introduced overhead is too high for most approximation-tolerant applications. Therefore, I proposed a method based on contract algorithms, as well as described best practices on how to transfer tasks into contract-based tasks and how to control them. Moreover, I presented innovative methods that adapt the ordering of input items to provide a better performance profile. Such tasks allow a user to continue execution for situations in which the QoR is insufficient and thus reduces overhead to reach a higher QoR compared to re-executing the task.

To conclude this thesis, I present the following key takeaway: Accuracy-aware computing provides an innovative way for system designers and programmers to reach certain design goals. I demonstrated that a one-size-fits-all strategy is not useful for accuracy-aware computer systems. Therefore, I presented a methodology for realizing accuracy-aware systems. The introduced methodology is a step towards controlling a system during runtime by tuning accuracy-aware and conventional methods according to the current system state, constraints, and inputs. Introducing input sensitivity is a milestone to increase the scope of accuracy-aware computing.

10.2 Outlook and Future Work

The hierarchical approach used to find global configurations at design time shifts the highest computational effort towards generating performance profiles for the task. Here, QoR must be determined for each parameter setting and budget by using a set of representative input data. This is achievable for the tasks considered in this thesis but generating local profiles can be a bottleneck when there are many accuracy-aware methods and corresponding parameter settings. Therefore, reducing the effort required to build merged profiles should be the subjective of future research.

Certain programming effort is currently required to integrate accuracy-aware methods into an application. The programmer must identify the tasks and transform them into an accuracy-aware task. This thesis presented straightforward methods to realize such a transformation, including the introduction of a mechanism to control the budget and best practices to implement contract-based tasks. In the future, this effort should be reduced by an automatic solution, which requires identifying suitable tasks, integrating accuracy-aware methods into the task, and building local merged performance profiles. For contract-based tasks, an easy solution would be to integrate a timer into a loop and to exit the loop as soon as the value of the timer is larger than the budget.

Future work should also identify innovative solutions on introducing input awareness into the system. For the tasks considered in this thesis, it was sufficient to use M5 models for the budget model and linear regression for the monitor. However, more complex tasks - especially in the domain of high-precision arithmetic - require more complex solutions than identified by this thesis because the correlation between input, parameter setting, and QoR is much more complex. Notably, this is not only a matter of improving the methods' accuracy, since computational effort must be low to benefit the accuracy-aware applications. Moreover, a completely automatic method to build the budget model and the monitor per task, and to integrate them into the application, would reduce the effort required of the programmer.

This thesis assumed that achieving a local desired QoR per task for a particular input results in the desired global QoR, which works well on the applications considered. However, there could be situations in which different local QoRs are required to achieve the same global QoR for different inputs. Hence, it would be useful to have a mechanism that estimates the local QoRs based on the current application input during runtime.

The AC design patterns for dynamic programming algorithms presented in this thesis are efficient for different domains, but not all patterns are as valuable as others for the various domains. Since these patterns can also be applied on a higher level, such as OpenCL, automatic design space exploration could identify useful AC patterns. Similar AC patterns could also be identified for other algorithms. Ideally, this would mean general AC design patterns to be considered in automatic design space exploration.

Accuracy-aware methods targeting memories, caches, or data transfers are considered individually in the literature. There are multiple expected advantages to combining multiple methods, but this has yet to be explored. For instance, the conversion unit could be combined with the load-value approximation approach and techniques for higher memory levels such as approximate compressing techniques or increasing the refresh cycle time for DRAMs.

BIBLIOGRAPHY

[1] Robert H. Dennard, Jin Cai, and Arvind Kumar. “A perspective on today’s scaling challenges and possible future directions”. In: *Solid-State Electronics* 51.4 (2007). Special Issue: Papers selected from the 2006 {ULIS} Conference, pp. 518–525. ISSN: 0038-1101.

[2] G. E. Moore. “Cramming More Components onto Integrated Circuits”. In: *Electronics* 38.8 (Apr. 1965), pp. 114–117. ISSN: 0018-9219.

[3] Hadi Esmaeilzadeh, Emily Blem, et al. “Dark silicon and the end of multicore scaling”. In: *ACM SIGARCH Computer Architecture News*. Vol. 39. ISCA ’11 3. ACM. San Jose, California, USA: ACM, 2011, pp. 365–376. ISBN: 978-1-4503-0472-6.

[4] Muhammad Shafique, Siddharth Garg, et al. “The EDA Challenges in the Dark Silicon Era: Temperature, Reliability, and Variability Perspectives”. In: *Proceedings of the 51st Annual Design Automation Conference*. DAC ’14. San Francisco, CA, USA: ACM, 2014, 185:1–185:6. ISBN: 978-1-4503-2730-5.

[5] Mark D. Hill and Michael R. Marty. “Amdahl’s law in the multicore era”. In: *Computer* 41.7 (July 2008), pp. 33–38. ISSN: 0018-9162.

[6] V. Chippa, S. Chakradhar, et al. “Analysis and characterization of inherent application resilience for approximate computing”. In: *50th ACM/EDAC/IEEE Design Automation Conference (DAC)*. May 2013, pp. 1–9.

[7] Jie Han and M. Orshansky. “Approximate computing: An emerging paradigm for energy-efficient design”. In: *2013 18th IEEE European Test Symposium (ETS)*. May 2013, pp. 1–6.

[8] Song Liu, Karthik Pattabiraman, et al. “Flikker: Saving DRAM Refresh-power Through Critical Data Partitioning”. In: *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Lan-*

guages and Operating Systems. ASPLOS XVI. Newport Beach, California, USA: ACM, 2011, pp. 213–224. ISBN: 978-1-4503-0266-1.

[9] Bert Moons and Marian Verhelst. “DVAS: Dynamic voltage accuracy scaling for increased energy-efficiency in approximate computing”. In: *IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE. 2015, pp. 237–242.

[10] Hadi Esmaeilzadeh, Adrian Sampson, et al. “Architecture Support for Disciplined Approximate Programming”. In: *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS XVII. London, England, UK: ACM, 2012, pp. 301–312. ISBN: 978-1-4503-0759-8.

[11] Mehrzad Samadi, Janghaeng Lee, et al. “SAGE: Self-tuning Approximation for Graphics Engines”. In: *46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. ACM, 2013, pp. 13–24. ISBN: 978-1-4503-2638-4.

[12] Atieh Lotfi, Abbas Rahimi, et al. “Grater: An approximation workflow for exploiting data-level parallelism in FPGA acceleration”. In: *Proceedings of the Conference on Design, Automation & Test in Europe (DATE)*. EDA Consortium. 2016, pp. 1279–1284.

[13] Stelios Sidiroglou-Douskos, Sasa Misailovic, et al. “Managing performance vs. accuracy trade-offs with loop perforation”. In: *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. ACM. 2011, pp. 124–134.

[14] Mehrzad Samadi, Davoud Anoushe Jamshidi, et al. “Paraprox: Pattern-based Approximation for Data Parallel Applications”. In: *SIGPLAN Notices* 49.4 (Feb. 2014), pp. 35–50. ISSN: 0362-1340.

[15] Hadi Esmaeilzadeh, Adrian Sampson, et al. “Neural Acceleration for General-Purpose Approximate Programs”. In: *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO-45. Vancouver, B.C., CANADA: IEEE Computer Society, 2012, pp. 449–460. ISBN: 978-0-7695-4924-8.

[16] Cindy Rubio-González, Cuong Nguyen, et al. “Precimonious: Tuning assistant for floating-point precision”. In: *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. ACM. 2013, p. 27.

[17] Woongki Baek and Trishul M Chilimbi. “Green: a framework for supporting energy-conscious programming using controlled approximation”. In: *ACM Sigplan Notices*. Vol. 45. 6. ACM. 2010, pp. 198–209.

[18] Adrian Sampson, Werner Dietl, et al. “EnerJ: Approximate data types for safe and general low-power computation”. In: *ACM SIGPLAN Notices*. Vol. 46. 6. ACM. 2011, pp. 164–174.

[19] Lakshminarayanan Renganarayana, Vijayalakshmi Srinivasan, et al. “Programming with relaxed synchronization”. In: *Proceedings of the 2012 ACM workshop on Relaxing synchronization for multicore and manycore scalability*. ACM. 2012, pp. 41–50.

[20] Subrata Mitra, Manish K Gupta, et al. “Phase-aware optimization in approximate computing”. In: *2017 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. IEEE. 2017, pp. 185–196.

[21] Henry Hoffmann. “JouleGuard: Energy Guarantees for Approximate Applications”. In: *Proceedings of the 25th Symposium on Operating Systems Principles*. SOSP ’15. Monterey, California: ACM, 2015, pp. 198–214. ISBN: 978-1-4503-3834-9.

[22] H. Hoffmann. “CoAdapt: Predictable Behavior for Accuracy-Aware Applications Running on Power-Aware Systems”. In: *26th Euromicro Conference on Real-Time Systems (ECRTS)*. July 2014, pp. 223–232.

[23] Daya Shanker Khudia, Babak Zamirai, et al. “Rumba: an online quality management system for approximate computing”. In: *Proceedings of the 42nd Annual International Symposium on Computer Architecture, Portland, OR, USA, June 13-17, 2015*. 2015, pp. 554–566.

[24] Divya Mahajan, Amir Yazdanbakhsh, et al. “Prediction-based quality control for approximate accelerators”. In: *Workshop on Approximate Computing Across the System Stack*. 2015.

[25] Michael A. Laurenzano, Parker Hill, et al. “Input Responsiveness: Using Canary Inputs to Dynamically Steer Approximation”. In: *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI ’16. Santa Barbara, CA, USA: ACM, 2016, pp. 161–176. ISBN: 978-1-4503-4261-2.

[26] Arnab Raha, Swagath Venkataramani, et al. “Energy-Efficient Reduce-and-Rank Using Input-Adaptive Approximations”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 25.2 (2017), pp. 462–475.

[27] David H Bailey, Roberto Barrio, and Jonathan M Borwein. “High-precision computation: Mathematical physics and dynamics”. In: *Applied Mathematics and Computation* 218.20 (2012), pp. 10106–10121.

[28] Alexander Backlund. “The definition of system”. In: *Kybernetes* 29.4 (June 2000), pp. 444–451. ISSN: 0368-492X.

[29] Benkt Wangler and Alexander Backlund. “Information Systems Engineering: What Is It?” In: *Proceedings of the Workshops at 17th International Conference on Advanced Information Systems Engineering CAiSE, Porto, Portugal, June 13-17, CAiSE’05 Workshops, Vol. 2*. 2005, pp. 427–437.

[30] Anita K. Jones. “Editor’s Introduction”. In: *ACM Transactions on Computer Systems (TOCS)* 1.1 (Feb. 1983), pp. 1–2. ISSN: 0734-2071.

[31] Michel Dubois, Murali Annavaram, and Per Stenström. *Parallel Computer Organization and Design*. New York, NY, USA: Cambridge University Press, 2012. ISBN: 9780521886758.

[32] John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2011.

[33] John Gantz and David Reinsel. *THE DIGITAL UNIVERSE IN 2020: Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East*. Tech. rep. International Data Corporation (IDC) Go-to-Market Services, Dec. 2012. URL: `http://idcdocserv.com/1414`.

[34] Krste Asanovic, Ras Bodik, et al. *The landscape of parallel computing research: A view from berkeley*. Tech. rep. UCB/EECS-2006-183, EECS Department, University of California, Berkeley, 2006.

[35] David A Patterson and John L Hennessy. *Computer organization and design: the hardware/software interface*. Newnes, 2013.

[36] David J. Lilja. *Measuring Computer Performance: A Practitioner’s Guide*. Cambridge University Press, 2005. ISBN: 0521646707.

[37] John L Henning. “SPEC CPU2006 benchmark descriptions”. In: *ACM SIGARCH Computer Architecture News* 34.4 (2006), pp. 1–17.

[38] Christian Bienia. “Benchmarking Modern Multiprocessors”. PhD thesis. Princeton University, Jan. 2011.

[39] Shuai Che, Michael Boyer, et al. “Rodinia: A benchmark suite for heterogeneous computing”. In: *IEEE International Symposium on Workload Characterization (IISWC)*. IEEE. 2009, pp. 44–54.

[40] Robert H Dennard, Fritz H Gaensslen, et al. “Design of ion-implanted MOSFET’s with very small physical dimensions”. In: *IEEE Journal of Solid-State Circuits* 9.5 (1974), pp. 256–268.

[41] Wm A Wulf and Sally A McKee. “Hitting the memory wall: implications of the obvious”. In: *ACM SIGARCH computer architecture news* 23.1 (1995), pp. 20–24.

[42] Brian Hayes. “Computing Science: Computing in a Parallel Universe”. In: *American Scientist* 95.6 (2007), pp. 476–480.

[43] Markus Kowarschik and Christian Weiß. "An overview of cache optimization techniques and cache-aware numerical algorithms". In: *Algorithms for Memory Hierarchies*. Springer, 2003, pp. 213–232.

[44] Kurt Mehlhorn and Peter Sanders. *Algorithms and data structures: The basic toolbox*. Springer Science & Business Media, 2008.

[45] Matteo Frigo, Charles E Leiserson, et al. "Cache-oblivious algorithms". In: *40th Annual Symposium on Foundations of Computer Science*. IEEE. 1999, pp. 285–297.

[46] Maria Malik, Setareh Rafatirah, et al. "System and architecture level characterization of big data applications on big and little core server architectures". In: *IEEE International Conference on Big Data (Big Data)*. IEEE. 2015, pp. 85–94.

[47] Vivek De and Shekhar Borkar. "Technology and Design Challenges for Low Power and High Performance". In: *Proceedings of the 1999 International Symposium on Low Power Electronics and Design*. ISLPED '99. San Diego, California, USA: ACM, 1999, pp. 163–168. ISBN: 1-58113-133-X.

[48] Nikos Hardavellas, Michael Ferdman, et al. "Toward Dark Silicon in Servers". In: *IEEE Micro* 31.4 (July 2011), pp. 6–15. ISSN: 0272-1732.

[49] Li Tan, Zizhong Chen, and Shuaiwen Leon Song. "Scalable Energy Efficiency with Resilience for High Performance Computing Systems: A Quantitative Methodology". In: *ACM Transactions on Architecture and Code Optimization* 12.4 (Nov. 2015), 35:1–35:27. ISSN: 1544-3566.

[50] Yavuz Yetim, Sharad Malik, and Margaret Martonosi. "EPROF: An energy/performance/reliability optimization framework for streaming applications". In: *17th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE. 2012, pp. 769–774.

[51] Gene M. Amdahl. "Validity of the single processor approach to achieving large scale computing capabilities". In: *AFIPS spring joint computer conference*. 1967.

[52] Sparsh Mittal. "A survey of techniques for improving energy efficiency in embedded computing systems". In: *International Journal of Computer Aided Engineering and Technology* 6.4 (2014), pp. 440–459.

[53] Canturk Isci, Alper Buyuktosunoglu, et al. "An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget". In: *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO 39. Washington, DC, USA: IEEE Computer Society, 2006, pp. 347–358. ISBN: 0-7695-2732-9.

[54] Intel Cooperation. “Intel Turbo Boost Technology in Intel® Core™ Microarchitecture (Nehalem) Based Processors (white paper)”. In: (Nov. 2008).

[55] James Charles, Preet Jassi, et al. “Evaluation of the Intel® Core™ i7 Turbo Boost feature”. In: *IEEE International Symposium on Workload Characterization (IISWC)*. IEEE. 2009, pp. 188–197.

[56] Andre R. Brodtkorb, Christopher Dyken, et al. “State-of-the-art in Heterogeneous Computing”. In: *Scitific Programming* 18.1 (Jan. 2010), pp. 1–33. ISSN: 1058-9244.

[57] Rainer Buchty, Vincent Heuveline, et al. “A survey on hardware-aware and heterogeneous computing on multicore processors and accelerators”. In: *Concurrency and Computation: Practice and Experience* 24.7 (2012), pp. 663–675. ISSN: 1532-0634.

[58] Javier Diaz, Camelia Munoz-Caro, and Alfonso Nino. “A survey of parallel programming models and tools in the multi and many-core era”. In: *IEEE Transactions on parallel and distributed systems* 23.8 (2012), pp. 1369–1386.

[59] Scott Hauck and Andre DeHon. *Reconfigurable computing: the theory and practice of FPGA-based computation*. Vol. 1. Morgan Kaufmann, 2010.

[60] Kenneth Pocek, Russell Tessier, and André DeHon. “Birth and adolescence of reconfigurable computing: A survey of the first 20 years of field-programmable custom computing machines”. In: *Highlights of the First Twenty Years of the IEEE International Symposium on Field-Programmable Custom Computing Machines*. 2013, pp. 3–19.

[61] Sparsh Mittal and Jeffrey S. Vetter. “A Survey of CPU-GPU Heterogeneous Computing Techniques”. In: *ACM Computing Surveys* 47.4 (July 2015), 69:1–69:35. ISSN: 0360-0300.

[62] Sparsh Mittal and Jeffrey S. Vetter. “A Survey of Methods for Analyzing and Improving GPU Energy Efficiency”. In: *ACM Computing Surveys* 47.2 (Aug. 2014), 19:1–19:23. ISSN: 0360-0300.

[63] Brahim Betkaoui, David B Thomas, and Wayne Luk. “Comparing performance and energy efficiency of FPGAs and GPUs for high productivity computing”. In: *International Conference on Field-Programmable Technology (FPT)*. IEEE. 2010, pp. 94–101.

[64] Jeremy Fowers, Greg Brown, et al. “A performance and energy comparison of convolution on GPUs, FPGAs, and multicore processors”. In: *ACM Transactions on Architecture and Code Optimization (TACO)* 9.4 (2013), p. 25.

[65] Shuai Che, Jie Li, et al. “Accelerating compute-intensive applications with GPUs and FPGAs”. In: *Symposium on Application Specific Processors (SASP)*. IEEE. 2008, pp. 101–107.

[66] Mohammad Khavari Tavana, Mohammad Hossein Hajkazemi, et al. “ElasticCore: enabling dynamic heterogeneity with joint core and voltage/frequency scaling”. In: *Proceedings of the 52nd Annual Design Automation Conference*. ACM. 2015, p. 151.

[67] N. Mitchell, E. Schonberg, and G. Sevitsky. “Four Trends Leading to Java Runtime Bloat”. In: *IEEE Software* 27.1 (Jan. 2010), pp. 56–63. ISSN: 0740-7459.

[68] Matthew Hertz, Yi Feng, and Emery D Berger. “Garbage collection without paging”. In: *ACM SIGPLAN Notices*. Vol. 40. 6. ACM. 2005, pp. 143–153.

[69] Guoqing Xu, Nick Mitchell, et al. “Software bloat analysis: finding, removing, and preventing performance problems in modern large-scale object-oriented applications”. In: *Proceedings of the FSE/SDP workshop on Future of software engineering research*. ACM. 2010, pp. 421–426.

[70] Erik Altman, Matthew Arnold, et al. “Performance analysis of idle programs”. In: *ACM Sigplan Notices*. Vol. 45. 10. ACM. 2010, pp. 739–753.

[71] Guoqing Xu, Michael D. Bond, et al. “LeakChaser: Helping Programmers Narrow Down Causes of Memory Leaks”. In: *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI ’11. San Jose, California, USA: ACM, 2011, pp. 270–282. ISBN: 978-1-4503-0663-8.

[72] Nick Mitchell and Gary Sevitsky. “The causes of bloat, the limits of health”. In: *ACM SIGPLAN Notices*. Vol. 42. 10. ACM. 2007, pp. 245–260.

[73] Suparna Bhattacharya, Mangala Gowri Nanda, et al. “Reuse, Recycle to De-bloat Software”. In: *Proceedings of the 25th European Conference Object-Oriented Programming*. Ed. by Mira Mezini. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 408–432. ISBN: 978-3-642-22655-7.

[74] Guoqing Xu. “Finding Reusable Data Structures”. In: *SIGPLAN Notices* 47.10 (Oct. 2012), pp. 1017–1034. ISSN: 0362-1340.

[75] Linhai Song and Shan Lu. “Statistical Debugging for Real-world Performance Problems”. In: *SIGPLAN Notices* 49.10 (Oct. 2014), pp. 561–578. ISSN: 0362-1340.

[76] Adrian Nistor, Linhai Song, et al. “Toddler: Detecting Performance Problems via Similar Memory-access Patterns”. In: *Proceedings of the 2013 International Conference on Software Engineering*. ICSE ’13. San Francisco, CA, USA: IEEE Press, 2013, pp. 562–571. ISBN: 978-1-4673-3076-3.

[77] D. Maplesden, E. Tempero, et al. “Performance Analysis for Object-Oriented Software: A Systematic Mapping”. In: *IEEE Transactions on Software Engineering* 41.7 (July 2015), pp. 691–710. ISSN: 0098-5589.

[78] Jeremy Lau, Matthew Arnold, et al. “Online Performance Auditing: Using Hot Optimizations Without Getting Burned”. In: *SIGPLAN Notices* 41.6 (June 2006), pp. 239–251. ISSN: 0362-1340.

[79] K. Seymour, H. You, and J. Dongarra. “A comparison of search heuristics for empirical code optimization”. In: *IEEE International Conference on Cluster Computing*. Sept. 2008, pp. 421–429.

[80] Richard W. Vuduc. “Autotuning”. In: *Encyclopedia of Parallel Computing*. Ed. by David Padua. Boston, MA: Springer US, 2011, pp. 102–105. ISBN: 978-0-387-09766-4.

[81] Tomofumi Yuki, Lakshminarayanan Renganarayanan, et al. “Automatic Creation of Tile Size Selection Models”. In: *Proceedings of the 8th Annual IEEE/ACM International Symposium on Code Generation and Optimization*. CGO ’10. Toronto, Ontario, Canada: ACM, 2010, pp. 190–199. ISBN: 978-1-60558-635-9.

[82] C. A. Schaefer. “Reducing search space of auto-tuners using parallel patterns”. In: *2009 ICSE Workshop on Multicore Software Engineering*. May 2009, pp. 17–24.

[83] Jason Ansel, Shoaib Kamil, et al. “OpenTuner: An Extensible Framework for Program Autotuning”. In: *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation*. PACT ’14. Edmonton, AB, Canada: ACM, 2014, pp. 303–316. ISBN: 978-1-4503-2809-8.

[84] Lianjie Luo, Yang Chen, et al. “Finding representative sets of optimizations for adaptive multiversioning applications”. In: *arXiv preprint arXiv:1407.4075* (2014).

[85] M Aater Suleman, Moinuddin K Qureshi, and Yale N Patt. “Feedback-driven threading: power-efficient and high-performance execution of multi-threaded workloads on CMPs”. In: *ACM Sigplan Notices*. Vol. 43. 3. ACM. 2008, pp. 277–286.

[86] Matthew Curtis-Maury, Filip Blagojevic, et al. “Prediction-based power-performance adaptation of multithreaded scientific codes”. In: *IEEE Transactions on Parallel and Distributed Systems* 19.10 (2008), pp. 1396–1410.

[87] Dong Li, Bronis R de Supinski, et al. “Hybrid MPI/OpenMP power-aware computing”. In: *IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*. IEEE. 2010, pp. 1–12.

[88] Ronald G Dreslinski, Michael Wieckowski, et al. “Near-threshold computing: Reclaiming moore’s law through energy efficient integrated circuits”. In: *Proceedings of the IEEE* 98.2 (2010), pp. 253–266.

[89] Michael Hübner and Cristina Silvano. *Near Threshold Computing: Technology, Methods and Applications*. 1st. Springer Publishing Company, Incorporated, 2015. ISBN: 9783319233888.

[90] Cor Claeys, Danielle Chiappe, et al. “(Invited) Advanced Semiconductor Devices for Future CMOS Technologies”. In: *ECS Transactions* 66.5 (2015), pp. 49–60.

[91] Dmitri E Nikonov and Ian A Young. “Overview of beyond-CMOS devices and a uniform methodology for their benchmarking”. In: *Proceedings of the IEEE* 101.12 (2013), pp. 2498–2533.

[92] Kelin J Kuhn, Uygar Avci, et al. “The ultimate CMOS device and beyond”. In: *IEEE International Electron Devices Meeting (IEDM)*. IEEE. 2012, pp. 1–8.

[93] An Chen, James Hutchby, et al., eds. *Emerging Nanoelectronic Devices*. Chichester, United Kingdom: John Wiley & Sons Ltd, Dec. 2014. ISBN: 9781118958254.

[94] Ian A Young, Uygar E Avci, and Daniel H Morris. “Tunneling field effect transistors: Device and circuit considerations for energy efficient logic opportunities”. In: *IEEE International Electron Devices Meeting (IEDM)*. IEEE. 2015, pp. 1–22.

[95] Richard Martel, T Schmidt, et al. “Single-and multi-wall carbon nanotube field-effect transistors”. In: *Applied Physics Letters* 73.17 (1998), pp. 2447–2449.

[96] An Chen. “A review of emerging non-volatile memory (NVM) technologies and applications”. In: *Solid-State Electronics* 125 (2016). Extended papers selected from {ESSDERC} 2015, pp. 25–38. ISSN: 0038-1101.

[97] KL Wang, JG Alzate, and P Khalili Amiri. “Low-power non-volatile spintronic memory: STT-RAM and beyond”. In: *Journal of Physics D: Applied Physics* 46.7 (2013), p. 074003.

[98] Hyper Memory Cube Consortium. *HMC Specification 2.1*. Hyper Memory Cube Consortium, 2014.

[99] John H Lau. “Overview and Outlook of 3D IC Packaging, 3D Si Integration, and 3D IC Integration”. In: *Journal of Electronic Packaging* 136.4 (2014).

[100] Jeffrey P. Gambino, Shawn A. Adderly, and John U. Knickerbocker. “An overview of through-silicon-via technology and manufacturing challenges”. In: *Microelectronic Engineering* 135 (2015), pp. 73–106. ISSN: 0167-9317.

[101] Jishen Zhao, Qiaosha Zou, and Yuan Xie. “Overview of 3-D Architecture Design Opportunities and Techniques”. In: *IEEE Design & Test* 34.4 (2017), pp. 60–68.

[102] Vinton G. Cerf. “Unconventional Computing”. In: *Communications of the ACM* 57.10 (Sept. 2014), pp. 7–7. ISSN: 0001-0782.

[103] Ivan K. Schuler and Rick Stevens. *Neuromorphic Computing: From Materials to Systems Architecture - Report of a Roundtable Convened to Consider Neuromorphic Computing Basic Research Needs*. Tech. rep. U.S. Department of Energy, Office on Science, Oct. 2015.

[104] Andrew Nere, Atif Hashmi, et al. “Bridging the semantic gap: Emulating biological neuronal behaviors with simple digital neurons”. In: *IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE. 2013, pp. 472–483.

[105] Steven K. Esser, Paul A. Merolla, et al. “Convolutional networks for fast, energy-efficient neuromorphic computing”. In: *Proceedings of the National Academy of Sciences* 113.41 (2016), pp. 11441–11446. eprint: `http://www.pnas.org/content/113/41/11441.full.pdf`.

[106] F. L. Traversa and M. Di Ventra. “Universal Memcomputing Machines”. In: *IEEE Transactions on Neural Networks and Learning Systems* 26.11 (Nov. 2015), pp. 2702–2715. ISSN: 2162-237X.

[107] James Mnatzaganian, Ernest Fokoué, and Dhireesha Kudithipudi. “A Mathematical Formalization of Hierarchical Temporal Memory Cortical Learning Algorithm’s Spatial Pooler”. In: *arXiv preprint arXiv:1601.06116* (2016).

[108] D. S. Holmes, A. M. Kadin, and M. W. Johnson. “Superconducting Computing in Large-Scale Hybrid Systems”. In: *Computer* 48.12 (Dec. 2015), pp. 34–42. ISSN: 0018-9162.

[109] J Joshua Yang, Dmitri B Strukov, and Duncan R Stewart. “Memristive devices for computing”. In: *Nature nanotechnology* 8.1 (2013), pp. 13–24.

[110] Armin Alaghi and John P Hayes. “Survey of stochastic computing”. In: *ACM Transactions on Embedded computing systems (TECS)* 12.2s (2013), p. 92.

[111] Hyungmin Cho, Larkhoon Leem, and Subhasish Mitra. “ERSA: Error resilient system architecture for probabilistic applications”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31.4 (2012), pp. 546–558.

[112] K. Ohashi, K. Nishi, et al. “On-Chip Optical Interconnect”. In: *Proceedings of the IEEE* 97.7 (July 2009), pp. 1186–1198. ISSN: 0018-9219.

[113] A. Karkar, T. Mak, et al. “A Survey of Emerging Interconnects for On-Chip Efficient Multicast and Broadcast in Many-Cores”. In: *IEEE Circuits and Systems Magazine* 16.1 (Jan. 2016), pp. 58–72. ISSN: 1531-636X.

[114] Olaf Neugebauer, Peter Marwedel, et al. “Quality Evaluation Strategies for Approximate Computing in Embedded Systems”. In: *Technological Innovation for Smart Systems*. Ed. by Luis M. Camarinha-Matos, Mafalda Parreira-Rocha, and Javaneh Ramezani. Cham: Springer International Publishing, 2017, pp. 203–210. ISBN: 978-3-319-56077-9.

[115] Ismail Akturk, Karen Khatamifard, and Ulya R Karpuzcu. “On quantification of accuracy loss in approximate computing”. In: *Workshop on Duplicating, Deconstructing and Debunking (WDDD)*. 2015, p. 15.

[116] Qiang Xu, Nam Sung Kim, and T. Mytkowicz. “Approximate Computing: A Survey”. In: *IEEE Design Test* 33.1 (Feb. 2016), pp. 8–22. ISSN: 2168-2356.

[117] Christian Plessl, Marco Platzner, and Peter J. Schreier. “Approximate Computing”. German. In: *Informatik-Spektrum* 38.5 (2015), pp. 396–399. ISSN: 0170-6012.

[118] Daniele Jahier Pagliari, Massimo Poncino, and Enrico Macii. “Energy-Efficient Digital Processing via Approximate Computing”. In: *Smart Systems Integration and Simulation*. Ed. by Nicola Bombieri, Massimo Poncino, and Graziano Pravadelli. Cham: Springer International Publishing, 2016, pp. 55–89. ISBN: 978-3-319-27392-1.

[119] Sparsh Mittal. “A Survey of Techniques for Approximate Computing”. In: *ACM Comput. Surv.* 48.4 (Mar. 2016), 62:1–62:33. ISSN: 0360-0300.

[120] S. Venkataramani, S. T. Chakradhar, et al. “Approximate computing and the quest for computing efficiency”. In: *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. June 2015, pp. 1–6.

[121] Lukas Sekanina. “Introduction to approximate computing: Embedded tutorial”. In: *IEEE 19th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*. IEEE. 2016, pp. 1–6.

[122] J. Henkel. “Approximate Computing: Solving Computing’s Inefficiency Problem?” In: *IEEE Design Test* 33.1 (Feb. 2016), pp. 4–5. ISSN: 2168-2356.

[123] T. Moreau, J. San Miguel, et al. “A Taxonomy of General Purpose Approximate Computing Techniques”. In: *IEEE Embedded Systems Letters* 10.1 (Mar. 2018), pp. 2–5. ISSN: 1943-0663.

[124] Amir Yazdanbakhsh, Divya Mahajan, et al. *AxBench: A Benchmark Suite for Approximate Computing Across the System Stack*. Tech. rep. Georgia Institute of Technology, 2016.

[125] Panos S Koutsovasilis, Christos Kalogirou, et al. “Affichee: A Benchmark Suite at the Meeting Point of Heterogeneous and Approximate Computing”. In: *Workshop on Parallel Programming for Resilience and Energy Efficiency at the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 2016.

[126] Peter Düben, Jeremy Schlachter, et al. “Opportunities for Energy Efficient Computing: A Study of Inexact General Purpose Processors for High-performance and Big-data Applications”. In: *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. DATE ’15. Grenoble, France: EDA Consortium, 2015, pp. 764–769. ISBN: 978-3-9815370-4-8.

[127] Inigo Goiri, Ricardo Bianchini, et al. “Approxhadoop: Bringing approximations to mapreduce frameworks”. In: *Computer Architecture News*. Vol. 43. 1. ACM. 2015, pp. 383–397.

[128] Schuyler Eldridge, Florian Raudies, et al. “Neural network-based accelerators for transcendental function approximation”. In: *Proceedings of the 24th edition of the great lakes symposium on VLSI*. ACM. 2014, pp. 169–174.

[129] Philipp Gschwandtner, Charalampos Chalios, et al. “On the potential of significance-driven execution for energy-aware HPC”. In: *Computer Science - Research and Development* 30.2 (2014), pp. 197–206. ISSN: 1865-2042.

[130] Lawrence McAfee and Kunle Olukotun. “EMEURO: A Framework for Generating Multi-purpose Accelerators via Deep Learning”. In: *Proceedings of the 13th Annual IEEE/ACM International Symposium on Code Generation and Optimization*. CGO ’15. San Francisco, California: IEEE Computer Society, 2015, pp. 125–135. ISBN: 978-1-4799-8161-8.

[131] S. Venkataramani, A. Ranjan, et al. “AxNN: Energy-efficient neuromorphic systems using approximate computing”. In: *2014 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. Aug. 2014, pp. 27–32.

[132] Q. Zhang, T. Wang, et al. “ApproxANN: An approximate computing framework for artificial neural network”. In: *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*. Mar. 2015, pp. 701–706.

[133] B. Moons, B. De Brabandere, et al. “Energy-efficient ConvNets through approximate computing”. In: *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*. Mar. 2016, pp. 1–8.

[134] Michael Figurnov, Dmitry P. Vetrov, and Pushmeet Kohli. “PerforatedCNNs: Acceleration through Elimination of Redundant Convolutions”. In: *CoRR* abs/1504.08362 (2015).

[135] M. Ayinala and K. K. Parhi. “Low-energy architectures for Support Vector Machine computation”. In: *2013 Asilomar Conference on Signals, Systems and Computers*. Nov. 2013, pp. 2167–2171.

[136] Sasa Misailovic, Daniel M. Roy, and Martin C. Rinard. “Probabilistically Accurate Program Transformations”. In: *Proceedings of the 18th International Static Analysis Symposium (SAS), Venice, Italy, September 14-16*. Ed. by Eran Yahav. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 316–333. ISBN: 978-3-642-23702-7.

[137] Jiayuan Meng, S. Chakradhar, and A. Raghunathan. “Best-effort parallel execution framework for Recognition and mining applications”. In: *IEEE International Symposium on Parallel Distributed Processing (IPDPS)*. May 2009, pp. 1–12.

[138] Sasa Misailovic, Deokhwan Kim, and Martin Rinard. “Parallelizing sequential programs with statistical accuracy tests”. In: *ACM Transactions on Embedded Computing Systems (TECS)* 12.2s (2013), p. 88.

[139] John Sartori and Rakesh Kumar. “Branch and Data Herding: Reducing Control and Memory Divergence for Error-tolerant GPU Applications”. In: *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*. PACT ’12. Minneapolis, Minnesota, USA: ACM, 2012, pp. 427–428. ISBN: 978-1-4503-1182-3.

[140] Shlomo Zilberstein and Stuart Russell. “Approximate Reasoning Using Anytime Algorithms”. In: *Imprecise and Approximate Computation*. Springer, 1995, pp. 43–62. ISBN: 978-0-7923-9579-9.

[141] Wyne Wyne Kywe, Daisuke Fujiwara, and Kazuhito Murakami. “Scheduling of image processing using anytime algorithm for real-time system”. In: *18th International Conference on Pattern Recognition (ICPR)*. Vol. 3. IEEE. 2006, pp. 1095–1098.

[142] R. Mangharam and A. A. Saba. “Anytime Algorithms for GPU Architectures”. In: *IEEE 32nd Real-Time Systems Symposium (RTSS)*. Nov. 2011, pp. 47–56.

[143] Joshua San Miguel, Ravi Nair, et al. *A systolic approach to deriving anytime algorithms for approximate computing*. Tech. rep. IBM Research Report RC25600, Tech. Rep, 2016.

[144] Joshua San Miguel and Natalie Enright Jerger. “The anytime automaton”. In: *ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE. 2016, pp. 545–557.

[145] Nitin, Mithuna Thottethodi, et al. *Stratified Online Sampling for Sound Approximation in MapReduce*. Tech. rep. Department of Electrical and Computer Engineering, Perdue University, 2015.

[146] Zeyuan Allen Zhu, Sasa Misailovic, et al. “Randomized Accuracy-aware Program Transformations for Efficient Approximate Computations”. In: *SIGPLAN Notices* 47.1 (Jan. 2012), pp. 441–454. ISSN: 0362-1340.

[147] Qian Zhang, Feng Yuan, et al. “Approxit: An approximate computing framework for iterative methods”. In: *Proceedings of the 51st Annual Design Automation Conference (DAC)*. IEEE. 2014, pp. 1–6.

[148] Dhanya R Krishnan, Do Le Quoc, et al. “IncApprox: A Data Analytics System for Incremental Approximate Computing”. In: *Proceedings of the 25th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee. 2016, pp. 1133–1144.

[149] Martin Rinard, Henry Hoffmann, et al. “Patterns and Statistical Analysis for Understanding Reduced Resource Computing”. In: *SIGPLAN Notices* 45.10 (Oct. 2010), pp. 806–821. ISSN: 0362-1340.

[150] Henry Hoffmann, Stelios Sidiroglou, et al. “Dynamic Knobs for Responsive Power-aware Computing”. In: *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS XVI. Newport Beach, California, USA: ACM, 2011, pp. 199–212. ISBN: 978-1-4503-0266-1.

[151] Vaibhav Gupta, Debabrata Mohapatra, et al. “IMPACT: imprecise adders for low-power approximate computing”. In: *Proceedings of the 17th IEEE/ACM international symposium on Low-power electronics and design*. IEEE Press. 2011, pp. 409–414.

[152] Jin Miao, Ku He, et al. “Modeling and synthesis of quality-energy optimal approximate adders”. In: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE. 2012, pp. 728–735.

[153] Avinash Lingamneni, Christian Enz, et al. “Designing Energy-Efficient Arithmetic Operators Using Inexact Computing”. In: *Journal of Low Power Electronics* 9.1 (2013), pp. 141–153.

[154] G. Zervakis, S. Xydis, et al. “Hybrid approximate multiplier architectures for improved power-accuracy trade-offs”. In: *IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. July 2015, pp. 79–84.

[155] H. Jiang, J. Han, et al. “Approximate Radix-8 Booth Multipliers for Low-Power and High-Performance Operation”. In: *IEEE Transactions on Computers* 65.8 (Aug. 2016), pp. 2638–2644. ISSN: 0018-9340.

[156] A. Mehta, S. Maurya, et al. “Accuracy-configurable approximate multiplier with error detection and correction”. In: *TENCON 2015 - 2015 IEEE Region 10 Conference*. Nov. 2015, pp. 1–4.

[157] A. Momeni, J. Han, et al. “Design and Analysis of Approximate Compressors for Multiplication”. In: *IEEE Transactions on Computers* 64.4 (Apr. 2015), pp. 984–994. ISSN: 0018-9340.

[158] G Tziantzioulis, AM Gok, et al. “Lazy Pipelines: Enhancing quality in approximate computing”. In: *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2016, pp. 1381–1386.

[159] Hang Zhang, Mateja Putic, and John Lach. “Low Power GPGPU Computation with Imprecise Hardware”. In: *Proceedings of the 51st Annual Design Automation Conference*. DAC ’14. San Francisco, CA, USA: ACM, 2014, 99:1–99:6. ISBN: 978-1-4503-2730-5.

[160] A. Yazdanbakhsh, D. Mahajan, et al. “Axilog: Language support for approximate hardware design”. In: *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*. Mar. 2015, pp. 812–817.

[161] Jin Miao, Andreas Gerstlauer, and Michael Orshansky. “Approximate Logic Synthesis Under General Error Magnitude and Frequency Constraints”. In: *Proceedings of the International Conference on Computer-Aided Design*. ICCAD ’13. San Jose, California: IEEE Press, 2013, pp. 779–786. ISBN: 978-1-4799-1069-4.

[162] A. Ranjan, A. Raha, et al. “ASLAN: Synthesis of approximate sequential circuits”. In: *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*. Mar. 2014, pp. 1–6.

[163] Kumud Nepal, Yueting Li, et al. “ABACUS: A Technique for Automated Behavioral Synthesis of Approximate Computing Circuits”. In: *Proceedings of the Conference on Design, Automation & Test in Europe*. DATE ’14. Dresden, Germany: European Design and Automation Association, 2014, 361:1–361:6. ISBN: 978-3-9815370-2-4.

[164] Chaofan Li, Wei Luo, et al. “Joint precision optimization and high level synthesis for approximate computing”. In: *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. June 2015, pp. 1–6.

[165] R. Venkatesan, A. Agarwal, et al. “MACACO: Modeling and analysis of circuits for approximate computing”. In: *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. Nov. 2011, pp. 667–673.

[166] M. Kamal, A. Ghasemazar, et al. “Improving efficiency of extensible processors by using approximate custom instructions”. In: *Design, Automation and Test in Europe Conference and Exhibition (DATE)*. Mar. 2014, pp. 1–4.

[167] Vinay Chippa, Debabrata Mohapatra, et al. “Scalable effort hardware design”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 22.9 (2014), pp. 2004–2016.

[168] Arnab Raha, Swagath Venkataramani, et al. “Quality Configurable Reduce-and-rank for Energy Efficient Approximate Computing”. In: *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*. DATE ’15. Grenoble, France: EDA Consortium, 2015, pp. 665–670. ISBN: 978-3-9815370-4-8.

[169] Joshua San Miguel, Mario Badr, and Natalie Enright Jerger. “Load value approximation”. In: *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society. 2014, pp. 127–139.

[170] Amir Yazdanbakhsh, Gennady Pekhimenko, et al. “RFVP: Rollback-Free Value Prediction with Safe-to-Approximate Loads”. In: *ACM Transactions on Architectures and Code Optimization (TACO)* 12.4 (Jan. 2016), 62:1–62:26. ISSN: 1544-3566.

[171] Joshua San Miguel, Jorge Albericio, et al. “Doppelgänger: A Cache for Approximate Computing”. In: *Proceedings of the 48th International Symposium on Microarchitecture*. ACM, 2015, pp. 50–61. ISBN: 978-1-4503-4034-2.

[172] San Miguel, J. Albericio, et al. “The Bunker Cache for spatio-value approximation”. In: *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Oct. 2016, pp. 1–12.

[173] M. Shoushtari, A. BanaiyanMofrad, and N. Dutt. “Exploiting Partially-Forgetful Memories for Approximate Computing”. In: *IEEE Embedded Systems Letters* 7.1 (Mar. 2015), pp. 19–22. ISSN: 1943-0663.

[174] Jacob Nelson, Adrian Sampson, and Luis Ceze. “Dense Approximate Storage in Phase-Change Memory”. In: *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), ACM*. 2011.

[175] Jian Zhou, Xunchao Chen, and Jun Wang. *ApproxSSD: Fast Data Sampling on SSD Arrays*. `http://acs.ict.ac.cn/asbd2016/Papers/ASBD2016_paper_2.pdf`. 2016.

[176] Swarat Chaudhuri, Sumit Gulwani, et al. “Proving programs robust”. In: *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. ACM. 2011, pp. 102–112.

[177] Carlos Alvarez, Jesus Corbal, and Mateo Valero. “Fuzzy memoization for floating-point multimedia applications”. In: *IEEE Transactions on Computers* 54.7 (2005), pp. 922–927.

[178] X. He, G. Yan, et al. “ACR: Enabling computation reuse for approximate computing”. In: *21st Asia and South Pacific Design Automation Conference (ASP-DAC)*. Jan. 2016, pp. 643–648.

[179] Mohsen Imani, Yeseong Kim, et al. “ACAM: Approximate computing based on adaptive associative memory with online learning”. In: *International Symposium on Low Power Electronics and Design (ISLPED)*. 2016, pp. 162–167.

[180] L. A. d. Silveira, M. Brandalero, et al. “The Potential of Accelerating Image-Processing Applications by Using Approximate Function Reuse”. In: *VI Brazilian Symposium on Computing Systems Engineering (SBESC)*. Nov. 2016, pp. 122–127.

[181] S. Sinha and W. Zhang. “Low-Power FPGA Design Using Memoization-Based Approximate Computing”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* PP.99 (2016), pp. 1–14. ISSN: 1063-8210.

[182] Thierry Moreau, Mark Wyse, et al. “SNNAP: Approximate computing on programmable SoCs via neural acceleration”. In: *21st IEEE International Symposium on High Performance Computer Architecture, HPCA, Burlingame, CA, USA, February 7-11, 2015*, pp. 603–614.

[183] Amir Yazdanbakhsh, Jongse Park, et al. “Neural Acceleration for GPU Throughput Processors”. In: *Proceedings of the 48th International Symposium on Microarchitecture*. MICRO-48. Waikiki, Hawaii: ACM, 2015, pp. 482–493. ISBN: 978-1-4503-4034-2.

[184] Zidong Du, Avinash Lingamneni, et al. “Leveraging the error resilience of machine-learning applications for designing highly energy efficient accelerators”. In: *19th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE. 2014, pp. 201–206.

[185] B. Grigorian, N. Farahpour, and G. Reinman. “BRAINIAC: Bringing reliable accuracy into neurally-implemented approximate computing”. In: *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. Feb. 2015, pp. 615–626.

[186] Jongse Park, Emmanuel Amaro, et al. “AxGames: Towards Crowdsourcing Quality Target Determination in Approximate Computing”. In: *SIGPLAN Notices* 51.4 (Mar. 2016), pp. 623–636. ISSN: 0362-1340.

[187] Xuanhua Li and Donald Yeung. “Exploiting soft computing for increased fault tolerance”. In: *Workshop on Architectural Support for Gigascale Integration*. 2006.

[188] Michael Carbin, Sasa Misailovic, and Martin C. Rinard. “Verifying Quantitative Reliability for Programs That Execute on Unreliable Hardware”. In: *SIGPLAN Notices* 48.10 (Oct. 2013), pp. 33–52. ISSN: 0362-1340.

[189] Jongse Park, Hadi Esmaeilzadeh, et al. “FlexJava: Language Support for Safe and Modular Approximate Programming”. In: *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering*. ESEC/FSE 2015. Bergamo, Italy: ACM, 2015, pp. 745–757. ISBN: 978-1-4503-3675-8.

[190] Michael Ringenburg, Adrian Sampson, et al. “Monitoring and Debugging the Quality of Results in Approximate Programs”. In: *SIGPLAN Notices* 50.4 (Mar. 2015), pp. 399–411. ISSN: 0362-1340.

[191] Jason Ansel, Yee Lok Wong, et al. “Language and compiler support for auto-tuning variable-accuracy algorithms”. In: *Proceedings of the 9th Annual IEEE/ACM International Symposium on Code Generation and Optimization*. IEEE Computer Society. 2011, pp. 85–96.

[192] Adrian Sampson, André Baixo, et al. “Accept: A programmer-guided compiler framework for practical approximate computing”. In: *University of Washington Technical Report UW-CSE-15-01* 1 (2015).

[193] Sasa Misailovic, Michael Carbin, et al. “Chisel: Reliability-and accuracy-aware optimization of approximate computational kernels”. In: *ACM SIGPLAN Notices*. Vol. 49. 10. ACM. 2014, pp. 309–328.

[194] P. Roy, J. Wang, and W. F. Wong. “PAC: Program Analysis for Approximation-aware Compilation”. In: *2015 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*. Oct. 2015, pp. 69–78.

[195] Vassilis Vassiliadis, Jan Riehme, et al. “Towards Automatic Significance Analysis for Approximate Computing”. In: *Proceedings of the International Symposium on Code Generation and Optimization*. Barcelona, Spain: ACM, 2016, pp. 182–193. ISBN: 978-1-4503-3778-6.

[196] Pooja Roy, Rajarshi Ray, et al. “ASAC: Automatic Sensitivity Analysis for Approximate Computing”. In: *Proceedings of the SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems*. LCTES ’14. Edinburgh, United Kingdom: ACM, 2014, pp. 95–104. ISBN: 978-1-4503-2877-7.

[197] Sasa Misailovic, Stelios Sidiroglou, et al. “Quality of Service Profiling”. In: *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*. ICSE ’10. Cape Town, South Africa: ACM, 2010, pp. 25–34. ISBN: 978-1-60558-719-6.

[198] Vassilis Vassiliadis, Konstantinos Parasyris, et al. “A Programming Model and Runtime System for Significance-aware Energy-efficient Computing”. In: *SIGPLAN Notices* 50.8 (Jan. 2015), pp. 275–276. ISSN: 0362-1340.

[199] Beayna Grigorian and Glenn Reinman. “Dynamically adaptive and reliable approximate computing using light-weight error analysis”. In: *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. IEEE. July 2014, pp. 248–255.

[200] Cheng Tan, Thannirmalai Somu Muthukaruppan, et al. “Approximation-aware scheduling on heterogeneous multi-core architectures”. In: *20th Asia and South Pacific Design Automation Conference (ASP-DAC),* IEEE. 2015, pp. 618–623.

[201] Anne Farrell and Henry Hoffmann. “MEANTIME: Achieving Both Minimal Energy and Timeliness with Approximate Computing”. In: *2016 USENIX Annual Technical Conference (USENIX ATC 16)*. Denver, CO: USENIX Association, 2016, pp. 421–435. ISBN: 978-1-931971-30-0.

[202] Beayna Grigorian and Glenn Reinman. “Improving coverage and reliability in approximate computing using application-specific, light-weight checks”. In: *First Workshop on Approximate Computing Across the System Stack (WACAS)*. 2014.

[203] D. S. Khudia, B. Zamirai, et al. “Quality Control for Approximate Accelerators by Error Prediction”. In: *IEEE Design Test* 33.1 (Feb. 2016), pp. 43–50. ISSN: 2168-2356.

[204] Sara Achour and Martin C. Rinard. “Approximate Computation with Outlier Detection in Topaz”. In: *SIGPLAN Notices* 50.10 (Oct. 2015), pp. 711–730. ISSN: 0362-1340.

[205] Ting Wang, Qian Zhang, et al. “On Effective and Efficient Quality Management for Approximate Computing”. In: *Proceedings of the International Symposium on Low Power Electronics and Design*. ISLPED ’16. San Francisco Airport, CA, USA: ACM, 2016, pp. 156–161. ISBN: 978-1-4503-4185-1.

[206] Xin Sui, Andrew Lenharth, et al. “Proactive Control of Approximate Programs”. In: *SIGOPS Operating Systems Review* 50.2 (Mar. 2016), pp. 607–621. ISSN: 0163-5980.

[207] David H Bailey and Jonathan M Borwein. “High-precision arithmetic in mathematical physics”. In: *Mathematics* 3.2 (2015), pp. 337–367.

[208] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. 2nd. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2002. ISBN: 0898715210.

[209] George Constantinides, Adam Kinsman, and Nicola Nicolici. “Numerical data representations for FPGA-based scientific computing”. In: *IEEE Design & Test of Computers* 28.4 (2011), pp. 8–17.

[210] Wei-Fan Chiang, Mark Baranowski, et al. “Rigorous Floating-point Mixed-precision Tuning”. In: *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*. POPL 2017. Paris, France: ACM, 2017, pp. 300–315. ISBN: 978-1-4503-4660-3.

[211] Mian Lu, Bingsheng He, and Qiong Luo. “Supporting Extended Precision on Graphics Processors”. In: *Proceedings of the Sixth International Workshop on Data Management on New Hardware*. DaMoN ’10. Indianapolis, Indiana: ACM, 2010, pp. 19–26. ISBN: 978-1-4503-0189-3.

[212] Mioara Joldes, Jean-Michel Muller, et al. “CAMPARY: Cuda Multiple Precision Arithmetic Library and Applications”. In: *Mathematical Software – ICMS 2016*. Ed. by Gert-Martin Greuel, Thorsten Koch, et al. Cham: Springer International Publishing, 2016, pp. 232–240. ISBN: 978-3-319-42432-3.

[213] Takatoshi Nakayama and Daisuke Takahashi. “Implementation of multiple-precision floating-point arithmetic library for GPU computing”. In: *Proceedings of the 23rd IASTED International Conference on Parallel and Distributed Computing and Systems*. 2011, pp. 343–349.

[214] Y. Lei, Y. Dou, et al. “Special-purposed VLIW architecture for IEEE-754 quadruple precision elementary functions on FPGA”. In: *IEEE 29th International Conference on Computer Design (ICCD)*. Oct. 2011, pp. 219–225.

[215] Yong Dou, Yuanwu Lei, et al. “FPGA Accelerating Double/Quad-double High Precision Floating-point Applications for ExaScale Computing”. In: *Proceedings of the 24th ACM International Conference on Supercomputing*. ICS ’10. Tsukuba, Ibaraki, Japan: ACM, 2010, pp. 325–336. ISBN: 978-1-4503-0018-6.

[216] Yuanwu Lei, Yong Dou, et al. “FPGA implementation of an exact dot product and its application in variable-precision floating-point arithmetic”. In: *The Journal of Supercomputing* 64.2 (2013), pp. 580–605. ISSN: 1573-0484.

[217] Ulrich Kulisch and Van Snyder. “The exact dot product as basic tool for long interval arithmetic”. In: *Computing* 91.3 (2011), pp. 307–313.

[218] John L Gustafson. “A radical approach to computation with real numbers”. In: *Supercomputing frontiers and innovations* 3.2 (2016), pp. 38–53.

[219] Laslo Hunhold. “The Unum Number Format: Mathematical Foundations, Implementation and Comparison to IEEE 754 Floating-Point Numbers”. In: *CoRR* abs/1701.00722 (2017).

[220] Khaing Yin Kyaw, Wang Ling Goh, and Kiat Seng Yeo. “Low-power high-speed multiplier for error-tolerant application”. In: *2010 IEEE International Conference of Electron Devices and Solid-State Circuits (EDSSC)*. Dec. 2010, pp. 1–4.

[221] Mark Horowitz. *Computing’s Energy Problem: and what we can do about it*. Keynote, International Solid-State Circuits Conference. (last visit 03/05/15). Feb. 2014. URL: `https://www.futurearchs.org/sites/default/files/horowitz-ComputingEnergyISSCC.pdf`.

[222] William Hadley Richardson. “Bayesian-Based Iterative Method of Image Restoration”. In: *Journal of the Optical Society of America* 62 (1 Jan. 1972), pp. 55–59.

[223] Paul Bourke. *Implementation of a 2D Fast Fourier Transformation*. URL: `http://paulbourke.net/miscellaneous/dft/`.

[224] Chi-Keung Luk, Robert Cohn, et al. “Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation”. In: *ACM SIGPLAN Notices* 40.6 (June 2005), pp. 190–200. ISSN: 0362-1340.

[225] Nicholas Nethercote and Julian Seward. “Valgrind: a framework for heavy-weight dynamic binary instrumentation”. In: *ACM Sigplan notices*. Vol. 42. 6. ACM. 2007, pp. 89–100.

[226] Jan Lucas, Mauricio Alvarez-Mesa, et al. “Sparkk: Quality-Scalable Approximate Storage in DRAM”. In: *The Memory Forum*. June 2014, pp. 1–9.

[227] Adrian Sampson, Jacob Nelson, et al. “Approximate Storage in Solid-state Memories”. In: *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO-46. Davis, California: ACM, 2013, pp. 25–36. ISBN: 978-1-4503-2638-4.

[228] Somayeh Sardashti, André Seznec, and David A Wood. “Skewed compressed caches”. In: *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society. 2014, pp. 331–342.

[229] Marc Baboulin, Alfredo Buttari, et al. “Accelerating scientific computations with mixed precision algorithms”. In: *Computer Physics Communications* 180.12 (2009), pp. 2526–2533.

[230] Bryan Hennelly, Damien Kelly, et al. “Zooming algorithms for Digital Holography”. In: *Journal of Physics: Conference Series* 206.1 (2010), p. 012027.

[231] Hardkernel. *Odriod-XU*. (last visit 03/05/15). URL: `http://odroid.com/dokuwiki/doku.php?id=en:odroid-xu`.

[232] Parallela Project. *Parallela board*. (last visit 03/05/15). URL: `http://www.parallella.org/board/`.

[233] Movidius Ltd. *Myriad 1*. `http://www.hotchips.org/wp-content/uploads/hc_archives/hc23/HC23.19.8-Video/HC23.19.811-1TOPS-Media-Moloney-Movidius.pdf`. (last visit 03/05/15).

[234] Andrew Waterman, Yunsup Lee, et al. *The RISC-V Instruction Set Manual. Volume 1: User-Level ISA, Version 2.2*. Tech. rep. California University of Berkeley, Department of Electrical Engineering and Computer Sciences, May 2017. URL: `https://riscv.org/specifications/`.

[235] Krste Asanovic, Rimas Avizienis, et al. “The rocket chip generator”. In: *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17* (2016).

[236] Shyamkumar Thoziyoor, Naveen Muralimanohar, et al. *CACTI 5.1*. Tech. rep. HPL-2008-20, HP Labs, 2008.

[237] Botang Shao and Peng Li. “A Model for Array-based Approximate Arithmetic Computing with Application to Multiplier and Squarer Design”. In: *Proceedings of the International Symposium on Low Power Electronics and Design*. ISLPED ’14. La Jolla, California, USA: ACM, 2014, pp. 9–14. ISBN: 978-1-4503-2975-0.

[238] Alaa R Alameldeen and David A Wood. “Adaptive cache compression for high-performance processors”. In: *Proceedings of the 31st Annual International Symposium on Computer Architecture*. IEEE. 2004, pp. 212–223.

[239] S. Sardashti and D. A. Wood. “Decoupled compressed cache: Exploiting spatial locality for energy-optimized compressed caching”. In: *2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Davis, California, Dec. 2013, pp. 62–73. ISBN: 978-1-4503-2638-4.

[240] A. Shafiee, M. Taassori, et al. “MemZip: Exploring unconventional benefits from memory compression”. In: *IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE. Feb. 2014, pp. 638–649.

[241] Angelos Arelakis and Per Stenstrom. “SC2: A statistical compression cache scheme”. In: *ACM SIGARCH Computer Architecture News*. Vol. 42. 3. IEEE Press. 2014, pp. 145–156.

[242] Shiro Kobayashi and GerhardP. Fettweis. “A Hierarchical Block-Floating-Point Arithmetic”. English. In: *Journal of VLSI signal processing systems for signal, image and video technology* 24.1 (2000), pp. 19–30. ISSN: 0922-5773.

[243] Animesh Jain, Parker Hill, et al. “Concise loads and stores: The case for an asymmetric compute-memory architecture for approximation”. In: *49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE. 2016, pp. 1–13.

[244] A. Ranjan, A. Raha, et al. “Approximate memory compression for energy-efficiency”. In: *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE. July 2017, pp. 1–6.

[245] Michael D. Linderman, Matthew Ho, et al. “Towards Program Optimization Through Automated Analysis of Numerical Precision”. In: *Proceedings of the 8th Annual IEEE/ACM International Symposium on Code Generation and Optimization*. CGO ’10. Toronto, Ontario, Canada: ACM, 2010, pp. 230–237. ISBN: 978-1-60558-635-9.

[246] KT Gribbon, Donald G Bailey, and Christopher T Johnston. “Using design patterns to overcome image processing constraints on FPGAs”. In: *Third IEEE International Workshop on Electronic Design, Test and Applications (DELTA)*. IEEE. 2006, pp. 7–56.

[247] Michael Bromberger, Pascal Bastian, et al. “FPGA-accelerated Richardson-Lucy deconvolution for 3D image data”. In: *IEEE International Symposium on Biomedical Imaging (ISBI)*. 2016, pp. 132–135.

[248] Richard Bellman. “On the theory of dynamic programming”. In: *Proceedings of the National Academy of Sciences* 38.8 (1952), pp. 716–719.

[249] M. Remmert, A. Biegert, et al. “HHblits: Lightning-fast iterative protein sequence searching by HMM-HMM alignment”. In: *Journal of Nature methods (Nature Publishing Group)* 9.2 (Feb. 2012), pp. 173–175. ISSN: 1548-7105.

[250] M. Farrar. “Striped Smith-Waterman speeds database searches six times over other SIMD implementations”. In: *Journal of Bioinformatics (Oxford University Press)* 23.2 (Jan. 2007), pp. 156–161. ISSN: 1367-4803.

[251] Johannes Söding. “Protein homology detection by HMM–HMM comparison”. In: *Bioinformatics* 21.7 (2004), pp. 951–960.

[252] T.F. Smith and M.S. Waterman. “Identification of Common Molecular Subsequences”. In: *Journal of Molecular Biology (Elsevier)* 147.1 (1981), pp. 195–197. ISSN: 0022-2836.

[253] Moritz Menze and Andreas Geiger. “Object Scene Flow for Autonomous Vehicles”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 3061–3070.

[254] Beau Tippetts, Dah Jye Lee, et al. “Review of stereo vision algorithms and their suitability for resource-limited systems”. In: *Journal of Real-Time Image Processing* 11.1 (2016), pp. 5–25. ISSN: 1861-8219.

[255] Rostam Affendi Hamzah and Haidi Ibrahim. “Literature Survey on Stereo Vision Disparity Map Algorithms”. In: *Journal of Sensors* 2016 (2016), p. 23.

[256] O. J. Arndt, D. Becker, et al. “Parallel implementation of real-time semi-global matching on embedded multi-core architectures”. In: *2013 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*. July 2013, pp. 56–63.

[257] Joseph Kruskall and Mark Liberman. “The Symmetric Time Warping Algorithm: From Continuous to Discrete.” In: *Time Warps, String Edits and Macromolecules: The theory and practice of sequence comparison*. Addison-Wesley, 1983, pp. 125–162.

[258] Hiroaki Sakoe and Seibi Chiba. “Dynamic programming algorithm optimization for spoken word recognition”. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 26.1 (1978), pp. 43–49.

[259] HT Kung and Charles E Leiserson. “Systolic arrays (for VLSI)”. In: *Sparse Matrix Proceedings 1978*. Vol. 1. SIAM. 1979, pp. 256–282.

[260] Hsiang-Tsung Kung. “Why systolic architectures?” In: *IEEE computer* 15.1 (1982), pp. 37–46.

[261] Richard P. Brent and H. T. Kung. “Systolic VLSI Arrays for Polynomial GCD Computation”. In: *IEEE Transactions on Computers* 33.8 (1984), pp. 731–736.

[262] André DeHon, Joshua Adams, et al. “Design patterns for reconfigurable computing”. In: *12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE. 2004, pp. 13–23.

[263] Intel Cooperation. *Intel FPGA SDK for OpenCL Best Practices Guide*. UG-OCL003. Dec. 2017. URL: `https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/opencl-sdk/aocl-best-practices-guide.pdf`.

[264] Altera Corperation. “Altera SDK for OpenCL - Programming Guide”. In: *https://www.altera.com/en_US/pdfs/literature/hb/opencl-sdk/aocl_programming_guide.pdf* (2015).

[265] EBV Elektronik. *Datasheet SoCrates II Cyclone V-SoC Evaluation Board*. 2015.

[266] Torbjørn Rognes and Erling Seeberg. “Six-Fold Speed-Up of Smith-Waterman Sequence Database Searches Using Parallel Processing on Common Microprocessors”. In: *Bioinformatics* 16.8 (2000), pp. 699–706.

[267] Chotirat Ann Ratanamahatana, Jessica Lin, et al. “Mining time series data”. In: *Data mining and knowledge discovery handbook*. Springer, 2009, pp. 1049–1077.

[268] Jessica Lin, Eamonn Keogh, et al. “A symbolic representation of time series, with implications for streaming algorithms”. In: *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*. ACM. 2003, pp. 2–11.

[269] Simon Malinowski, Thomas Guyet, et al. “1d-SAX: A novel symbolic representation for time series”. In: *International Symposium on Intelligent Data Analysis*. Springer. 2013, pp. 273–284.

[270] Rohit J Kate. “Using dynamic time warping distances as features for improved time series classification”. In: *Data Mining and Knowledge Discovery* 30.2 (2016), pp. 283–312.

[271] Il-Lyong Jung, Tae-Young Chung, et al. “Consistent Stereo Matching Under Varying Radiometric Conditions”. In: *IEEE Transactions on Multimedia* 15.1 (Jan. 2013), pp. 56–69. ISSN: 1520-9210.

[272] Cuong Cao Pham and Jae Wook Jeon. “Domain Transformation-Based Efficient Cost Aggregation for Local Stereo Matching”. In: *IEEE Transactions on Circuits and Systems for Video Technology* 23.7 (July 2013), pp. 1119–1130. ISSN: 1051-8215.

[273] Kurt Konolige. “Projected texture stereo”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2010, pp. 148–155.

[274] Yanping Chen et al. *The UCR Time Series Classification Archive*. www.cs.ucr.edu/~eamonn/time_series_data/. July 2015.

[275] Gustavo EAPA Batista, Eamonn J Keogh, et al. “CID: an efficient complexity-invariant distance for time series”. In: *Data Mining and Knowledge Discovery* 28.3 (2014), pp. 634–669.

[276] Thanawin Rakthanmanon, Bilson Campana, et al. “Addressing big data time series: Mining trillions of time series subsequences under dynamic time warping”. In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 7.3 (2013), p. 10.

[277] Sang-Wook Kim, Sanghyun Park, and Wesley W Chu. “An index-based approach for similarity search supporting time warping in large sequence databases”. In: *Proceedings of the 17th International Conference on Data Engineering*. IEEE. 2001, pp. 607–614.

[278] Eamonn Keogh and Chotirat Ann Ratanamahatana. “Exact indexing of dynamic time warping”. In: *Knowledge and information systems* 7.3 (2005), pp. 358–386.

[279] Daniel Lemire. “Faster Retrieval with a Two-pass Dynamic-time-warping Lower Bound”. In: *Pattern Recognition* 42.9 (2009), pp. 2169–2180.

[280] Erich Gamma. *Design patterns: elements of reusable object-oriented software*. Pearson Education India, 1995.

[281] Raghu Prabhakar, David Koeplinger, et al. “Generating configurable hardware from parallel patterns”. In: *ACM SIGOPS Operating Systems Review* 50.2 (2016), pp. 651–665.

[282] Enzo Rucci, Carlos García, et al. “State-of-the-Art in Smith–Waterman Protein Database Search on HPC Platforms”. In: *Big Data Analytics in Genomics*. Ed. by Ka-Chun Wong. Cham: Springer International Publishing, 2016, pp. 197–223. ISBN: 978-3-319-41279-5.

[283] Andrzej Wozniak. “Using video-oriented instructions to speed up sequence comparison”. In: *Bioinformatics* 13.2 (1997), pp. 145–150.

[284] Jeff Daily. “Parasail: SIMD C library for global, semi-global, and local pairwise sequence alignments”. In: *BMC Bioinformatics* 17.1 (Feb. 2016), p. 81. ISSN: 1471-2105.

[285] Yongchao Liu, Adrianto Wirawan, and Bertil Schmidt. “CUDASW++ 3.0: accelerating Smith-Waterman protein database search by coupling CPU and GPU SIMD instructions”. In: *BMC Bioinformatics* 14.1 (Apr. 2013), p. 117. ISSN: 1471-2105.

[286] Dan Zou, Yong Dou, and Fei Xia. “Optimization schemes and performance evaluation of Smith-Waterman algorithm on CPU, GPU and FPGA”. In: *Concurrency and Computation: Practice and Experience* 24.14 (2012), pp. 1625–1644. ISSN: 1532-0634.

[287] Enzo Rucci, Carlos Garcia, et al. “OSWALD: OpenCL Smith–Waterman on Altera’s FPGA for Large Protein Databases”. In: *The International Journal of High Performance Computing Applications* (2016).

[288] Haidong Lan, Weiguo Liu, et al. “Accelerating large-scale biological database search on Xeon Phi-based neo-heterogeneous architectures”. In: *IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE. 2015, pp. 503–510.

[289] Jure Zbontar and Yann LeCun. “Stereo matching by training a convolutional neural network to compare image patches”. In: *Journal of Machine Learning Research* 17.1-32 (2016), p. 2.

[290] Heiko Hirschmüller, Maximilian Buder, and Ines Ernst. “Memory efficient semi-global matching”. In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 3 (2012), pp. 371–376.

[291] Daniel Hernández Juárez, Alejandro Chacón, et al. “Embedded Real-time Stereo Estimation via Semi-Global Matching on the GPU”. In: *Procedia Computer Science* 80 (2016), pp. 143–153.

[292] Stefan K. Gehrig, Reto Stalder, and Nicolai Schneider. “A Flexible High-Resolution Real-Time Low-Power Stereo Vision Engine”. In: *International Conference on Computer Vision Systems, Copenhagen, Denmark*. July 2015, pp. 69–79. ISBN: 978-3-319-20904-3.

[293] Jaco Hofmann, Jens Korinth, and Andreas Koch. “A Scalable High-Performance Hardware Architecture for Real-Time Stereo Vision by Semi-Global Matching”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2016, pp. 27–35.

[294] W. Wang, J. Yan, et al. “Real-Time High-Quality Stereo Vision System in FPGA”. In: *IEEE Transactions on Circuits and Systems for Video Technology* 25.10 (Oct. 2015), pp. 1696–1708. ISSN: 1051-8215.

[295] Daolu Zha, Xi Jin, and Tian Xiang. “A real-time global stereo-matching on FPGA”. In: *Microprocessors and Microsystems* 47, Part B (2016), pp. 419–428. ISSN: 0141-9331.

[296] Peiheng Zhang, Guangming Tan, and Guang R. Gao. “Implementation of the Smith-Waterman Algorithm on a Reconfigurable Supercomputing Platform”. In: *Proceedings of the 1st international workshop on High-performance reconfigurable computing technology and applications (HPRCTA): held in conjunction with ACM/IEEE Supercomputing Conference*. ACM, 2007, pp. 39–48. ISBN: 978-1-59593-894-7.

[297] Zilong Wang, Sitao Huang, et al. “Accelerating Subsequence Similarity Search Based on Dynamic Time Warping Distance with FPGA”. In: *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays FPGA*. ACM, 2013, pp. 53–62. ISBN: 978-1-4503-1887-7.

[298] S. Huang, G. Dai, et al. “DTW-Based Subsequence Similarity Search on AMD Heterogeneous Computing Platform”. In: *IEEE 10th International Conference on High Performance Computing and Communications & IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_-EUC)*. Nov. 2013, pp. 1054–1063.

[299] L. Xiao, Y. Zheng, et al. “Parallelizing Dynamic Time Warping Algorithm Using Prefix Computations on GPU”. In: *IEEE 10th International Conference on High Performance Computing and Communications & IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC)*. 2013, pp. 294–299.

[300] R. Miniakhmetov, A. Movchan, and M. Zymbler. “Accelerating time series subsequence matching on the Intel Xeon Phi many-core coprocessor”. In: *38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. 2015, pp. 1399–1404.

[301] D. Sart, A. Mueen, et al. “Accelerating Dynamic Time Warping Subsequence Search with GPUs and FPGAs”. In: *IEEE 10th International Conference on Data Mining (ICDM)*. 2010, pp. 1001–1006.

[302] Muhammad Shafique, Rehan Hafiz, et al. “Cross-layer approximate computing: From logic to architectures”. In: *Proceedings of the 53rd Annual Design Automation Conference*. ACM. 2016, p. 99.

[303] Amir Yazdanbakhsh, Divya Mahajan, et al. “AxBench: A Multiplatform Benchmark Suite for Approximate Computing”. In: *IEEE Design & Test* 34.2 (2017), pp. 60–68.

[304] Paul I Pénzes and Alain J Martin. “Energy-delay efficiency of VLSI computations”. In: *Proceedings of the 12th ACM Great Lakes symposium on VLSI*. ACM. 2002, pp. 104–111.

[305] *SMHasher - A test suite for testing hash functions*. 2016. URL: `https://github.com/aappleby/smhasher` (visited on 02/15/2018).

[306] Piotr Indyk and Rajeev Motwani. “Approximate nearest neighbors: towards removing the curse of dimensionality”. In: *Proceedings of the 13th annual ACM symposium on Theory of computing*. ACM. 1998, pp. 604–613.

[307] Moses S Charikar. “Similarity estimation techniques from rounding algorithms”. In: *Proceedings of the thirty-fourth annual ACM symposium on Theory of computing*. ACM. 2002, pp. 380–388.

[308] Piotr Indyk Alexandr Andoni. *E^2LSH 0.1 - User Manual*. June 21, 2005. URL: `http://www.mit.edu/~andoni/LSH/`.

[309] Mark Boddy and Thomas Dean. “Solving Time-dependent Planning Problems”. In: *Proceedings of the 11th International Joint Conference on Artificial Intelligence - Volume 2*. IJCAI’89. Detroit, Michigan: Morgan Kaufmann Publishers Inc., 1989, pp. 979–984.

[310] Eric J Horvitz. “Reasoning about Beliefs and Actions under Computational Resource Constraints”. In: *In Proceedings of the Workshop on Uncertainty in Artificial Intelligence*. 1987.

[311] K.J. Lin, S. Nataraja, et al. “Concord: A system of imprecise computations”. In: *Proceedings of COMPSAC, Tokoyo, Japan*. 1987, pp. 75–81.

[312] Daniel S. Bernstein, Theodore J. Perkins, et al. “Scheduling Contract Algorithms on Multiple Processors”. In: *Proceedings of the Eighteenth National Conference on Artificial Intelligence*. Edmonton, Alberta, 2002, pp. 702–706.

[313] Shlomo Zilberstein. “Operational Rationality through Compilation of Anytime Algorithms”. PhD thesis. University of California at Berkeley, 1993.

[314] Joshua Grass. “Reasoning about computational resource allocation”. In: *Crossroads* 3.1 (1996), pp. 16–20.

[315] Shlomo Zilberstein. “Optimizing Decision Quality with Contract Algorithms”. In: *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*. IJCAI’95. Montreal, Quebec, Canada: Morgan Kaufmann Publishers Inc., 1995, pp. 1576–1582. ISBN: 1-55860-363-8.

[316] Shlomo Zilberstein. “Using anytime algorithms in intelligent systems”. In: *AI magazine* 17.3 (1996), p. 73.

[317] Eric A Hansen and Shlomo Zilberstein. “Monitoring the progress of anytime problem-solving”. In: *Proceedings of the National Conference on Artificial Intelligence*. 1996, pp. 1229–1234.

[318] Eric A Hansen and Shlomo Zilberstein. “Monitoring anytime algorithms”. In: *ACM SIGART Bulletin* 7.2 (1996), pp. 28–33.

[319] Eric A Hansen and Shlomo Zilberstein. “Monitoring and control of anytime algorithms: A dynamic programming approach”. In: *Artificial Intelligence* 126.1-2 (2001), pp. 139–157.

[320] Lev Finkelstein and Shaul Markovitch. “Optimal schedules for monitoring anytime algorithms”. In: *Artificial Intelligence* 126.1-2 (2001), pp. 63–108.

[321] A. J. Garvey and V. R. Lesser. “Design-to-time real-time scheduling”. In: *IEEE Transactions on Systems, Man, and Cybernetics* 23.6 (Nov. 1993), pp. 1491–1502. ISSN: 0018-9472.

[322] Joshua Grass and Shlomo Zilberstein. “Anytime algorithm development tools”. In: *ACM SIGART Bulletin* 7.2 (1996), pp. 20–27.

[323] Marcelo Brandalero, Leonardo Almeida da Silveira, et al. “Accelerating error-tolerant applications with approximate function reuse”. In: *Science of Computer Programming* 165 (2018). Special issue on VI Brazilian Symposium on Computing Systems Engineering, pp. 54–67. ISSN: 0167-6423.

[324] Intel Cooperation. *Intel 64 and IA-32 Architectures Software Developer’s Manual - Combined Volumes:1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D and 4*. 2017.

[325] Jan Treibig, Georg Hager, and Gerhard Wellein. “LIKWID: A Lightweight Performance-Oriented Tool Suite for x86 Multicore Environments”. In: *Proceedings of the 2010 39th International Conference on Parallel Processing Workshops*. ICPPW ’10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 207–216. ISBN: 978-0-7695-4157-0.

[326] Intel Cooperation. *Enhanced Intel SpeedStep Technology for the Intel Pentium M Processor (White Paper)*. Tech. rep. 2004.

[327] AMD. *The Zen Core Architecture*. 2017. URL: `http://www.amd.com/en-gb/innovations/software-technologies/zen-cpu` (visited on 02/05/2018).

[328] C-L Hwang and Abu Syed Md Masud. *Multiple objective decision making–methods and applications: a state-of-the-art survey*. Vol. 164. Springer Science & Business Media, 2012. ISBN: 978-3-642-45511-7.

[329] Ricardo Gonzalez and Mark Horowitz. “Energy dissipation in general purpose microprocessors”. In: *IEEE Journal of solid-state circuits* 31.9 (1996), pp. 1277–1284.

[330] Alain J Martin. “Towards an energy complexity of computation”. In: *Information Processing Letters* 77.2-4 (2001), pp. 181–187.

[331] C-H Hsu, W-C Feng, and Jeremy S Archuleta. “Towards efficient supercomputing: A quest for the right metric”. In: *19th Proceedings of the IEEE International Parallel and Distributed Processing Symposium*. IEEE. 2005, 8–pp.

[332] Christopher M Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006. ISBN: 978-0-387-31073-2.

[333] John R Quinlan et al. “Learning with continuous classes”. In: *5th Australian joint conference on artificial intelligence*. Vol. 92. Singapore. 1992, pp. 343–348.

[334] Rulequest Research. *Cubist*. `https://www.rulequest.com/cubist-info.html`. 2016.

[335] J. S. Miguel and N. E. Jerger. “The Anytime Automaton”. In: *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. June 2016, pp. 545–557.

[336] Kevin Barker and Thomas Benson. *PERFECT Benchmark Suite Manual*. `http://hpc.pnnl.gov/projects/PERFECT/`. Pacific Northwest National Laboratory and Georgia Tech Research Institute. Dec. 2013.

[337] Hui Ding, Goce Trajcevski, et al. “Querying and Mining of Time Series Data: Experimental Comparison of Representations and Distance Measures”. In: *Proceedings of the VLDB Endowment* 1.2 (Aug. 2008), pp. 1542–1552.

[338] Y. Wu, P. Wawrzusin, et al. “Spatially isotropic four-dimensional imaging with dual-view plane illumination microscopy”. In: *Nature biotechnology* 31.11 (Nov. 2013), pp. 1032–1038. ISSN: 1087-0156.

[339] Muhammad Shafique, Waqas Ahmad, et al. “A low latency generic accuracy configurable adder”. In: *52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE. 2015, pp. 1–6.

[340] Arun Chandrasekharan, Daniel Große, and Rolf Drechsler. “ProACt: A Processor for High Performance On-demand Approximate Computing”. In: *Proceedings of the on Great Lakes Symposium on VLSI 2017*. ACM. 2017, pp. 463–466.

[341] N. M. Ho, E. Manogaran, et al. “Efficient floating point precision tuning for approximate computing”. In: *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. Jan. 2017, pp. 63–68.

[342] Arjun Suresh, Bharath Narasimha Swamy, et al. “Intercepting Functions for Memoization: A Case Study Using Transcendental Functions”. In: *ACM Transactions on Architecture Code Optimization* 12.2 (June 2015), 18:18:1–18:18:23. ISSN: 1544-3566.

[343] Arjun Suresh, Erven Rohou, and André Seznec. “Compile-time Function Memoization”. In: *Proceedings of the 26th International Conference on Compiler Construction*. CC 2017. Austin, TX, USA: ACM, 2017, pp. 45–54. ISBN: 978-1-4503-5233-8.

[344] X. He, S. Jiang, et al. “Exploiting the Potential of Computation Reuse Through Approximate Computing”. In: *IEEE Transactions on Multi-Scale Computing Systems* 3.3 (July 2017), pp. 152–165. ISSN: 2332-7766.

[345] Jose-Maria Arnau, Joan-Manuel Parcerisa, and Polychronis Xekalakis. “Eliminating redundant fragment shader executions on a mobile gpu via hardware memoization”. In: *ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*. IEEE. 2014, pp. 529–540.

[346] Arnab Raha and Vijay Raghunathan. “qLUT: Input-Aware Quantized Table Lookup for Energy-Efficient Approximate Accelerators”. In: *ACM Transactions on Embedded Computing Systems (TECS)* 16.5s (Sept. 2017), 130:1–130:23. ISSN: 1539-9087.

[347] Chris Wilcox, Michelle Mills Strout, and James M Bieman. “Mesa: automatic generation of lookup table optimizations”. In: *Proceedings of the 4th International Workshop on Multicore Software Engineering*. ACM. 2011, pp. 1–8.

[348] Aurangzeb and Rudolf Eigenmann. “HiPA: history-based piecewise approximation for functions”. In: *Proceedings of the International Conference on Supercomputing, ICS 2017, Chicago, IL, USA, June 14-16, 2017*. 2017, 23:1–23:10.

[349] Kaveh R. Ghazi, Vincent Lefèvre, et al. “Why and How to Use Arbitrary Precision.” In: *Computing in Science and Engineering* 12.3 (2010), p. 5.

[350] Suzana Milutinovic, Jaume Abella, et al. “Speeding up static probabilistic timing analysis”. In: *International Conference on Architecture of Computing Systems*. Springer. 2015, pp. 236–247.

[351] Ding Ma and Michael A Saunders. “Solving multiscale linear programs using the simplex method in quadruple precision”. In: *Numerical Analysis and Optimization*. Springer, 2015, pp. 223–235.

[352] M. Blair, S. Obenski, and P. Bridickas. *GAO/IMTEC-92-26 Patriot Missile Defense: Software Problem Led to System Failure at Dhahran, Saudi Arabia*. Tech. rep. 1992. URL: `http://archive.gao.gov/t2pbat6/145960.pdf`.

[353] D Richard Kuhn, Dolores R Wallace, and Albert M Gallo. “Software fault interactions and implications for software testing”. In: *IEEE Transactions on Software Engineering* 30.6 (2004), pp. 418–421.

[354] David H. Bailey, Yozo Hida, et al. *ARPREC: An arbitrary precision computation package*. Tech. rep. 2002.

[355] Werner Hofschuster and Walter Krämer. “C-XSC 2.0: A C++ Library for Extended Scientific Computing”. English. In: *Numerical Software with Result Verification*. Vol. 2991. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2004, pp. 15–35. ISBN: 978-3-540-21260-7.

[356] Laurent Fousse, Guillaume Hanrot, et al. “MPFR: A Multiple-precision Binary Floating-point Library with Correct Rounding”. In: *ACM Transactions on Mathematical Software (TOMS)* 33.2 (June 2007). ISSN: 0098-3500.

[357] Markus Grimmer, Knut Petras, and Nathalie Revol. “Multiple precision interval packages: Comparing different approaches”. In: *Numerical Software with Result Verification*. Springer, 2004, pp. 64–90.

[358] Anne Kielmann. *Lanczos-Verfahren bei exponentiell abfallenden Eigenwerten*. diploma thesis. Christian-Albrechts-Universität zu Kiel, 2008.

[359] B. Parhami. *Computer Arithmetic: Algorithms and Hardware Designs*. Computer Arithmetic: Algorithms and Hardware Designs. Oxford University Press, 2000. ISBN: 9780195125832.

[360] David W Matula and Peter Kornerup. “Finite precision rational arithmetic: Slash number systems”. In: *IEEE Transactions on Computers* 1 (1985), pp. 3–18.

[361] Roy D Merrill. “Improving digital computer performance using residue number theory”. In: *IEEE Transactions on Electronic Computers* 2 (1964), pp. 93–101.

[362] J.L. Gustafson. *The End of Error: Unum Computing*. Chapman & Hall/CRC Computational Science. Taylor & Francis, 2015. ISBN: 9781482239867.

[363] John L Gustafson and Isaac T Yonemoto. “Beating Floating Point at its Own Game: Posit Arithmetic”. In: *Supercomputing Frontiers and Innovations* 4.2 (2017), pp. 71–86.

[364] Robert Morris. “Tapered floating point: A new floating-point representation”. In: *IEEE Transactions on Computers* 100.12 (1971), pp. 1578–1579.

[365] Gene H. Golub and Henk A. van der Vorst. “Eigenvalue computation in the 20th century”. In: *Journal of Computational and Applied Mathematics* 123.1 (2000). Numerical Analysis 2000. Vol. III: Linear Algebra, pp. 35–65. ISSN: 0377-0427.

[366] Christopher C Paige. “Computational variants of the Lanczos method for the eigenproblem”. In: *IMA Journal of Applied Mathematics* 10.3 (1972), pp. 373–381.

[367] Lawrence Page, Sergey Brin, et al. *The PageRank citation ranking: Bringing order to the web.* Tech. rep. Stanford InfoLab, 1999.

[368] Scott Deerwester, Susan T Dumais, et al. “Indexing by latent semantic analysis”. In: *Journal of the American society for information science* 41.6 (1990), p. 391.

[369] Alex Pothen, Horst D Simon, and Kang-Pu Liou. “Partitioning sparse matrices with eigenvectors of graphs”. In: *SIAM Journal on Matrix Analysis and Applications* 11.3 (1990), pp. 430–452.

[370] Hong-Yi Chen, R Wortis, and WA Atkinson. “Disorder-induced zero-bias anomaly in the Anderson-Hubbard model: Numerical and analytical calculations”. In: *Physical Review B* 84.4 (2011), p. 045113.

[371] *IEEE Standard for Floating-Point Arithmetic*. Tech. rep. 3 Park Avenue, New York, NY 10016-5997, USA: Microprocessor Standards Committee of the IEEE Computer Society, Aug. 2008, pp. 1–70. URL: `http://dx.doi.org/10.1109/ieeestd.2008.4610935`.

[372] W. Hart B. Gladman and et al. J. Moxham. *MPIR: Multiple Precision Integers and Rationals*. Version 2.7.0, `http://mpir.org`. 2015.

[373] The GMP development team. *GNU MP: The GNU Multiple Precision Arihmetic Library*. 2017. URL: `https://gmplib.org` (visited on 11/02/2017).

[374] Pavel Holoborodko. *MPFR C++ WWW Homepage*. `http://www.holoborodko.com/pavel/mpfr/`. 2015.

[375] Gaël Guennebaud, Benoît Jacob, et al. *Eigen v3*. http://eigen.tuxfamily.org. 2010.

[376] Hartwig Anzt, Jack Dongarra, and Enrique S Quintana-Ortí. “Adaptive precision solvers for sparse linear systems”. In: *Proceedings of the 3rd International Workshop on Energy Efficient Supercomputing*. ACM. 2015, p. 2.

[377] Michael Schaffner, Frank K Gurkaynak, et al. “An approximate computing technique for reducing the complexity of a direct-solver for sparse linear systems in real-time video processing”. In: *51st ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE. 2014, pp. 1–6.

[378] Alexander Schöll, Claus Braun, and Hans-Joachim Wunderlich. “Applying efficient fault tolerance to enable the preconditioned conjugate gradient solver on approximate computing hardware”. In: *Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2016 IEEE International Symposium on*. IEEE. 2016, pp. 21–26.

[379] Qian Zhang, Ye Tian, et al. “ApproxEigen: An approximate computing technique for large-scale eigen-decomposition”. In: *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. IEEE Press. 2015, pp. 824–830.

[380] Alexander Schöll, Claus Braun, and Hans-Joachim Wunderlich. “Energy-efficient and Error-resilient Iterative Solvers for Approximate Computing”. In: *Proceedings of the 23rd IEEE International Symposium on On-Line Testing and Robust System Design (IOLTS’17)*. 2017, pp. 237–239.

[381] Hartwig Anzt, Edmond Chow, and Jack Dongarra. “Iterative Sparse Triangular Solves for Preconditioning”. In: *Euro-Par 2015: Parallel Processing*. Ed. by Jesper Larsson Träff, Sascha Hunold, and Francesco Versaci. Vol. 9233. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2015, pp. 650–661. ISBN: 978-3-662-48095-3.

[382] Stig Larsson and Vidar Thomee. *Partial Differential Equations with Numerical Methods*. Springer Verlag Berlin Heidelber New York, 2003. ISBN: 3-540-01772-0.

[383] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. PWS, 1996. ISBN: 978-0534947767.

[384] Jonathan Richard Shewchuk. *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*. article. School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA, Aug. 1994.

[385] Michele Benzi. “Preconditioning Techniques for Large Linear Systems: A Survey”. In: *Journal of Computational Physics* 182 (2002), pp. 418–477.

[386] Roberto Bagnara. “A Unified Proof for the Convergence of Jacobi and Gauss Seidel Methods”. In: *SIAM Review* 37 (1995), pp. 93–97.

[387] D. Chazan and W Miranker. “Chaotic Relaxation”. In: *Linear Algebra and its Applications* 2 (1969), pp. 199–222.

[388] Stuart Russell, Peter Norvig, and Artificial Intelligence. “A modern approach”. In: *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs* 25 (1995), p. 27.

[389] J. Ross Quinlan. “Induction of decision trees”. In: *Machine learning* 1.1 (1986), pp. 81–106.

[390] Silvano Martello and Paolo Toth. *Knapsack Problems: Algorithms and Computer Implementations*. New York, NY, USA: John Wiley & Sons, Inc., 1990. ISBN: 0-471-92420-2.

[391] J. Huang, T. Pretz, and Z. Bian. “Intelligent solid waste processing using optical sensor based sorting technology”. In: *Image and Signal Processing (CISP), 2010 3rd International Congress on*. Vol. 4. Oct. 2010, pp. 1657–1661.

[392] Mahmood R Golzarian and Ross A Frick. “Classification of images of wheat, ryegrass and brome grass species at early growth stages using principal component analysis”. In: *Plant Methods* 7.1 (2011), p. 28.

[393] Cong Liu, Jie Han, and Fabrizio Lombardi. “A low-power, high-performance approximate multiplier with configurable partial error recovery”. In: *Proceedings of the conference on Design, Automation & Test in Europe*. European Design and Automation Association. 2014, p. 95.

[394] Andrew B Kahng and Seokhyeong Kang. “Accuracy-configurable adder for approximate arithmetic designs”. In: *Proceedings of the 49th Annual Design Automation Conference*. ACM. 2012, pp. 820–825.

[395] Ting Wang, Qian Zhang, and Qiang Xu. “ApproxQA: A unified quality assurance framework for approximate computing”. In: *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2017, pp. 254–257.

[396] Chengwen Xu, Xiangyu Wu, et al. “On Quality Trade-off Control for Approximate Computing Using Iterative Training”. In: *Proceedings of the 54th Annual Design Automation Conference 2017*. ACM. 2017, p. 52.

[397] Swagath Venkataramani, Anand Raghunathan, et al. “Scalable-effort classifiers for energy-efficient machine learning”. In: *Proceedings of the 52nd Annual Design Automation Conference*. ACM. 2015, p. 67.

[398] Frank Hutter, Lin Xu, et al. “Algorithm runtime prediction: Methods & evaluation”. In: *Artificial Intelligence* 206 (2014), pp. 79–111.

[399] Ling Huang, Jinzhu Jia, et al. “Predicting execution time of computer programs using sparse polynomial regression”. In: *Advances in neural information processing systems*. 2010, pp. 883–891.