Karlsruhe Institute of Technology

Thermal Management for Dependable On-Chip Systems

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

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

genehmigte

Dissertation

von

Thomas Ebi

aus Basel

Tag der mündlichen Prüfung:	8. Juli 2014
Erster Gutachter:	Prof. Jörg Henkel
Zweiter Gutachter:	Prof. David Atienza
Dritter Gutachter:	Prof. Mehdi Tahoori

for my father,
the original Dr. Ebi

First and formost, I would like to thank my wife and my family for their continued support without which none of this would be possible. I would also like to thank Prof. Jörg Henkel for his advice and guidance that have led me here. Additional thanks go out to my many collegues that have accompanied me through the years. Thank you also to my co-supervisors David Atienza and Mehdi Tahoori for their valuable input.

List of Own Publications Included in this Thesis

1. Major contributions to thesis

[43] Thomas Ebi, M Faruque, and Jörg Henkel. TAPE: Thermal-aware agent-based power econom multi/many-core architectures. In *Computer-Aided Design-Digest of Technical Papers, 2009. ICCAD 2009. IEEE/ACM International Conference on*, pages 302–309. IEEE, 2009.

[45] Thomas Ebi, David Kramer, Wolfgang Karl, and Jörg Henkel. Economic learning for thermal-aware power budgeting in many-core architectures. In *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2011 Proceedings of the 9th International Conference on*, pages 189–196. IEEE, 2011.

[44] Thomas Ebi, Hussam Amrouch, and Jörg Henkel. Cool: control-based optimization of load-balancing for thermal behavior. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 255–264. ACM, 2012.

[56] Jörg Henkel, Thomas Ebi, Hussam Amrouch, and Heba Khdr. Thermal management for dependable on-chip systems. In *ASP-DAC*, pages 113–118, 2013.

2. Minor contributions to thesis

[46] Thomas Ebi, Holm Rauchfuss, Andreas Herkersdorf, and Jörg Henkel. Agent-based thermal management using real-time i/o communication relocation for 3d many-cores. *Integrated Circuit and System Design. Power and Timing Modeling, Optimization, and Simulation*, pages 112–121, 2011.

[15] Hussam Amrouch, Thomas Ebi, and Jörg Henkel. Stress balancing to mitigate nbti effects in register files. In *The 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'13)*, 2013.

[67] Muhammad Ismail, Osman Hasan, Thomas Ebi, Muhammad Shafique, and JÖrg Henkel. Formal verification of distributed dynamic thermal management. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, pages 248–255, 2013.

The Big Picture

The Chair for Embedded Systems (CES) at the Karlsruhe Institute of Technology (KIT) provided me with a productive research environment within which to complete my thesis. Currently, the focus at the CES for architecture and design lie in multi-core systems, dependability, and low power. This includes, for instance, both adaptive and self-organizing on-chip systems as well as leveraging dark silicon to achieve thermal/dependability goals.

As with any work of this nature, this thesis builds upon the foundation laid by past explorations and has benefited from various collaborations. My initial work in the realm of distributed run-time adaptive management was done during my Diploma thesis [CES1], which developed AdNoC, an adaptive Network-on-Chip architecture. This work also manifested itself in two publications of the CES: In [CES2], the adaptive Network-on-Chip router architecture is presented, and in ROAdNoC [CES3] the corresponding distributed monitoring infrastructure. Al Faruque et al subsequently published their landmark paper ADAM [CES4] for the CES introducing agent-based resource management to multi-core systems, and thereby overcoming growing scalability concerns.

The developed concepts were then incorporated and expanded in the Digital On-demand Computing Organism (DodOrg) project. This collaboration between Computer- and Electrical-Engineering departments at the KIT (including the CES) along with the Embedded Systems department of the University of Frankfurt, aimed to create a computing architecture inspired by processes occurring in nature [CES5] [CES6]. Initially the contribution of the CES focused on power management. This was later extended to also include thermal management (TAPE [CES7]). This was realized through the application of economic principles to agent-based management. The DodOrg project itself was part of a wider initiative, the Priority Program (*Schwerpunktprogramm, SPP*) 1183, "Organic Computing", of the German Research Foundation (DFG).

The CES also played a pivotal role in the creation of a further Priority Program, the DFG SPP 1500 "Dependable Embedded Systems". This ongoing SPP deals with the growing dependability concerns in nano-scale CMOS [CES8] due to aging effects, high power densities, high process variation, etc. The SPP 1500 unifies various individual projects throughout Germany, working together to identify future trends and to model/quantify and improve dependability [CES9]. An example of the challenges it aims to address is managing the aforementioned dark silicon [CES10]. The projects cover a wide scope: spanning from the architecture level over the OS level to the compiler level. The SPP 1500 also cooperates with NSF Variability Expedition, a US initiative for the development of variability-aware software for nano-scale devices. One of the contributions of the CES is the VirTherm-3D project – in collaboration with TU Munich (TUM) [CES11]. In it, mulit-core adaptivity is employed in order to achieve dependability in a 3D layered architecture. The VirTherm-3D project is also responsible for the joint hardware platform available to all SPP 1500 members.

The CES is also involved in multiple sub-projects of the DFG Transregional Collaborative Research Center (TCRC) 89 “Invasive Computing”. This ambitous undertaking of KIT, TUM, together with the Friedrich Alexander University in Erlangen (FAU) has set its goal to facilitate self-adaptive and resource-aware programming by unifying new programming concepts, languages, compilers, operating systems together with a customized multi-core architecture [CES12]. The key idea therein being that programs are given the possibility to spread compution of code fragments over a dynamic area in a multi-core system – in a sense “invading” additonal resources. This results in a high degree of paralellism. One of the contributions of the CES is an agent-based approach that manages computational resources and facilitates coordination of invasion [CES13] in a distributed manner, thus maintaining scalability.

[CES1] T. Ebi, “Design and analysis of adaptive networks on chip,” Diploma Thesis, University of Karlsruhe, 2007.

[CES2] A. Faruque, M. Abdullah, T. Ebi, and J. Henkel, “Run-time adaptive on-chip communication scheme,” in *Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on.* IEEE, 2007, pp. 26–31.

[CES3] M. A. A. Faruque, T. Ebi, and J. Henkel, “ROAdNoC: Runtime observability for an adaptive network on chip architecture,” *IEEE/ACM International Conference on Computer-Aided Design, 2008. ICCAD 2008*, pp. 543–548, Nov. 2008.

[CES4] M. A. A. Faruque, R. Krist, and J. Henkel, “Adam: Run-time agent-based distributed application mapping for on-chip communication,” *Design Automation Conference (DAC)*, pp. 760–765, 2008.

[CES5] J. Becker, K. Brändle, U. Brinkschulte, J. Henkel, W. Karl, T. Köster, M. Wenz, and H. Wörn, “Digital on-demand computing organism for real-time systems.” in *ARCS Workshops*, vol. 81, 2006, pp. 230–245.

[CES6] T. Ebi, D. Kramer, C. Schuck, A. von Renteln, J. Becker, U. Brinkschulte, J. Henkel, and W. Karl, “Dodorg - a self-adaptive organic many-core architecture,” in *Organic Computing - A Paradigm Shift for Complex Systems.* Springer, 2011, pp. 353–368.

[CES7] T. Ebi, M. A. A. Faruque, and J. Henkel, “TAPE: Thermal-aware agent-based power economy for multi/many-core architectures,” in *27th IEEE/ACM International Conference on Computer-Aided Design (ICCAD ’09)*, 2009, pp. 302–309.

[CES8] J. Henkel, L. Bauer, J. Becker, O. Bringmann, U. Brinkschulte, S. Chakraborty, M. Engel, R. Ernst, H. Härtig, L. Hedrich *et al.*, “Design and architectures for dependable embedded systems,” in *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2011 Proceedings of the 9th International Conference on.* IEEE, 2011, pp. 69–78.

[CES9] J. Henkel, L. Bauer, N. Dutt, P. Gupta, S. Nassif, M. Shafique, M. Tahoori, and N. Wehn, “Reliable on-chip systems in the nano-era: lessons learnt and future trends,” in *Proceedings of the 50th Annual Design Automation Conference.* ACM, 2013, p. 99.

[CES10] M. Shafique, S. Garg, D. Marculescu, and J. Henkel, “The eda challenges in the dark silicon era: Temperature, reliability, and variability perspectives,” pp. 185:1–185:6, 2014.

[CES11] T. Ebi, H. Rauchfuss, A. Herkersdorf, and J. Henkel, “Agent-based thermal management using real-time i/o communication relocation for 3D many-cores,” in *Integrated Circuit and System Design. Power and Timing Modeling, Optimization, and Simulation workshop (PATMOS)*, 2011, pp. 112–121.

[CES12] J. Henkel, A. Herkersdorf, L. Bauer, T. Wild, M. Hübner, R. K. Pujari, A. Grudnitsky, J. Heisswolf, A. Zaib, B. Vogel, V. Lari, and S. Kobbe, “Invasive manycore architectures.” in *ASP-DAC*, 2012, pp. 193–200.

[CES13] S. Kobbe, L. Bauer, D. Lohmann, W. Schröder-Preikschat, and J. Henkel, “Distrm: distributed resource management for on-chip many-core systems,” in *Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis.* ACM, 2011, pp. 119–128.

Abstract

Today's fabrication technology, operating at technology nodes of 22 nm and below, allows the integration of several billion transistors on a single chip. A major drawback of the high integration density, is the resulting increase in power per chip area. This power density directly results in elevated on-chip temperatures. The small structure sizes also lead to a number of reliability issues, of which the most concerning are again dependent on temperature. Examples of such issues are electromigration and NBTI ("Negative Bias Temperature Instability") aging mechanisms. Recently, NBTI has emerged as the most dominant in current technologies.

Initially, this thesis addresses the reliability issues from a thermal perspective, focusing mostly on NBTI. This includes an explanation and analysis of models developed together with our partners from the semiconductor industry, in order to show the relationship between reliability and temperature. Additionally multiple thermal management strategies are examined in order to analyze their effects on reliability. This analysis shows that it is possible to double chip lifetime by choosing the suitable thermal management strategy.

Subsequently, multiple novel methods for on-chip thermal management are introduced aiming to optimize thermal properties. This includes both methods for reducing peak temperatures as well as for reducing both spatial and temporal thermal variation. The State-of-the-Art divides thermal management methods into two classes: so-called *reactive* methods act when a given temperature threshold is hit; *proactive* methods, on the other hand, rely on thermal or statistical models to make predictions on expected future temperatures, and act based on these predictions (i.e. before a threshold is hit).

The presented methods are first applied to the processor microarchitecture. Here, the *Control-Based Optimization of Load-Balancing* (COOL) is examined whose goal it is to distribute computational activity among individual components of a processor in order to minimize both peak temperatures and thermal variations. This is realized by partitioning a component and performing a self-optimizing assignment of computational activity to each component part.

Next, thermal management methods for multi-core processors are presented and analyzed. These are designed to exhibit a high degree of scalability to remain applicable to computer

architectures with a high number of processing cores. This means avoiding a central "bottleneck" and reducing communication and computational overhead. The underlying mechanism employed is an agent-based system that operates distributed over the chip. Thus the problem complexity can be reduced by exploiting locality when making thermal management decisions.
The most important methods, that were developed are the following:

- *Thermal-Aware Agent-Based Power Economy* (TAPE): A completely *decentralized* approach where each processor core is assigned a power budget. This budget is traded by agents of neighboring cores based on supply and demand (i.e. how much power is currently being used to run application tasks and how much additional power is available). The temperature influences this trading by increasing the cost of the power budget. The power budget can be distributed throughout the entire chip through propagation from core to core over multiple successive trading intervals.
- *Economic Learning for Thermal-Aware Power Budgeting* (EcoLe): A *hierarchical* approach where power budget is distributed among clusters. In this context, a cluster is a grouping of a subset of cores located in a defined continuous region of the chip. Agents acting at cluster-level are responsible for both the power budget within the cluster as well as providing information to higher hierarchy levels which can in turn disseminate this information among other clusters. This allows exploiting locality for management decisions within a cluster while also retaining access to global system knowledge (e.g. the temperature distribution).

Analysis of the methods is done partially through simulation and partially through implementation on an FPGA-based hardware platform whose thermal behavior is captured using both on-chip sensors and an infrared thermal camera. This allows evaluation of the methods under real operating conditions. On average, the methods for multi-core architectures achieve a reduction of peak temperature by 15°C with only a small decrease in performance. The method optimizing the microarchitecture reduces the peak temperature by 13°C. This reduction is slightly smaller, as a smaller area footprint of the processor is affected. Through these methods, chip lifetime can be increased by at least a factor of two.

Zusammenfassung

Heutige Fertigungstechnologien, welche Strukturgrößen von 22 nm und darunter haben, erlauben es, mehrere Milliarden Transistoren auf einem einzelnen Chip zu integrieren. Ein großer Nachteil hoher Integrationsdichte ist es, dass sich der Leistungsverbrauch pro Chipfläche ebenfalls erhöht ("Leistungsdichte"), was direkt steigende Temperaturen auf dem Chip mit sich bringt. Mit kleinen Strukturgrößen treten auch eine Reihe von Zuverlässigkeitsproblemen auf, von denen die mit den gravierendsten Auswirkungen wiederum von der Temperatur abhängig sind. Als Beispiel hierfür gelten z.B. Elektromigration und NBTI ("Negative Bias Temperature Instability"), von denen vor allem letzteres in jüngster Zeit immer häufiger in den Vordergrund rückt. Die vorgestellte Arbeit befasst sich zunächst mit diesen Zuverlässigkeitsproblemen und setzt dabei einen Schwerpunkt auf NBTI. Diesbezüglich werden die von uns, in Zusammenarbeit mit unserem industriellen Partner entwickelten Modelle erläutert und analysiert, um das Verhältnis zwischen Temperatur und Zuverlässigkeit zu verdeutlichen. Weiterhin werden verschiedene Temperaturmanagement-Strategien und deren Auswirkungen auf die Zuverlässigkeit der Chips untersucht. Hierbei wird gezeigt, dass es möglich ist, durch eine geeignete Temperaturmanagement-Strategie die Lebensdauer von Chips zu verdoppeln.

Zunächst werden mehrere neue Methoden zum Management der Chiptemperatur vorgestellt, mit dem Ziel Temperatureigenschaften zu optimieren. Dies betrifft sowohl Methoden zur Senkung der Höchsttemperatur wie auch Methoden zur Reduzierung der räumlichen und zeitlichen Variation der Temperatur. Der Stand der Forschung teilt entsprechende Methoden in zwei Klassen auf: sogenannte *reaktive* Methoden agieren wenn ein vorgegebener Temperaturschwellwert übertroffen wird; *proaktive* Methoden setzen hingegen auf thermische oder statistische Modelle um Vorhersagen über zu erwartende Temperaturen zu treffen und agieren dann, um den vorhergesagten Zustand zu vermeiden. Zunächst werden die vorgestellten Methoden angewandt auf der Mikroarchitektur eines Prozessors. In diesem Rahmen wird ein Verfahren (*Control-Based Optimization of Load-Balancing*, COOL) vorgestellt welches zum Ziel hat, die Rechenaktivität auf individuelle Komponenten des Prozessors zu verteilen, so dass sich dessen Höchsttemperatur wie auch dessen Temperaturvariationen minimieren. Dies erfolgt durch eine Partitionierung einer Komponente und einer selbst-optimierenden Zuweisung der Rechenaktivität an diese Teilkomponenten.

Des Weiteren werden Methoden für Multiprozessorsystemen vorgestellt und analysiert.

Diese sind dazu konzipiert, einen hohen Grad an Skalierbarkeit zu erzielen, um auf Architekturen mit einer Vielzahl von Prozessorkernen anwendbar zu bleiben, d.h. ohne einen (zentralen) "Flaschenhals" zu haben bzgl. sowohl Kommunikations- als auch Rechenaufwand. Der zugrunde liegende Mechanismus hierfür ist ein agenten-basiertes System, welches über den gesamten Chip verteilt arbeitet. Dadurch wird die Problemkomplexität durch die Ausnutzung von Lokalität auf einem kleineren Suchraum reduziert. Die wichtigsten Methoden, welche entwickelt wurden:

- *Thermal-Aware Agent-Based Power Economy* (TAPE): Ein komplett dezentrales Verfahren, in welchem jeder Prozessorkern ein Energiebudget zugewiesen bekommt. Dieses Budget wird dann, basierend auf Angebot und Nachfrage (d.h. wie viel Energie gerade gebraucht wird um Anwendungen auszuführen bzw. wie viel Energie verfügbar ist) zwischen benachbarten Kernen getauscht. Die Temperatur beeinflusst diesen Austausch durch eine Erhöhung der Kosten der Energie. Durch mehrere aufeinander folgende Tauschintervalle ist es möglich, Energiebudgets von Kern zu Kern über den gesamten Chip zu propagieren.
- *Economic Learning for Thermal-Aware Power Budgeting* (EcoLe): Ein hierarchisches Verfahren welches Energiebudgets auf Clusterebene verteilt. Hierbei bezeichnet ein Cluster eine Gruppierung von einer Menge von Kernen innerhalb eines definierten Bereichs des Chips. Agenten, welche einen Cluster verwalten, sind dafür zuständig sowohl das Energiebudget in ihrem Cluster zu verteilen, als auch den Informationsaustausch über höher gelegene Hierarchieebenen mit entsprechenden Agenten anderer Cluster durchzuführen. Somit kann auch hierbei räumliche Lokalität ausgenutzt werden und das System hat dennoch Zugriff auf die globalen Temperaturinformationen der höheren Ebenen.

Die Analyse der Verfahren verlief teilweise durch Simulationen, teilweise aber auch durch eine Implementation auf einer FPGA-basierten Hardwareplattform, dessen thermisches Verhalten von einer Infrarotkamera erfasst wurde. So war es möglich, die Verfahren unter realen Bedingungen zu evaluieren. Die Verfahren erzielen im Durchschnitt eine Senkung der Höchsttemperaturen um etwa 15°C bei geringer Steigerung der Laufzeiteffizienz (vergleichbare Verfahren benötigen 44% mehr Laufzeit). Bei der Optimierung der Mikroarchitektur werden die Höchsttemperaturen im Durchschnitt um 13°C gesenkt (die Reduktion ist hier etwas geringer, da die betroffene Chipfläche insgesamt kleiner ist als bei dem Management von Multiprozessorsystemen), ohne Erhöhung der Laufzeit. Die Lebensdauer des Chips wird durch die Verfahren um mindestens einen Faktor zwei erhöht.

Contents

1. Introduction

Temperature has long been a limiting factor in electronics, defining boundaries on performance and power consumption as well as limiting dependable operation. This holds particularly true in current VLSI design where the market demands ever-increasing performance. Temperature management is a means for dealing with these limits and pushing the performance envelope by altering power consumption or increasing heat dissipation. The first line of defense lies in packaging design. Adding a heat sink greatly increases heat dissipation – even more so the use of active cooling methods, e.g. using fans or liquid cooling. While designing these physical means for removing heat from a chip are indispensable, the means themselves are often limited due to cost or area constraints. As the power per area grows, e.g. cooling fans have increased in both size and in their own power consumption, incurring high energy cost. This particularly presents a problem in systems where space is severely limited, e.g. as is the case in embedded systems. All this makes it necessary to add another layer of temperature management: system-level Dynamic Thermal Management (DTM) with the goal of managing temperature by controlling the power usage of a chip during runtime. This need for DTM is continually rising as temperatures and dependability concerns increase, largely due to the effects of technology scaling.

In 1965, G. Moore published his seminal work predicting the trend in sizing and complexity of semiconductor devices. Often referred to as Moore's law, this prediction calls for the doubling of the number of transistors every two years [91]. It remains valid until the present day as feature sizes have steadily shrunk over the past decades. Technology scaling has opened up the path to ever increasingly complex systems. The large amount of transistors available allow placing multiple processors or even whole systems consisting of several components joined together by an on-chip network on a single chip, for instance.

The exponential growth comes at a price due to the pitfalls that accompany small structure sizes. These structures are inherently less robust and as a result, more susceptible to failure than larger ones. Small cracks or erosions that were unnoticeable at the micrometer level are catastrophic regarding failures in nano-scale devices. These defects can occur during the manufacturing process, but may also occur due to degradation over time caused by temperature-dependent aging effects. The small structures are also plagued by a great deal of variability. While the absolute variance has remained the same or decreased, it is its relative impact that is concerning. While size variations of a few nm were negligible in larger structures (i.e. $>$ 45 nm), these can have a great impact as sizes fall below 20 nm. Device characteristics are no longer deterministic at design time, necessitating wider safety margins to deal with the expected range of characteristics.

The problems arising from both aging-effects as well variability are compounded by the decrease in voltage bias. Where a large potential difference may be able to compensate some of these effects (e.g. when developing defects increase resistance and thereby reduce currents), a small voltage bias fails a lot sooner. Since the difference between supply and threshold voltages is also shrinking, the potential barrier hindering charge carriers is lower. As a result safety margins are considerably reduced, thereby for instance decreasing the critical charge Q_{crit} of a circuit and increasing susceptibility to soft errors. In a sense, this is a double edged sword. While on the one hand a higher voltage is able to compensate device degradation, on the other it also accelerates its effects by increasing power consumption and thereby temperature.

While in general the power consumption of a specific design decreases when moving from one technology node to the next, the power *densities* can increase dramatically due to the decrease in occupied area. Typically, however, designs in smaller technology nodes also contain more features – either the designs are inherently more complex or contain duplications (as is the case in multi-core architectures) – and thereby also increase the overall power consumption, especially if these also target higher frequencies. Together this means elevated temperatures which can threaten dependability.

There is also a great deal of complexity incurred when managing systems running on designs consisting of billions of transistors. This creates a high-dimensional problem space where solutions are sought for problems that are inherently complex – many management problems, i.e. assigning resources, already being in the realm of NP-hard problems. This calls for solutions which offer a large degree of scalability, oftentimes based on heuristics which aim to find approximations for optimal solutions where finding optimal solutions is not feasible.

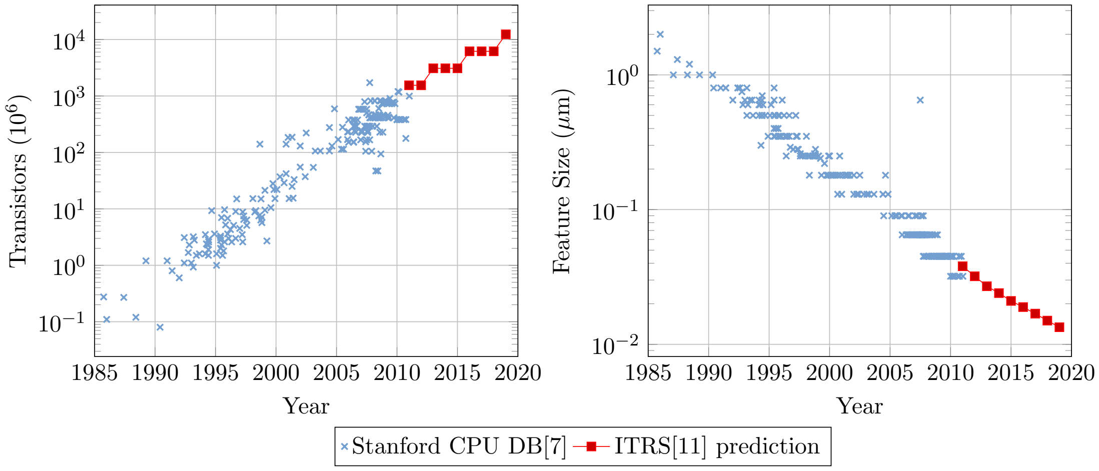


Figure 1.1.: Trends in number of transistors and feature size. Each point from the CPU DB represents a unique processor.

1.1. A Closer Look at Processor Trends

The Stanford CPU DB [7] affords us with a repository of statistics on a large portion of current and past processors. Additionally, the International Technology Roadmap for Semiconductors (ITRS) regularly publishes predictions on the future of semiconductor devices. Together, these sources provide data which can give many insights on technology development and future trends. From a thermal perspective it is of particular interest how these trends affect the power consumption of a chip.

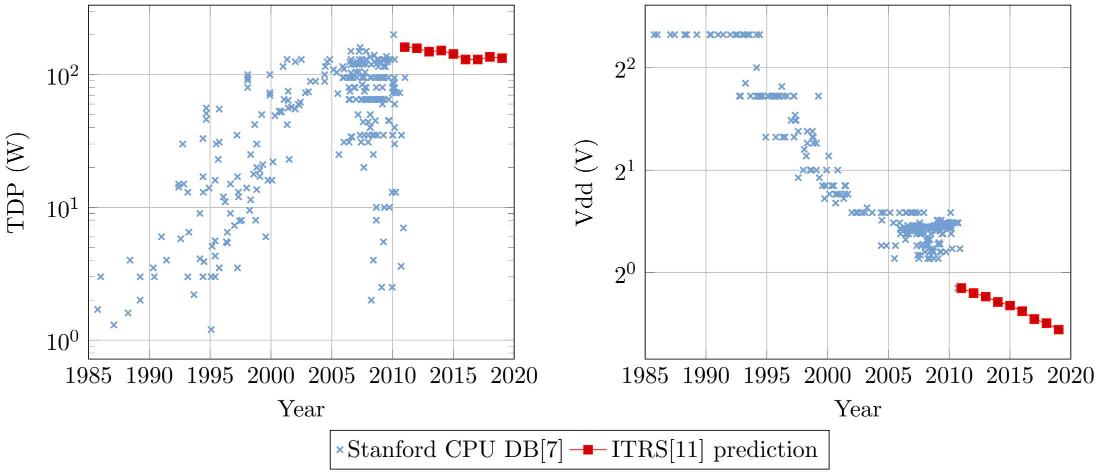


Figure 1.2.: Trends in Thermal Design Power (TDP) and supply voltage (Vdd)

Concerning general scalability, Figure 1.1 corroborates the validity of Moore's law. The amount of transistors has indeed experienced an exponential growth. The same holds true for the used feature sizes over time, although these are limited to discrete values corresponding to technology nodes. As we can see, the ITRS predicts this trend to continue into the future. More importantly, when considering temperature is how technology scaling affects Thermal Design Power (TDP). TDP is a metric supplied by chip manufacturers that represents the power consumed when the chip is under maximum workload. This serves as a guide for designers of temperature management by giving the maximum power that needs to be removed from the chip when transferred into heat. As can be seen, the TDP underwent exponential growth up until around 2005, but has since begun to level out in its maximum. Why this is the case is explained in conjunction with the discussion of voltage below. It is also evident that current processors exhibit a wide spectrum of TDPs, especially compared to past generations. This demonstrates a current trend of diversification of processors as many now specifically target low power (for battery-powered mobile devices, for instance) while others target high performance.

The voltage of a processor determines how fast parasitic capacitances can be charged and thus determines how fast, e.g. logic gates can operate. Smaller feature sizes result in smaller capacitances, meaning a given design will require less voltage to operate at the same speed compared to larger feature sizes. As a result, the required voltage for a given feature size depends on the desired frequency. Initially, as can be seen in Figure 1.2, supply voltage (Vdd) was fixed at 5 V. Theoretically this convention would have allowed ever-increasing frequencies, but fixing the voltage became less and less feasible as the number of transistors grew causing power consumption to explode.

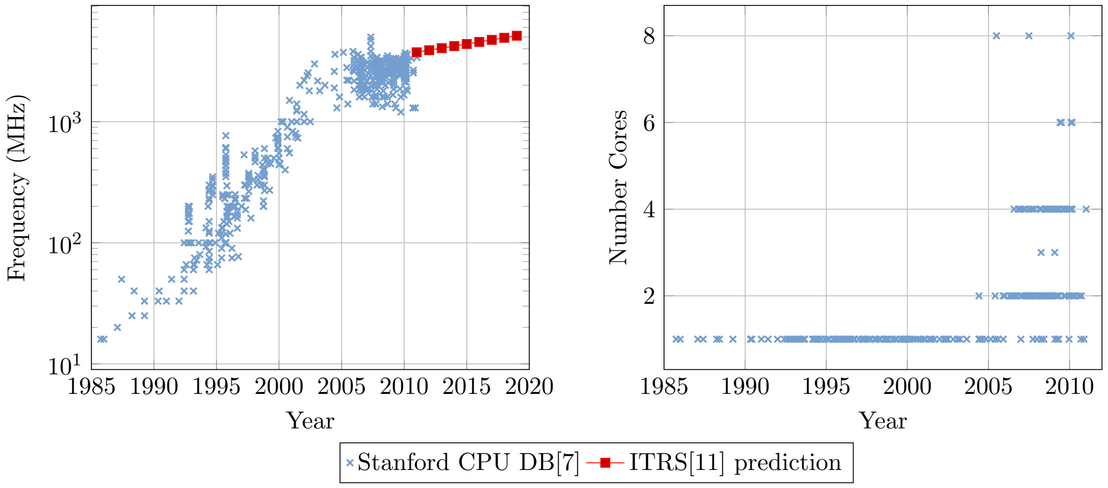


Figure 1.3.: Trends in operating frequency and typical number of cores per design

Typically voltage by itself scales with the square root of feature size. However, the demand for extra performance has historically called for an increase in operating frequency which again means increasing the reduced voltage. Thus, the voltage decrease from one technology node to the next has typically been reduced, i.e. the potentially attainable voltage decrease was sacrificed for the sake of performance. An interesting change happens in Figure 1.3 after 2005. As can be seen, at this time frequency begins to rise at a much lower rate. At the same time processors begin to incorporate multiple cores. This paradigm shift represents the beginning of the multi-core era where increased performance demands are additionally met through parallelization. This was made necessary to avoid hitting a power wall where the produced power is unable to be dissipated as heat.

As a continuation of this trend the ITRS predicts the number of cores to double each 18 months [11] analog to Moore's predicted doubling of transistors. Current front runners in this domain include the Intel Single-chip Cloud Computer (SCC) with 48 cores [10] and the Tilera TILE-Gx family of processors with up to 72 cores [51]. Future multicore architectures are predicted to integrate thousands of cores [26] on a single chip. The performance of these architectures will depend largely on the amount of parallelization that can be achieved which is limited by Amdahl's law [59]. Theoretically, a large improvement is possible in terms of performance per Watt for highly parallelizable applications.

The most relevant trend when regarding temperature is the development of power densities over time. Figure 1.4 depicts the average worst-case power densities, i.e. the average power density on chip when under a worst-case workload. The presented graph is similar to that of TDP, indicating that overall chip area has largely remained the same. Power densities also grew continuously until around 2005.

To achieve a complete picture of predicted future trends, however, it is necessary to adopt a more differentiated view of processor technologies. To this effect, the ITRS defines two processor classifications: *high-performance* and *cost-performance*.

For high-performance processors, the severe increase in power density that was predicted at that time has failed to come into effect. Instead, due to the aforementioned paradigm shift to multi-cores along with the research into new technologies (high-k dielectrics, FinFET), power densities have stopped increasing and are predicted by the IRTS to level out around

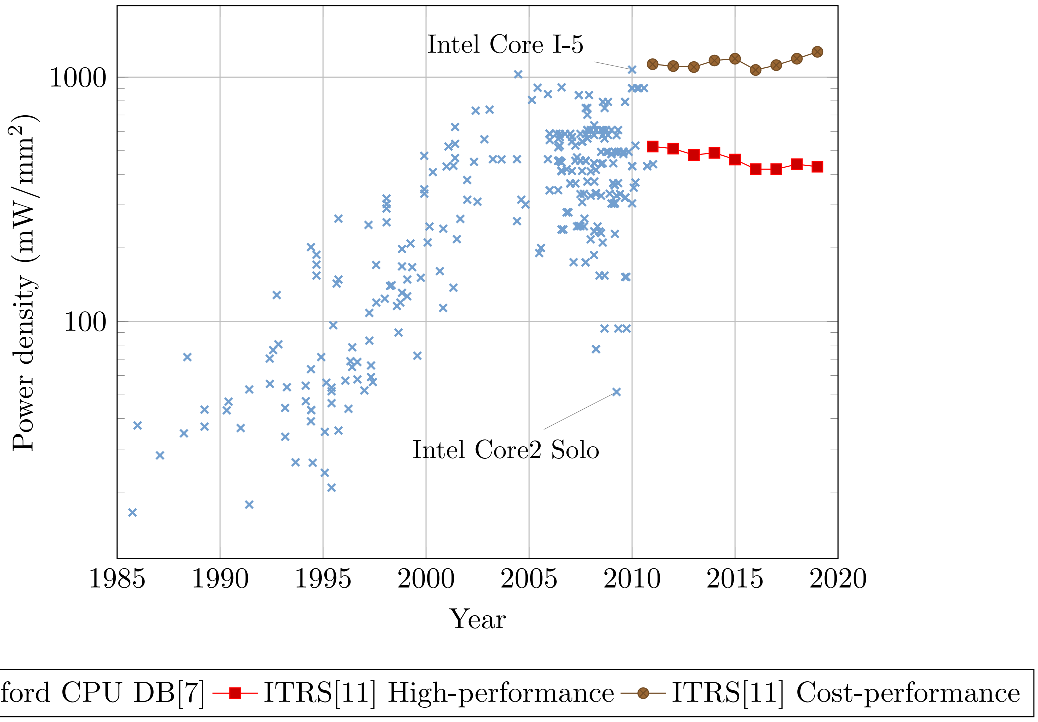


Figure 1.4.: Development of power density over time calculated using data from [7] and ITRS [11] predicted future stabilization of power density

0.42 W/mm^2 in the coming years. While this is true on average, a higher integration density can still result in higher peak power densities. For instance, these can range from around 2 W/mm^2 in the 65 nm technology node to 7.2 W/mm^2 for 45 nm [81].

Unfortunately, although it is possible to manufacture high-performance processors with little or no increase in power density, this comes with inhibitive design and production costs that are contrary to ubiquitous computing demands. As such, a large number of future processors will instead be optimized for cost-performance. For these, the future trend in Figure 1.4 predicts further rise in power density. This in itself presents a number of challenges to hardware-software co-design to be able to utilize a chip to its full potential – for instance leveraging supply voltage with dark silicon [111]. Regardless whether power densities increase or not, the growing susceptibility of small structure sizes mean thermal management will still grow in importance from a dependability viewpoint. Only the employed strategies may shift to favor ones that minimize long-term aging effects.

More recently, 3D chip architectures have emerged to increase performance per chip area. So far, the most common manufacturing technique involves multiple individual chip layers stacked on top of each other and connected through Through Silicon Vias (TSVs). Unfortunately, surface power densities are compounded in these 3D architectures. In the general case, heat must first travel through multiple layers before reaching the heat sink, which is only located at the top most layer.

1.2. Key Challenges

Dependable system operation has long been expected from hardware. In this context, *dependability* is a property that encompasses both *reliability* and *availability*. That is, execution is expected to terminate correctly and resources are expected to remain available throughout the lifetime of a chip. The temperature-dependent aging effects typically first threaten reliable operation. These gradually degrade device parameters, altering their operating characteristics, e.g. such as circuit delay. Eventually, they may lead to failures of components and thereby limit their availability.

Typically DTM techniques target optimizing performance within given thermal constraints. Little work has been done examining the effects of different thermal management strategies on system lifetime. Intuitively, any form of thermal management will reduce most temperature-dependent aging effects. Thus, constraining temperatures has the added bonus of increased general dependability. Not all strategies are equal, though. One that achieves the highest performance may not necessarily be the same one that minimizes aging. Moreover, reducing one aging mechanism may result in worsening the effects of another.

- Therefore, it is necessary to **bridge the gap between thermal management and dependability**. In this scope, considering the effects of temperature on multiple aging mechanisms is required to determine the overall lifetime of a chip. In particular this means analyzing thermal management strategies and their effect on aging mechanisms. This can be used to determine situations where minimizing one aging effect will worsen another, for instance as is the case with Negative Bias Temperature Instability (NBTI) and thermal cycling.

As mentioned, the main consideration of DTM is keeping temperatures within predefined bounds which are deemed to be a safe range. When optimizing for performance, the system generally operates at maximum allowable power that keeps temperatures as close to a specified fixed threshold as possible. This is far from optimal when targeting dependability, however. New strategies need to be developed that reduce temperature, ideally with little impact on performance.

- Management strategies targeting dependability to increase system lifetime must minimize aging effects. From a thermal standpoint this means a shift from optimizing performance within thermal constraints to **optimizing temperature** while keeping performance constraints. This entails both reducing peak temperatures as well as thermal gradients, as both can have a negative impact on dependability (cf. Chapter 3).

Processors in the multi-core era offer a great deal of potential for dynamic thermal management. Apart from traditional methods such as Dynamic Voltage and Frequency Scaling (DVFS), they allow controlling the power density distribution through the migration of tasks to cores. Management problems such as these are typically computationally expensive. Even when considering single cores, in its simplest form, the assignment of tasks reduces to the np-hard[1] knapsack problem [27]. Expanded to multiple cores, it becomes the even more complex multiple knapsack problem. In this sense, general thermal man-

[1] While determining *if* a solution to the knapsack problem exists is np-complete, finding the solution itself is np-hard

agement through task mapping can be expressed as the following:

$$\begin{aligned} \text{maximize :} \quad & \sum_{i=1}^{m}\sum_{j=1}^{n} p_{i,j}x_{i,j}, \\ \text{subject to :} \quad & \mathrm{MAX}_{i=1}^{m} T(\sum_{j=1}^{n} x_{i,j}) < T_{\max}, \end{aligned} \qquad x_{i,j} \in \{0,1\} \qquad (1.1)$$

when mapping n tasks to m cores. $x_{i,j} = 1$ if task j is mapped to core i. $p_{i,j}$ is the value of mapping task j to core i, i.e. in terms of performance. Here, T is a temperature function which determines the temperature of each core (i.e. for a static mapping this would correspond to a steady-state temperature). In reality, of course, determining temperatures is more difficult, as heat is also conducted to and from the surroundings of a core. Furthermore, when performing *dynamic* thermal management, the knapsack problem needs to be solved periodically during runtime. This also entails continuously updating the temperature function. In practice, this requires either the use of models for predicting future temperatures or continuous observational feedback and control.

Typically, State-of-the-Art thermal management is done by a central instance. This creates both a central point of failure as well as a communication bottleneck. To be able to make DTM decisions, the central instance must be kept up-to-date on the state of each core, which requires continuous transmission of temperature data. The only way to deal with these problems is shifting DTM to novel distributed techniques.

Even with the large communication overhead, the major advantage of a central instance is the availability of global system knowledge. It will have information about all tasks and all cores and their current state, e.g. temperature, currently running tasks, etc. As such, a central instance always has more potential for finding optimal solutions than a distributed one. However, the algorithmic complexity rapidly grows, as architectures with several cores running several tasks create a large problem space. Centralized approaches quickly become infeasible even when using a heuristic solution.

A major challenge is thus finding a way to efficiently provide access to global system knowledge while reducing problem complexity – combining the benefits of a centralized approach with those of distributed ones.

- Novel management techniques are required which **scale to large multi-core architectures**. These techniques should be distributed to reduce bottlenecks (both in communication as well as computation) and avoid having a central point of failure. Furthermore, techniques for distributing and aggregating system knowledge need to be developed to allow distributed DTM to achieve a similar quality as is possible in centralized ones. A general obstacle hindering the adoption of distributed techniques is the difficulty to verify their **stability**. It is thus necessary to analyze these techniques in this respect to guarantee that the individual instances do not contradict each other in their actions. Local optimizations should not lead to e.g. localized temperature minima that diverge too far from, and thus prevent, a global optimization.
- Furthermore, **emerging 3D architectures** exacerbate all temperature problems. Herein, the challenge lies in adapting DTM techniques for planar multi-core archi-

tectures to ones with multiple layers stacked on top of each other. This requires consideration of the high vertical heat conductance.

While DTM has been a necessity for some time, the above challenges reflect the change in requirements brought on by the recent trends in semiconductor devices and the shift in focus from performance optimization to increasing dependability – all while dealing with the complexity of large-scale multi-core architectures.

1.3. Thesis Contribution

In the scope of this thesis, the above key concerns are addressed with the following:

- This thesis presents an overview of the effects of temperature on key reliability concerns, covering the most dominant aging mechanism as well as timing errors. Here particular emphasis is placed on the analysis of Negative Bias Temperature Instability (NBTI). The studied effects are then used to evaluate the overall effect of thermal management on system lifetime.
- Thermal management is in essence a control problem. Temperatures are observed by the DTM infrastructure which in turn changes system configuration. This can either occur as a reaction to the observed temperatures directly, or proactively by considering predicting temperatures based on observations. Using control-theory principles, a management method is presented which optimizes temperatures using *extremum-seeking* control to drive the output temperature function of a system to a minimum.
- Scalability is addressed using a fully distributed agent-based system to manage temperature. This system is based on economic principles to distribute a limited power budget among the cores of a multi-core architecture. It exploits the concept of *locality*, making management decisions only in local regions in order to keep the problem space small. Additionally, an analysis of stability is presented to be able to guarantee proper functionality.
- The fully distributed approach is augmented by a hierarchical one. This has been designed with the goal of facilitating access to global system knowledge while sacrificing as little locality as possible. The approach also contains a basic learning technique to be able to increase knowledge about local thermal states. Additionally, this approach is extended to cover the realm of 3D multi-core architectures.

1.4. Thesis Structure and Overview

The rest of this thesis is organized into five chapters; each one dedicated to its own facet of thermal analysis and management. First, Chapter 2 examines the basics of the temperature development and analysis techniques employed in today's systems. This includes a look at the physics underlying heat generation and heat transfer. The analysis section focuses on different forms of temperature measurement – from on-chip sensors to infrared thermal cameras – and thermal simulation techniques. Additionally, an overview of thermal management techniques in multi-core architectures is presented including a glimpse into 3D architectures and their particular thermal caveats.

Chapter 3 explores the link between chip temperature and key reliability concerns. Initially, an overview of some of the most pressing temperature-related aging effects is given. In this context, particular focus is given to Negative Bias Temperature Instability (NBTI) which includes our own models. Afterwards, the effect of thermal management on dependability is examined by taking different DTM strategies and quantifying their impact on chip lifetime. The content of this chapter is based on and expanded from my work in [56] and [15].

Chapter 4 examines thermal temperature from a control-theoretic standpoint and presents an approach for optimizing temperature that first appeared in [44]. This approach specifically targets microarchitectural components where the performance overhead of migrating computation can be kept low. Analysis is performed using measurements of an FPGA-based implementation as well as through simulation of an implementation in a superscalar register file.

Chapter 5 carries thermal management concepts into the realm of multi-core systems. This chapter presents two proactive thermal management approaches. The first of these is a fully distributed approach based on the economic principle of supply and demand. The second is a hierarchical approach based on auctions that encompasses rudimentary learning. This approach also deals with the challenge of managing temperature in 3D multi-core architectures. The approaches were first introduced in [43] and [45], respectively.

Chapter 6 presents a summary of this thesis and provides an outlook into future work.

2. Thermal Basics and Related Work

Before analyzing the effects of temperature on dependability, it is first necessary to examine the thermal characteristics of a chip. It must be clear how and where heat is created and how it spreads. In this chapter we take a look at these issues and present both theoretical and experimental methods for quantifying these characteristics.

Temperature distribution itself is a well-studied topic, and its development over time can be expressed by the first law of thermodynamics: a change in system temperature is equal to the amount of heat energy supplied/removed to the system (Q) plus the work (W) done by the system. In semiconductor devices, this work equates to power consumption. In the form of a differential equation, the first law of thermodynamics is the following [30]:

$$dT = \delta Q + \delta W \tag{2.1}$$

For CMOS circuits, the power consumption consists of both dynamic and static power, which are dominated by switching and leakage power, respectively. Switching power, which is expressed as [87]

$$P_{switch} = \alpha C f v^2, \tag{2.2}$$

is determined by the load capacitance C of a circuit, the operating frequency f, and the voltage V. Here α is an *activity* factor that states the fraction of the capacitance affected, i.e. which logic gates of the circuit are switching and what fraction of C they represent.

It becomes evident from Equation 2.2 what parameters can be adjusted to perform thermal management based on switching power. Since it factors in quadratically, reducing voltage is the most efficient way of reducing temperature. Reducing activity is possible by changing the physical hardware design or distributing activity over a larger area[1]. Of course, these parameters exhibit interdependencies. For instance voltage and frequency have a quasi-linear relationship since the voltage determines how fast capacitances can be charged. Thus a given frequency will have a minimum required voltage and a given voltage will have a maximum possible frequency.

[1] Technically, we are still dealing with volumes, but assuming the height of a circuit's layers are constant, it is simpler to regard the power density as power per area.

Unlike dynamic power, leakage power for a given circuit and fabrication technology depends primarily on voltage. The most prevalent source of leakage power consumption is due to subthreshold leakage currents. In current technologies the difference between V_{DD} and the threshold voltage V_{th} is reduced in an effort to reduce P_{switch}. This has the side effect that it also reduces the potential barrier preventing leakage currents flowing from V_{DD} to V_{SS} (Gnd) when the transistor is in the subthreshold region, i.e. $V_{GS} < V_{th}$. The percentage of power dissipated as leakage is becoming more and more, having recently surpassed switching power in magnitude [79]. Of course, the small structure sizes themselves also play a role. The reduced channel width increases the electric field between source and drain. It also means that subthreshold leakage currents must only travel a small distance to pass through the transistor.

These currents can be modeled as a diode with the following equation [119]

$$i_{sub} = i_s(e^{q^{V/kT}} - 1) \tag{2.3}$$

where i_s is the reverse saturation current, V the diode voltage, q electronic charge, and k is Boltzmann's constant. As can be seen, leakage currents themselves are also temperature dependent. A common rule of thumb is that leakage current doubles for every 10°C [107]. This can lead to a feedback loop resulting in a so-called thermal runaway scenario: increasing temperatures results in a rise in leakage power which in turn causes temperatures to increase further [14].

Additionally, other leakage currents also contribute to the overall leakage power consumption, for instance gate leakage. Gate leakage is similar to subthreshold leakage, only with currents flowing from the gate to the source/drain or substrate. These currents are the result of various tunneling effects [29]. Gate leakage is reduced through the introduction of high-κ metal gates, where the standard dielectric SiO_2 is replaced with another that has a high dielectric constant. This results in a weaker electric field between gate and bulk.

Not all materials generate the same temperatures through power consumption. The amount of heat energy required to raise the temperature one degree (i.e. J/K) is given by a material's *heat capacity*. This can be given, for instance, as a *volume-specific heat capacity* $J \cdot m^{-3} \cdot K^{-1}$ which is near constant[2] for a particular material at temperatures relevant when observing chip temperatures.

There are three types of heat transfer affecting stationary bodies: *conduction*, *convection*, and *radiation*. Conduction occurs in physically connected substances through the interaction of molecules. Heat is transferred when vibrating molecules relinquish energy to stimulate vibrations in surrounding molecules. Conductivity (k) is a material property expressing its ability to conduct heat, expressed in $Wm^{-1}K^{-1}$. This can be used to determine the rate of heat passing through a given area, i.e. the heat flux (Φ_q), which is dQ/dt per area. Convection is the transfer of heat through fluid motion, e.g. in air or coolant liquid flowing around a heat sink. With radiation, heat is emitted and absorbed as energy in the electromagnetic spectrum. An overview of heat transfer is given in Figure 2.1.

When examining on-chip temperatures, the major heat transfer method of concern is conduction. As chips are typically surrounded by packaging, often including a heat sink, the chips are shielded from direct convection and heat radiation. To serve their purpose

[2] The heat capacity approaches 0 when temperatures approach absolute zero.

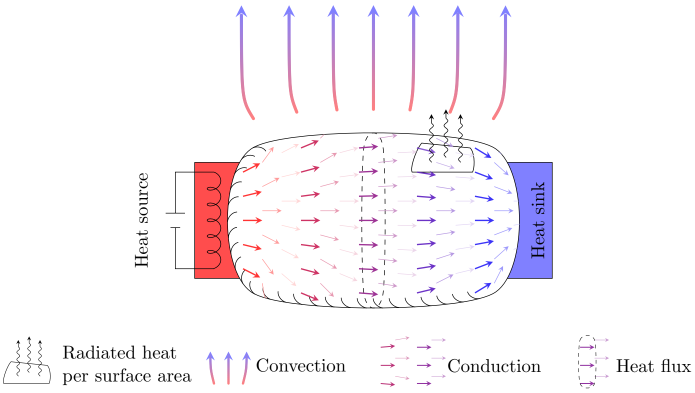


Figure 2.1.: Schematic diagram illustrating forms of heat transfer

of removing heat from the chip, heat sinks must have a high thermal conductivity k. As a result they are near isothermal, abstracting away convection- and radiation-based heat transfer as a fixed thermal sink to the ambient temperature.

Since heat production and conduction are physical processes, there are always latencies involved. How a system reacts to a change in power consumption can be expressed as a step response. This step response is proportional to $\pm e^{-t/\tau}$ (± depending on whether cooling or heating). Here, τ is the so-called *thermal constant* which determines the rate of heating/cooling (cf. Chapter 4.4 for more details). The asymptote that the step response nears is the so-called *steady state temperature* that is reached when the heat generated in the system and the heat dissipating from it reach an equilibrium. This quantity is determined for a fixed power profile, e.g. assuming constant power consumption. In theory, reaching the steady state temperature requires infinite time. However, in practice chip temperatures can be said to reach the steady state in finite time due to the exponential nature of the step response and the finite precision of thermal measurements. In our experiments, this was typically in the order of minutes.

Heat conduction together with spatial variances in power consumption within the chip results in thermal gradients ∇T. These can negatively influence dependability as will be seen in Chapter 3.

2.1. Thermal Simulation

Often, it becomes necessary to perform offline analysis of thermal behavior. This is particularly relevant during the design phase, for instance when determining the TDP which serves as a guide for developing cooling solutions, or performing thermal-aware layouting of system components on a chip in order to avoid a spatial concentration of thermal hotspots. Since it is not feasible to manufacture chips with all possible configurations, this step is done using thermal simulations.

In essence, all simulators aim to solve Equation 2.1, however doing so at a molecular level is not possible, in particular when performing transient simulations. Determining temperature distributions is a computationally intensive task and as a result certain abstractions need to be made. A balance between desired accuracy and computation needs to be found. And, while different simulation methods are designed to exhibit different accuracies and computational overhead, it is not possible to rank them on these properties *per se* as they depend largely on the accuracy and complexity of their input data. In fact a great deal hinges on this data, and much of the difficulties encountered when performing thermal simulation stems from obtaining detailed traces of power consumption and area layout.

Finite-Element Analysis (FEA) [65] can potentially offer the highest accuracy in thermal simulation. Here, the input is a geometry CAD file of arbitrary complexity. The employed software then performs a numerical analysis to approximate the solutions of the differential equation. An example FEA is presented in Section 2.3, which shows both a detailed and abstract finite element analysis. The detailed simulation utilizes the layout and structure of the metal layers, where the abstract one combines multiple chip structures into an abstract rectangular component.

HotSpot [63] is a thermal simulation tool often employed for abstract simulations. Its approach for solving Equation 2.1 is to define an RC equivalent circuit. The capacitor (C) represents heat stored (i.e. the temperature) in a given volume and corresponds to the heat capacity. The resistors (R) represents the heat conduction from one volume to neighboring adjacent ones, i.e. $R \propto 1/k$. In practice, the height of a given chip layer is fixed. The volume is then again given as a rectangular component or *block*. Figure 2.2 shows an example RC circuit for two blocks consuming power. At this abstraction level, each block has a single averaged power value and a single temperature output, regardless of the actual distribution of power consumption within each block.

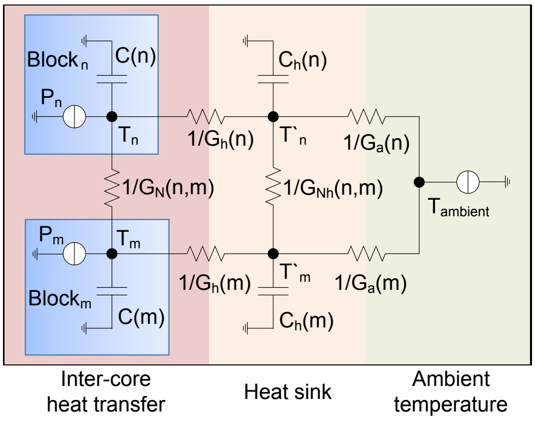


Figure 2.2.: RC equivalent circuit for coarse-grained thermal simulation

2.2. Thermal measurement

Performing DTM at runtime requires knowledge of the temperature distribution. Thermal measurements are a key requirement for determining the temperature development of a running system. The most common measurements are taken using on-chip sensors. Depending on the requirements different measurement techniques exist. The simplest way is to use the thermal sensors that are available in modern processors. These are typically implemented as thermal diode sensors [89] that have a varying voltage based on temperature. In these types of sensors, their voltage is compared to a reference voltage. This difference is then used to determine temperature. Typical thermal diodes have an accuracy of $\pm 1K$ [103]. However, they also have limitations. Their implementation consumes significant area and power by requiring an analog to digital converter to compare voltages. For instance, $\pm 1K$ accurate measurements in [103] correspond to 2 W power consumption. Diode placement is also fixed at design time. The sensor will not be able to measure the temperature of thermal hotspots outside of its proximity if there are high thermal gradients.

Another effect that can be exploited to construct thermal sensors is the dependency of propagation delay on temperature. For instance, in [123, 37] a 20°C increase of temperature results in 5-6% increase in the interconnect delay. Thus, by measuring the delay, temperature can be inferred. While these sensors are less accurate than thermal diodes, a major advantage is that they can be placed arbitrarily when measuring temperatures on an FPGA. An example implementation of these *softsensors* is presented in the next section. Further details on the temperature-dependence of delay and how it affects timing errors is given in Chapter 3.1.7.

For testing purposes, the most accurate and fine-grained measurements can be obtained using external infrared thermography [102]. However, this approach is limited to analysis and cannot be employed in production chips. It also requires altering the physical properties of the chip somewhat, e.g. by removing the packaging. An example of a thermal imaging setup is presented in Section 2.2.2.

2.2.1. Soft On-Chip Sensor

Since temperature changes delays, a way to measure temperature is to measure the speed of an asynchronous circuit. A common method is to use a *Ring Oscillator* (RO) and calibrate its output using a reference temperature measurement. This establishes a direct relation between temperature and oscillation frequency [4]. A ring oscillator consists of a feedback loop consisting of n inverters, where $n = 2k + 1, k \in \mathbb{N}$. An odd number of inverters are required for the RO output to toggle at each oscillation. An example RO is depicted in Figure 2.3. The frequency of a ring oscillator depends on number of delay elements n and the temperature-dependent propagation delay of a single inverter Δt_i. The approximate frequency can be given as $f_{\text{RO}} = \frac{1}{2 \cdot n \cdot \Delta t_i}$. To measure the frequency, the RO output is used to increment a counter. After a fixed period, i.e. determined by a fixed reference clock, the value of the counter represents the frequency and can be used to determine temperature.

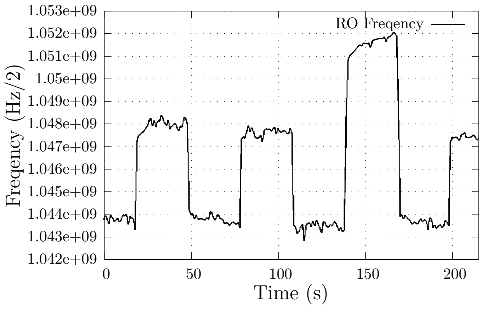


Figure 2.4.: Voltage dependent RO Frequency, 2s sampling interval

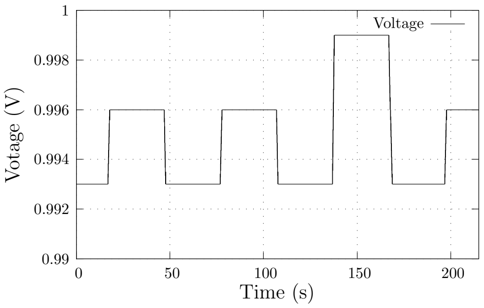


Figure 2.5.: Voltage changes in Figure 2.4

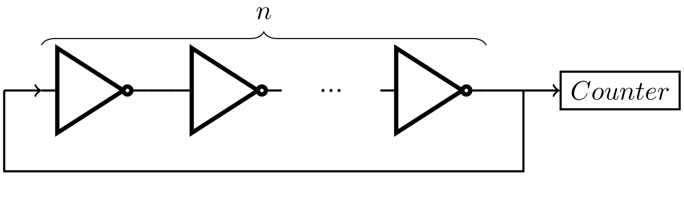


Figure 2.3.: Ring Oscillator consisting of an odd number n of inverters

For examining the soft sensors, we implemented various configurations of ring oscillators. Unfortunately, ring oscillator frequency is not only dependent on temperature. Changes in voltage will also influence the frequency. This can be observed in Figure 2.4 where

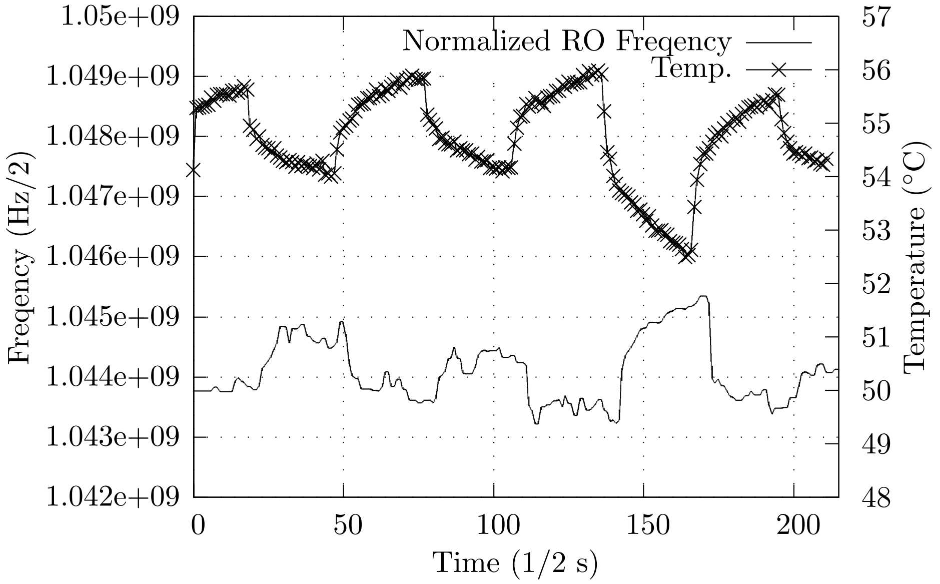


Figure 2.6.: RO frequency normalized to voltage and corresponding temperature

jumps in the ring oscillator frequency correlate to Voltage levels (Figure 2.5) obtained from the Xilinx System Monitor. In this example, the lowest and middle voltages correspond to a 200MHz and 100MHz toggle rate respectively in a design using slices only. The highest voltage corresponds to an idle system without any switching activity. The changes in voltage are a result of changes in the overall chip resistance. Temperature variance in this experiment is 2°C. In the evaluation, the maximum voltage difference of 6 mV results in a 0.7% change in RO frequency whereas the maximum change in temperature of 2°C results in a 0.09% change in frequency, meaning the effect of voltage on the RO is about one order of magnitude higher than that of temperature. According to [47], the effect of voltage on the ring oscillator delay is proportional to

$$\frac{V_{DD}}{(V_{DD} - V_{th})^{\alpha}} \tag{2.4}$$

where V_{DD} and V_{th} are the supply and threshold voltages, respectively, and $\alpha \in [0, 1]$. In order to compensate the effect of different voltages, we experimentally determine the factor by which Equation 2.4 changes the ring oscillator frequency. This is done separately for each individual voltage allowing all ring oscillator frequencies to be normalized to one voltage, and thus removing the dependency of the ring oscillator on voltage. An example of the graph in Figure 2.4 normalized to one voltage is shown in Figure 2.6, along with the temperature measured by the thermal camera.

When using a ring oscillator to estimate temperature, its parameters can be adjusted in order to achieve higher accuracy or precision, for instance by using different *numbers of delay elements*. Having more delay elements will increase the overall delay of one oscillation and thus increase the temperature sensitivity of the ring oscillator. The above evaluations have all employed ring oscillators comprised of 100 delay elements. Figure 2.9 shows the effects of reducing the number of delay elements to 24.

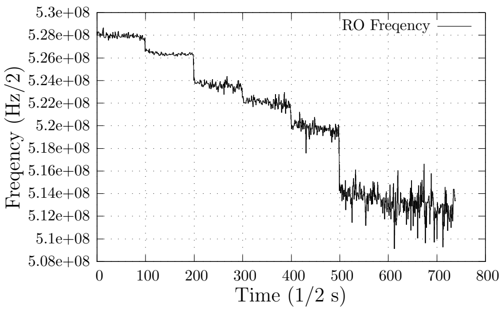


Figure 2.7.: RO Freqency with 1/2s sampling interval

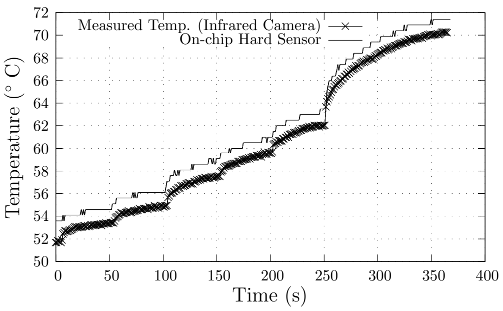


Figure 2.8.: Temperature measured by thermal camera and the on-chip sensor corresponding to Figure 2.7

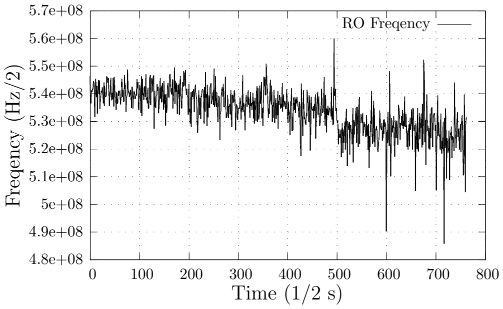


Figure 2.9.: RO Freqency with 1/2s sampling interval and 24 delay elements

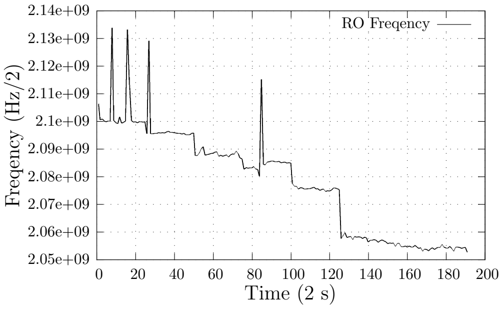


Figure 2.10.: RO Freqency with 2s sampling interval and 100 delay elements

The number of delay elements also influences the effectiveness of the second parameter, the *sampling interval.* The longer the sampling interval, the less susceptible the ring oscillator is to measurement noise. This is shown in Figure 2.10 in which a RO with 100 delay elements is used that is sampled every two seconds. The overall measurements are much smoother than using a 0.5s sampling interval (Figure 2.7), however a few peaks still remain. It should be noted that the magnitude of these peaks is comparable to those present using a 0.5s sampling interval, however their occurrence rate is considerably lower and they can be removed by applying a median filter.

2.2.1.1. Temperature Sensors of the Intel SCC

In the course of our temperature exploration, we also performed experiments on Intel's Single Chip Cloud computer [10]. This 48 core research platform is designed to provide insight into the benefits and limitations of future multi-core architectures. The SCC architecture is divided into 24 tiles, each encompassing two cores. Every tile holds two performance counters. As they adhere to the properties described in Section 2.2.1, these can, and have been employed, as temperature sensors [40, 71].

Unfortunately, the sensors are subject to a great deal of noise. This is illustrated in Figure 2.11, which presents the distributions of readings of three example sensors of the SCC. In this experiment, all cores are placed under the same computational stress in order

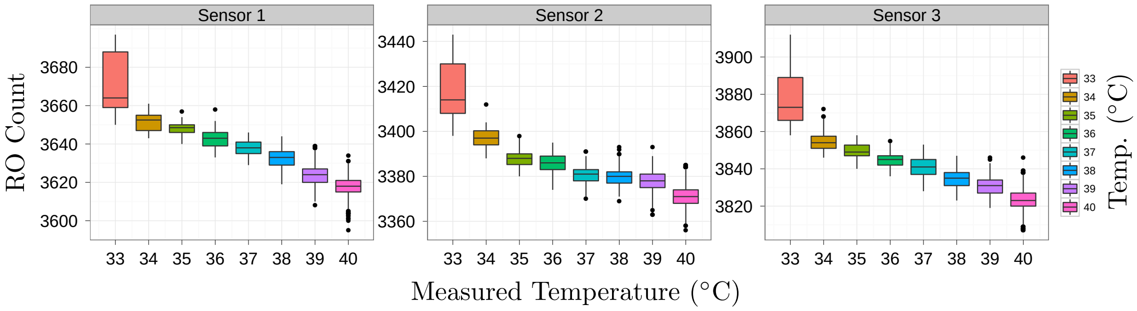


Figure 2.11.: RO sensor readings of different cores at a measured reference temperature

to create a temperature distribution that is as uniform as possible. All readings are taken in reference to the measured chip temperature. While the correlation between temperature and average sensor reading is evident, there is also a large overlap between readings at different temperatures. As a result, single readings cannot be used alone to determine the temperature.

The readings of each sensor are in different ranges and must be calibrated separately. A possible solution is to take the average of multiple readings. This creates additional overhead and it will take longer to determine temperatures. Since the sensors are also voltage dependent, this calibration must be done for all available voltages.

2.2.2. Infrared Thermal Cameras

Infrared cameras provide a means of obtaining a detailed map of the temperature distribution across an object. Before being able to analyze a given chip, it is necessary to remove everything obscuring the radiated heat. Unfortunately, this means removing any cooling infrastructure which may be in place (e.g. a heat sink or fan). The packaging itself is usually designed (i.e. assuming a metal packaging) to spread heat as uniformly as possible over the chip. Thus it needs to be circumvented as well. Generally, removing the packaging is no easy task, often requiring, for example, chemical etching of the protective housing to reach the actual chip – while taking care not to damage the wiring connecting the chip to a board (PCB). Fortunately, recent flip-chip packaging removes these wires and connects the metal layers of the chip directly to a PCB through a ball grid array. As a result, only the reverse side (i.e. the side of the silicon die without logic) is exposed to the packaging. This facilitates the removal of the packaging to expose the reverse side of the chip, which can then be analyzed through an infrared camera to obtain the temperature distribution of the chip along with its development over time.

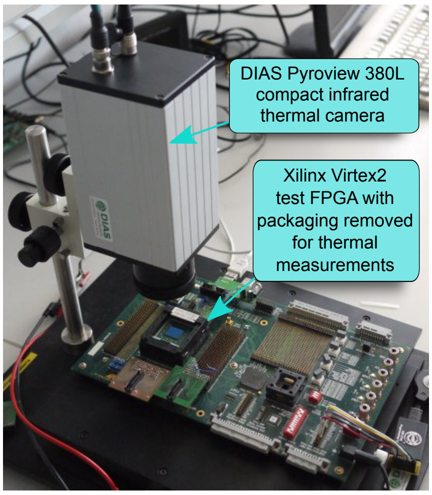

DIAS Pyroview 380L compact infrared thermal camera
Xilinx Virtex2 test FPGA with packaging removed for thermal measurements

Figure 2.12.: Experimentation measurement setup using thermal camera

In our lab we employ a DIAS Pyroview 380L infrared camera [3] which accurately (±1°C) measures temperatures of structure sizes down to 50 μm. The setup is shown in Figure 2.12.

The *emissivity* of the tested material is one of the key aspects to consider in any infrared measurement. It represents the percentage of heat that can be emitted in the infrared spectrum from the surface of a material compared to so-called ideal *black bodies* whose emissivity is defined as 1.0. For a silicon die, the emissivity is between 0.75 and 0.9, depending on how smooth the surface is. Before performing measurements, it is thus necessary to first determine the correct emissivity. This is done by applying a coating with a known emissivity in the desired spectrum to a part of the die and comparing the calibrating the temperature of the silicon to that of the coating, as seen in Figure 2.13. For our camera we used masking tape for this purpose, as its emissivity in the range of the camera (8–14 μm wavelengths) is known to be 0.9 [52]. Masking tape also has the desirable property of being easily removed, making it less obtrusive than, for instance, black paint. Once the emissivity is determined, the camera software can compensate accordingly. The second consideration is expressed in Figure 2.13. Often, polished metals are also reflective in the infrared spectrum. Care must be taken that no infrared emissions from outside heat sources are in the proximity of the examined chip.

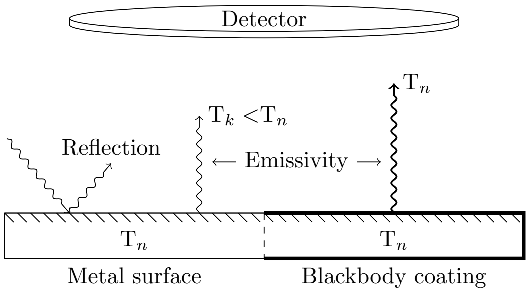


Figure 2.13.: Emissivity of metal surface vs. that of a surface with a (near) black body coating. T_k and T_n are compared to calibrate detector

One question that remains open is how realistic the temperatures measured from the back of the silicon die are compared to the temperatures at the metal layers and how much removing the packaging changes the temperature distribution. The finite element analysis presented in Section 2.3 examines this by presenting a simulation of an abstract chip, both with and without a metal heat spreader. As can be seen, the temperatures obtained from the back of the altered chip remain representative of the temperatures generated at the chip's metal layers.

2.3. Examining the Microarchitecture: SRAM Cell

All microarchitectural components together determine the power distribution of a chip. However the contribution of individual components needs to be analyzed in order to find a suitable abstraction level for thermal analysis.

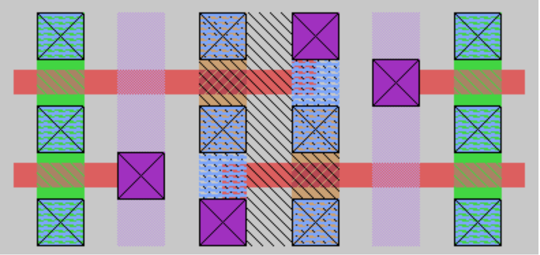


Figure 2.14.: Thin-film layout of 6-T SRAM cell modeled in magic [75].

Figure 2.14 shows the general layout of a thin SRAM cell consisting of two inverters and two NMOS access transistors. It is modeled using magic VLSI layout tool [75] using its design rule checking to verify layout constraints (e.g. keeping the minimum required distance between structures). To obtain equal rise and fall times in an inverter, the gate channel width of the PMOS transistors is typically larger to compensate for the decreased mobility of the majority

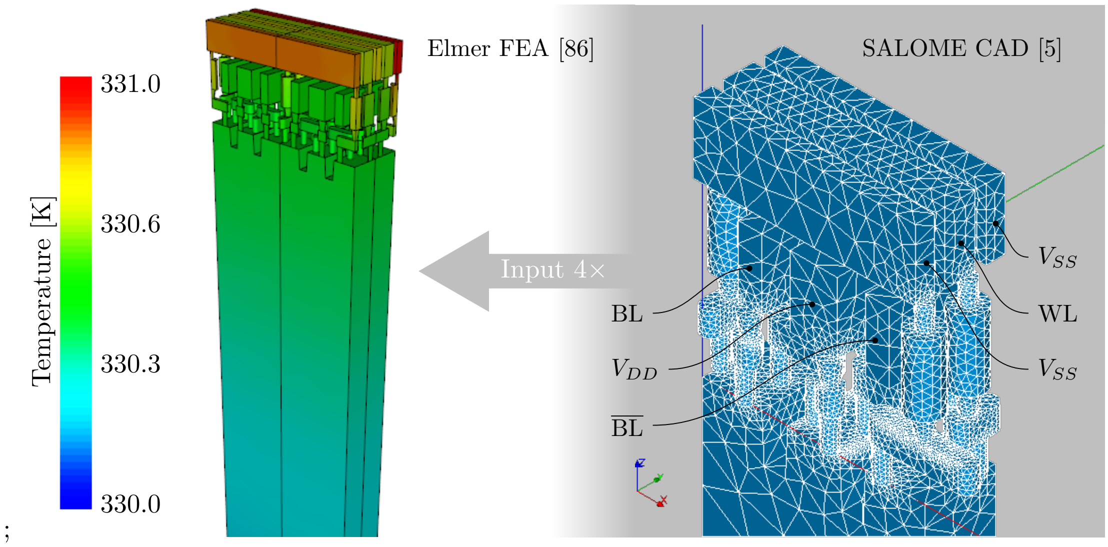


Figure 2.15.: Thermal finite element analysis of SRAM

carriers (i.e. holes) compared to NMOS transistors (i.e. electrons) [119]. Here, for simplicity, both PMOS and NMOS devices have the same dimensions.

To analyze the impact of a single microarchitectural component, a CAD geometry file was created based on the SRAM cell layout and sizings obtained from [2]. This served as the input for finite element analysis. Figure 2.15 shows the result for four adjacent cells. The supplied power consumption in the metal layers is 1 GW/kg, which amounts to approximately 2.25 W/mm^2. There is less than 1 K difference between the maximum and minimum temperatures, which includes the silicon layer. The highest temperature of the simulation is located in V_{SS} lines on the outside of the cells. This results from the smaller contact area between the V_{SS} lines and the silicon surface compared to the other metal lines. However, this is only the case on the outer edges of the combined SRAM cells, since the V_{SS} lines are shared between adjacent cells. Thus, adding more cells will result in more contacts to the silicon, and thereby create a more even temperature distribution, e.g. with the V_{SS} temperature being similar to the other metal lines. This can be seen with the V_{SS} line in the middle of the *metal 3* layer.

The overhead of thermal simulations of individual components makes this simulation approach unsuitable for larger systems, since the FEA would require considerable computation time. Instead, abstractions must be made. Figure 2.16 shows the FEA of the temperature of an entire chip where one square section is assumed to consist of SRAM cells with a uniform power distribution, equivalent to being filled with SRAM cells with the same power consumption as in Figure 2.15. Additionally, Figure 2.17 replaces the copper heat sink of Figure 2.16, leaving the silicon die exposed to air similar to the thermal camera setup presented in Section 2.2. Heat transfer in the air “block” is subject to both conduction and convection. Following key observations can be made:

Observation 1 Despite heat conductance being considerably higher in copper than silicon, only a small amount of heat is transfered horizontally in the chip’s metal layers due to their small height (2 μm). In particular, the vertical conduction into the silicon layer is significantly higher.

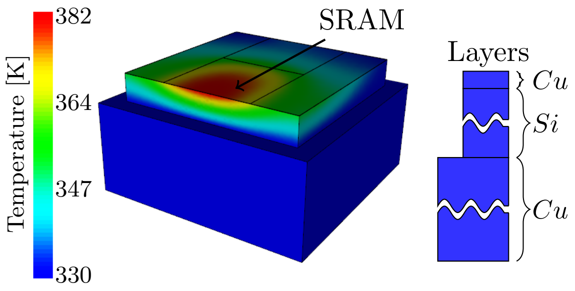


Figure 2.16.: Thermal Characteristics of Chip with one uniform SRAM component and copper heat sink

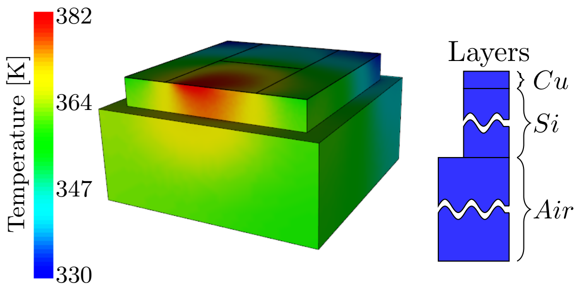


Figure 2.17.: Copper heat spreader is replaced by air

Observation 2 The copper heat spreader is very effective at distributing the temperature to the extent that it is nearly isothermal. This means that the surface of the silicon die to which it is attached is also mostly at the same temperature. In fact, the largest effect different components of the chip have on each other in terms of mutual heating is through the heat spreader, i.e. by raising its temperature. The temperature of the die surface depends on how effective the heat spreader (and attached heat sink) are at dissipating heat from the die. Here the boundary condition of 330 K on the bottom surface of the heat spreader is fixed.

Observation 3 The temperature on the silicon surface opposite the metal layers is representative of the temperature distribution on the metal layers when the heat spreader is removed (replaced by air). And, while there is more heat conducted vertically in the silicon block than in the metal layer, it is only enough to blur the temperature distribution seen on the metal layer, not distort it entirely. As a result temperature measurements taken with an infrared thermal camera observing the bottom side of the silicon die show a realistic picture of the temperature distribution.

2.4. Multi-Core Systems

Multi-core systems offer an additional dimension to reducing power densities. While DVFS focuses on reducing the power consumed over a fixed area, multiple cores can be used to spread a given activity over a larger area through migration.

FEA Summary:
Figures 2.16,2.17

Metal layer (Cu)	2 μm
Die layer (Si)	150 μm
Die area	1 mm × 1 mm

2.4.1. Related Work: Thermal Management in Multi-Core Systems

Traditionally, *dynamic thermal management* (DTM) is implemented using *dynamic voltage and frequency scaling* (DVFS) [58] in order to reduce the power consumption and thereby also reducing the generated heat. However, besides DVFS, multi-core architectures additionally have the potential for managing on-chip temperatures by distributing workloads throughout the chip by means of task migration [43]. Early approaches utilize a centralized reactive approach that migrates tasks to cooler cores once a predefined thermal threshold is reached on a core [97]. This approach has gradually given way to centralized proactive schemes [36, 35, 127] which use predictions based on thermal models [36, 127] and online learning [35] in order to migrate tasks before a predefined temperature threshold is hit. However, all centralized approaches have in common that they suffer from poor scalability [43], as finding an optimal task distribution is an NP-hard problem and thus grows considerably in complexity with the number of cores. This will especially be a problem considering future multi-core architectures with hundreds and even thousands of cores [26]. Additionally, relying on a central instance also introduces a central point of failure within the system.

To deal with this, decentralized semi-proactive approaches such as [43] have emerged for deployment in future multi-core architectures. There, so-called *power units* are propagated across the chip by trading with adjacent cores based on what is identified as local supply (available power units) and demand (computational requirements). The approach is semi-proactive since while the temperature is distributed across the chip proactively, actual power trading is reactionary as it is done depending on the current temperature as opposed to future predictions.

2.4.2. 3D Architectures

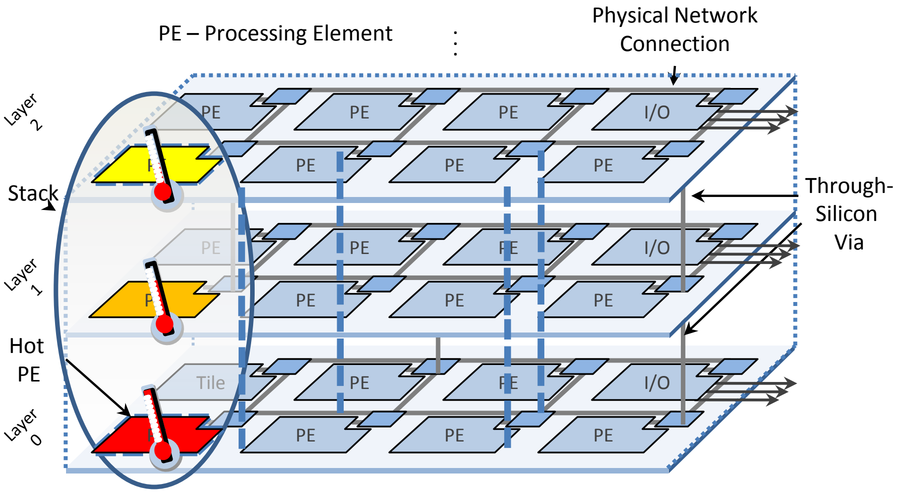
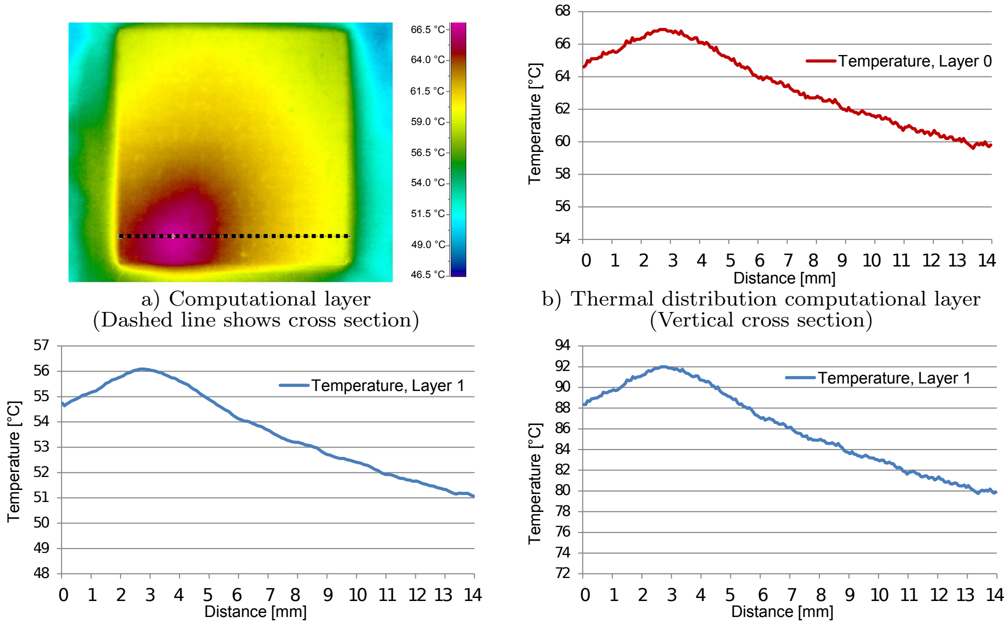


Figure 2.18.: Example of a layered 3D multi-core architecture

While stacked 3D architectures have been shown to have increased performance compared to their 2D counterparts [24] – especially with regard to the throughput and latencies of the communication architecture [50, 72] – they are much more prone to excessive heating as the heat needs to dissipate through multiple layers before reaching the heat sink [24, 99].

Activity migration is particularly effective in 3D multi-core architectures where a task running on a particular core has a significant effect on surrounding cores, especially in the vertical direction. To motivate this, we briefly examine a simplified analysis of temperatures in 3D architectures:



a) Computational layer
(Dashed line shows cross section)

b) Thermal distribution computational layer
(Vertical cross section)

c) Layer 0: computation; Layer 1: Idle;
Temperature Layer 1

d) Layer 0: computation; Layer 1:
computation; Temperature Layer 1

Figure 2.19.: Measurement of temperatures in one layer for 3D thermal analysis

Here, to analyze the thermal behavior of a homogeneous 3D layered architecture, the thermal distribution of each layer is measured individually using a thermal infrared camera. An example of such a measurement is shown in Figure 2.19 a), where an area of homogeneous computation is located in the bottom left, while the rest of the chip is idle with the resulting measured temperature values shown in Figure 2.19 b). Assuming this measurement represents the lower layer, *layer 0*, of a 3D architecture consisting of two layers, the heat conduction to an idle upper layer, *layer 1*, can be calculated using the thermal conduction/capacitance of the layers with respect to one another, resulting in the temperature distribution of Figure 2.19 c) in *layer 1*. Here the thermal conductivity of Silicon is assumed to be 100 W/mK, and that of copper to be 400 W/mK. The metal layer is assumed to take up 1% of the die height. If assuming both layers utilize identical computation temperature peaks will increase significantly as shown in Figure 2.19 d).

Device Summary:
Figure 2.19

Die area	1.4 cm × 1.4 cm
Layer height	500 μm
Metal layer (% of total height)	1%
Conductivity:	
Si	100 W/mK
Cu	400 W/mK

Due to the relatively large die height of 500 μm, this example exhibits more horizontal heat conduction than more compact layers for 3D stacking. Also, the layers are assumed to be directly on top of each other. The characteristics of temperature distribution can be improved significantly through the use of TSVs (s. Section) and interface materials. While these are not considered in this motivation, they are for the 3D evaluation in Chapter 5. However, the main observation of localized elevated temperature when computation is layered on top of other computation, remains valid.

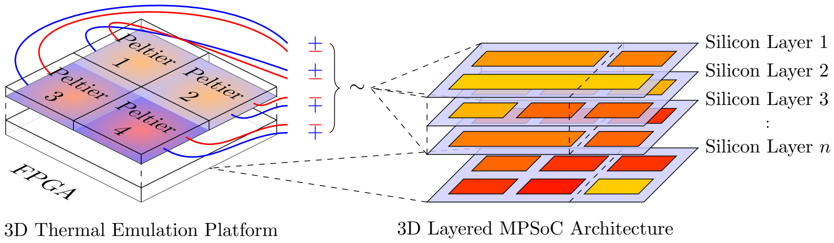


Figure 2.20.: Emulation of 3D temperature distribution on 2D FPGA

Emulating the thermal conditions of a 3D architecture on two-dimensional chips enables the analysis of 3D thermal management using conventional hardware. Figure 2.20 presents an approach where simulated 3D layers can be used to influence the temperature of a single additional layer running on an FPGA. The output of the simulation is used to control heating elements (e.g. Peltier devices [39]) corresponding to the total heat generated by multiple upper layers. Of course, such approaches are limited in granularity and cannot encompass the full range of temperature distribution that can be present across one layer.

2.4.2.1. Related Work: Thermal Management in 3D Architectures

Since elevated temperatures are a key concern in 3D architectures, a number of techniques have appeared specifically targeting 3D thermal management [104]. First are static design time approaches which increase heat dissipation by altering the physical characteristics of the chip. This can be accomplished by different means including thermal aware floorplaning [32] and advanced techniques such as thermal TSVs [33] and microfluids [38, 105]. While these approaches are able to lower temperatures, they may not be a sufficient replacement for dynamic thermal management but simply delay its invocation and are unable to deliver optimal results when unpredictable scenarios, such as bursts in computational load due to user interaction or varying input data, occur.

Dynamic approaches on the other hand rely on reducing the power consumption at thermal hot spots through DVFS and power gating. Additionally, multi-core architectures have the opportunity to divert the power consumption away from thermal hot spots through runtime task migration. This technique is particularly effective in 3D multi-core architectures where a task running on a particular core has a significant effect on surrounding cores, especially in the vertical direction.

State-of-the-art dynamic thermal management approaches also differ in when they are invoked. Simple reactive threshold based approaches react when a preset temperature has been reached. As such they fail to consider the costs of thermal management such as the task migration overhead. Proactive approaches, on the other hand, act before a threshold is reached, thereby avoiding thermal hot spots all together. These approaches are typically based on stochastic models or on-line learning.

Ayse K. Coskun et al [34] propose a reactive dynamic thermal management scheme for 3D multicore architectures. They identify task remapping as a key solution to thermal problems in 3D environments because of the severe performance hits that measurements like straightforward DVFS or clock gating (stalling) of hot cores come with. They consider a homogeneous stack of UltraSPARC cores and do not state how task migration is triggered, i.e. if from a central instance or how such a controlling instance would be organized. A number of migration strategies are proposed and evaluated: The simplest strategy swaps the task from the highest-temperature core with the one on the coolest core, while the so-called “adaptive-random” policy takes thermal histories of each core into account to make more effective remapping decisions. The “Adapt3D” policy is designed to address the characteristics of 3D systems, most importantly, the correlation of the temperature of vertically stacked chips.

Therefore, stack (cylinder)[3] temperatures are considered for remapping decisions over individual core temperatures. This work does not consider scalability concerns for large systems because no distributed controlling scheme is introduced. Additionally, communication volume and –latency is not taken into account and task remapping cost is not modeled despite the fact that each task remapping is assumed to constantly require 1ms. Zhou et al [128] presented a reactive OS-level task scheduling approach to thermal management for 3D systems. They augmented a Linux 2.6 kernel to schedule tasks so to avoid hardware DTM measurement invocations such as DVFS and do not require any hardware changes. Thermal correlation in vertically overlapping cores is considered and their work focuses around tasks as causes for thermal issues. Each task is considered to cause thermal stress to the cores, and thus, cores that cause a large amount of thermal stress are considered hot tasks while computationally inexpensive tasks that cause less thermal stress are considered to be cold tasks. Consequently, hot tasks are assigned less time slices than cold tasks by the scheduling algorithm. They also shorten time slice length from a proposed 10-200ms interval to a 8ms interval, which comes with increased scheduling overhead (context switches) but allows for finer-grain decisions to address thermal problems. Both the target core and the executed task are chosen at each scheduling interval. They consider only small (2x2x2) chip arrays and make no assumptions on interconnection architecture or task migration requirements.

To overcome the shortcomings of previous approaches, Changyun Zhu et al [129] consider dynamic thermal management in 3D architectures not as an emergency measurement whose invocation should be avoided as much as possible, but rather find that because large 3D systems are likely to operate continuously close to the thermal threshold, thermal management needs to be invoked continuously on a regular basis. Therefore, thermal management techniques that come with a severe performance overhead are considered impractical. Specifically, they stress the importance on global decision making because they find local considerations insufficient (which is not proven in experiments). As a consequence, their work proposes a proactive global thermal budgeting algorithm that optimizes for chip performance under a constraint of peak temperature. Additionally, they regard the operation of different cores in a heterogeneous environment at different voltage and frequency levels being key to effective dynamic thermal management because of the different thermal characteristics of different cores.

They guide thermal management through the calculation of the thermal-impact-per-performance-gain (TIP), which allows optimizing the performance for a given thermal budget. Another guideline for DTM favors the mapping of demanding tasks to cores with a higher thermal efficiency. These guidelines have been implemented in ThermOS, a solution con-

[3] a stack or cylinder is a connected subset of cores that vertically overlap each other, ranging from the topmost to the lowermost layer

sisting of hardware-based temperature and workload monitoring and an augmented Linux 2.6 kernel that maps and schedules tasks following guidelines derived from these considerations. Each core does run-time DVFS triggered by a thermal budget calculation, resulting in a proactive thermal management. They claim that DVFS comes with little hardware overhead as most modern CPUs are already equipped with DVFS units. Only additional power supply wiring is required. Last-resort emergency clock throttling is done for cores exceeding a thermal threshold value. Considering an application of the outlined approach to a large system consisting of hundreds of cores, a major drawback of this work is the lack of scalability considerations for large systems. Additionally, the availability of fine-grain DVFS in each core is unlikely to hold in large systems for cost and complexity reasons.

2.5. Summary

Although grounded in the basic laws of thermodynamics, the study of temperature and heat distribution remains multifaceted. The level of thermal details relies on the component of interest in the system. Examining the microarchitecture requires a fine-grained analysis which is able to capture all of the intricacies of small features. These can be more of a hindrance than an asset in larger views, i.e. of entire multi-core architectures where it is more beneficial to use less detailed thermal distributions, e.g. assuming one temperature value per core. Both in simulation and measurement, proper abstractions need to found between the desired accuracy and complexity.

The basis for all dynamic thermal management techniques lies in controlling the power consumption per area. This can be done in two ways: first by reducing the power and second by increasing the area where power is consumed. Power can be reduced by changing the voltage and frequency. The area can be increased by periodically migrating switching activity form one area to another, for instance using task migration when performing DTM at core-level.

DTM techniques can be classified into two categories. The simpler ones, so-called reactive techniques, rely on a predefined temperature threshold and perform thermal management only once the threshold has been hit. In contrast, proactive techniques predict future temperatures based on current and past measurements using thermal models or statistical methods. DTM is then done based on these predictions. Proactive management is able to reduce temperatures more than their reactive counterparts, but also require more calculations and may perform DTM more frequently than reactive techniques.

3. On Temperature-related Dependability

As previously mentioned, dependable operation depends on two factors. The first of these is reliability. For a system to be deemed reliable, it must produce the correct output for a given input. In a general sense, system noise, e.g. as voltage or timing fluctuations, threatens reliable operation. Additionally, individual failures, e.g. resulting in stuck-at faults, prevent reliability. Failures are also responsible for the second factor of dependability: the availability. A system is no longer available, once it does not produce output for any input. Any reliability degradation may eventually lead to diminished availability. In the following, different temperature-dependent mechanisms are described that threaten reliability by increasing susceptibility to noise or by inducing failures, as presented in [56].

3.1. Reliability Concerns from a Thermal Perspective

In the broad sense, temperature has two ways to impact the dependability of a system: timing errors and aging effects. Timing errors are often transient due to spatial thermal gradients and thus do not directly impact system lifetime. However, some aging effects such as NBTI can increase the system's susceptibility to timing errors. Aging effects in general degrade system performance and ultimately lead to system failure. All aging effects rely on physical processes which degrade a chip over time. As such they have an Arrhenius relation to temperature [120] and their temperature dependence can be expressed as:

$$Ae^{-E_a/kT} \tag{3.1}$$

where A is the frequency factor (i.e. the number of collisions per second), E_a is the activation energy of the particular process, and k the Bolzmann constant. The main differentiating factor between the temperature-dependence of each process is the activation energy. For aging effects, it is often possible to determine a Mean Time To Failure (MTTF) based on models parameterized with experimental data. When dealing with dependability, it is not enough to view the mean effect. Additionally, the distribution of the effect must also be taken into account. In the following, some of the most prominent reliability concerns are outlined individually. This includes the most dominant aging effects as well as timing errors.

3.1.1. Electromigration

Through the movement of ions in the metal layers of CMOS circuits, metal interconnects are gradually eroded over time. This effect – referred to as electromigration – ultimately leads to a failure of the interconnect. Regarding temperature, both peak temperatures and thermal gradients influence the degree of electromigration.

Electromigration Model

Electromigration is commonly modeled using Black's equation [93, 25]:

$$\text{MTTF} = AJ^{-n}e^{q/(kT)} \tag{3.2}$$

where A is a constant, J the current density; n is the technology parameter, q the activation energy, and k is the Boltzmann constant. The MTTF due to electromigration is further affected by spatial thermal gradients [93]. Along with Equation 3.2, it is modified to the following:

$$\text{MTTF} = AJ^{-n}e^{q/(k(T+\Delta T_{joule}))} \tag{3.3}$$

where T_{joule} represents the joule heating, i.e. the temperature obtained through local power consumption and not through conduction. While Equation 3.3 gives the mean time to failure, it does not provide us with a variance of electromigration. According to [110], the variance of electromigration is also temperature dependent and is given as:

$$\sigma^2 = 2.3\frac{1}{k^2}(\frac{1}{\theta} - \frac{1}{T})^2\sigma_q^2$$

where σ_q^2 is the variance of the activation energy q and θ the *iso-kinetic temperature* with $\theta \approx 300$ K according to [48].

3.1.2. Negative Bias Temperature Instability

NBTI is caused by traps near the interface layer between the silicon die and the gate dielectric of PMOS transistors [108]. While NBTI is not yet fully understood, it is believed that two types of traps contribute to the overall effect. On the one hand, pre-existing filled traps are found in the dielectric which are emptied under voltage stress, and on the other, interface traps are generated by the break-down of Si-H bonds located at the Si-SiO_2 interface. NBTI is becoming more noticeable as transistor sizes decrease due to the overall decrease in the number of molecules which increases the significance of individual traps effected by NBTI.

NBTI manifests itself in a circuit through an increase in gate threshold voltages, and thus induces a delay into circuit running at a fixed voltage. Since different transistors may be affected differently by NBTI, timing errors can occur and result in transient faults. These can be overcome by either increasing the voltage – which will accelerate aging effects – or by running the system at lower frequencies – which may violate performance constraints.

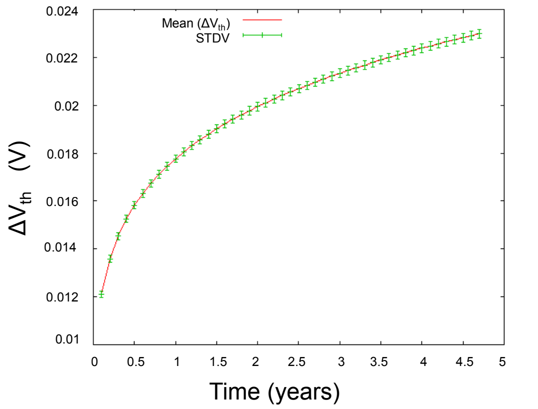


Figure 3.1.: NBTI induced shift in threshold voltage over time in 65nm latch

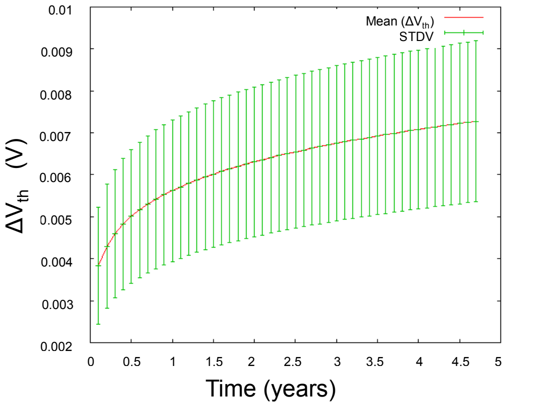


Figure 3.2.: NBTI induced shift in threshold voltage over time in 32nm latch

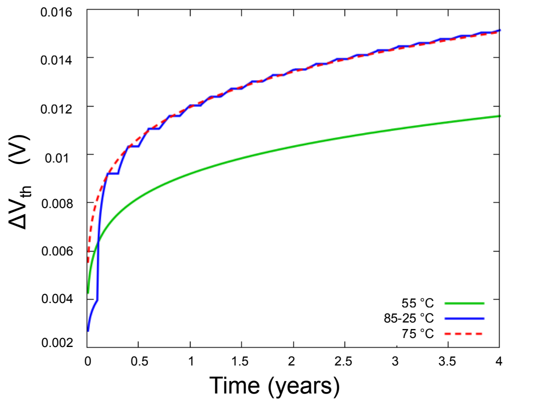


Figure 3.3.: NBTI induced shift in threshold voltage for different temperatures

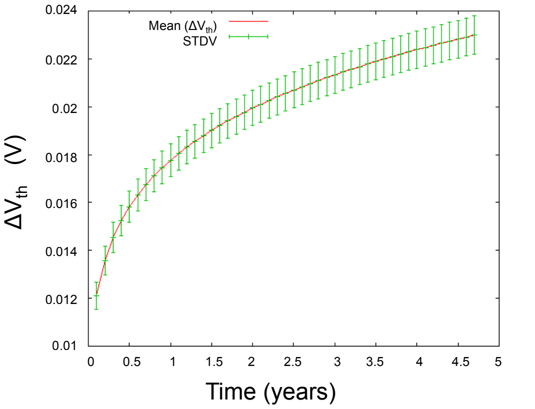


Figure 3.4.: NBTI induced shift in threshold voltage over time in 65nm SRAM cell

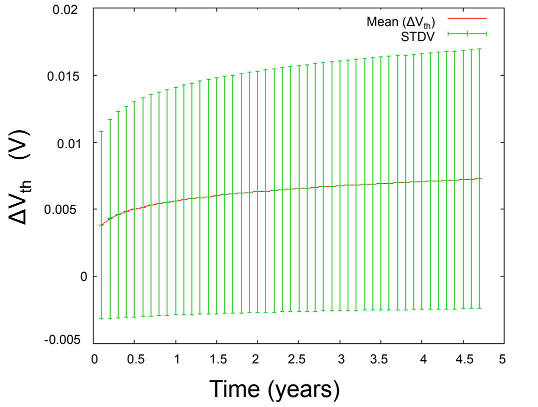


Figure 3.5.: NBTI induced shift in threshold voltage over time in 32nm SRAM cell

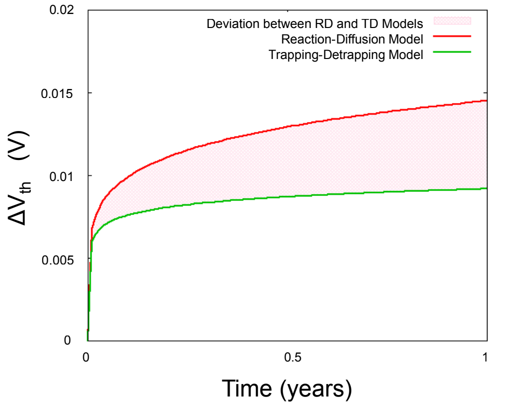


Figure 3.6.: Discrepancy between the *reaction-diffusion* and *trapping-detrapping* NBTI models over large time spans

NBTI Models Our NBTI models which we have obtained together with our industrial partner are based on measured data which is extrapolated using different model assumptions. The most commonly used NBTI model is based on a *reaction-diffusion* model [13] with exponential time dependency. Using sample data from 65nm to 22nm, we arrive at the following equation:

$$< \Delta V_{th} >= 0.05 \cdot e^{-1500/T} \cdot V_{dd}^4 \cdot t^{1/6} \cdot d^{1/6} \tag{3.4}$$

where $< \Delta V_{th} >$ is the expected mean V_{th}, V_{dd} is the supply voltage, T is the temperature in K, and d is the duty cycle.

According to [118] the *reaction-diffusion* model given by Equation 3.4 becomes inaccurate over long time intervals. Instead, they claim that the *trapping-detrapping* model, with its logarithmic time dependency, more closely represents the shift in V_{th}. By fitting the measured data to this model, we obtain:

$$< \Delta V_{th} >= 0.05 \cdot e^{-1500/T} \cdot V_{dd}^4 \cdot d^{1/6} \cdot A \cdot \log(1 + Ct) \tag{3.5}$$

where $A = 0.856$ and $C = 0.0163$. As can be seen in Figure 3.6, the *trapping-detrapping* model (Equation 3.5) results in a 45% lower shift in V_{th} than the *reaction-diffusion* model after one year.

From the data, it can be observed that the variance of the shift in threshold voltage increases with both the value of ΔV_{th} and with the decrease in transistor size. The resulting standard deviation is as follows:

$$\sigma = \sqrt{\frac{2.52 \cdot 10^{-19} \cdot \Delta Vth}{W \cdot L}}$$

for transistors of length L and width W. How this affects circuits can be seen in Figures 3.1-3.2 and 3.4-3.5, showing the shift in V_{th} and its standard deviation in a latch (representing computational logic) and in an SRAM cell (representing memory). It can be seen that the variance in the SRAM cell is considerably larger compared to the latch at the same technology node. This is due to the fact that the SRAM transistors are smaller than those of latches.

3.1.2.1. Understanding Traps

As briefly mentioned previously, two types of traps are responsible for NBTI.

Interface traps are generated between the silicon and the gate dielectric, i.e. at the $Si - SiO_2$ interface. During manufacturing, hydrogen atoms are introduced at this layer in order to stabilize the surface [55]. This process covalently bonds Si surface atoms to h, leaving it chemically inert. This is known as hydrogen passivation. When under negative biased voltage stress $V_{gs} < 0$, the vertical electric field beneath the transistor gate paired with thermal vibrations can cause these $Si - H$ bonds to break. Resulting are dangling bonds Si^x which are electrically charged and slightly counteract the electric field.

Freed H^+ atoms have two different possibilities. The first is *diffusion* into the oxide layer. This can occur either in its atomic form (h diffusion) or on a molecular level

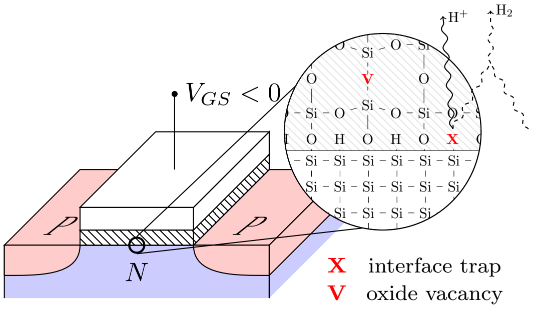


Figure 3.7.: NBTI affects PMOS transistors at the $Si - SiO_2$ interface as well as in the gate oxide

when two hydrogen atoms combine (H_2 diffusion). The difference between these two manifests in the time dependency resulting from different diffusion rates. The $t^{1/6}$ commonly used in reaction-diffusion models, e.g. as in Equation 3.4, corresponds to H_2 diffusion only [85], ignoring the slower $hDiffusion$ ($\propto t^{1/4}$). This overestimates total diffusion and, as a result, also NBTI. As a second possibility, the H^+ may re-bond with a Si^x and repassivate the surface. This *reaction* is responsible for the long-term recovery effect of NBTI.

An example of an interface trap is shown in Figure 3.7. The number of possible interface traps N_{IT} is limited by the total number of $Si - H$ bonds.

Pre-existing Oxide traps , also known as hole traps, are the result of vacancies in the oxide layer. Unlike in a crystalline structure, the amorphous oxide layer exhibits large variations in local bonding [82]. In PMOS transistors, holes are the majority carrier which flow when the transistor is conducting. When these holes enter the gate dielectric, again due to the vertical electric field, they may encounter oxide traps. Once holes have filled the traps, they result in a weakening of the electric field. As the name suggests, these traps are pre-existing, that is they are a result of the manufacturing process. Trapping holes when under voltage stress and the subsequent recovery when the stress is removed, are both significantly faster than the generation/recovery of interface traps [85]. The number of trapped holes is represented as N_{HT}.

Under accelerated aging conditions, i.e. when voltage stress is increased, additional oxide traps will form over time. This effect has been shown to be negligible under nominal V_{DD} [85].

Since both kinds of traps reduce the electric field, they both manifest themselves as an increase in the threshold voltage V_{th}. The change in threshold voltage ΔV_{th} is directly proportional to the total number of both kinds of traps [69].

3.1.2.2. Modeling Short-term NBTI Degradation

The NBTI model given in Equation 3.4 is effective when determining the expected effects of NBTI over long time scales, e.g. days, weeks, out even years. However, it does not explicitly model the recovery effect but instead includes it implicitly as an average change to the monotonously increasing aging effect. When examining smaller time scales, this is no longer accurate.

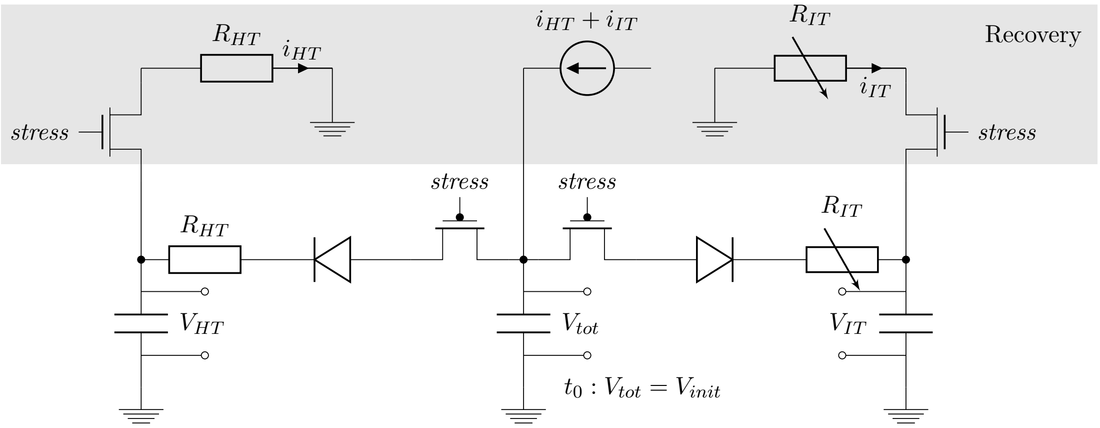


Figure 3.8.: RC model for ΔV_{th} including short-term recovery and custom diffusion modeling through variable resistors

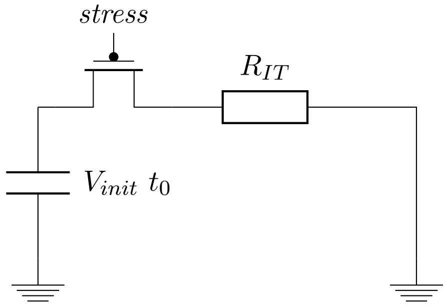


Figure 3.9.: Simple modeling of ΔV_{th} as discharging capacitor

One way to model the reaction-diffusion process of NBTI is by creating an RC circuit that mimics its behavior. An initial example of such a circuit is given in Figure 3.9. Here, the formation of traps is modeled by the discharging of a capacitor. In essence, the voltage of the capacitor represents the number of potential traps that can be formed. That is, for a starting voltage V_0, the number of traps is proportional to $V_0 - V(t)$. As it corresponds to the number of traps [69], $V_0 - V(t)$ is also proportional to the shift in threshold voltage ΔV_{th}. For simplicity, the starting voltage V_0 is normalized to one.

Whether the circuit is in stress phase is determined by the PMOS transistor which has the stress signal of the corresponding transistor being modeled as their input. Resistors are used to regulate the diffusion rate of the capacitor charge and should therefore correspond to the diffusion rate of hydrogen from the gate interface.

This model has two shortcomings: first, just like the model given by Equation 3.4 it does not address the recovery effect and also only models interface trap generation. Second, the time dependency of discharging a capacitor is given as:

$$V(t) = V_0 e^{-\frac{1}{RC}t} \tag{3.6}$$

which of course does not result in the same $\propto t^{1/6}$ time dependency as Equation 3.4. While this can still be used as a piecewise approximation of interface trap generation, properly modeling the time dependency will require changing the timing properties of the resistor.

To do this, it is necessary to revisit the underlying physical models and determine where the time dependency comes from. As stated, the change in threshold voltage is proportional to the number of traps formed, $\Delta V_{th} \propto (N_{IT} + N_{HT}) \propto (V_0 - V_{tot}(t))$. [69] gives the time dependency for N_{IT}. Apart from the diffusion rate of hydrogen, the rate of interface trap generation is also dependent on the total number of $Si-H$ bonds and the current number of traps. That is, V_0 and $V_0 - V(t)$, respectively.

It has been noted, however, that considering only interface traps is not sufficient when exploring very short time scales [85]. Fortunately, the time dependency in Equation 3.6 correspond to that of hole trapping in preexisting oxide traps [69]. Modeling these remains discharging a capacitor over a resistor.

The modified full NBTI modeling circuit is given in Figure 3.8. Diodes are added in order to limit the direction of current, and to avoid that, e.g., the two trap capacitors C_{HT} and C_{IT} influence each other. Since all currents leaving the circuit are reintroduced, the total amount of charge in the circuit always stays the same. In this circuit, the number of total traps, i.e. N_{IT} and N_{HT} are lumped together to form one voltage V_{tot}. The percentage of each type of trap is determined by the size of the capacitor chosen for each one.

A further issue that needs to be addressed is the modeling of the temperature-dependence of NBTI, which the circuit does not consider by default. Fortunately, SPICE provides first and second order temperature parameters to determine the temperature-dependency of resistor resistance. These two terms need to be configured so that the resistor approximates the Arrhenius relation from Equation 3.1. For all practical purposes this approximation is sufficient, but will exhibit considerable error in extreme cases, e.g. near absolute zero.

When modeled in SPICE, the NBTI RC circuit can be directly simulated along with the transistors being examined. An implementation of the model is given in Appendix A.

3.1.3. Positive Bias Temperature Instability

Analog to NBTI in PMOS transistors, Positive Bias Temperature Instability (PBTI), affects NMOS devices, due to their positive gate-to-source voltage bias. Instead of hole trapping, the main source of PBTI is electron trapping [125], i.e. corresponding to the majority carrier in NMOS. Typically, the shift in V_{th} PBTI is an order of magnitude lower than NBTI [19] in conventional devices withe a SiO_2 gate dielectric. With the introduction of High-k dielectrics, PBTI becomes more prominent [69]. This mainly due to the increased electron trapping in the high-κ insulator.

3.1.4. Hot Carrier Injection

Similar to NBTI, Hot Carrier Injection (HCI) based degradation is caused by traps at the interface to the gate dielectric. Here, carriers are accelerated in the MOSFET channel by the horizontal electric field between source and drain. These carriers are referred to as *hot* if they have sufficient energy to cause impact ionization, forming an electron hole pair in the silicon layer. Through coulomb scattering, these can be injected into the oxide where they break $Si-h$ bonds. While this is the main contributing factor additional mechanisms also contribute to HCI degradation. These are individual electrons that escape directly into the oxide, minority carriers from secondary impact ionization, and due to direct tunneling effects [120]

A main characterizing difference compared to NBTI lies in which electric field is responsible for the effect [84]. Since transistor channel width is decreasing with technology scaling, the strength of the horizontal electric field is increasing. Combined with the adoption of high-κ metal gates reducing the vertical fields, the impact of HCI will continue to grow in importance compared to NBTI.

3.1.5. Time Dependent Dielectric Breakdown

Unlike the previous aging mechanisms, Time Dependent Dielectric Breakdown (TDDB) has only a minor degradation impact on device characteristics (i.e. expressed through a shift in V_{th}). Instead, its outcome is complete device failure. The process behind TDDB takes place in two phases [66]. TDDB first occurs through the gradual deterioration of the gate dielectric through the filling of pre-existing oxide traps along with the creation of new oxide traps[100]. As such, the mechanisms behind TDDB correspond to those behind NBTI. The amassing trap densities in the oxide can then lead to a percolation effect, i.e. the clustering behavior of random processes, resulting in the so-called runaway phase [66]. This ultimately leads to a conducting path formed through the gate oxide. The time to breakdown t_{BD} can be modeled as [109]:

$$t_{BD} = \tau_0(T)e^{\frac{G(T)}{\kappa}} \tag{3.7}$$

where $\tau_0(T)$ is the Arrhenius dependency of TDDB, and $G(T) \propto 1/kT$. The time to breakdown is directly related to the strength of the electric field between gate and bulk as can be seen by its dependency on the oxide permittivity κ. The use of high-κ gate dielectrics decrease the effect of TDDB accordingly.

3.1.6. Thermal Cycling

Periodic phases of heating and cooling – often referred to as thermal cycling – can add additional thermal stress on system materials even at low temperatures as only the temperature range is relevant and not the absolute temperature. Unfortunately, thermal management techniques themselves often induce thermal cycling as a means of reducing peak temperatures.

Thermal cycling is commonly modeled using the Coffin-Manson equation [114]:

$$N = C(\frac{1}{\Delta T})^q \tag{3.8}$$

where N is the expected number of cycles until a system failure occurs, ΔT is the change in temperature and C is a material constant. $q \in [1, 3]$ represents the Coffin-Mason exponent which is determined through experiments. In particular, the lifetime degradation to thermal cycling is not dependent on the cycling rate, but instead only on the difference between the maximum and minimum temperature and the number of total cycles.

3.1.7. Timing errors

Varying temperatures across a chip will also result in varying timing properties. If the temperatures vary considerably, timing errors can occur. This is especially the case for components that span over large areas of the chip. For instance, the clock tree has been proven to be particularly vulnerable to spatial thermal gradients which induce uneven interconnect delay. The resulting clock skew problems induce timing errors[31].

To analyze the general effect of temperature on the timing of circuits, we need to examine its effect on individual transistor delay. Unfortunately, the effect of temperature on delay is not a direct linear relationship. The delay is however directly related to the drain current

I_D – the higher the current, the faster it can charge subsequent capacitances driven by transistor output, and thereby reduce circuit delay. As a result, any change in delay is not only dependent on temperature but also in the nature of the circuit. An empirical approximation of drain current can be given by the following alpha power law [106].

$$I_D = \frac{W}{2L}\mu(T)C_{ox}(V_{GS} - V_{th}(T))^{\alpha} \tag{3.9}$$

where C_{ox} is the gate capacitance, W and L are the gate dimensions. Typical values of alpha vary in the interval $[1, 2]$ [106] and differ between NMOS and PMOS. As can be seen there are two factors which are dependent on temperature: first the carrier mobility μ and the threshold voltage V_{th}.

BSIM [1] presents us with models describing MOSFET behavior. In the current BSIM6.1, the temperature-dependence of μ and V_{th} is given as follows[1]:

$$\mu(T) = \mu_0(\frac{T}{T_0})^{\beta_\mu} \tag{3.10}$$

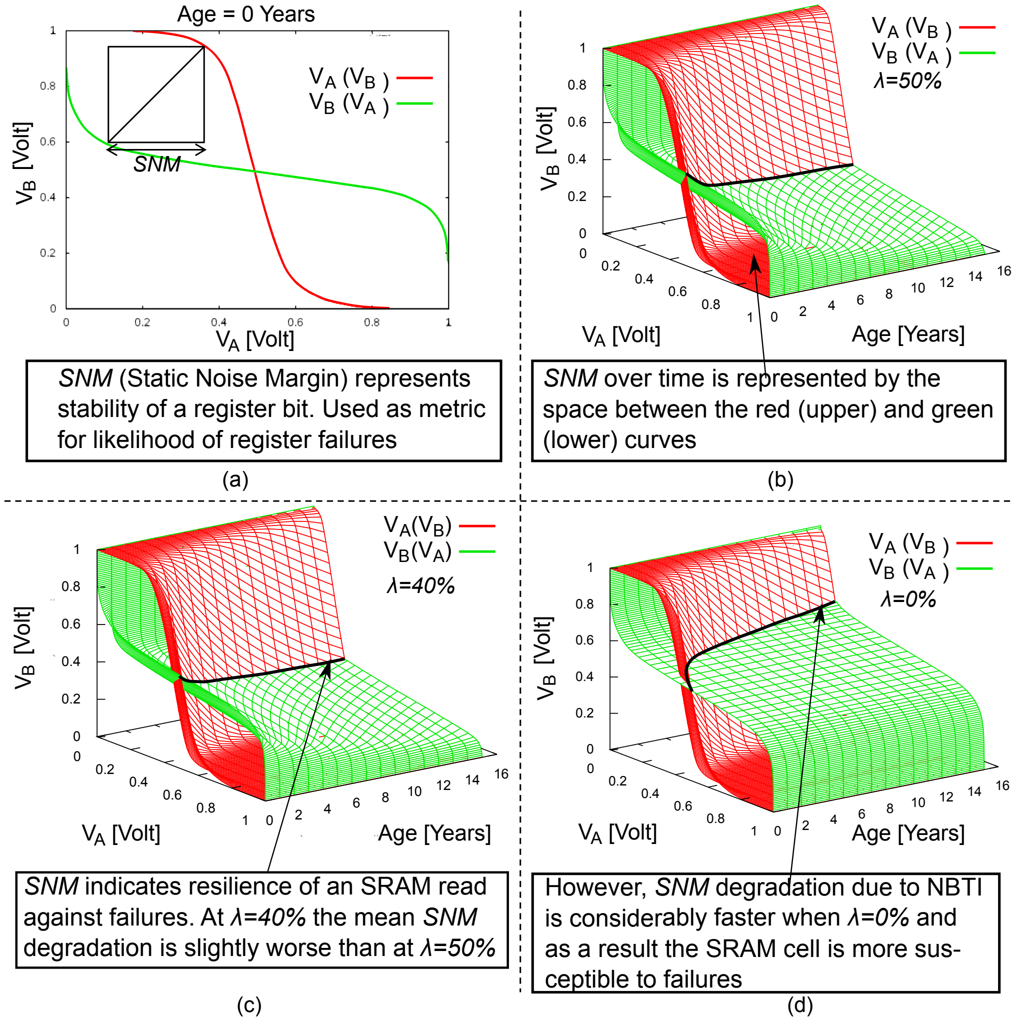
$$V_{th}(T) = V_{th}(T_0) + \gamma((\frac{T}{T_0})^{\beta_{V_{th}}} - 1) \tag{3.11}$$

with the mobility temperature exponent β_μ and the threshold voltage temperature coefficient γ, which are generally both negative. The default values in BSIM6.1 are: $\beta_\mu = -1.5$, $\beta_{V_{th}} = 1$ (i.e. no effect), $\gamma = -0.11V$. As temperatures rise from the nominal temperature T_0, both μ and V_{th} decrease.

Revisiting Equation 3.9, we see that here, μ and V_{th} have opposing effects. Whereas decreasing the mobility also decreases the drain current, decreasing V_{th} increases I_D. For a fixed system, whether the overall change in I_D is dominated by its relation to the change in V_{th} or to the change in μ depends on $V_{GS} - V_{th}(T)$, i.e. is determined by what supply voltage is applied. It has been shown that a *temperature insensitive voltage* V_{TIV} exists [21] where, if $V_{SS} = V_{TIV}$, temperature has no effect on delay when considering normal operating conditions (i.e. under 125°C), as the temperature effects of μ and V_{th} cancel each other out in I_D. For higher voltages, $V_{SS} > V_{TIV}$, the circuit is under *normal temperature dependence* [121] where the mobility effects are dominant, and temperature increases cause higher delays. At the same time, $V_{SS} < V_{TIV}$ causes *reverse temperature dependence* [121] where $V_{th}(T)$ is dominant and higher temperatures decrease delays. Through the use of high-κ gate dielectrics, V_{TIV} increases. It is around 0.4 V in 45 nm [122].

Additionally, variations of timing can be the result of the aforementioned aging effects. For instance, the increase of V_{th} over time due to NBTI also results in an increase in delay. Since these aging effects are hardly uniformly distributed, delays of individual transistors are affected differently. Of course it should be mentioned that not all changes in delay can or will result in timing errors. Errors depend highly on the circuit and whether it is on a critical path.

[1] Assuming default channel length temperature dependence of 0 Vm, as well as zero body-bias



SNM (Static Noise Margin) represents stability of a register bit. Used as metric for likelihood of register failures

(a)

SNM over time is represented by the space between the red (upper) and green (lower) curves

(b)

SNM indicates resilience of an SRAM read against failures. At *λ=40%* the mean *SNM* degradation is slightly worse than at *λ=50%*

(c)

However, *SNM* degradation due to NBTI is considerably faster when *λ=0%* and as a result the SRAM cell is more susceptible to failures

(d)

Figure 3.10.: Degradation of reliability over time due to NBTI quantified by SNM

3.2. Aging Effects and Reliability: Analysis of Effects on SRAM

This section takes a closer look at the effects of temperature-dependent aging mechanisms on specific circuits through an exemplary analysis of the effects on a standard 6-T SRAM Cell. Each cell consists of two cross-coupled CMOS inverters – made up of one PMOS and one NMOS transistor – and two NMOS access transistors [15]. The focus of this analysis is limited to NBTI. Thus, aging primarily affects the PMOS transistors.

The *Static Noise Margin (SNM)* is an often employed metric for expressing the reliability of an SRAM cell. It illustrates the resiliency of an SRAM cell against voltage noise and provides a design margin for avoiding failures. A design can compensate for any SNM above zero by increasing the sensitivity of the SRAM sense amplifier. Thus the sense amplifier is typically designed to meet desired yield criteria (e.g. six sigma [22]) considering the SNM.

The *SNM* is measured from the so-called *butterfly* curve that describes the transfer characteristics of the SRAM cell. Figure 3.10(a) depicts the read SNM of a 22 nm six transistor

(6-T) SRAM cell, with the *SNM* being equal to the length of the side of the square located between the two curves. Since there can be both an upper and a lower square, the smaller one of the two is taken to determine the SNM. A and B represent the outputs of the inverters of the 6-T SRAM cell. $V_A(V_B)$ plots the output voltage resulting from a voltage sweep in B from 0 to V_{DD}. $V_B(V_A)$ is the voltage in B when V_A sweeps from 0 to V_{DD}, with V_{DD} here being 1 V. The SNM was calculated using SPICE, together with the 22 nm Predictive Technology Model (PTM) [126]. A threshold voltage increase in the PMOS transistors due to NBTI degrades the *SNM* and consequently increases SRAM cell susceptibility to noise – decreasing reliability. Fig 3.10 (b, c and d) show the potential impact of NBTI on the reliability of an SRAM cell over years under different voltage stress scenarios. In this context, the percentage of time where voltage stress is applied is referred to as the *duty cycle* λ. Since the inverters are cross-coupled, if the PMOS transistor of one is under stress, the other is not, and vice versa. Thus the duty cycle of the individual PMOS transistors are $\lambda_1 = \lambda$ and $\lambda_2 = 1 - \lambda$, respectively.

When both PMOS transistors are subject to equal stress, as is the case in Figure 3.10 (b), both experience comparable degradation. This results in a symmetric decline in the upper and lower halves of the butterfly curve. The opposite is the case for $\lambda = 0\%$ in Figure 3.10 (c). Due to the uneven degradation, one half of the butterfly curve degrades faster than the other, resulting in an overall faster decrease of the SNM.

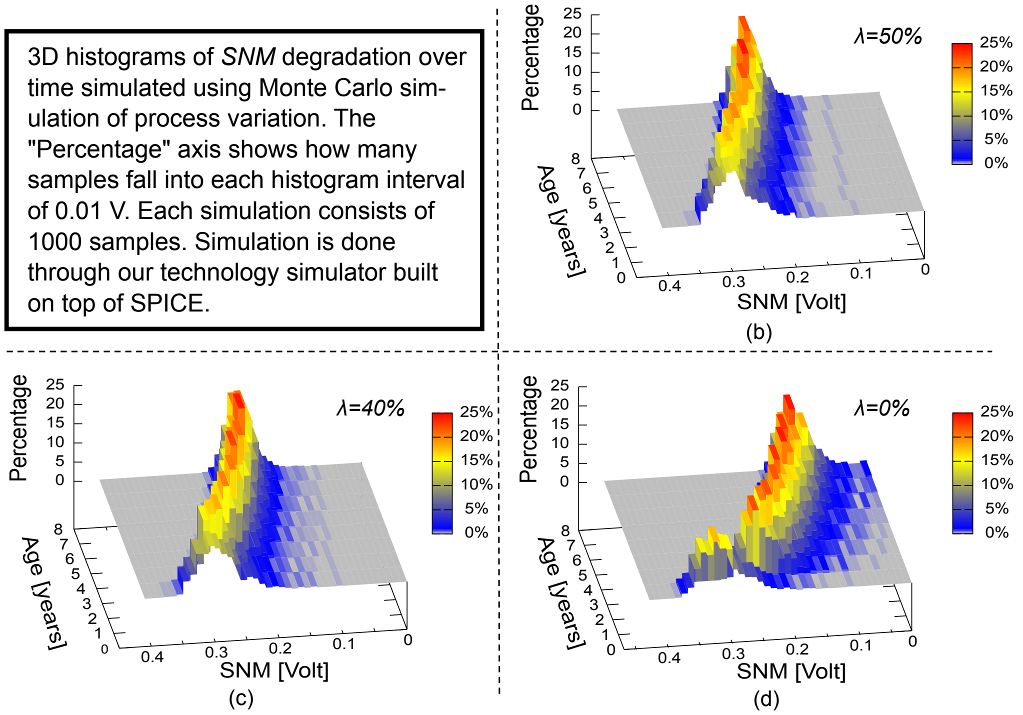

3D histograms of *SNM* degradation over time simulated using Monte Carlo simulation of process variation. The "Percentage" axis shows how many samples fall into each histogram interval of 0.01 V. Each simulation consists of 1000 samples. Simulation is done through our technology simulator built on top of SPICE.

Figure 3.11.: SNM degradation considering process variation

The *SNM* degradation becomes even more critical when accounting for transistor *variability*. Fig 3.11 shows the effect of process variation on the *SNM* for the same *duty cycles* λ as Fig 3.10, as a 3D histogram of Monte Carlo simulations at discrete time intervals. It can be seen that although the mean *SNM* degradation for $\lambda = 0\%$ is only about 40% after eight years, some SRAM cells are already failing, and the number of low *SNM* values close to zero is also not negligible. In this low range, the SRAM cell is much more susceptible to

DC voltage noise, which can result in a reduction of V_{DD} of up to 20% [16]. In contrast, Fig 3.11(e) and (f) show the histogram of the *SNM* over time considering process variation at the 45 nm and 32 nm technology nodes, respectively, for the worst-case $\lambda = 0\%$. It can be seen that there is little divergence from the mean *SNM*, showing the higher overall resilience of larger nodes.

3.3. Tackling Dependability Through Thermal Management

As becomes evident in Section 3.1, different dependability concerns need to be addressed using different – often contradictory – methods. For instance, the increase in threshold voltage due to NBTI can be avoided by increasing the system voltage. This, however, increases both dynamic and leakage power and, in turn, also increases temperature, resulting in an increase in aging due to electromigration. Conversely, lowering the system voltage reduces electromigration, but increases a system's susceptibility to a shift of the threshold voltage. Both effects can be reduced by a reduction of frequency, however this is often undesirable due to loss of performance.

3.3.1. Aging Budgeting

Assuming a multi-core system $\mathcal{S}$, we define the aging budget of each core $i \in \mathcal{S}$ as $\mathcal{B}_i = \{B_i, \Delta V_{th,i}, \hat{n}_i\}$, where B_i represents the effect of aging due to electromigration and NBTI, and is given as the time it takes for the dependability of a system to fall below a threshold at a reference temperature T_r and voltage V_r. $\hat{n}_i$ is the expected number of thermal cycles to failure for a normalized temperature difference $\hat{\Delta T}$. $\Delta V_{th,i}$ is the estimated mean shift in threshold voltage due to NBTI. It is regarded in addition to B_i in order to be able to react to changes in threshold voltage induced delay before a core failure occurs. Since aging takes place over long time periods, the aging budget only needs to be updated relatively infrequently, e.g. once per day. However, short-term thermal management is still a necessity in order to keep peak temperatures below a given threshold.

To enable aging budgeting based on temperature, it is assumed that thermal sensors are deployed throughout the system. In the short-term (e.g. every few ms) these are used to perform thermal management as well as collect statistics which are used to adjust the aging budget. For B_i and $\Delta V_{th,i}$ it is sufficient to record the average of the temperature terms (the e terms) and use these together with the models from Section 3.1 to calculate the change in the aging budget. When implemented, the e terms can be pre-computed for all temperatures achievable through thermal sensors (i.e. which typically have a resolution around 1°C) and stored in memory. For electromigration, the difference of the average temperatures of neighboring (i.e. adjacent) cores must also be recorded in order to incorporate spatial thermal gradients.

Adjusting $\hat{n}_i$ is a little more complex. To observe thermal cycles, it is necessary to examine whether the change in temperature over time dT/dt is either negative or positive. If dT/dt changes from positive to negative, a maximum temperature T_{max} has been reached. As soon as dT/dt becomes positive again, T_{min} has been reached. This is regarded as one thermal cycle with $\Delta T = T_{max} - T_{min}$. Assuming that the normalized temperature difference of a thermal cycle $\hat{\Delta T} = 1$, one thermal cycle with ΔT corresponds to $(\Delta T)^q$ normalized thermal cycles at $\hat{\Delta T}$. The number of normalized thermal cycles are then totaled and used when updating the aging budget.

3.3.2. Correcting Aging Budget at Runtime

In order to be able to perform corrections to the aging budget, it is necessary to observe the system state at runtime. This can be accomplished by deploying various aging sensors throughout the system. These sensors can empower on-chip systems to quantify the amount of aging induced by the major effects such as NBTI before it appears. It is expected that such sensors will be present in future microprocessors for use as real-time aging monitors. Intel, for instance, has developed a technology code-named Foxton, which measures the amount of propagation delay along specific critical signal paths within a microprocessor [8]. An interesting approach introduced by [12] proposed a new sensor integrated inside a flip-flop for predicting NBTI-induced PMOS aging. A novel sensor design able to predict aging induced by NBTI is proposed in [74] based on measuring the threshold voltage difference between an NBTI stressed and an NBTI unstressed MOSFET device using an inverter chain and a phase comparator. While the presence of aging sensors is not a requirement of our approach, they can serve to enhance its effectiveness.

3.3.3. Power Profiles

As a method for controlling temperatures with regard to long-term aging effects, we define various power profiles which determine allowed power dissipation of each core.

Idle: The *idle* power profile is used when no application is running on a core. If available, clock and or power gating should be applied to the core.

Low Performance (LP): From a dependability viewpoint, it is most beneficial to perform computation on cores running at low voltages and low temperatures at the cost of performance. Thus, if performance constraints allow, cores are set to the *LP* profile and limited to low voltages and frequencies. This is e.g. the case for low-priority non-critical applications. Furthermore, applications which offer a high degree of parallelism can also be run in the *LP* profile even while keeping performance constraints if the application is executed on multiple cores.

While the *LP* profile still produces higher temperatures than the *idle* profile, if a core is not running an application between two periods at the *LP* power profile it is not always beneficial to change to the *idle* profile due to the competing effects of e.g. NBTI, electromigration and thermal cycling. Once the B or $\hat{n}$ terms of the aging budget fall under a specified threshold, cores are automatically limited to the *LP* profile even if this results in performance constraints not being met.

High Performance (HP): In general, it is desirable to spread computation across all cores and run these at low frequencies and voltages. However, it is often the case that an application cannot be parallelized and must run on a single core at a high frequency in order to meet performance needs. When running in the high performance power profile, applications must be frequently migrated so that peak temperatures are reduced and the negative effects of NBTI are reduced. Ideally, between periods of running at the *HP* profile, a core is either running at the Idle or *LP* profile. This is controlled by the short-term aging budget.

High Performance - Low Dependability (HPLD): Once the V_{th} term of a core's energy budget sinks below a threshold, it is no longer able to reliably run at the *HP* power profile. In order to continue to be able to run at a high frequency, it is necessary to increase the voltage. While this measure is needed to meet performance demands, it unfortunately also accelerates the further aging process.

3.3.4. Thermal Management

Thermal management is determined by the aging budget together with the power profiles and temperature measurements. How the thermal management is performed depends on what strategy is employed, as outlined below.

Greedy Strategy: When using the greedy strategy, all *HP* executions are run on a two core subset of all cores $\mathcal{H} \subset \{i\}$. Once the aging budget no longer allows *HP* execution, the core is then blocked for high performance applications and removed from $\mathcal{H}$ allowing it to only run applications requiring the *LP* power profile. A new core is then added to $\mathcal{H}$. What happens once there are no cores left for *HP* execution depends on which terms in the aging budget have fallen under a threshold. If it is ΔV_{th}, the process continues using the *HPLD* power profile. For B and $\hat{n}_i$, however, the cores remain in the *LP* profile and performance constraints can no longer be met. The benefit of this approach is that core failures do not occur for all cores during one short time interval, but instead the cores lose their capabilities successively and temporally distributed. However, limiting the *HP* applications to a small number of cores can result in higher temperatures and thus accelerated aging.

Balanced Strategy: In the balanced strategy, all cores are in $\mathcal{H}$ and high performance applications are initially mapped to the core with the highest aging budget. This allows aging to occur more evenly across the chip. The downside of this strategy is that all cores will reach the aging budget threshold at which *HP* executions switch to the *HPLD* profile, as well as all cores failing at approximately the same time.

Adaptive Strategy: This strategy employs a mixture of the two previous ones. In it, the size of $\mathcal{H}$ is adapted over time depending on the current state of the system. Whether a core is in $\mathcal{H}$ depends on which factor of the aging budget is more expected to cause failure. If B_i is more critical, core i is inserted into $\mathcal{H}$. Once $\hat{n}_i$ becomes critical, it is removed. Note, however, that until all $\hat{n}_i$ fall under a determined threshold, it is necessary to have at least two cores in $\mathcal{H}$, even if all $\hat{n}_i$ are more critical than all B_i. In this case, the two cores with the highest absolute $\hat{n}_i$ are chosen.

Regardless of the employed strategy, the thermal management always reacts when the peak temperature of a core reaches a threshold.

Algorithm 1 presents an additional outline of one time-step of the adaptive strategy. First, while waiting for the time-step to complete, temperatures are continuously monitored (Lines 3-10). For each thermal cycle – when the temperature reaches its minimum – the number of normalized thermal cycles corresponding to the temperature differnce is added to the total number of thermal cycles in this time-step. Additionally, the average temperature is calculated to be used in the aging models. Second, the current aging budget and the set of cores allowing HP execution, $\mathcal{H}$, are updated from time t to $t+1$, based on the monitor data of the first step (Lines 13-25). In Line 14, the available aging budget components are decreased for each core. If these are now below the thresholds, the core is removed from $\mathcal{H}$. If they are above the thresholds the core is added.

3.4. Experimental Setup and Results

In order to evaluate our approach we first analyze the temperature characteristics of various applications of the SPEC2000 and SPEC2006 benchmark suite using the SimAlpha [42]

Algorithm 1 Aging Budgeting with Adaptive Strategy

```
Input: minimum threshold for $\hat{n}_i$: $\hat{n}_\tau$
Input: cores $i$, initial $\hat{n}_{i,0} = \hat{n}_i$ at $t = 0$
Input: $\mathcal{B}_i = \{B_i, \Delta V_{th,i}, \hat{n}_i\}, \mathcal{H}$ at time $t$
Output: $\mathcal{B}_i = \{B_i, \Delta V_{th,i}, \hat{n}_i\}, \mathcal{H}$ at time $t + 1$
begin
    $n_i = 0$
    $T_{max,i} = T_{min,i} = m_{i,0}$
    while $time \neq t + 1$ do
        foreach sensor measurement $m_{i,l}$ do
            if temperature maximum then
                $T_{max,i} = m_i$
            end
            if temperature minimum then
                $T_{min,i} = m_i$
                $n_i = n_i + (T_{max,i} - T_{min,i})^q$
            end
            $T_{avg,i} = (\sum_l m_{i,l})/l$
        end
    end
    foreach $i$ do
        $\hat{n}_i = \hat{n}_i - n_i$
        $B_i = B_i -$ EM_NBTI($T_{avg,i}$)
        $\Delta V_{th,i} = \Delta V_{th,i} +$ NBTI($T_{avg,i}$)
        if $((t + 1)/(\hat{n}_{i,0} - \hat{n}_i)) \cdot \hat{n}_i < B_i$ or $\hat{n}_i < \hat{n}_\tau$ then
            if $i \in \mathcal{H}$ then
                $\mathcal{H} = \mathcal{H} \setminus \{i\}$
            end
            “
        else
            if $i \notin \mathcal{H}$ then
                $\mathcal{H} = \mathcal{H} \cup \{i\}$
            end
        end
    end
    return $B_i$, $\mathcal{H}$
end
```

instruction level simulator combined with power values from McPAT [80] as input to the HotSpot thermal simulator [63]. We assume register files and cache accesses are evenly distributed using a technique such as [44], and therefore assume an even distribution of temperature inside the components. We further assume that a subset of the applications requires the high performance power profile (e.g. running at 2 GHz) and the rest are simulated at a low frequency and voltage akin to the low performance power profile (e.g. running at 800 MHz). These simulations are then used to develop a thermal profile for *HP*, *LP*, and *HPLD* applications. Using these thermal profiles we perform a random simulation for

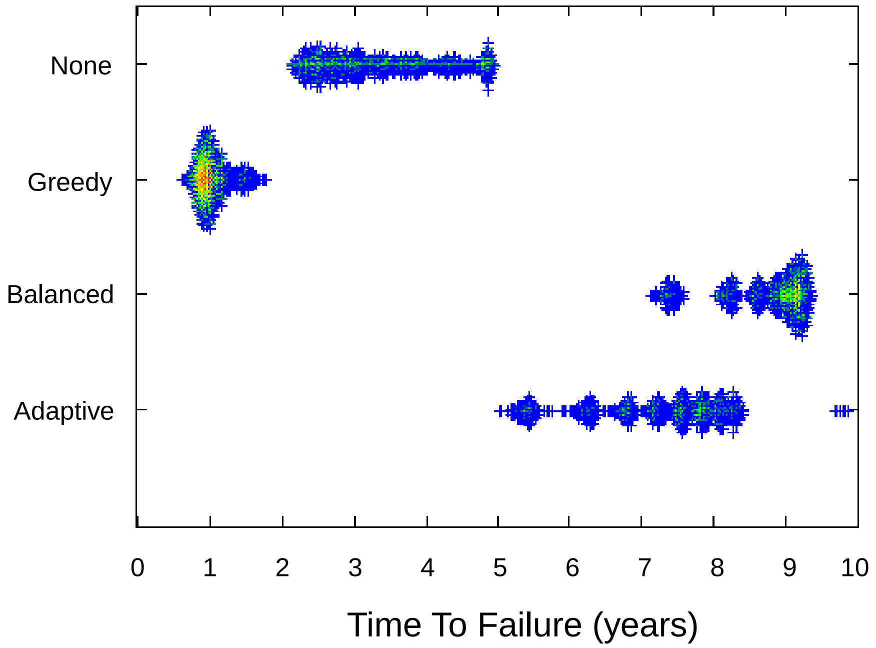


Figure 3.12.: Time to failure of individual cores for different approaches

each of the presented strategies to find the distribution of failures over time assuming a 16 core multi-core system with the number of applications simultaneously requiring a high performance profile varying uniformly from 1 to 10.

The results of the simulation runs are shown in Figure 3.12, which compares the core failures of the three presented strategies with the case where thermal management is only done to avoid peak temperature thresholds and not based on the aging budget. As can be seen, the greedy strategy performs worst out of all four. This is due to the limited number of cores in the set of *HP* cores $\mathcal{H}$ constantly reaching a higher temperature than when more cores are in $\mathcal{H}$. The balanced strategy, on the other hand, outperforms even the adaptive strategy in terms of dependability. However this comes at a cost: the balanced strategy is no longer able to meet performance constraints. In fact, the balanced strategy is no longer able to meet performance constraints in all runs on average after 1.95 years, whereas the adaptive strategy is able to meet performance constraints (until the last core fails) in 58% of all runs. In the remaining 42%, performance constraints are met until 7.62 years, on average.

Figure 3.13 shows the cumulative failures for one run of the approach while running 1 high-performance application requiring the *HP* power profile. It can be seen that, while the balanced strategy outperforms the adaptive strategy on average, the time to the overall system failure is still higher for the adaptive strategy in this case. As the number of *HP* applications increases, however, the time from the first core failure to last core failure decreases, and the last failure of the adaptive strategy is generally before that of the

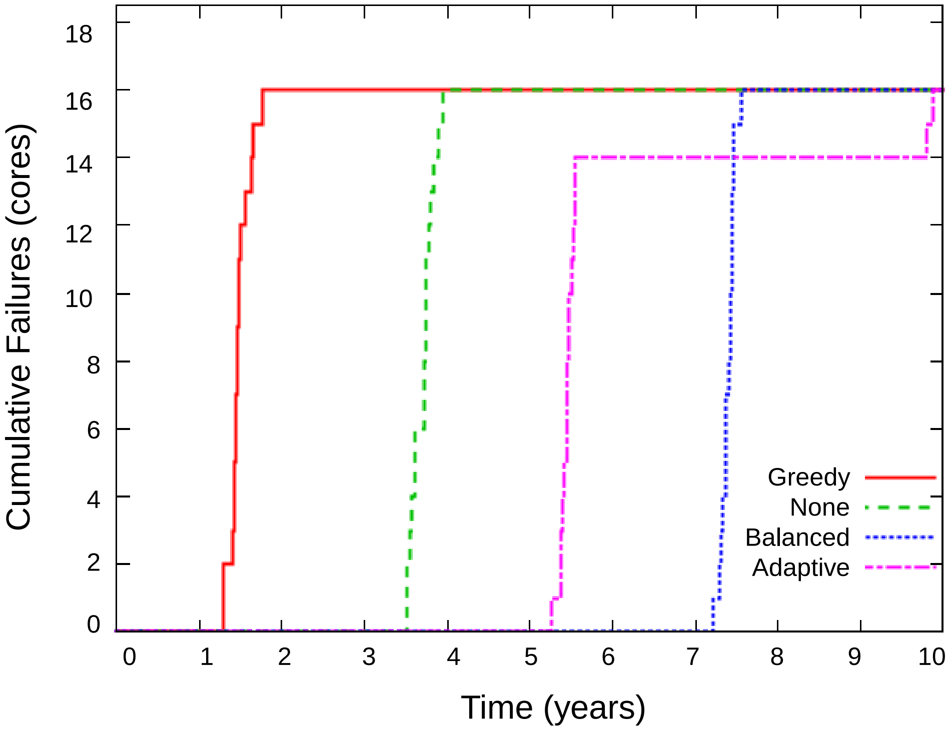


Figure 3.13.: Cumulative failures for different strategies

balanced one. Additionally, a higher *HP* application count also results in shorter lifetimes in the case of not performing thermal management targeted at dependability. Overall, the adaptive strategy achieves an increase in chip lifetime by a factor of 2.1 (on average) while keeping performance constraints.

It should be reiterated that these gains come entirely through control of temperature stress at the system level and are thus able to reduce multiple aging mechanisms at once – in particular by slowing electro-chemical processes. This approach is largely independent of microarchitecture-level approaches which aim to control voltage stress for specific aging mechanisms in specific components (provided these approaches do not increase temperature). For instance, [77] and [15] deal with voltage stress in SRAM and register files, respectively. Both have negligible impact on temperature since they are active infrequently (e.g. cell flipping occurs once per day in [77]). As such, the presented temperature management can be used to complement the microarchitecture-level approaches.

3.5. Summary

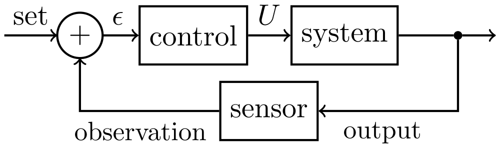
Manufacturing process advancements that enable nano-scale fabrication (i.e. with feature sizes of 45 nm and below) produce devices plagued by dependability issues. From a thermal perspective, the predominant issues are aging mechanisms such as NBTI, PBTI, HCI degradation, electromigration and TDDB occurring during device lifetime. As these are all electro-chemical processes, their rate is temperature-dependent in accordance with Arrhenius law. Higher temperatures accelerate reactions and thereby increase aging. While it is impossible to stop these effects at reasonable temperatures (i.e. above absolute zero), reducing temperature will at least delay them. Thermal management is expected to play a key role in ensuring that on-chip systems maintain dependable operation for as long as possible.

From a system point of view, the choice in thermal management strategy can have a considerable impact on lifetime degradation to aging mechanisms. For instance, using the presented adaptive strategy, it is possible to double lifetime expectancy in a multi-core system while being able to meet all deadlines. Longer operating times are also possible, but require DTM to perform a trade-off between maximizing lifetime and keeping performance constraints.

4. Control of Thermal Systems

A system undergoing thermal management is an example of a so-called *Observer-Controller* architecture. That is, an output value of the system is constantly observed to provide feedback to a controlling instance, that in turn changes the system configuration. This can be expressed as a classical control model as shown in Figure 4.1. When applied to thermal management, the system's output value is temperature. An example system configuration U might be, for instance, a DVFS setting or a task mapping.

In general, thermal management techniques can be divided into two categories: those whose goal is optimizing performance while keeping thermal constraints, and those aiming to optimize temperature under performance constraints. The key differences of these is highlighted in the next two sections.



ϵ - error, U - controller output

Figure 4.1.: Generic control system

4.1. Optimizing Performance

By far the most common type of thermal management is done by setting temperature constraints and optimizing for performance within them. Control systems are typically only effective if they are able to avoid oscillations that can bring the system out of a stable operating point. However, what this means is dependent on the type of optimization being done. For instance, when performing thermal management through task or activity migration but optimizing for performance, it is beneficial to avoid performing frequent migrations if migration overhead is high. This may result in a large oscillation of temperature. However, it is these oscillations that allow optimization of performance. It is important to note here that thermal oscillations are an essential part of thermal management. In fact thermal management using activity migration relies on oscillations in temperature: activity is migrated away from hot components to allow these to cool – at which point they can execute new activity and heat up again. For these types of oscillations it is merely important that they are constrained. Mostly this is done by limiting the allowed peak temperature to a given threshold.

4.2. Optimizing Temperature

Optimizing for temperature is a more specialized approach employed when it is necessary to minimize temperatures or variations of temperature. This allows maximizing system lifetime from the perspective of thermal dependability concerns (cf. Chapter 3) but comes at the cost of performance. Generally trivial solutions to thermal optimization exists that are, however, counter productive. As an example, a system employing dynamic voltage and frequency scaling (DVFS) to optimize temperature would simply run at as low a frequency as possible. As this is not practical, usually constraints on performance degradation are given.

An example of using control-based optimization of temperature through periodic migration of computational activity is presented in Section 4.3. The performance overhead here is mainly incurred through the overhead of activity migration. Thus optimizing for temperature may call for very frequent migrations which in turn increase the overhead affecting performance.

4.3. COOL: Control-based Optimization Of Load-balancing for Thermal Behavior

In COOL [44], an approach is presented that aims to optimize a system's temperature. As previously stated, this generally comes at the cost of a loss of performance guarantees, unless the cost of performing activity migration is negligible.

Figure 4.2 shows an example where such an approach is beneficial: by controlling the activity of a register file. Activity in the register file in Figure 4.2 is controlled by limiting the write accesses to a subset of the registers (in the example: component R) of the register file through the register renaming unit. Thus, activity migration is performed through the renaming unit by limiting write accesses to a different subset of registers (e.g. component L). There may be some read accesses to the now idle component (R). However, as shown in Section 4.6.1, these are infrequent across all observed benchmarks. Figure 4.2 shows two different rates of activity migration in a register file with two components: a) using infrequent migration, and b) using frequent migrations. Case b) is able to reduce thermal variations between the two components but, due to mutual heating resulting from the thermal conductivity between the components and insufficient cooling periods, it may lead to higher peak temperatures if the migrations are too frequent.[1] Case a) on the other hand, has a lower peak temperature, but a higher thermal variation. Our goal is to balance the load between components by finding the optimal rate of activity migration which minimizes the temperature T and dT/dx for a given workload.

4.4. Activity and Thermal Models

In order to formulate the properties of the thermal impact of activity migration, we examine the case where activity is migrated between two homogeneous components. To express the properties mathematically, we first assume an ideal system, i.e. where the overhead of activity migration is considered to be zero, and in the evaluation we apply our approach

[1] What is deemed as "too frequent" is highly dependent on what the components are. In the case of the register file, this is around more frequent then once every 15K execution cycles.

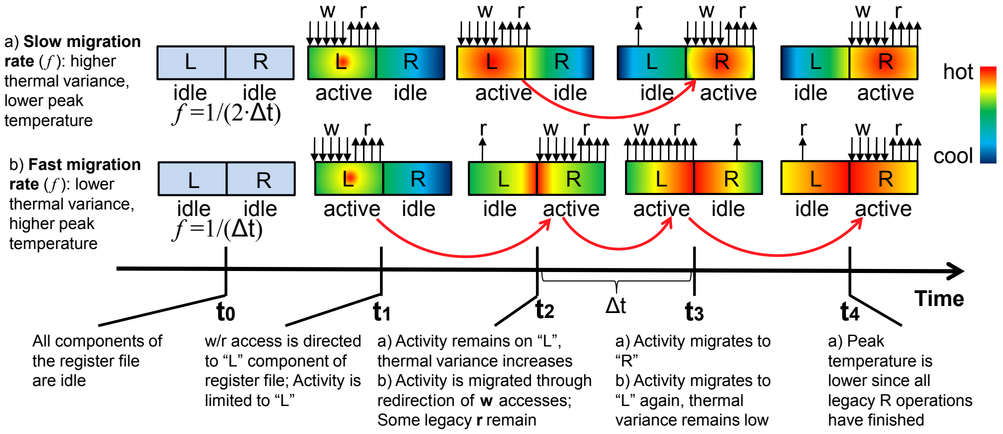

a) **Slow migration rate** (f): higher thermal variance, lower peak temperature
b) **Fast migration rate** (f): lower thermal variance, higher peak temperature
L
R
idle
active
w
r
f=1/(2·Δt)
f=1/(Δt)
hot
cool
Δt
Time
t0
t1
t2
t3
t4
All components of the register file are idle
w/r access is directed to "L" component of register file; Activity is limited to "L"
a) Activity remains on "L", thermal variance increases
b) Activity is migrated through redirection of **w** accesses; Some legacy **r** remain
a) Activity migrates to "R"
b) Activity migrates to "L" again, thermal variance remains low
a) Peak temperature is lower since all legacy R operations have finished

Figure 4.2.: Example scenario of activity migration in register file components and possible effects on temperature.

using both real thermal measurements (obtained from a thermal camera) and thermal simulations and we account for incurred performance penalty. Furthermore, only within this section, it is assumed that the power consumption is constant during execution. The ideal activity on the two components can thus be modeled using the *heaviside* function Θ.

$$\Theta(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases} \tag{4.1}$$

with the execution on each component modeled as $\Theta_1(f,t) := \Theta(\sin 2\pi f t)$ and $\Theta_2(f,t) := \Theta(-\sin 2\pi f t)$, respectively, with the migration rate f determining the frequency of activity migrations. The thermal development over time is then the *step response* for Θ, $\mathcal{R}(\Theta)$ as is shown in Figure 4.3, and can be expressed as

$$\mathcal{R}(\Theta) \equiv T(t) = S_n + (T(n) - S_n)e^{-t/a(n)}$$

where $T(n)$ is the initial temperature at a change in Θ, S_n is the *steady state* temperature after the change in Θ, and $a(n)$ controls the rate of heating/cooling. If $T_n < S_n$, the component is heating up, and if $T_n > S_n$, it is cooling. This corresponds to the solution of the differential equation of the first law of thermodynamics ($[c \cdot m \cdot \frac{dT}{dt} = P \cdot \Theta + \dot{Q}]$, $c \cdot m$ is the thermal capacity, P is the power consumption, and $\dot{Q}$ is the heat transferred to or from the system) under the assumption that the two components are symmetrical (i.e. both generate the same amount of heat). To calculate the thermal properties, it is necessary to examine the temperatures of both components T_1 and T_2. For each different value of f, we can use these temperatures to determine the peak and average temperatures, as well as the thermal variation between the two components for thermal balancing. As can be seen in Figure 4.3, these different migration rates f will result in different $T_{\max}$ and $T_{\min}$, allowing us to optimize thermal behavior – and thereby chip reliability and lifetime – by finding an optimal f. To accomplish this, we employ the principles of *extremum-seeking control*.

4.5. Application of Extremum-Seeking Control

Control theory deals with the control of dynamic systems using a feedback loop to drive a system into a desired state. Akin to the goal of optimizing the temperature/thermal

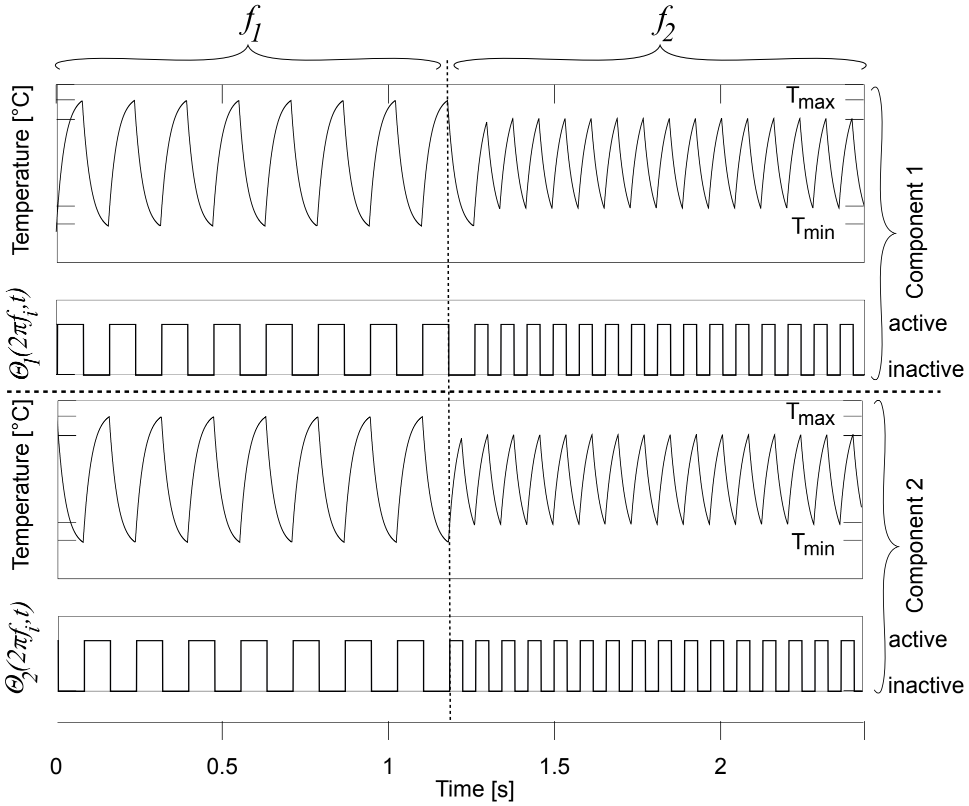


Figure 4.3.: Ideal activity migration and resulting temperature considering homogeneous components

balancing, the model of *extremum-seeking control*, also referred to as *self-optimizing* control, is exploited in order to optimize the output function g of a system as long as this function has an *extremum* value (minimum/maximum) and is at least differentiable two times ($g \in C^2$). Its notable difference to standard control theory is that it does not rely on a predetermined set point, i.e. a predetermined value for the output function to reach, but instead, it aims to drive the output function to an extreme value, e.g. to a minimum, and is thus particularly suited for the challenge of minimizing peak temperatures and thermal variations. To fulfill this goal, the controller controls components of an on-chip processor architecture. The output function g of the components is based on the temperature measured during runtime, as shown in the right part of Figure 4.4. In order to influence this temperature function, the controller changes the activity migration rate between the components. By observing how this change affects the temperature function, the controller can adapt the activity migration rate to drive the temperature function to a minimum.

According to Taylor's theorem, we can approximate f around a neighborhood a as:

$$g(f) \approx g(a) + g'(a)(f-a) + \frac{g''(a)}{2}(f-a)^2 + \epsilon \tag{4.2}$$

with an error term ϵ. Since we are interested in finding the extremum of the output function (e.g. the minimum temperature), we choose a such that $g(a)$ is a minimum/maximum, and therefore $g'(a) = 0$.

In summary, our *extremum-seeking* control works as follows: assuming the output function

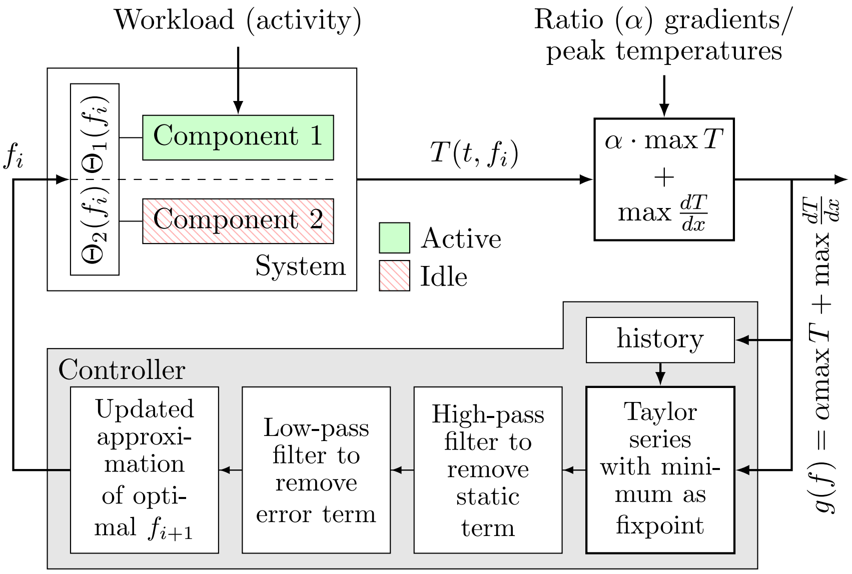

Workload (activity)
Ratio (α) gradients/ peak temperatures
f_i
$\Theta_2(f_i)$ $\Theta_1(f_i)$
Component 1
Component 2
System
$T(t, f_i)$
$\alpha \cdot \max T$ + $\max \frac{dT}{dx}$
🟩 Active
▨ Idle
$g(f) = \alpha \max T + \max \frac{dT}{dx}$
history
Controller
Updated approximation of optimal f_{i+1}
Low-pass filter to remove error term
High-pass filter to remove static term
Taylor series with minimum as fixpoint

Figure 4.4.: Overview of *extremum-seeking* controller

of the system $g(f)$ can be expressed as Equation 4.2 with an error ϵ, passing $g(f)$ through a high-pass "washout" filter will filter out the constant term $g(a)$. Likewise, the low-pass filter serves the purpose of filtering out the error ϵ, which can be seen as high-frequency noise. The result is an approximation for $g''(a)$, which is given in [17]:

$$\hat{f} \approx -\frac{kbg''(a)}{2}\tilde{f} \tag{4.3}$$

where $\tilde{f}$ is the estimation error $(a - \hat{f})$, k is the *adaptation gain* of the low-pass filter, b the amplitude of the probing signal used for modulation. This is then employed to determine the changed input rate f to the activity migration in order to drive the output function towards $g(a)$.

First we examine the case of optimal *thermal balancing*, in which the goal is to keep the system temperature as steady as possible, i.e. having a minimal thermal gradient between the two components, and avoiding thermal cycling. This is the case when the largest difference between the temperature of the first component, T_1, and the temperature on the second, T_2, is minimal.

$$g(f) = \max_{\Delta \mathrm{t}} \frac{(T_1(t, f) - T_2(t, f))}{dx} \tag{4.4}$$

Applying *extremum-seeking* control of Equation 4.3 shows that an optimal thermal balancing occurs when $f \to \infty$, or in other words, when activity migration is applied as frequent as possible. However, that is not feasible for practical thermal management, since too frequently applied migrations will negatively impact system performance. As such, when only optimizing for thermal balancing, we need to additionally specify an upper bound for f. Furthermore, a system optimized only for thermal balancing does not necessarily have the best overall thermal characteristics (i.e. it may still exhibit a high peak temperature whose negative effects on chip reliability and lifetime may outweigh the gain in reliability through minimizing thermal variations). This is a result of the latency of heating and cooling, i.e. there is not sufficient time for cooling if the component is only idle momentarily resulting in a system where the components being controlled are constantly heating up.

It is thus necessary to consider the additional thermal properties, i.e. the average and peak temperatures. We can express the average temperature as:

$$g(f) = \frac{1}{2}(\frac{\sum_t T_1(t,f)}{\Delta t} + \frac{\sum_t T_2(t,f)}{\Delta t}) \tag{4.5}$$

Likewise, to optimize peak temperatures we consider

$$g(f) = \max_{\Delta \mathrm{t}}(T_1(t,f), T_2(t,f)) \tag{4.6}$$

In practice, all three of these factors must be considered, and hence, the output function is a weighted combination of Equations 4.4, 4.5, and 4.6 with respective weights. However, in our experiments we found a correlation between average and peak temperatures (see Figure 4.10), making it sufficient to only optimize for peak temperatures and thermal balancing. This is typically the case when the components occupy a small chip area footprint, due to thermal conductance within the components. If both the area of the components is large and activity is focused on a small part of the components (e.g as may be the case if a component is an entire core, and not only a part of its microarchitecture), however, the average and peak temperatures may no longer be correlated to each other and the average temperature must be taken into consideration and included in the output function. Determining the weight values (α) is device-specific, and as such they must be obtained using an architecture-dependent device exploration. In Figure 4.4, the weight is given by α which determines the ratio of optimization between the peak temperature and thermal balancing.

Our approach can be implemented using either online or offline optimization depending on how much *a priori* knowledge is available, i.e. if the activity during runtime is known, the optimal migration strategy can be obtained by running our approach offline and applying the devised strategy for runtime activity migration. If this is not possible, e.g. due to non-deterministic activity resulting from different application inputs or user behavior, the optimization of the migration strategy can be accomplished online using our controller. While the online approach is more flexible, it also requires additional hardware resources to implement our controller.

It is important to differentiate between heterogeneous and homogeneous components. Homogeneous components use the same amount of switching activity (and occupy the same area footprint) for computation and thereby produce equal amounts of heat. In this case migration is applied between the components with each component being active for the same time duration. In the general case, the components are heterogeneous and consume different amounts of power and/or occupy a different amount of area, and, therefore, generate different amounts of heat. To optimize thermal behavior, these heterogeneous components must therefore be active for different amounts of time. The activity can thus be modeled using an extension of Equation 4.1, using two components as an example:

$$\begin{aligned}\Theta_1(f,\beta,t) &:= \Theta(\sin 2\pi f t + \beta) \text{ and}\\ \Theta_2(f,\beta,t) &:= \Theta(-\sin 2\pi f t - \beta)\end{aligned}$$

where $\beta \in [0, 1)$. As a result, Θ_1 and Θ_2 are alternatingly active for the intervals $\frac{f}{2}(1+\beta)$ and $\frac{f}{2}(1-\beta)$, respectively, and $\beta = 0$ corresponds to the heterogeneous case. Without loss of generality, Θ_2 represents the component with increased computation time, i.e. the component that generates less heat. The values for β are optimized offline with the *extremum-seeking* controller using a fixed f.

4.5.1. Extension to higher-dimensional problem space

To cover a larger range of architectures, i.e. architectures with more than two components and one activity, the above example must be extended to the multi-dimensional case with n components and m activities. This can be accomplished by using a chain of *extremum-seeking* controllers [41], each responsible for deciding when to migrate activity from one component. An example of such a controller chain is shown in Figure 4.5 and described in Algorithm 2. The inputs of a component are averaged between *extremum-seeking* controllers it shares with adjacent components, i.e. components located physically next to each other. Thus, in assuming a two-dimensional mesh architecture, each component will have between two and four inputs depending on its location (e.g. if it is somewhere in the middle or at the border).

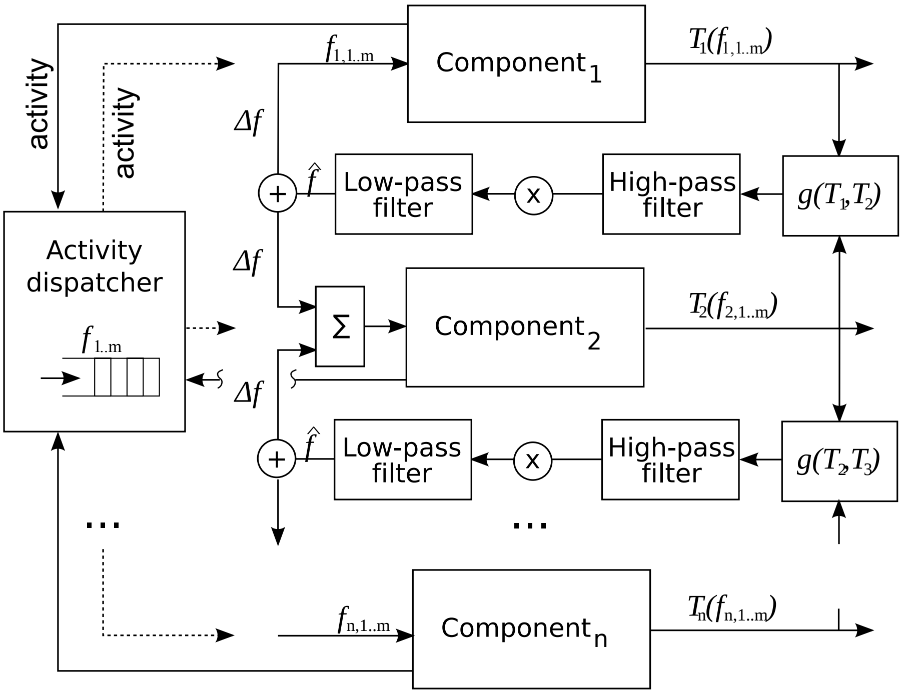


Figure 4.5.: Chaining of multiple *extremum-seeking* controllers

Here, the migration rate is no longer fixed, but instead each activity $\tau_j, j \in \{1, .., m\}$ has its distinct migration rate f_{τ_j}. Thus each of the n cores has as its inputs f of the activity currently running on the component. For instance, Component$_i$, $i \in \{1, .., n\}$ will have the inputs f_{i,τ_j} if activity τ_j is running. Each component can have up to m inputs as shown in Figure 4.5. For each activity migration, the activity is migrated from the component it is running on and placed to the queue of the *activity dispatcher* which chooses a new component to migrate the activity to.

Algorithm 2 COOLChain: Chained *extremum-seeking* control at time t

```
Input: Set of $m$ Activities: $\{\tau_j\}$
Input:  set of $n$ components: $\{C_i\}$
Input: Initial allocation at time $t$: $A_{ij}(t) = \{\tau_j \rightarrow C_i\}$
Input: Corresponding migration rates $f_{\tau_j}(t)$
Output: Modified migration rates $f_{\tau_j}(t+1)$
Output: Modified allocation: $A_{ij}(t+1)$
begin
    foreach Controller $\mathfrak{C}_k$ do
        foreach Input $C_{i'} \subset \{C_i\}$ do
            Get sensor data $T_{i'}(t)$   foreach Activity $\tau_{j'} \in A(i'j)$ do
                // Using Equation 4.3:
                $tildef_{\tau_{j'}} = \mathfrak{C}_k(f_{\tau_{j'}}, T_{i'}(t))$
            end
        end
        $l$ =MAX$_i$ $\tilde{f}_{\tau_{j'}}$   $\Delta f$ =VECTOR$_m$ (0..0)   $\Delta f_l = \tilde{f}_{\tau_{j'}}$
    end
    foreach $C_i$ do
        foreach Input from $\mathfrak{C}_k$ do
            $\{f_{\tau_j}(t+1)\} = \{f_{\tau_j}(t)\} + \Delta f$
        end
    end
    foreach Activity $\tau \in \{\tau_j\}$ do
        if $f_\tau = n \cdot t, n \in \mathbb{N}$ then
            $A_{ij}(t+1) = A_{ij}(t) \setminus \{\tau\}$   dispatcherQueue.add($\tau$)
        end
    end
    foreach activity $\tau$ in dispatcherQueue do
        $C_{\tilde{i}} \leftarrow$ getNextComponent(policy)   $A_{ij}(t+1) = A_{ij}(t) \cup \{\tau \rightarrow C_{\tilde{i}}\}$
    end
end
```

There are two parameters which influence the effectiveness of the chained controller.

- The first one is the policy of the activity dispatcher. Where it decides to migrate activity to is dependent on the weight α (see Figure 4.4). For instance, when optimizing for spatial thermal variation, it is most efficient to migrate an activity to the coolest core in the region (where the variation is maximal).
- Second, in the general case where there is more than one activity scheduled on a component, it must be decided how $\sum(\Delta f)$ influences the individual f_τ. For homogeneous activity workloads, we simply reduce the maximum f_τ. However for heterogeneous workloads, the load also needs to be considered. Thus, assuming that a normalized load parameter L_τ is known (i.e. through performance counters), the f_τ is reduced where $L_\tau \cdot f_\tau$ is maximal.

Due to the properties of *extremum-seeking control*, we are able to manage on-chip temperatures, minimizing both peak temperatures and thermal variations. In the following we analyze the effectiveness of the application of COOL for thermal optimization in different systems. First we analyze the effect of activity migration in FPGA-based systems which enable us to obtain detailed thermal measurements. Subsequently, as one possible application example, we apply our approach to the register file of a superscalar microarchitecture in an ASIC and analyze its effectiveness using thermal simulation.

4.6. Thermal Analysis

To analyze the COOL control-based temperature optimization using real measurements, a thermal camera is used to monitor the infrared emissions from a die. Using Xilinx Virtex-II and Virtex-5 FPGAs we are able to examine the effects of various values for f which determine how often the activity is migrated between two switching regions. The on-chip temperature distributions are observed using a DIAS Pyroview 380L compact infrared thermal camera [3]. The utilized infrared camera is able to measure the emitted temperatures from the FPGA die with an accuracy of ($\pm 1°C$).

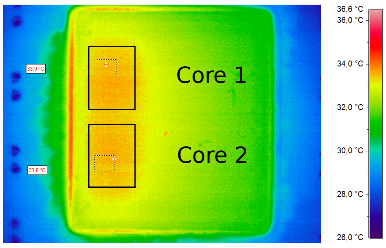


Figure 4.6.: Infrared image of 77.5 MHz activity migration (every cycle)

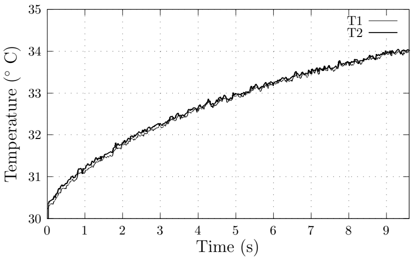


Figure 4.7.: Temperatures due to 77.5 MHz activity migration

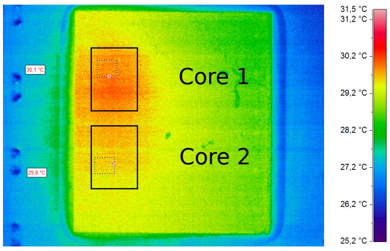


Figure 4.8.: Infrared image of 7.75 Hz activity migration

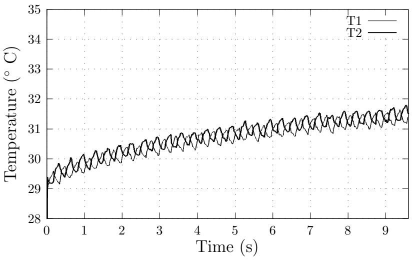


Figure 4.9.: Temperatures due to 7.75 Hz activity migration

The implemented designs on the tested FPGAs emulate two computational components by means of switching activity using Toggle-Flip Flops (T-FFs) in rectangular regions on the FPGA, representing the worst-case of switching activity. Each of these components can be in one of two states: *active* or *idle*. A running component consumes 100% switching power (a normalized value corresponding to the average switching activity of that component), and, in our implementation, an idle component is clock gated, consuming 0% switching power. Activity migration is performed by swapping the states of the two components, i.e. the active component is clock gated and becomes idle, whereas the clock of the idle component is enabled, making it active. While the FPGA chip is unable to produce temperatures as high as ASIC chips of current technology nodes may reach, the concepts presented can be used at any technology node and are equally applicable to higher temperatures, as is seen in Section 4.6.1.

Device Characteristics:

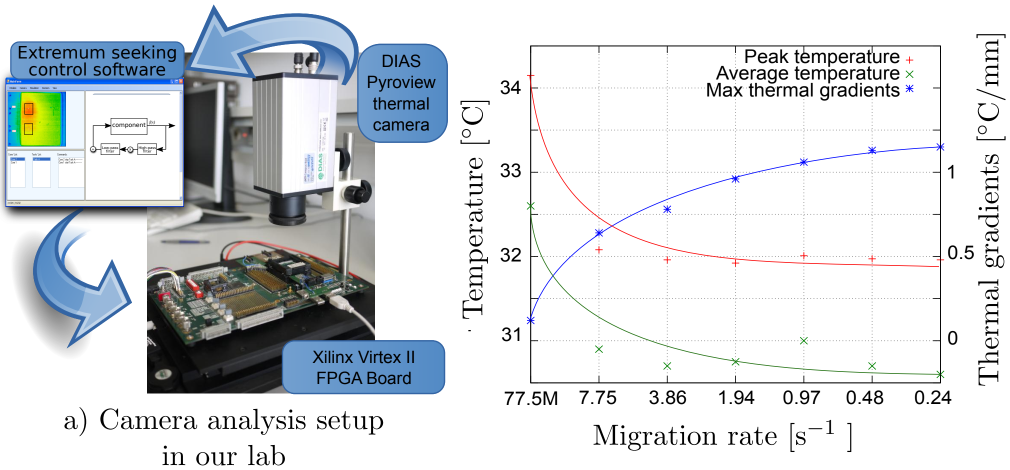
Virtex-II	XC2VP30 77.5 MHz
Virtex 5	XC5VLX110T 400 MHz

We begin our evaluation using the Virtex-II FPGA and examining the effects of various values for f which determine how often the activity is migrated between the two emulated cores, each consisting of 1000 T-FFs. The thermal images of two of these runs are shown in Figures 4.6 and 4.8. The temperature scale of the two images is generated dynamically based on the total temperature range, and is therefore not consistent between the two images. In Figure 4.6, the activity migration frequency, given by the angular frequency f from Section 4.4, is the maximum possible with the task switching cores after every cycle. As such power consumption and heat production are as balanced as possible, resulting in a symmetric thermal image. In Figure 4.8, however, the switching of the task between the cores is less frequent, allowing one of the cores to cool down while the other heats up. This results in an asymmetric thermal distribution. It should additionally be noted that the idle core cannot cool down completely to the ambient temperature, as it is still affected by heat conducted from the active core, as well as by static power consumption, e.g. due to leakage power.

Figures 4.7 and 4.9 show the temperatures of the two cores over time for two activity migration frequencies – once after every cycle and once after every 10 million cycles, corresponding to around once every 13 ns and 130 ms, respectively when running at 77.5 MHz. Both runs show a steady increase in both the maximum and minimum temperatures over time as the system gradually heats up towards the steady state temperatures of the individual runs. However, it can be seen that, in the case with the optimal thermal balancing (Figure 4.7), the peak temperature is greater than in the other cases. This is largely due to the heat conducted from the active core to the idle one and that the idle core does not have enough time to cool down during one cycle. In this case, optimizing the system for only thermal balancing will result in the worst case for peak temperature. This observation also holds true for the four core Virtex-5 case below, as well as in the register file in Section 4.6.1.

The temperature values given for the input f in Figures 4.7-4.9 present an insight on the observability and controllability of the system. Through the observed temperatures, we are directly able to infer the current system state set by the input frequency f by regarding the periods of heating/cooling in each core. Likewise, changing the input f will result in a predictable change in the system output function.

The analysis of the effects of the chosen f (i.e. without using the controller) on the thermal characteristics of the chip are summarized in Figure 4.10. We can see that optimizing for peak temperature corresponds to an optimization for average temperature, i.e. here the f resulting in the lowest peak temperature will also have the lowest average temperature. As such, we limit our optimization to peak temperatures and thermal balancing. Since the maximum thermal variation, which quantifies the degree of thermal balancing, ranges from 0 to $1.6°C$, we choose $\alpha = 1/30$ (see Figure 4.4) such that the impact of thermal balancing is comparable to that of peak temperatures. The obtained weight can then be used in the *extremum-seeking* controller to optimize activity migration at runtime.



a) Camera analysis setup in our lab

Figure 4.10.: b) Summary of thermal characteristics for various activity migration rates f.

Since the input to the controller is discrete, the continuous Laplace is replaced by the discrete z-transform. The *rate of control* is limited by the camera frequency (50 Hz), meaning that changing the activity migration rate can at most be done whenever new temperatures are observed. However, it is also necessary to wait for multiple temperature measurements in order to accurately regard the current state and its relation to the extremum $g(a)$, and especially also in order to decrease the effect of measurement noise. Through our thermal measurements, we have determined that for our evaluation platform, a rate of control of once per second is sufficient since the heating of the chip to the peak temperature takes several minutes (between 5 and 10 minutes depending on ambient temperature). However, if a system requires more dynamic thermal management, and has faster on-chip thermal sensors (e.g. reading a thermal sensor takes 54ms in [20]), the rate of control can be in the range of milliseconds (e.g. once every 10-100ms).

Figure 4.11 a) shows the application of the controller implementation on our evaluation platform. After the interval of one second, determined by the rate of control, the controller performs a change to the input f. Initially, the thermal balancing component of the output function is the deciding factor, and thus the input is changed by increasing the frequency of activity migrations in order to achieve a better thermal balancing. At the second controller iteration at $t = 2$s, the peak temperature has risen above the peak temperature of the previous input f, resulting in the next input decreasing the migration frequency. This process continues until the optimal f is reached, decreasing the peak temperature in the example scenario by 6.3% (°C) compared to only optimizing for thermal balancing. It should be noted, however, that due to the monotonous increase in peak temperature during the initial heating phase, the *extremum-seeking* control drives the output function towards *local* maxima/minima. If this is no longer optimal, once the initial heating phase is complete, the controller will again change the input f towards the global optimal state. The initial heating phase is shown here in order to illustrate the dynamics of the approach, since in the worst-case scenario (and in particular since the total activity is constant, i.e. not application dependent), f will become constant. In Figure 4.11 b), the change in f over multiple control intervals is shown in terms of activity migrations per second, which illustrates the convergence of the controller towards an optimal f.

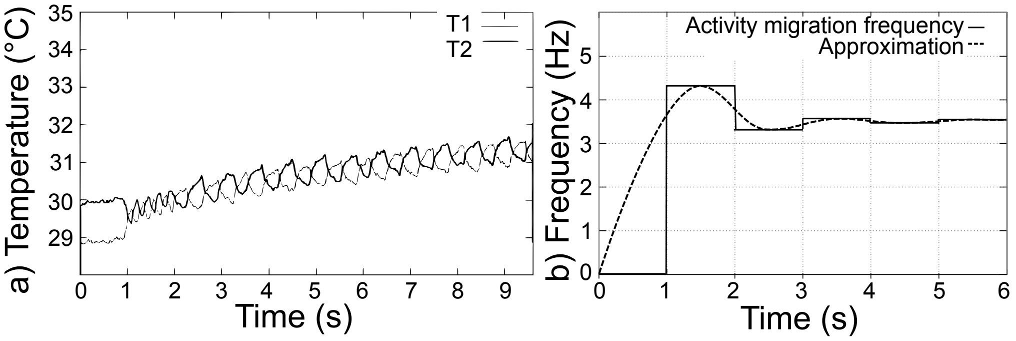


Figure 4.11.: a) Typical runtime control example showing the temperatures T_1 and T_2 of the two components and b) Refinement of f by *extremum-seeking* controller towards optimal value

For the evaluation with the Virtex-5, we examine the temperatures after several minutes of operation when the thermal state of the system has stabilized to the steady state temperature. At this point, the peak/average temperatures and max thermal variation no longer changes (except for noise temperature variations). Since the temperature has stabilized, optimizing for f will result in a fixed value. For instance, when using the *extremum-seeking* controller to find optimal values with the 2000 T-FF design, the resulting f is 4.167 Hz (once every 96 M cycles), with a peak temperature of $58.2°C$ and a thermal variation of $0.98°C$, a reduction of $9°C$ and $5°C$, respectively. Table 4.1 shows a summary of the thermal characteristics of several evaluation runs.

To evaluate the multi-dimensional problem with multiple components, we have implemented a four component version of the 2000 T-FF Virtex-5 design running two activities, τ_1 and τ_2. Each activity has its own corresponding f value, f_{τ_1} and f_{τ_2}. The thermal exploration of this setup is shown in Figures 4.12 and 4.13. When regarding peak temperatures in Figure 4.12, we can see two effects which contribute to the peak temperature. The first is that a high migration rate results in higher temperatures, analogous to the two component system. The other is that a correlation between f_{τ_1} and f_{τ_2} can result in more mutual heating. Thermal gradients, however, are still mostly dependent on f, with larger values (and thus lower migration rates) resulting in higher gradients. The maximum peak temperature measured is $65.8°C$, and the maximum thermal variation $1.5°C$. In our approach, the thermal optimization for four components consists of four controllers, each with two inputs and two outputs. When starting from the worst case ($f_{\tau_1} = f_{\tau_2} = 1$), the COOL first results in an increase in both f values. The increase of both is slightly different resulting from the different controllers and shifting the peak temperature away from the center in Figure 4.12 (i.e. where there are higher peak temperatures resulting from correlated values of f). The resulting values for f obtained are $f_{\tau_1} = 5.2$ Hz (once every 76.8M cycles), and $f_{\tau_2} = 15.625$ Hz (once every 25.6M cycles) with an overall peak temperature of $63.1°C$ and thermal variation of $0.84°C$.

In summary, the analysis presented in this section shows the runtime behavior of our approach measured using an infrared thermal camera. This provides us with a verification of the concepts presented in Section 5.5, showing that there is an optimal activity migration rate f and that our controller is able to determine that. We achieve a reduction of peak temperature of $9°C$ and a reduction in thermal variation from $6°C$ to $1°C$ by optimiz-

FPGA	T-FFs	Clock	Migration Rate f	T_A	T_{max}	T_{avg}	ΔT
		$[10^6\ s^{-1}]$	$[s^{-1}]$	$[°C]$	$[°C]$	$[°C]$	$[°C]$
Virtex-II	1000	77.5	77.5×10^6	29.0	34.15	32.60	0.12
			7.75	29.0	32.08	30.90	0.64
			3.875	29.0	31.96	30.70	0.78
			1.938	29.0	31.36	30.75	0.96
			0.969	29.0	32.14	31.00	1.06
			0.484	29.0	32.01	30.70	1.13
Virtex-5	2000	400	50×10^3	29.4	57.62	56.43	0.40
			1×10^3	29.4	56.10	55.30	0.46
			500	29.4	55.80	54.93	0.49
			196	29.4	55.46	54.60	0.46
			50	32.0	64.91	57.50	0.64
			1	32.0	57.90	55.50	2.74
			0.5	32.0	57.64	55.22	3.06
			0.196	32.0	58.40	55.54	3.10
	4000	400	50×10^3	29.5	68.70	67.28	0.51
			1×10^3	29.5	68.16	66.70	0.76
			500	29.5	68.16	66.31	0.80
			196	29.5	67.12	65.63	0.71

Table 4.1.: Summary of Thermal Behavior (peak temperature and thermal variation)

ing f in the case of two components in a Virtex-5 FPGA. When more components and activities are considered, the reductions achieved are lower due to the overall increase in activity, however the controller is still able to reduce temperatures to the minimums seen in Figures 4.12 and 4.13.

4.6.1. A Look at Specific Components: The Register File

The analysis has thus far focused on a generalized concept of components. To demonstrate the application of the approach, additional experiments focus on one of the parts of a processor that traditionally exhibits the highest power densities [54]: the register file. This section takes a brief look at the analysis of COOL applied to the register file of a superscalar architecture.

As an example superscalar architecture, the Alpha 21464 processor [98] has 32 architecture registers and a larger number of physical registers (e.g. 40 registers). In addition to this, 8 shadow registers are reserved for the operating system. Regarding register access patterns, experiments for different applications from the SPEC 2006 and SPEC 2000 [6] benchmark suites reveal the following: Due to the unbalanced access to the register file, the majority

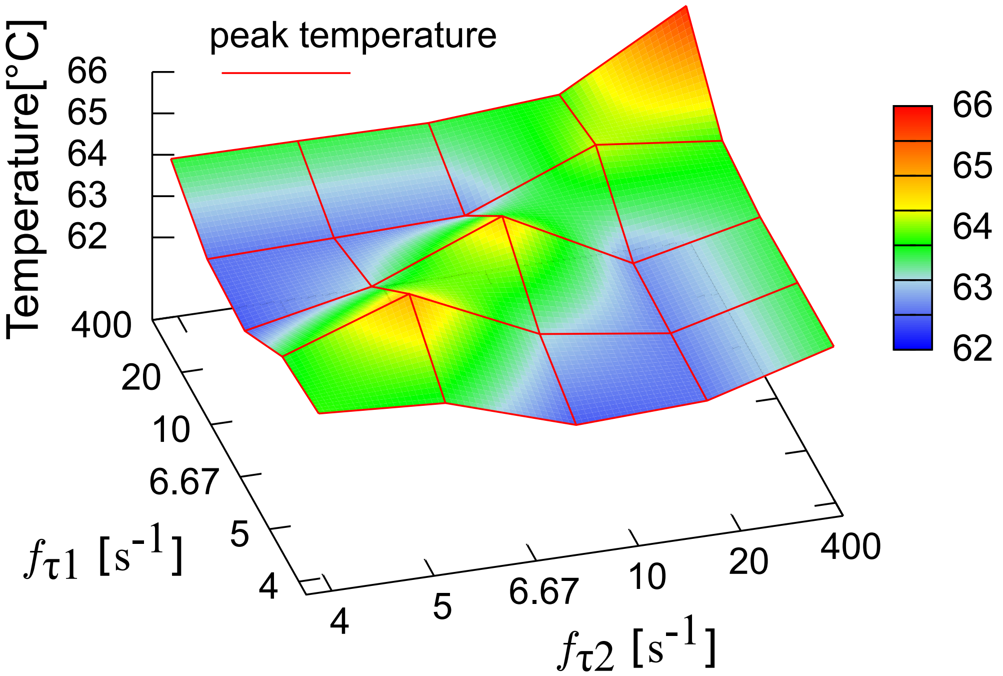


Figure 4.12.: Peak temperatures with 4 cores dependent on f measured by thermal camera

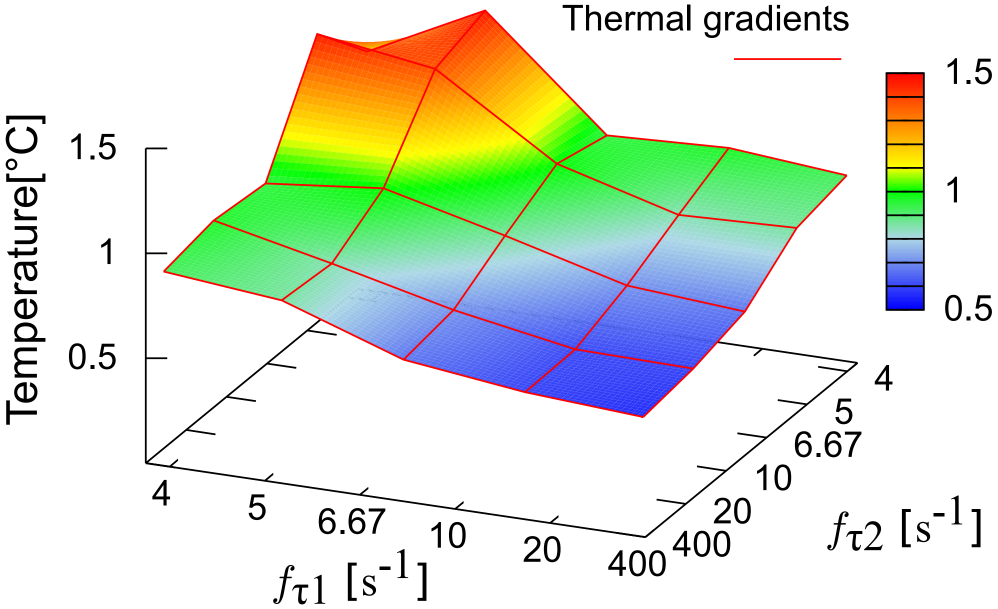
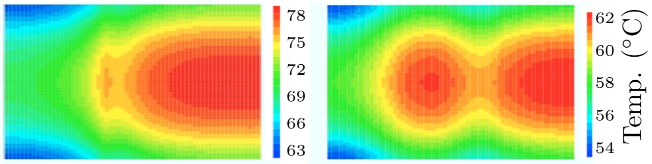


Figure 4.13.: Thermal gradients with 4 cores dependent on f measured by thermal camera

of heat often builds up in a small section. This can be seen in the simulated thermal map in Figure 4.14(a). Additionally, not all the registers are used at a certain point of time and the average occupancy of the register file is relatively low across the benchmarks[2]. About half of the physical registers in the register file are not occupied in around 90% of clock

[2] In spite of this observation, it is worthy to note that decreasing the register file size too much would have a severe impact on the performance, and thus an extensive exploration of the benchmarks is highly needed.

cycles. This observation is consistent with related work [61, 96, 90].



a) RF: No activity migration

b) RF: Extremum-seeking control

Figure 4.14.: Simulated thermal maps of the register file (RF) with total 80 registers for the gzip benchmark (a) Base case and (b) after applying the *extremum-seeking* controller

The aforementioned observations motivate the use of an *extremum-seeking* controller to migrate the activity of a set of registers that are responsible for the thermal hotspots.

The most straightforward way to realize two components for activity migration is to simply duplicate the register file. This additional area overhead may not be desirable, due to area limitations, or even possible, as it requires changing the physical chip design. Another way to realize two components is by dividing the existing register file into two. This is done logically by limiting the access of the renaming unit (`IntMap`) to only part of the register file during each interval, i.e. corresponding to the example presented in Figure 4.2. Practically, this results in only half of the physical registers being available to `IntMap` at any point of time during execution. Unfortunately, reducing the effective size of the register file results in a loss of performance. The impact on the performance degradation is expected to be low, however, (as presented later: around 1.4% increase in execution time) due to the observation that at least half of the physical registers are not occupied during the majority of the execution time.

The *extremum-seeking* controller migrates the activity between the unused half (the cool region) and the occupied half (the hot region) of the physical registers. The optimal migration rate (f) required for migrating between the active and idle region are computed offline. It is necessary to note that f is not the same for each benchmark as shown in Table 4.2 and it highly depends on the utilization of the register file of each application.

Due to its small size which limits its observability under the infrared camera, the analysis of the register file components is simulated.

Table 4.2 summarizes the temperature results achieved for various applications of the SPEC benchmark suites. As seen, the peak temperature reduction after implementing our approach reaches up to $21°C$ and, on average, around $13°C$ and $11°C$ for the SEC 2006 and SPEC 2000 benchmark suite, respectively. On the other hand, the activity migration rate derived from the *extremum-seeking* controller effectively minimizes the thermal variation. Compared to the *Base* version, the thermal variation has been reduced, on average, by 64% and 62% for the SEC 2006 and SPEC 2000 benchmark suite, respectively. It also achieves better results compared to State-of-the-Art techniques. For instance, the thermal variation inside the register file has been decreased, on average, by 38% and 49% in comparison to *"Odd Even"*[73] and *"Bank Switching"*[96] approaches, respectively.

Overhead: for controlling the thermal behavior of the register file, the *extremum-seeking* optimization was applied offline for each application as it is possible to estimate the register file utilization by analyzing the application scenario. Thus, the only overhead that comes with COOL is the overhead from the controller implemented in the `IntMap` component

SPEC 2000				SPEC 2006			
Benchmark	Migr. Rate f [10^3 s^{-1}]	Peak T [K]	T Red. [T]	Benchmark	Migr. Rate f [10^3 s^{-1}]	Peak T [K]	T Red. [K]
vortex	20	332	10.8	libquantum	100	340	20.9
gcc	100	331	10.4	hmmer	0.5	332.3	12.6
bzip	2	332.8	13.7	sjeng	100	322.8	3
mcf	4	330	2.3	specrand	2	336	17.3
gzip	20	334	18.9				
parser	2	328.7	10				
average	4.42			average	1.59		

Table 4.2.: Optimal activity migration rates and corresponding temperatures/temperature reductions.

of the Alpha processor. Towards investigating the overhead of the employed controller, we designed it in VHDL and synthesized it using the Synopsys Design Compiler with the 65nm TSMC technology library [9]. The required area is 826 μm^2. More precisely, the controller increases the area of `IntMap` by only 1%. From the power perspective, the controller leads to an increase of the dynamic power and leakage of `IntMap` by 4.6% and 0.02% respectively.

On average, the incurred performance penalty is 1.4% with a maximum overhead of 6%.

4.7. Summary

Dynamic thermal management that targets optimizing temperature instead of performance under thermal constraints had a higher potential for reducing temperatures and thermal gradients. This results in increased dependability. However, temperature-optimizing DTM is only feasible when the incurred performance overhead is constrained. This limits its applicability.

Load balancing through activity migration is a powerful means to optimize the thermal behavior of on-chip systems. COOL utilizes the concept of *extremum-seeking* control and applies it to dynamic thermal management for components of an on-chip microarchitecture. Thermal imaging analysis shows a reduction of peak temperatures by $9°C$ and a reduction of thermal spatial variation from $6°C$ to $1°C$ in an FPGA chip, and shows that the activity migration rates derived from the controller are optimal with respect to minimizing peak temperature and thermal spatial variation. When applying our approach to the register file of a superscalar ASIC microarchitecture, we are able to achieve a reduction of peak temperatures compared to a State-of-the-Art approach for register file thermal management (with an average peak temperature reduction of $13°C$) while at the same time reducing thermal variation by 49% (which increases the lifetime of a system) at a performance loss of only 1.4%.

While the presented approach lends itself to temperature optimization in microarchitectural components, the performance overhead of task migration limits its applicability to multi-core architectures. These will require more lightweight DTM solutions which scale to large system sizes. The problem of multi-core DTM will be addressed in the next chapter.

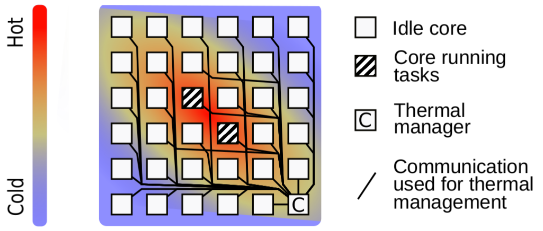
5. Proactive Techniques for Multi-Core Systems

5.1. Thermal Management and Multiple Cores

As discussed in the previous chapter, thermal management fundamentally consists of influencing the power distribution over time and area in order to reduce power densities. Through their increased number of components, multi-core systems offer a great deal of freedom for spreading computation over larger areas. In particular, this is accomplished due to the possibility of *task migration* between cores.

5.2. On the Importance of Scalability

Given the growing number of cores present in today's multi-core architectures, with numbers approaching the thousands [26], the problem of distributing computation over a chip becomes increasingly complex. Task mapping itself is a NP-hard problem [49], which is compounded by requiring runtime re-mapping to address dynamic thermal management over time.



Centralized management scheme. Manager forms bottleneck for communication as well as computation.

Figure 5.1.: Central management and scalability

Considering the central approach in Figure 5.1, we see that all cores must communicate with the central manager, resulting in a high communication volume and a communication bottleneck. While the central approach can potentially achieve the best thermal management, as it retains global knowledge, it must be considered that finding an optimal task mapping is an NP-hard problem and thus it grows considerably in complexity with the number of cores.

5.2.1. Related Work: Distributed Decision Making

When dealing with the growing number of cores in a multi-core architecture, it becomes necessary to make decisions –e.g. task mapping, resource allocation, etc. – in a distributed fashion in order for these to be able to scale with increasing system size. To this effect, local entities, hereafter referred to as *agents* [68], calculate these decisions for a designated region (i.e. a core or cluster of cores) while organizing themselves with other agents. In order to maximize scalability, a completely decentralized approach can be used [43]. This approach greatly reduces computational complexity. However if system knowledge needs to be propagated through the system, this approach may even lead to even more communicational overhead than the centralized approach.

The work done in [101, 49], have presented agent-based approaches for task mapping in 2D multi-core architectures. Both these approaches use clustering techniques to limit the mapping problem space. The agents map tasks inside of the resulting clusters based on a heuristic. The focus in both lies in optimizing communication distances and neither thermal effects nor communication in 3D architectures are considered.

Additionally to these static approaches, there has also been work done in incorporating learning techniques in order to improve decision making. In [88], Martinez et al. present a resource management for multi-core architectures which uses a machine learning mechanism. These include using reinforcement learning and artificial neural networks (ANN) to make decisions for DRAM scheduling whereas only the reinforcement learning is done by "intelligent agents" with the ANN requiring a central instance. Furthermore, the agents act independently from each other with each agent only managing its own resource.

Intelligent agents have also been deployed in a more cooperative manner in the balancing of resources in grid computing [28, 115]. Of these, [115] presents an overview of various negotiation models between the agents, such as an auction model and a game theory based model whereas [28] presents an approach using a genetic algorithm and quantifies the results. These show that the agent approach is highly scalable in the grid context and achieves comparable performance to a centralized approach in terms of application execution times.

5.2.2. Agents and Agent Systems

In order to deal with this growing complexity needed to manage multi-core architectures, new approaches have been proposed for system design. With the autonomic computing initiative introduced by IBM [62], there has been a consent that future systems will need to be *self-configuring* and *self-optimizing.* That is, system components must configure themselves and optimize their resource usage independently. Therefore, the *self-x* properties that are employed in the system may facilitate *DTM* in a pro-active manner. Agent-based systems are one of the most promising ways of realizing these properties. The power distribution approaches presented in this chapter, [43, 45], are among the first to apply agent-based systems to achieve runtime DTM. They are inspired by the techniques deployed in *agent-based computational economics* [117].

Since the term agent, is not well-defined, we instead formulate their required properties. As such, an agent system is composed of agents with the following inherent properties:

Scalable As mentioned above, scalability is the main goal which the agent system sets out to attain. This is not directly a property of individual agents, but instead one of the overall agent system. However, it does set limitations on individual agent behavior. As a result, the following properties need to adhere to specified constraints.

Situated Agents possess either a physical or logical area for which they are responsible. The agent acts on behalf of this area. Examples include one agent per core, one agent per cluster of cores, or one agent per microarchitectural component (as discussed in the previous chapter).

Social Agents must be able to communicate between each other to disseminate system knowledge and information on their own actions.

Proactive There is always some latency associated with the communication required to for implementing agents' social behavior, especially when multiple agents are involved. It is not beneficial for agents to only act once a problem has been identified (e.g. elevated temperatures or advanced aging).

Reactive While actions performed are ideally proactive, agents are still required to react to outside stimuli. These are quantified by sensors which supply the agents with data. Since proactivity is also a desired property, the agent system does not activate as a result of these stimuli, but is instead always active and only adapts its proactive strategy based on an analysis of the supplied data. Additionally, each agent must also react to information it receives from other agents with which it communicates.

Light-weight Agents must exhibit small computational and memory footprints. This is in part to facilitate the scalability of the agent system, but also to minimize the intrusiveness of agents implemented in software that compete with tasks for resources.

For achieving scalable management, the topology of the agent system can be a limiting factor. Topologies with communication and computational bottlenecks (e.g. star topologies) should be avoided. At the same time communication paths should be kept low. Of course, these two goals can be contradictory. There is always a trade-off between scalability and dissemination or aggregation of system knowledge that facilitates thermal management.

5.3. TAPE - Thermal-aware Agent-based Power Economy

This section presents a highly scalable distributed agent-based power distribution approach, (*TAPE* [43]), which focuses on distributing power and the resulting heat evenly over a multi-core architecture. The approach incorporates thermal penalties in the agent negotiation to deal pro-actively with potentially developing *thermal hotspots* explicitly.

5.4. Problem Formulation

Given a multi-core architecture M made up of tiles $n \in M$, we define a region R_j as the area covered by a rectangular connected subset of M (i.e. groups of adjacent tiles forming rectangles), then, using the principles described in Chapter 2, temperature $T(t_1, r)$ for $r \in R_j$ at time t_1 can be expressed as:

$$T(t_1, r) = kh \int_0^{t_1} P(r, t)dt - \int_0^{t_1} \dot{Q}_{R_j}(t)dt + T(0, r) \tag{5.1}$$

where $P(r, t)$ is the power used at r at time t, $\dot{Q}_{R_j}$ is the heat transfer rate to and from R_j, h is the thickness of the silicon layer, $T(0, r)$ is the temperature at time 0, and k is a unit constant. We aim to minimize the temperature of a region by avoiding power being concentrated in smaller regions,

$$\forall j, k\colon\ |R_j| < |R_k| \Rightarrow \int_{R_j} P(r,t)dr < \int_{R_k} P(r,t)dr \tag{5.2}$$

and by spreading power equally over regions of the same area A, that is *minimizing* $\text{MAX}_{R_i} T(t,r)$.

$$\forall R_i\colon\ |R_i| = A\colon\ \text{MAX}_{R_i} T(t,r) = \text{MAX}_{R_i} \int_{R_i} P(r,t)dr \tag{5.3}$$

5.5. System Models

In this section we present the models of different parts for our complete multi-core architecture where we have used our proposed agent-based runtime *DTM* scheme. We assume our multi-core architecture is made up of tiles consisting mainly of a PE, a communication element, an agent for power negotiation, and a power management unit (see Fig. 5.2). A **task** is considered to be a computational entity running on the PE of a tile. There can be multiple tasks running simultaneously on each PE.

- Each task (task$_i$) has a gate parameter $\alpha_{z,i} = \text{AVG}(\frac{g_{z,i}}{c_{z,i} \cdot g_{z,tot}})$ for each PE type PE$_z$ (i.e. considering heterogeneous PEs), where $g_{z,i}$ are the number of gates switched when running task$_i$ on PE$_z$, $c_{z,i}$ are the cycles it takes to run task$_i$ on PE$_z$, and $g_{z,tot}$ is the total amount of gates in PE$_z$. It specifies the percentage of gate switches per cycle required on an average when running task$_i$ on a PE$_z$.
- Furthermore, it is assumed that each task has a deadline D_{task_i} as well as a limit for the execution time measured in cycles: $0 < c_{z,i} \leq \text{MAX}(c_{z,i})$.

5.5.1. Determining Power Budgets

For each PE inside a tile, the given power budget limits the processing that can be accomplished. This power budget is measured in the number of *power units* available in a tile which define the amount of power the tile is allowed to consume. Using the simplified power formula for CMOS gates: $P = k_1 \alpha_{z,i} C_{\text{PE}_z} V^2 f + k_2 V I_{\text{leak}}$, we may observe the influence of the power budget (k_1 and k_2 are constant). Since I_{leak} and C_{PE_z} are assumed to be constant for PE$_z$, the parameters that the agents can change are limited to V and f. These two parameters are roughly proportional to each other and there is a maximum f for a given V. Furthermore, in our particular model it is assumed that V is limited to n discrete values, while f can have any value achievable by a *Digital Clock Manager* (DCM).

For a PE to be able to run a task, it must be running at a minimum frequency of f_{min} so that $D_{\text{task}_i} \geq f_{\text{min}}^{-1} \cdot \text{MAX}(c_{z,i})$. Since a given frequency has a minimum required voltage (frequency and voltage being roughly linearly proportional), the voltage is set to the lowest voltage above the minimum available in the tile's power profile. Thus, **a power unit is defined by the frequency it is able to achieve on** PE$_z$. The characteristics of our power budgeting are explained below.

- The **power unit granularity** gr_z is defined by the amount of power units needed to run at a given frequency f_0. If coarse-grained power units are used, i.e. when one power unit is able to achieve a higher frequency relative to f_0, less power units are required to run at f_0 (making trading, see below, less complex) but may allocate more power than is actually needed. On the other hand, using fine-grained power units, i.e. those only able to achieve low frequencies, allows more precise power allocation but results in more complexity by requiring more trading. A measure for the power unit granularity gr_z for PE$_z$ can be given as:

$$gr_z = \frac{\sum_{\mathrm{i}} \text{power units } \textit{per} \text{ task}_i}{\# \text{ of tasks}} \tag{5.4}$$

We then use gr_z to define the relative PE factor ξ_z which describes the power unit utilization of PE_z relative to other PEs (PE$_{z'}$, $z \neq z'$). This value is used later to calculate the *buy* and *sell* values (see Section 5.6.2)), and allows our approach to be used for heterogeneous multi-core architectures. Assuming there are N total PEs, ξ_z is defined as follows:

$$\xi_z = \frac{(N-1) \cdot gr_z}{\sum_{z'} gr_{z'}} \tag{5.5}$$

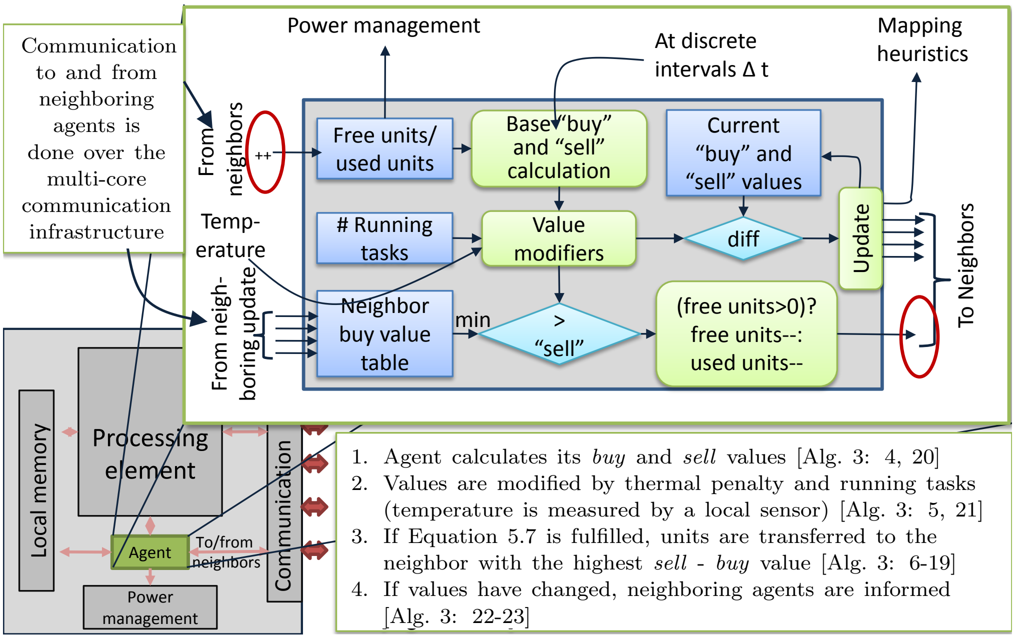
- The **task parameter** $\alpha_{z,i}$ can reduce the number of power units required to achieve a given frequency f_0. To be able to determine the average $\alpha_{z,i}$, first the worst case number of power units are allocated ($\alpha_{z,i} = 1$). The power usage is then measured and averaged over time.
- The **longest execution time measured** $\textsc{max}(c_{z,i})$ along with its **deadline** D_{task_i} determine the required frequency f_0. The longest execution time measured has no worst case scenario known *a priori*. For each missed task deadline, the exceeding time must be added to the current longest execution time measured. The frequency must then be changed accordingly.

5.6. Agent-based Power Economy

Before incorporating the models presented in the previous section into our approach, we first introduce our novel agent-based power economy for managing a runtime *DTM*, corresponding to the definition given in Section 5.2.2. The definition of an agent motivated by [117] is given below:

Definition of TAPE Agents:

An agent in TAPE is a *situated* computational entity that acts *autonomously* and *flexibly* on the behalf of others. For our approach it is necessary to situate our agents at the same level as at which power management is done. Since TAPE performs power management and runtime *DTM* at tile level, each tile also has a single agent. Agents act autonomously in order to accomplish a common goal (in our case evenly distributing power) and are not dependent on other agents (although they react to the actions of each other). In particular, there is no higher instance controlling the agents. An agent is considered flexible when it is *responsive* (i.e. responds to input from other agents), *proactive*, and *social* (i.e. communicates with other agents). In a nutshell, TAPE agents implement these properties with each of them acting on behalf of one tile. Plus, an agent trades power units with the agents of adjacent tiles using the agent negotiation presented below.



1. Agent calculates its *buy* and *sell* values [Alg. 3: 4, 20]
2. Values are modified by thermal penalty and running tasks (temperature is measured by a local sensor) [Alg. 3: 5, 21]
3. If Equation 5.7 is fulfilled, units are transferred to the neighbor with the highest *sell* - *buy* value [Alg. 3: 6-19]
4. If values have changed, neighboring agents are informed [Alg. 3: 22-23]

Figure 5.2.: Functionality of an agent during negotiation

Benefits of using Agents:

Agents exhibiting the above properties may be able to individually distribute power units between tiles of a multi-core architecture through local trade, i.e. only with their neighbors. They may exhibit an emergent behavior, i.e. many agents individually following simple rules are able to accomplish the power distribution when working together on a local basis only. As the hardware and software overhead of individual agents does not grow with system size (unlike the *CI* used in [113, 35, 124, 57]), agent-based approaches are highly scalable for next generation thousand-core chips [26]. The proactive characteristics of agents allow our *DTM* to proactively minimize the possibility of *thermal hotspots* during execution time compared to [113, 35, 124, 57]. To the best of our knowledge, current usage of agents in multi-core architectures so far has been limited to task mapping [49].

5.6.1. Agent Negotiation

Our agents are able to negotiate with their immediate neighbors in all directions (e.g. 4 neighbors in the North, South, East, and West directions for a multi-core architecture having a regular 2D Mesh topology) to optimize power budgets.

Initially, a predefined number of power units are distributed evenly (depending on the PE factor ξ_z in heterogeneous multi-core architectures) among all agents. Each of these power units belong to one of the following two categories: • *used* units are currently deployed by the PE to run tasks, i.e. *used* units are used to set the tile's frequency and voltage as specified above (Section 5.5.1), and • *free* units are power units available on a tile, but not currently needed by the tile to run its allocated tasks since the required frequency and voltage are already set by *used* units. These units are used by each agent to calculate *buy* and *sell* values for trading power units. These *buy* and *sell* values are not constant but vary depending on the *supply/demand* of power units calculated by each agent. The underlying main concept is that it becomes increasingly difficult to obtain power units if there are already many power units being used in a local area (high demand) even though there is no defined upper bound, as agents at the edge of a local area will trade with immediate neighboring agents outside this local area. This whole model may be compared to the classical ***supply/demand*** **model of computational economics**.

An overview of the agent negotiation is given in Fig. 5.2,5.3, and in Alg. 3. At each time interval (see Section 5.7 for details of the time interval), the agent first calculates its base *buy* and *sell* values. The calculation of these values at agent n is given as follows (Alg. 3, L. 4-5):

$$\begin{aligned} \text{sell}: \; & sell_n = w_{u,s} \cdot used_n + w_{f,s} \cdot free_n \\ \text{buy}: \; & buy_n = w_{u,b} \cdot used_n - w_{f,b} \cdot free_n + \gamma \end{aligned} \tag{5.6}$$

where, $free_n$ and $used_n$ are the number of *free* and *used* power units respectively, $w_{i,j}$ are weight parameters, and γ is a normalized offset parameter. $free_n$ may also be negative if new power units are required to execute a new task (see Section 5.6.3).

Once the base *buy* and *sell* values have been determined, these are then decreased based on the current temperature of the tile and increased based on the currently executing tasks. The final values are given as follows:

- *Buy* modifier:
  buy_n= $buy_n - a_b \cdot T_n$ (Alg. 3, L. 6)
  Here, a_b is the normalizing factor showing the relationship between the temperature and the power units. The modifier of the *buy* value ensures that it becomes increasingly difficult for an agent whose tile has a high temperature to acquire additional power units. The modifier is only applied once the temperature reaches a given minimum temperature T_0 (a pro-active behavior) above which the modifier is to be applied as T_n is measured in degrees above T_0 and is referred to as the *thermal penalty.*

- *Sell* modifier:
  $sell_n$= $sell_n + \sum_{\text{task}_i} p_i - a_s \cdot T_n$ (Alg. 3, L. 7)
  The *sell* modifier at the same time decreases the *sell* value when temperatures increases (*thermal penalty*) making it more likely that neighboring agents buy power units (again, the *thermal penalty* is only applied once the tile temperature surpasses the given minimum temperature T_0). At the same time, it also considers the running tasks task_i (with the weights p_i dependent on the task execution time characteristics i.e. hard real-time constraints). This ensures that power units are first traded by tiles running fewer tasks or tasks with no hard real-time constraints.

It should be noted that afore mentioned thermal penalties are specific to temperature reduction. This may not always be the only thermal goal. It may happen that a minimum temperature is also desired in order to keep the system in an optimal operating range and to further reduce thermal cycling (i.e. in addition to balancing tasks). Fortunately, the thermal penalty can trivially be extended to the general case by applying an additional thermal penalty proportional to the temperature difference below a corresponding threshold.

The neighbors of a specific tile n, i.e. all tiles adjacent to n, are informed when its modified *buy/sell* values have changed over time. The neighboring tiles then store the updated

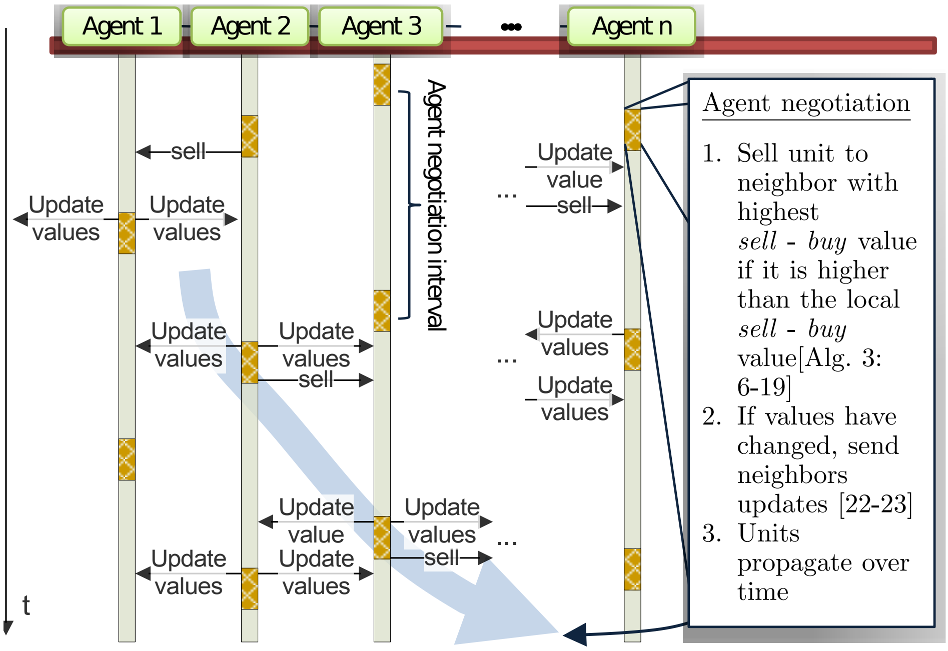

Agent 1
Agent 2
Agent 3
•••
Agent n
t
sell
Update values
Update values
Agent negotiation interval
Update value
sell
Update values
Update values
sell
Update values
Update values
Update value
Update values
sell
Update values
Update values
...
Agent negotiation
1. Sell unit to neighbor with highest *sell* - *buy* value if it is higher than the local *sell* - *buy* value[Alg. 3: 6-19]
2. If values have changed, send neighbors updates [22-23]
3. Units propagate over time

Figure 5.3.: Sequence of the agent negotiation (for simplicity only 1 dimension is shown)

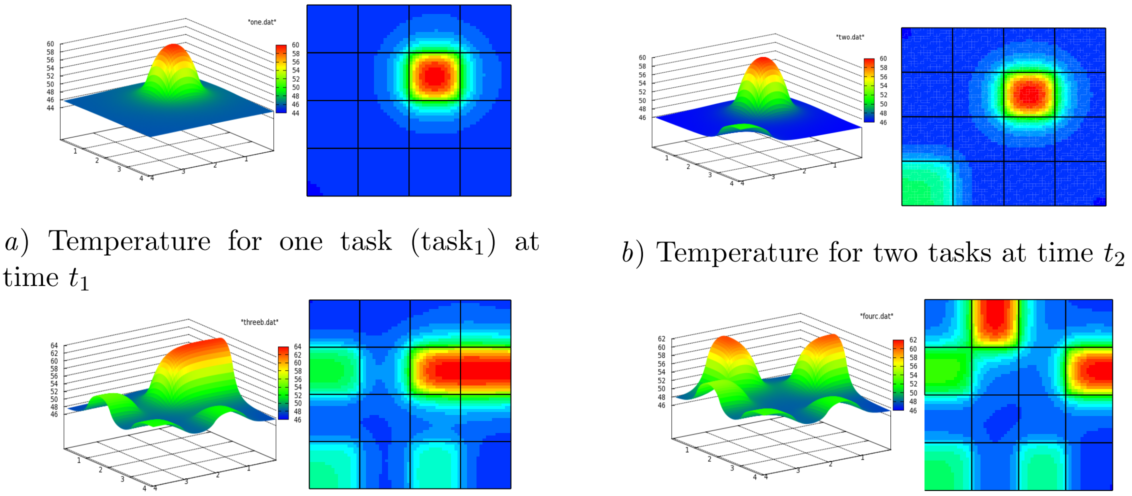
buy/sell values locally in order to minimize redundant communication. At the next trade time (i.e. after waiting the rest of their given interval time, see Section 5.7), the agents within the tiles compare their own resulting modified *buy/sell* values with the *buy/sell* values they have obtained from their neighbors ($i \in N$) to establish where more power units are needed and where there are “too many” power units being used, i.e. if the peak temperature has crossed a threshold or one PE is consuming a considerable amount of power while others are idle, using the following equation (Alg. 3, L. 8):

$$(sell_n - buy_n) - (sell_i - buy_i) > \tau_n \tag{5.7}$$

where τ_n is the sell threshold of tile n, with $\tau_n \approx w_{f,s}$. This threshold is to ensure that there is no redundant trading, i.e. two agents continuously trading one *free* unit back and forth (a ping-pong effect in trading).

If any of the neighbors $i \in N_n$ fulfill Eq. 5.7, agent n relinquishes a power unit to the neighbor with the maximum $(sell_i - buy_i)$ value. If the local agents have *free* power units then this is simply a matter of decrementing its number of *free* power units. However, if there are no *free* power units, then the local agent must give up one of its *used* power units. This means the agent must also reduce its power consumption through *DVFS* (Alg. 3, L. 12-13). If as a result of *DVFS* tasks will be unable to meet their deadlines then they must be (re-)mapped (Alg. 3, L. 15). This procedure is repeated until the system reaches a stable state, i.e. when the left side of Eq. 5.7 is less than τ_n in all tiles.

In this work we have assumed that the global amount of power units is a fixed number given at design time. Additional power scaling may be done by removing or adding power units to the multi-core architecture at runtime. However this is beyond the scope of this work.



a) Temperature for one task (task$_1$) at time t_1

b) Temperature for two tasks at time t_2

c) Five tasks at time t_i

d) Task$_1$ (re-)mapped at time t_{i+1}

$$\underbrace{\underset{used}{\begin{pmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{pmatrix}}\;\underset{buy}{\begin{pmatrix}0.5&\cdots&0.5\\\vdots&\ddots&\vdots\\0.5&\cdots&0.5\end{pmatrix}}\;\underset{free}{\begin{pmatrix}3&3&3&3\\3&3&3&3\\3&3&3&3\\3&3&3&3\end{pmatrix}}\;\underset{sell}{\begin{pmatrix}0.6&\cdots&0.6\\\vdots&\ddots&\vdots\\0.6&\cdots&0.6\end{pmatrix}}}_{t_0}$$

$$\underbrace{\underset{used}{\begin{pmatrix}0&0&0&0\\0&0&2&0\\0&0&0&0\\0&0&0&0\end{pmatrix}}\;\underset{buy}{\begin{pmatrix}0.5&0.5&0.6&0.5\\0.5&0.6&2.4&0.5\\0.5&0.5&0.6&0.5\\0.5&0.5&0.5&0.5\end{pmatrix}}\;\underset{free}{\begin{pmatrix}3&3&2&3\\3&2&5&2\\3&3&2&3\\3&3&3&3\end{pmatrix}}\;\underset{sell}{\begin{pmatrix}0.6&0.6&0.4&0.6\\0.6&0.4&2.2&0.4\\0.6&0.6&0.4&0.6\\0.6&0.6&0.6&0.6\end{pmatrix}}}_{t_1}$$

$$\underbrace{\underset{used}{\begin{pmatrix}0&0&0&0\\0&0&2&0\\0&0&0&0\\1&0&0&0\end{pmatrix}}\;\underset{buy}{\begin{pmatrix}0.5&0.5&0.6&0.5\\0.5&0.6&0.3&0.5\\0.6&0.5&0.6&0.5\\1.2&0.6&0.5&0.5\end{pmatrix}}\;\underset{free}{\begin{pmatrix}3&3&2&3\\3&2&5&2\\2&3&2&3\\4&2&3&3\end{pmatrix}}\;\underset{sell}{\begin{pmatrix}0.6&0.6&0.4&0.6\\0.6&0.4&2.2&0.4\\0.4&0.6&0.4&0.6\\1.4&0.4&0.6&0.6\end{pmatrix}}}_{t_2}\cdots$$

time

$$\cdots\underbrace{\underset{used}{\begin{pmatrix}0&0&0&0\\1&0&2&2\\0&0&0&0\\1&0&1&0\end{pmatrix}}\;\underset{buy}{\begin{pmatrix}0.6&0.5&0.6&0.6\\1.2&0.7&-1.5&2.5\\0.6&0.5&0.6&0.7\\1.2&0.6&1.3&0.6\end{pmatrix}}\;\underset{free}{\begin{pmatrix}2&3&2&2\\4&1&5&4\\2&3&2&1\\4&2&3&2\end{pmatrix}}\;\underset{sell}{\begin{pmatrix}0.4&0.6&0.4&0.4\\1.4&0.3&-1.1&1.9\\0.4&0.6&0.4&0.3\\1.4&0.4&1.3&0.4\end{pmatrix}}}_{t_i}$$

$$\underbrace{\underset{used}{\begin{pmatrix}0&2&0&0\\1&0&0&2\\0&0&0&0\\1&0&1&0\end{pmatrix}}\;\underset{buy}{\begin{pmatrix}0.7&2.5&0.6&0.6\\1.2&0.7&0.7&2.4\\0.6&0.5&0.6&0.6\\1.2&0.6&1.2&0.6\end{pmatrix}}\;\underset{free}{\begin{pmatrix}1&4&2&2\\4&2&2&5\\2&3&2&2\\4&2&4&2\end{pmatrix}}\;\underset{sell}{\begin{pmatrix}0.3&1.9&0.4&0.4\\1.2&0.4&0.4&2.0\\0.4&0.6&0.4&0.4\\1.2&0.4&1.2&0.4\end{pmatrix}}}_{t_{i+1}}\longrightarrow$$

Figure 5.4.: An Exemplary Runtime Scenario explaining TAPE's Functionality (Section 5.6.4)

5.6.2. Weights and Buy/Sell Value Characteristics

An important factor in ensuring that the system reaches a stable state is the choice of weights used to determine the *buy* and *sell* values. These weights are constant over time. Different tiles have different weights depending on their PE type (a heterogeneous multicore architecture is considered). The weights can be seen as the value of a power unit for each PE. When choosing weights, the following factors must be considered:

- The number of *used* units has a greater influence in the *buy/sell* value than the *free* units as these are the actual cause of supply/demand in a local area. In an abstract sense, the *used* units can be seen as generating "value" which the agents use to acquire *free* units. The weights for *used* units are proportional to the power

Algorithm 3 Agent-based power economy for *DTM*

```
Input: $sell_{base_n}$: Base sell value of a tile $n$ at time $t$
Input: $buy_{base_n}$: Base buy value of a tile $n$ at time $t$
Input: $T_n$: Temperature of a tile $n$ at time $t$
Input: $sell_{T_n}$: Sell value of a tile $n$ at a temperature $T_n$
Input: $buy_{T_n}$: Buy value of a tile $n$ at a temperature $T_n$
Input: $N_n$: Set of all the neighboring tiles of tile $n$
Input: $lastbuy$, $lastsell$: Last buy/sell values of a tile $n$ sent to all $i \in N$
Input: $buy[N]$, $sell[N]$: List of buy/sell values of neighboring tiles stored in $n$
Input: $free_n$: Free power units of tile $n$
Input: $used_n$: Power units used for running tasks on tile $n$
Input: $t_j$: Tasks running on tile $n$ at time $t$
Input: $\tau_n$: sell threshold of tile $n$
begin
    forall the tiles $n$ in parallel do
        foreach time interval $\Delta_n t$ do
            // Calculate base sell/buy values
            $sell_{base_n} \leftarrow (w_{u,s} \cdot used_n + w_{f,s} \cdot free_n)$   $buy_{base_n} \leftarrow (w_{u,b} \cdot used_n - w_{f,b} \cdot free_n + \gamma)$  // The temperature increase may happen due to change in PE activity. Modify buy/sell value
            $sell_{T_n} \leftarrow sell_{base_n}$ - $a_s \cdot T_n$   $buy_{T_n} \leftarrow buy_{base_n}$ - $a_b \cdot T_n + \sum_{t_j} p_j$   if $\exists i \in N_n : ((sell_{T_n} - buy_{T_n}) - (sell[i] - buy[i]) > \tau_n$ then
                if any free power units are left then
                    decrement $free_n$
                else
                    apply DVFS on $n$ to get more free power units   decrement $used_n$   if the task does not meet the given deadline as DVFS is used then
                        (re-)mapping needs to be invoked
                    end
                    graceful performance degradation if allowed
                end
                increment $free_i$
            end
            if $buy_{T_n} \neq lastbuy$ or $sell_{T_n} \neq lastsell$ then
                send $buy_{T_n}$ to all $i \in N$   send $sell_{T_n}$ to all $i \in N$   $lastbuy \leftarrow buy_{T_n}$   $lastsell \leftarrow sell_{T_n}$
            end
            // This procedure will propagate until a stable state is reached.
            if received updated buy/sell values from any $l \in N_n$ then
                update $buy[l]$, $sell[l]$
            end
            if new task mapped to $n$ requiring $k$ power units then
                $free_n \leftarrow free_n - k$   apply DVFS to PE on tile $n$   $used_n \leftarrow used_n + k$
            end
        end
    end
end
```

unit granularity gr_z and consequently also to ξ_z to allow heterogeneous multi-core architectures (Eq. 5.4).

$$w_{u,s}, w_{u,b} \propto gr_z, \xi_z \tag{5.8}$$

- The number of *free* units however must also influence *buy/sell* values in order to prevent one agent from acquiring all *free* units from its neighbors. The weight parameters of *free* units are proportional to the total amount of units (i.e. the initial an maximum number of *free* units), and also to gr_z (and ξ_z).

$$\begin{aligned} w_{f,s}, w_{f,b} &\propto \sum(used + free) \\ w_{f,s}, w_{f,b} &\propto gr_z, \xi_z \end{aligned} \tag{5.9}$$

$$\gamma = k \cdot \xi_z \cdot \frac{\sum(used + free)}{m} \tag{5.10}$$

where m is the number of total tiles in the multi-core architecture M.

From these properties we get the following equations for the weights

$$w_{u,b} = a \cdot \gamma, \qquad w_{u,s} = b \cdot \gamma,$$
$$w_{f,b} = c \cdot \tfrac{\gamma}{k}, \qquad w_{f,s} = d \cdot \tfrac{\gamma}{k}$$

To normalize these weights we define $\gamma := 1$, which also gives us k, and choose $a \in [1, 2)$, $b, c, d \in (0, 1)$ with $a > b, c < d$.

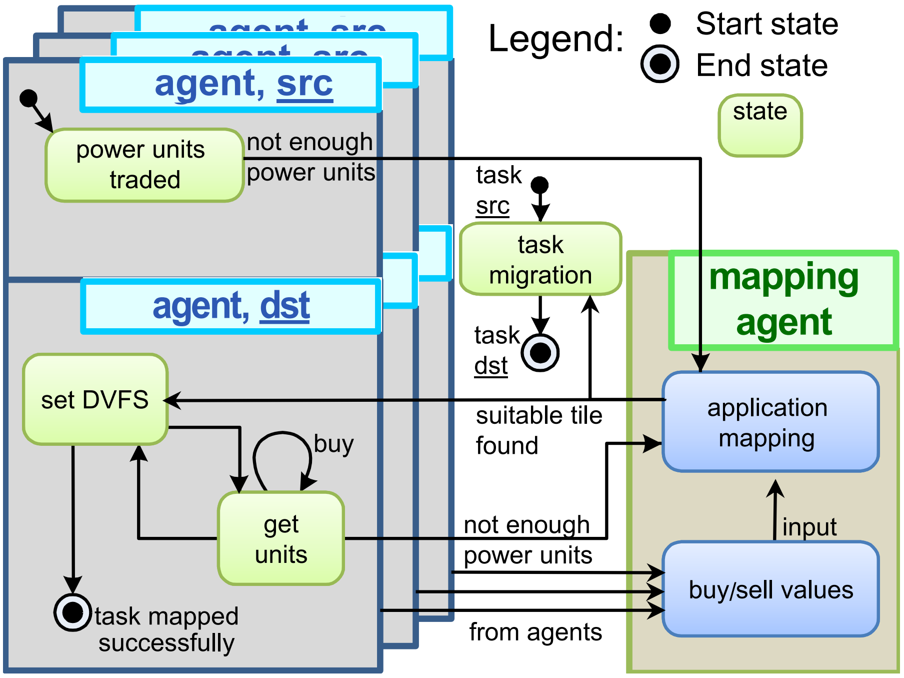


Figure 5.5.: Runtime application (re-)mapping overview

5.6.3. Runtime Application (Re-)Mapping

An agent-based runtime application (re-)mapping algorithm that is invoked due to power trading in the agents is integrated in our TAPE approach. Our (re-)mapping algorithm is based on the one presented in [49]. The goal of runtime application (re-)mapping is to determine which tile is best suited to migrate a given task to. The mapping agents are realized separately from the agents for power trading and are implemented in software running on one or more tiles (the implementation of the power trading agents is discussed in Section 5.7). The software entity that is responsible for our application (re-)mapping may be executed in any of the existing PEs inside the multi-core architecture. An overview of the interaction between the power trading agents and the mapping agents is presented in Figure 5.5. Depending on the situations during *DTM* the following scenarios may happen that require application (re-)mapping:

- If a new task needs to be mapped to a tile, it is mapped to one in which the most power units are available (either locally or through trade using agent-based negotiation). In our economic approach this is simply the tile where the difference between its *sell* value and its *buy* value is maximal: $\text{MAX}_n(sell_n - buy_n)$ (if more than one tile fulfills this criterion, one is chosen randomly among the ones with the lowest absolute *buy* value). Once a tile is chosen, that particular tile decrements its *free* power units and increments its *used* power units. If not enough *free* power units are available, the '*free*' count becomes negative and the agent must acquire new power units from its neighbors.
- If during runtime a tile does not have enough power units for a currently allocated task to meet its deadline (i.e. the number of *used* power units is less than the required number of power units – meaning that the PE is thereby not operating at a high enough frequency to complete the task by its deadline – and there are no *free* power units) then the task needs to be (re-)mapped. A new tile that maximizes the difference between *sell* and *buy* values is chosen to migrate the task to.
- In case of failure to (re-)map a task the system rejects the new task and in case of a currently executing task it uses the concept of graceful performance degradation similar to [49].

An important consideration that comes with task (re-) mapping is how the task migration will be done. In TAPE, we have adopted the approach presented in [49] during *DTM*. It realizes *task-relocation* by shutting down a task running on tile n and transferring its context to the destination tile. Here, the context is then loaded and the task is resumed. In [92] it is shown that the task migration overhead similar to our approach ranges from 10 000 to 10 000 000 cycles depending on the task size.

5.6.4. An Exemplary Runtime Scenario explaining TAPE's Functionality

In order to further illustrate the agent-based power economy, we present an exemplary scenario detailed in Fig. 5.4. The 4 × 4 2D-Mesh architecture can be seen as a part of a larger multi-core architecture. However, to keep the example clear and be able to present all the *buy/sell*, *free/used* values at each time t, only the 4 × 4 tiles are shown. First, we consider a multi-core architecture where all tiles are idle. All agents have only *free* units and all tiles have the same *buy/sell* values (a homogeneous multi-core architecture is considered here). At time $\boldsymbol{t_0}$ with no tasks running on the system, the left side of Eq. 5.7 is zero (i.e. less than τ_n) for all agent pairs, resulting in a stable state. Now we assume at time $\boldsymbol{t_1}$ task$_1$ needs to be mapped onto the system. At this point, from a thermal and power perspective, all tiles are equally suited to run task$_1$. The (re-)mapping algorithm randomly maps the task to tile n (Fig. 5.4a).[1] Tile n then converts the number of power units needed to run task$_1$ from *free* to *used.* This results in both an increase in *buy* and

[1] These example figures include power traces from two applications (gcc and FFT, using the *wattch* simulator), the thermal values are calculated using the *Hotspot thermal simulator* [63], and the temperature T_0 after which the thermal penalty is applied is assumed to be 62 °C.

sell values and a decrease in the difference ($sell_n - buy_n$). Assuming that the difference of n's *buy/sell* values ($sell_n - buy_n$) is now more than τ_n less than ($sell_{N_n} - buy_{N_n}$) of any of its neighbors N_n, n buys *free* power units from $l \in N_n$ in the next trade intervals of the neighbors until Eq. 5.7 again holds. At the same time, any l that has given up power units will have increased *buy* (and *sell*) values. These l then in turn trade with their other neighbors. The trading thus propagates throughout the chip until once again a stable state is reached. Now, assuming a second task, task$_2$ needs to be mapped on the chip, it will be mapped to a tile away from n at time $\boldsymbol{t_2}$ (Fig. 5.4b).

Next, we will assume that at a future time $\boldsymbol{t_i}$, several tasks are running on the system and that, unfortunately, there is a potential *thermal hotspot* forming at tile n (seen in Fig. 5.4c, the tile temperature exceeds 62 °C). The rise in temperature will cause buy_n as well as $sell_n$ to decrease (shown in red in the timeline). Once the temperature has become high enough to decrease $sell_n$ to the point where the left side of Eq. 5.7 is larger than τ_n for one of the l, n will be forced to give up a power unit. Assuming n has no *free* power units, n must give up a *used* unit. We assume that n is now not able to accommodate task$_1$. Thus the mapping agent is informed and task$_1$ is (re-)mapped (Fig. 5.4d) at time $\boldsymbol{t_{i+1}}$. The remaining *used* units are converted into *free* ones.

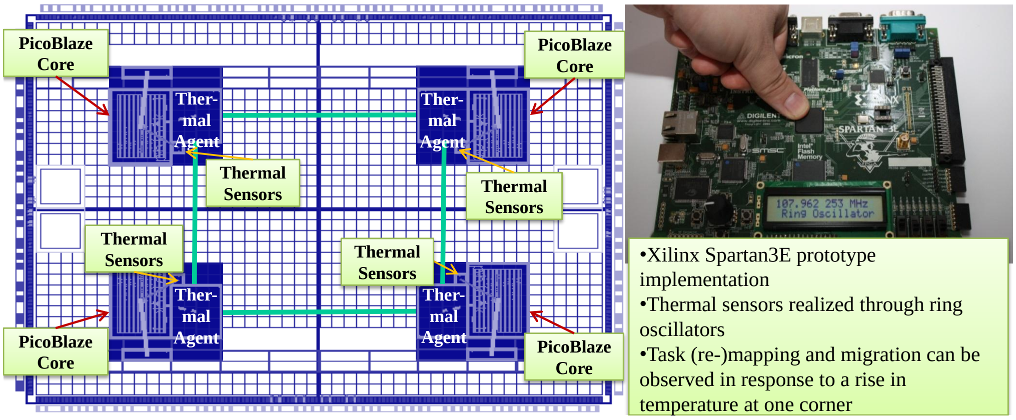
5.7. Implementation Details and Hardware Prototype

The agents store information (e.g. *buy/sell* values of neighboring agents, the current *free* and *used* units, and tasks currently allocated to the tile) in local memory. They receive thermal information from a thermal sensor also located in the tile. Each tile has one sensor as one thermal value is needed for the *thermal penalty* in the agent negotiation and the area requirement is marginal (see hardware prototype below). Neighboring agents communicate (i.e. trade negotiation) with each other through the multi-core architecture's communication infrastructure. Agents can be implemented either in hard- or software. Both have their advantages and disadvantages. However, software agents may not be available in all tile types, e.g. for a dedicated DSP or ASIC.

- **Hardware Agents** When not using a general purpose PE, it is necessary to implement agents in hardware. Hardware agents require additional on-chip area. Our hardware agent prototype takes up 143 slices of a Xilinx FPGA (i.e. less than 1% of the slices of a Virtex-4). Hardware agents do not interfere with tasks running on the tile's PE.

- **Software Agents** are realized as non-transferable tasks running on the PE of a tile. Therefore, they take power and processing time away from the power units allocated to the tile to run its other tasks. Software agents can also serve as a fallback for hardware agents, that is if a hardware agent were to fail, its functionality may be replaced through a software agent. If this is not possible (i.e. software agents are not available) the power manager may simply choose to use a predetermined voltage/frequency setting, thus the agents do not present a central point-of-failure in a tile.

The frequency with which agent negotiation occurs, i.e. due to the time interval between agent activity, is based on the maximum rate of temperature increase: it takes at least 100k cycles to raise the temperature by 0.1°C [113]. We choose the negotiation interval to have a maximum increase of 0.1°C – 1.0°C, which is enough to detect significant increases in temperature. Even when choosing a small interval (e.g. every 100 μs), the power consumed by the agents is still negligible compared to the rest of the architecture. It should also be noted that *buy/sell* values in general do not change at each interval and thus agents do not have to communicate at each interval, thus further lowering the overhead.



- Xilinx Spartan3E prototype implementation
- Thermal sensors realized through ring oscillators
- Task (re-)mapping and migration can be observed in response to a rise in temperature at one corner

Figure 5.6.: Hardware implementation (PlanAhead)

To test the effectiveness of our agent-based power economy in avoiding *thermal hotspots* we have also implemented a small-scale version of TAPE on a Xilinx Spartan3E FPGA. For convenience, the agents of all tiles in our example implementation are realized in hardware. Each tile also includes its own thermal sensor implemented as a ring oscillator which occupies one CLB (*Configurable Logic Block*) on the FPGA. The layout of the hardware implementation is shown in Fig. 5.6. In this case there are 4 PicoBlaze processors. As these processors do not produce a lot of heat, we have decreased the minimum temperature T_0 (and thereby increase the influence of the temperature on the *buy/sell* values) in order to achieve an effect. We have observed tasks migrating away from a corner when placing a thumb on that corner of the chip to increase the temperature showing that the agent-based power negotiation is successful.

5.8. Experimental Setup and Results

For the thermal model we have used the following values: Each tile occupies a $1mm \times 1mm$ area on the chip. The silicon and copper (heat sink) layers are $0.6mm$ and $1mm$ thick respectively. The specific heat capacities of the two materials are $0.7\,kJkg^{-1}K^{-1}$ and $0.385\,kJkg^{-1}K^{-1}$ (with a material density of $2330\,kgm^{-2}$ for silicon and $8920\,kgm^{-2}$ for copper) and the thermal conductivity $148\,Wm^{-1}K^{-1}$ and $401\,Wm^{-1}K^{-1}$ respectively. The ambient temperature is 45°C. We use the weights $w_{b,u} = 0.8, w_{s,u} = 1$ and $w_{b,f} = 0.1, w_{s,f} = \tau_n = 0.2$ which were determined at design time based on the equations in Section 5.6.2. The *thermal penalty* values are $a_b = 0.1$ and $a_s = 0.2$ and the temperature used (T_n) is the difference between the measured temperature T_m and

Simulation Summary:

Ambient temperature	45°C
Tile size	4.9 mm × 4.9 mm
Die height	150 μm
Conductivity:	
Si	148 W/mK
Cu	401 W/mK

the ambient temperature T_0 ($T_n = T_m - T_0$).
All agents are simulated in software and thus their own power usage is explicitly considered.

To evaluate TAPE, we run our agent-based thermal management approach with diverse applications of the MiBench benchmark suite [53] (e.g. calculating SHA hashes or encoding mp3s with LAME). These results are then compared to the same applications running without thermal management, using the *Heat and Run Thermal Management* (HRTM) technique described in [97], and using the state-of-the-art *Predictive Dynamic Thermal Management* (PDTM) from [124] which, is reported to achieve the highest reduction in peak temperature by using a predictive thermal model to migrate tasks before a thermal threshold is reached to tiles where the predicted temperature is the lowest. The HRTM technique is chosen to serve as a basis for comparison with other state-of-the-art techniques as it is a simple responsive DTM approach – migrating tasks when a threshold temperature is reached. The power traces used as input are generated by *SimpleScalar/Panalyzer 2.0* [18] and the temperature is calculated by the *Hotspot* [64] simulator. For the simulation we have used a 3×3 multi-core architecture and used agents implemented in software. Fig. 5.7 shows that TAPE is successfully able to keep the maximum temperature below a dynamic threshold of 47 °C and thus avoid *thermal hotspots*. It should be noted however that this dynamic threshold is not given but dependent on the agent economy (or effectively on the power distribution on the chip).[2] When running the same scenario without our thermal-aware power economy, we see that the on-chip temperature peaks at 51.47 °C. Our results are similar to other *DTM* methods [124] achieving a 7.7% drop in peak temperature compared to having no *DTM* [49] (e.g. here there is a gain of 1.9% compared to [124]). Therefore, besides being highly scalable without added complexity, the proactive behavior of our *DTM* approach keeps the maximum temperature well below the design-time-determined threshold at runtime compared to other approaches. The peak temperatures for further applications are shown in Fig. 5.8 where our approach achieves a

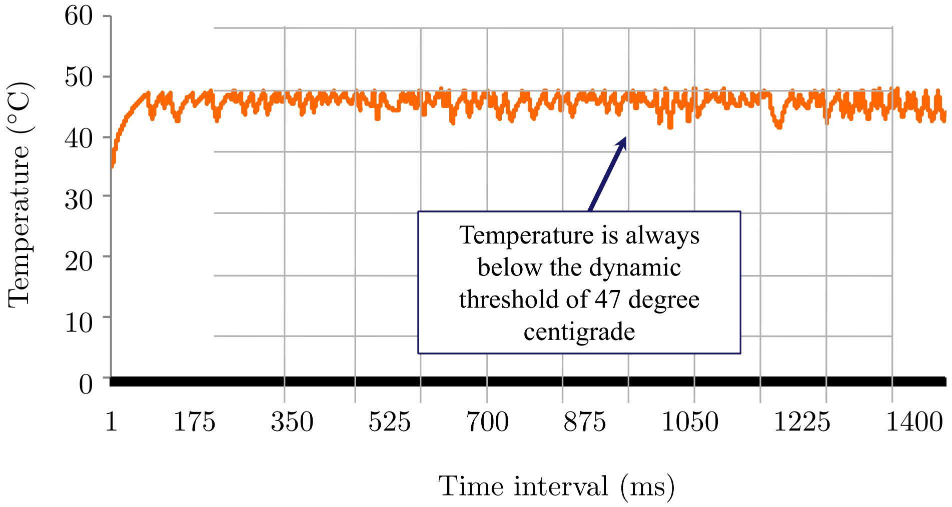


Figure 5.7.: Maximum core temperature in simulation

[2] This is by design, a hard threshold can be implemented by not allowing tasks to be (re-)mapped to tiles whose agents have negative *buy/sell* values.

similar decrease in peak temperature as the HRTM and PDTM approaches (e.g. 54.32 °C for TAPE versus 54.12 °C and 53.89 °C for HRTM and PDTM respectively for the LAME benchmark).

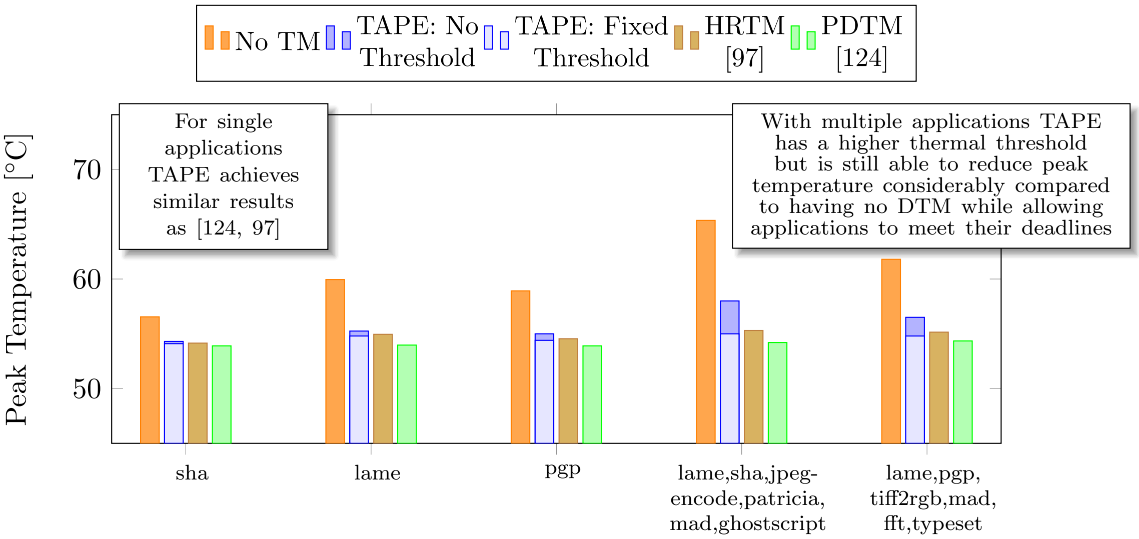


Figure 5.8.: Peak temperature when running single/multiple benchmarks

When multiple applications are running on the multi-core architecture our approach exhibits a higher dynamic thermal threshold as there are more *used* units which raises the average *sell* (and *buy*) values and thereby decreases the impact of the *thermal penalty* (see Fig. 5.8). Here, our approach does not achieve the same reduction of peak temperature as the HRTM/PDTM approaches (TAPE: 58 °C, HRTM: 55.3 °C, PDTM: 54.2 °C) with PDTM achieving 4.65% less peak temperature, but it still reduces the peak temperature by 11.23% compared to having no *DTM* (Fig. 5.8).

The 15.9% increase in execution time compared to having no *DTM* seen in Fig. 5.9 is based on two reasons. First, the application is interrupted in discrete intervals (in our simulation every 4ms) to run the agent negotiation. This interval is based on the rate at which thermal measurements are received; in our simulation every 10ms. The second factor is the task migration which takes considerably more time than the agent negotiation (in our simulation, 100 000 cycles are assumed). This factor is responsible for the HRTM and PDTM approaches having a longer execution time/energy usage than our approach with HRTM and PDTM taking 39.4% and 44.2% longer respectively, while using 31.59% and 44.4% more energy.

The reason behind this is that our approach requires a lesser number of task migrations than the HRTM and PDTM approaches because our thermal threshold value is not fixed and has a certain degree of tolerance. Even though there are idle cores, tasks need to be migrated more frequently in HRTM and PDTM approaches to ensure that the temperature does not reach a strict threshold. This is due to the heat sink. As it is already spreading heat from other tiles, it is unable to conduct heat efficiently away from a new task on a new tile making the tile heat up significantly faster. For instance, when the thermal sensor reaches a threshold, it will immediately migrate the task to another tile when using HRTM. In our approach however, the agent will begin selling some *free* units when it starts

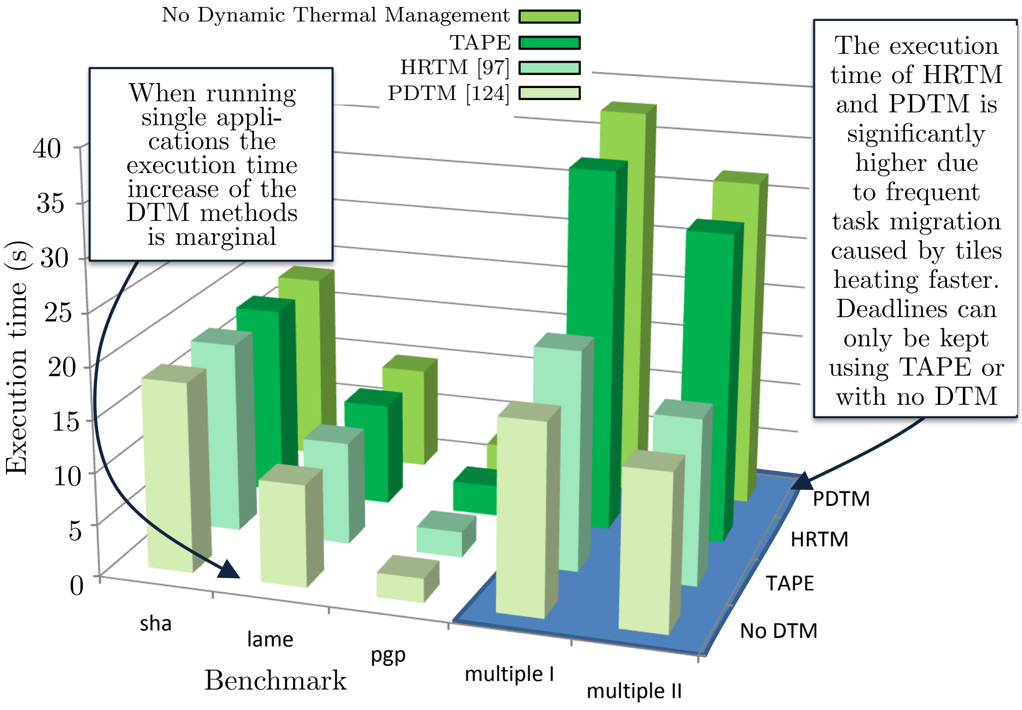
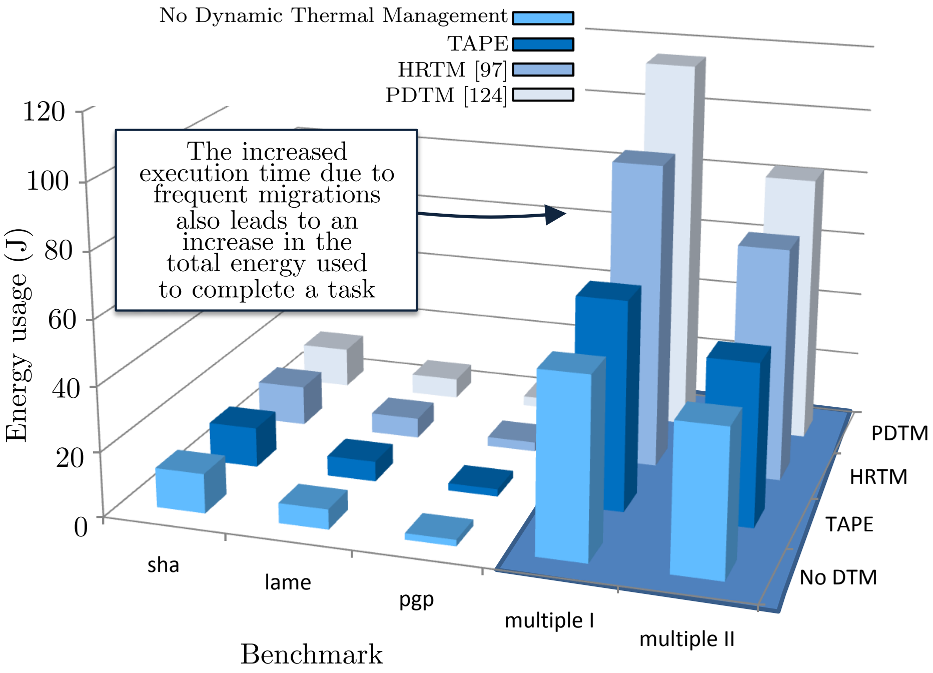


Figure 5.9.: Total execution time

approaching the threshold. The agent must first sell any *free* units it may still have before being forced to migrate a task. The energy used for each application (Fig. 5.10) directly correlates to the execution time of the application.

We require 4 bytes to encode the *buy/sell* values in our approach. The communication volume required for power trading in Fig. 5.11 is shown for both the fully distributed TAPE approach and for an approach using a *CI*, comparable to [113, 35, 124, 57]. Here, both approaches implement the same functionality, i.e. reporting of *buy/sell* values and subsequent power unit distribution. In particular, most packets to and from the *CI* will require multipe hops and thereby occupy multiple communication resources (e.g. links). The exponential increase in overall communication of the *CI* clearly shows their lack of scalability for larger multi-core architectures. Our TAPE approach already requires 11.9 times less communication with as few as 96 tiles compared to an approach using a *CI*. The communication overhead in collecting the state information for (re-)mapping is caused by: 1) each tile reporting its state (*buy/sell* values) to a mapping agent and 2) synchronization between mapping agents, which only requires the transfer of $\textsc{max}_n(sell_n - buy_n)$. Thus the scalability of the runtime application (re-)mapping communication depends largely on the amount of mapping agents deployed. Fig. 5.12 shows the communication overhead for gathering the system state of the multi-core architecture dependent on the number of tiles and the number of mapping agents. The overall computational overhead for (re-)mapping and power trading remains the same, i.e. both power trading and (re-)mapping are done in $O(N)$ using a *CI*, whereas TAPE requires $O(1)$ for power trading in each of the N tiles (and similarly mapping is done in $O(\log N)$), but is spatially distributed and parallelized when using agents, thereby eliminating a central point of failure.



The increased execution time due to frequent migrations also leads to an increase in the total energy used to complete a task

Figure 5.10.: Total energy usage

In summary our approach is able to pro-actively distribute power over a multi-core architecture and achieve on average a 7.7% decrease in peak temperature compared to having no thermal management; when running multiple applications this is increased to 11.23%. While this is 4.65% more than when using PDTM [124], our approach requires 44.2% less execution time and uses 44.2% less energy compared to PDTM. These savings, together with the scalability of TAPE, result in an ideal solution for future multi-core architectures.

5.9. On the Stability of Distributed Techniques

A key problem when employing distributed techniques such as TAPE – especially when considering large agent systems – is verifying their stable operation. Before analyzing stability, it is important to first specify the exact meaning of stability. From a control theoretical view, a stable system is simply defined as one where the integral over the impulse response $h(t)$ is finite, i.e. the system output settles after a perturbation of its input. More formally, this is expressed through the following:

$$\int_0^\infty |h(t)|dt < \infty \tag{5.11}$$

In the context of agent-based thermal management, there are actually three different kinds of stability. The first is *thermal stability.* This is given if the temperature is guaranteed to return to the safe operation range (i.e. below a thermal threshold) after a change in system input (e.g. a new task is started). Designing a stable thermal system is trivial as one can simply run the system at the low operating frequencies or even halt tasks in

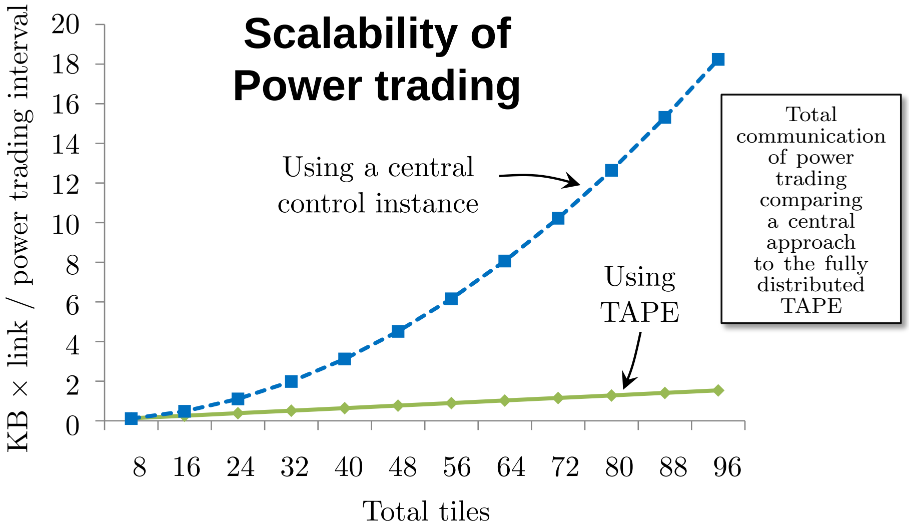
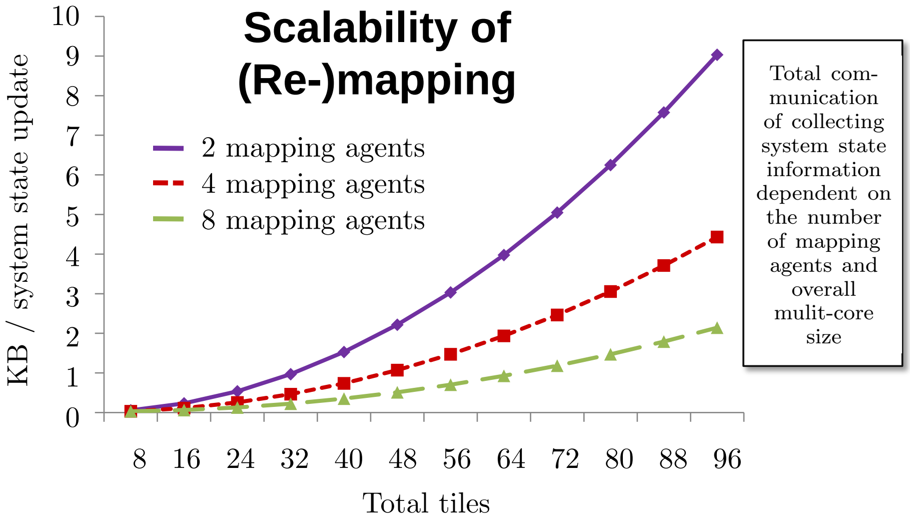


Figure 5.11.: Scalability of power trading communication volume

order to control temperature. Mostly additional constraints are added such as maximizing performance while keeping temperatures stable. The second form of stability is the inherent *stability of the agent system.* This stability is given if the agent-based system is guaranteed to find a task-to-core mapping for any change in input (tasks) in finite time. The last form is the *overall stability.* Even if a system is thermally stable and has a stable agent system, it is still possible that these work against each other. For instance, oscillations in temperature may cause the MAS to underestimate the actual temperature if it is only active periodically. Since this form of stability is difficult to determine directly, it is easier to determine stability by analyzing the outcome of an unstable system: if a system is unstable in any of the three forms, the input responses are not finite and thus no upper bounds can be given for incurred latencies or performance degradation. It follows, that if a worst-case execution time for an arbitrary task configuration can be determined, the system is stable. Note that this is a sufficient and not a necessary condition for stability. Some stable systems exist for which the worst-case execution time cannot be determined (being able to do so would effectively solve the halting problem). In our analysis we are only interested in finding systems from the class of stable systems where guarantees can be given.

The choice of tuning parameters for agent negotiation has been identified as a critical issue in ensuring a stable system [70]. Up until now distributed DTM techniques have been mainly analyzed using either simulations or running on real systems. However, due to the non-exhaustive nature of simulation, such analysis alone is not enough to account for and guarantee stability in all possible system configurations. Especially when considering large multi-core systems, the number of different configurations, e.g., task-to-core mappings, grows exponentially with the number of cores. Even if some corner cases can be specifically targeted, there is no proof that these represent a worst-case scenario, and it is never possible to consider or even foresee all corner cases. Moreover, using simulation, we may show that for a given set of tasks and cores, some mappings result in localized minima. In distributed DTM approaches, this means that a local region of cores may be successfully applying DTM from their point of view although from the global view, temperatures are



Total communication of collecting system state information dependent on the number of mapping agents and overall mulit-core size

Figure 5.12.: Scalability of (re-)mapping communication

really maximal.

Model checking has been successfully used for analyzing some single-core DTM schemes (e.g., [112, 94]). Similarly, probabilistic model checking of a DTM for multi-core architectures is presented in [83]. The focus of this work is to conduct a probabilistic analysis of frequency effects through DVFS, time and power spent over budget along with an estimate of required verification efforts. In order to raise the level of formally verifying complex DTM schemes, statistical model checking of power gating schemes has been recently reported [76]. However, to the best of our knowledge, so far no formal verification method, including model checking, was used for the verification of a distributed DTM for multi-core systems.

To address these concerns, formal verification techniques are used to examine the stability of TAPE agents in [67]. There, the proposed methodology uses the SPIN model checker [60] (an open source tool designed for formal verification of distributed software systems) together with Lamport timestamps [78] to determine the order of events in a distributed system. First, the distributed thermal management behavior of TAPE is modeled in the PROcess MEta LAnguage (PROMELA) language. These models serve as input to the SPIN model checker that can then verify desired functional properties. The Lamport timestamps algorithm is introduced in the PROMELA model of TAPE to facilitate the verification of timing properties via the SPIN model checker. Timing is given in the abstract unit of events corresponding to agent trading intervals.

The events to stability in Table 5.1 represents the number of agent trading events required to reach a stable state, i.e., a state where power units are distributed and no redundant trading of power units takes place. The analysis showed that $a_s + a_b < 1$ is a necessary condition in order for a stable state to be reached. This adds additional constraints on the choice of parameters in TAPE. Table 5.1 shows that the number of events increase proportionally with the increase in the number of tasks.

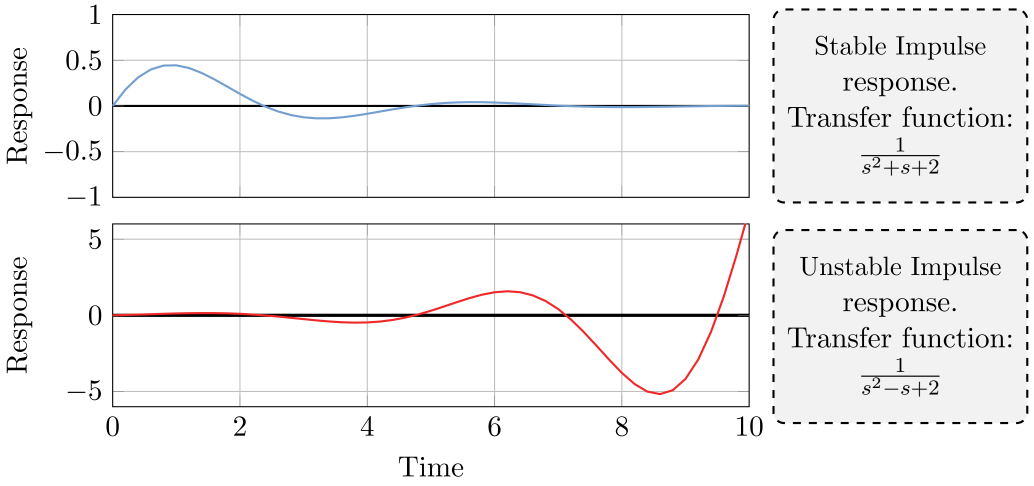


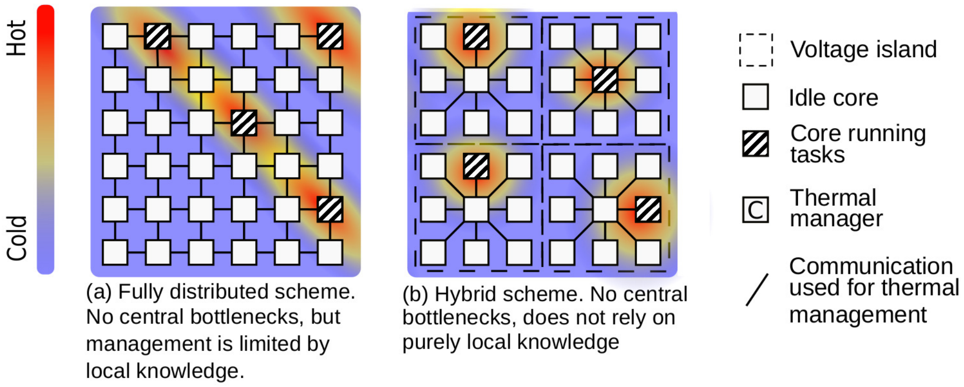
Figure 5.13.: Impulse responses of stable and unstable systems

Table 5.1.: Effect of number of tasks on stability and temperature (Total Power units:128, a_s:2/10, a_b:2/10, w_{us}:2/7, w_{fs}:2/7, w_{ub}:8/7, w_{fb}:1/7)

Tasks	Events to Stability	T_m(max)	Tasks Re-mapped
1	8	34	0
2	9	34	0
3	9	34	0
4	10	34	0
5	10	34	0
10	14	38	0
15	28	42	0
20	40	42	0
25	53	42	0
30	59	46	0
35	23	46	0
40	43	50	0
45	35	54	0
50	32	54	0
55	30	86	1
60	67	86	6

5.10. EcoLe - Economic Learning

TAPE is highly scalable because it limits the agents to a purely local view of the system, i.e. they have only local system knowledge. As a result it cannot reach the same DTM efficiency as a centralized approach. In Figure 5.14 (a) we can see the problems that can occur when only considering local knowledge in the fully distributed approach. The approach can result in configurations where tasks fail to be migrated across the chip due to local temperature maxima between regions. It may for instance be beneficial for the task in the top right to be migrated to the bottom left of the chip. However, this is hindered, e.g. due to power unit trading not passing the local thermal maxima generated



(a) Fully distributed scheme. No central bottlenecks, but management is limited by local knowledge.

(b) Hybrid scheme. No central bottlenecks, does not rely on purely local knowledge

Figure 5.14.: Management hierarchies and scalability

by the other three active cores in [43]. In such a case, the fully distributed approach must wait until the thermal threshold is hit, at which point the task will be remapped. In Figure 5.14 (b) we see how the above mentioned negative effects can be mitigated using a hierarchical approach that uses regional knowledge to achieve a trade-off between optimal results and scalability. In the next section the EcoLe [45] approach is presented. It combines a hierarchical agent system with the concepts behind TAPE to achieve scalable DTM with access to global knowledge.

5.11. System Models

The EcoLe approach considers multi-core architectures which are realized as a set of heterogeneous *processing elements* (PEs) connected through an on-chip communication network, e.g. through a bus or a network-on-chip. In a 3D architecture, two or more separate layers of these PEs and communication networks are vertically stacked on top of each other and connected using *Through-Silicon Vias* (TSVs). An example of such an Architecture is shown in Figure 5.15. We assume furthermore that the architecture implements dynamic power management, using both frequency and voltage scaling. Frequency scaling (e.g. implemented through a DCM) allows arbitrary frequencies which can be switched in one clock cycle and is available to every PE. Voltage scaling, on the other hand, is limited to a few discrete voltages which are distributed over voltage islands [95].

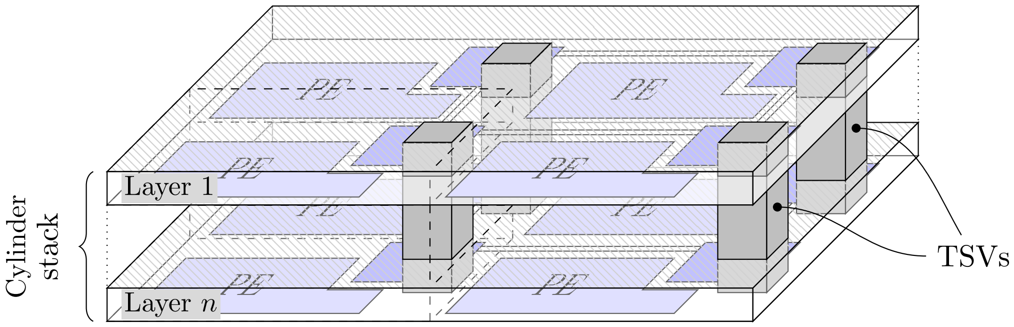


Figure 5.15.: Example of a layered 3D multi-core architecture

A **task** $\tau = \{\alpha(t, PE), \mathfrak{p}\}$ is a computational entity which is run on a PE. Each task is characterized by its switching activity α on each PE and has a priority $\mathfrak{p}$. A **periodic**

task $\hat{\tau} = \{\alpha(t, PE), \mathfrak{p}, I, D\}$ is a specific task which is reoccurring at interval I and has a deadline D. As such, the minimal frequency f_{min} is known at which the task must run on a particular PE in order to complete successfully, whereas regular tasks use best-effort execution.

In order to simplify power budgeting we further define a set of similar tasks as a **task class** $\mathcal{T} = \{\tau_i | \alpha_i(t, PE) \approx \alpha_j(t, PE)\}$, combining tasks with similar switching activity. Specifically, this will reduce the overhead of economic learning in Section 5.12.3 if there are a high number of tasks, as the learning will not be separate for each task, only for each task class. In the general case where not enough information is available to form a classification of the tasks, each task corresponds to its own unique task class.

The power consumption in CMOS is dominated by dynamic switching power and static leakage power. Both are already almost equal in today's technology nodes of 45 nm, with leakage power being expected to surpass switching power in future technology nodes. As such, the power consumed on a PE at time t can be approximated by:

$$P(t, PE) = AC_{\mathrm{PE}} f V^2 + \beta V \tag{5.12}$$

where $A = \sum_{\tau on PE} \alpha(t, PE)$ and β is the sum of all leakage currents. Furthermore, the maximal frequency f at which a PE can run is dependent on the voltage V. An increase of f will require an increase of V. Since we consider voltage to be limited to a few discrete levels, each voltage corresponds to a possible frequency range $[0, f_{max_V}]$. Power is supplied by an outside source (e.g. battery) which sets a global limit $P_{\mathrm{m}ax}$ on the amount of power available to the system.

5.12. Agent Negotiation

In order to keep our approach scalable, we use a distributed approach for thermal-aware budgeting and mapping decisions which grows linearly in both communication and computation with growing system size compared to the exponential growth of central approaches [43]. Additionally, this allows us to avoid a central point of failure. These decisions are made by so-called *agents*.

In our approach, agents are realized as software in order to minimize the impact on the architecture. There are two types of agents:

Local agents (LA) are situated at each PE. They are responsible for setting the PEs frequency based on the amount of power budget available to them, the voltage currently set, and the tasks which are running on the PE. These agents can be seen as consumers of the system, which "buy" their power budget for the next trading interval from *market agents* and convert the power into useful activities like computation or communication. They each have a fixed *income* (see Section 5.12.3) which can be used to buy their power budget. For thermal management, this *income* is adjusted through the monitoring infrastructure and the economic learning (Section 5.12.3).

Market agents (MA) are the agents where power budget trading is accomplished based on the requirements of each LA. Additionally, these agents are responsible for adjusting the voltage level of the island based on the currently allocated budget and the maximum frequency of the PEs of the LAs.

The number of *market agents* per thermal domain[3] depends largely on the size of the domain. If the thermal domains are large, i.e. consisting of several tens of cores, there may be several *market agents* per domain, each responsible for a part of it. If, on the other hand, thermal domains are only made up of one core, multiple domains can share one *market agent*. Unfortunately, having multiple *market agent* in one domain complicates the setting of the island voltage. In this case, all *market agent* in one voltage island must aggregate their consumption so that a decision may be made which takes all consumption of the island into consideration. These agents may be migrated from one PE to another, but ideally remain in a thermal domain they are responsible for in order to be able to exploit locality. To keep these migrations efficient and transparent for the agent system, agent migration is done using communication virtualization described in [46].

5.12.1. Agent Power Budgeting

Trading is performed in iterations which occur in fixed intervals Δt. This interval is set at design time as a trade-off between the overhead of budgeting and the possible thermal effect throughout the interval[4], limited by the *specific heat capacity* of the chip's materials. Each kind of agent follows its own goals: The *market agent* aims to sell all power allocated to it from the global power source to minimize the wasted power in each trading interval. Local agents, on the other hand, aim to maximize their utilization of communication and computational resources by buying as much power as possible.

The power budget corresponds to the power formula (Equation 5.12). As such, as the discrete voltage level rises, power becomes more expensive. Initially, all *local agents* have enough *income* to reach their maximal operating frequency. As a result, they will generally have an excess *income* which is added onto their *savings* (these *savings* are then later used to bid for newly deployed tasks, see Section 5.12.2). However, the *income* of *local agents* depends on the current temperature, preventing PEs from consuming a lot of power as temperatures increase (see Section 5.12.3).

Agent trading is outlined in the *Power-trading Algorithm per Market* (PAM), Algorithm 4 and illustrated in Figure 5.16. First, each LA sends its request to its MA. This request is based on the power it needs for executing its tasks τ_i at their required frequency calculated using Equation 5.12 for one task using the task's average switching activity i.e. $A = avg(\alpha_\tau)$. However, the request is limited by the LAs *income*, and thus may be less than the power needed to run the task at the required frequency which would result in task migration. The MA then distributes the power budget to the LAs through round-robin arbitration weighted by task priority $\mathfrak{p}$. If there is a left over power budget, or the power budget is too small to fulfill all LA requests, the MA increases its power budget request to the global power source in order to fulfill the LA requests in future iterations. Also, if an LA request could not be met, the LA must perform frequency scaling in order to reduce its power consumption. If, as a result, task deadlines can no longer be met, task migration is needed. In the case where there are multiple tasks running on a PE, the tasks with the lowest priorities $\mathfrak{p}$ are stopped (i.e. removed from the waiting queue) sequentially to determine if there is enough power budget to complete high priority tasks by their

[3] In 2D architectures, we consider a thermal domain to correspond to voltage islands, since voltage is the dominant part of power consumption. Thermal domains in 3D architectures are discussed in Section 5.13.2

[4] Through measurements with an infrared thermal camera, we were able to observe at most 1° C increase every 200 ms

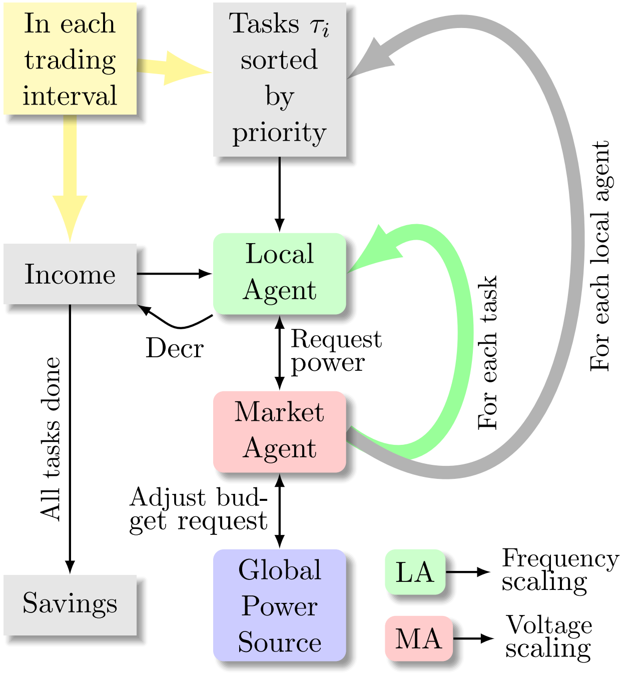


Figure 5.16.: Distribution of power budget through PAM algorithm, resulting in frequency and voltage scaling

deadlines. If so, the lower priority tasks are migrated one at a time starting with the one with the highest priority. If not, the highest priority task is migrated first and the lower priority tasks are started again.

5.12.2. Bidding for Tasks

Once a task needs to be mapped to the multi-core system, LAs place bids in order to obtain the tasks. This situation occurs when either a LA is unable to sustain task execution on its associated PE due to power budget limitations or when a new task is scheduled. Bids for new tasks are based on savings accumulated by the LA and submitted to the MA. The MAs then aggregate their highest bid amongst all other MAs. In practice, each MA constantly receives updates on the savings of its LAs and the aggregation is also done periodically. This way, the overhead will be hidden when mapping needs to occur as all MAs are already informed of which MA has the LA with the highest savings. However, since task mapping is less frequent than power budgeting and the LA with the highest savings generally does not change over short periods, this aggregation is done in multiple budget trading intervals. After each interval, one MA informs its *neighboring* MAs of its maximum known LA savings (either from its own domain or from one of its neighbors), where MAs are considered to be neighbors if they are in adjacent domains. The neighboring MAs that receive the maximum savings value will, in turn, send the maximum between the received value and the savings of the LAs in its domain to its own neighbors when it

Algorithm 4 PAM: Power-trading Algorithm per Market

```
Input: Market agent (MA), local agent (LA)
Output: Voltage, frequencies
begin
    repeat
        LA.sort($\tau_i$) by priority $\mathfrak{p}$
        MA.budget ← MA.budget + MA.request
        foreach LA do
            foreach task $\tau_i$ on PE of LA do
                send request to MA
                receive budget from MA
                LA.buget ← LA.budget + budget
                decrease LA.income
            end
            LA.savings ← LA.savings + LA.income
            if LA.budget < requiredBudget then
                frequency scaling
            end
        end
        modify MA.request
        send MA.request to global power source
        voltage scaling
    until each trading interval $\Delta t$;
end
foreach New Task do
    get highest bid
    send highest bid to middleware
    adjust MA.request
end
```

is active in the future. However, it only needs to send values back to previous MAs if its own LA savings value is higher in order to reduce redundant communication. The task is then mapped to the LA with the local highest bid in the domain of the MA with the globally highest bid. Once a task is mapped, the LAs savings are reset to zero. Generally, this results in the task being mapped to the PE which has been idle for the longest period of time, assuming that all PEs have the same temperature.

An LA can place a bid on a periodic task only if *income* is high enough to support the required PE frequency. For bids on general tasks, a bid can be placed as long as the LA has a positive *income*, as they are executed best-effort.

5.12.3. Monitoring, State Classification, and Economic Learning

In order to be able to determine the quality of agent decisions with respect to their effect on temperature, we need to have a monitoring infrastructure that is able to provide information on the quality of the monitored thermal state. This allows us to classify the monitored state in one of three categories: *good*, *neutral*, or *bad*. Which state a LA is in depends on the monitored values and how they relate to the transition values vector $X = (S_1, S_2)$ shown in Figure 5.17.

The initial configuration of the points S_1 and S_2 is determined by evenly dividing the interval between the ambient and threshold temperature into three smaller intervals. The interval border between the lowest and middle interval is assigned to S_1 and the border between the middle and upper interval S_2. If the monitored temperature is less than S_1, the LA and its PE are in a *good* state, between S_1 and S_2 in the *neutral* state, and above S_2 in *bad*.

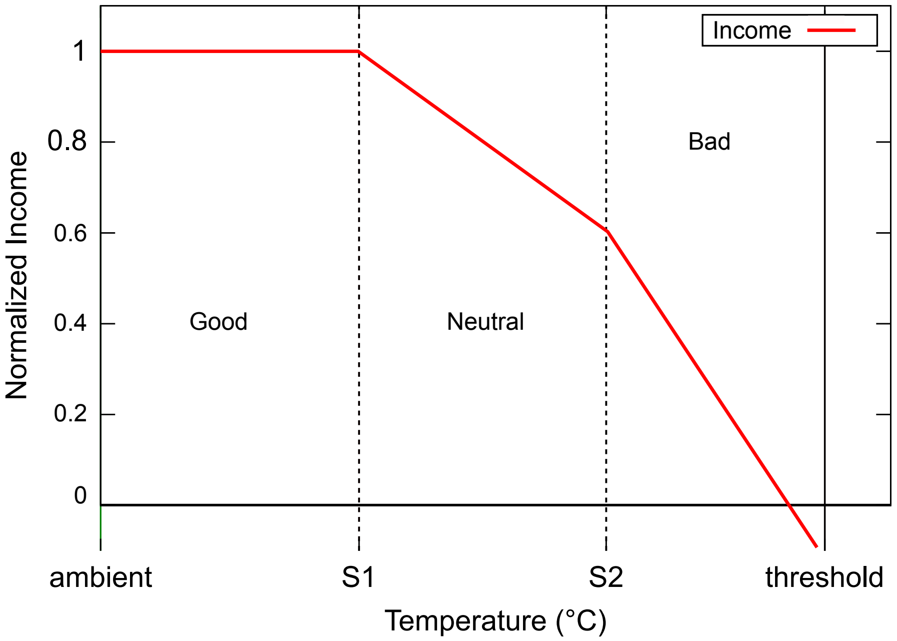


Figure 5.17.: State classification in relation to *income*

Shown in Figure 5.17 is how the *income* of an LA is dependent on the current state. In the *good* state, each LA has its full *income*, i.e. its income is equivalent to the maximum power given in Equation 5.12. In the other two states, the *income* decreases linearly with the measured temperature, with the rate of decrease being higher in the *bad* state. It should be noted that the *income* value can be negative. In this case, no task can be executed on the corresponding PE and the savings begin to decrease. This generally occurs when the threshold temperature is hit. However, the exact temperature is dependent on the economic learning when the *income* becomes negative.

Whether a *local agent* is able to "buy" all the power it needs from the *market agent*, depends on its *income* per trading interval, while the *income* in turn is determined by its state. Using the information gathered by the state classification, we are able to utilize the outcome of past decisions to influence future ones. Through Reinforcement Learning [116], we are able to reinforce decisions resulting in *good* states while penalizing decisions resulting in *bad* states.

An important aspect to consider for learning is not the state, but the state transition, as these supply us with an indication of an improvement/worsening of the system state. If, for instance, the state switches from *good* to *neutral* after a decision has been made, this decision is penalized and avoided in the future, whereas a transition from *neutral* to *good* will result in a positive reinforcement of the agent decision.

This is realized by adapting the values X, at which state transitions occur (see previous Section). However, since the currently running tasks differ in their power consumption, we must separately consider the state transition values for each task class. Therefore, we express the overall X as an average of task class transition values X_i, weighted by their activity.

$$X = \frac{\sum_{\mathcal{T}_i} \alpha_i X_i n_i}{\sum_{\mathcal{T}_i} \alpha_i n_i} \tag{5.13}$$

where α_i is the activity of $\mathcal{T}_i$ and n_i is the number of tasks of task class $\mathcal{T}_i$ currently running on the PE. Initially, all X_i are set to the predetermined X. We are now able to apply learning by modifying the values X_i of each task class individually:

$$X_i = X_i + \alpha_i \Gamma \tag{5.14}$$

where Γ is taken from Table 5.2. Here γ also depends on measured temperature value in order to keep X_i values from reaching maximum/minimum values.

$$\gamma = \hat{\gamma} \cdot 4 \frac{(T_{threshold} - T)(T_{ambient} - T)}{T_{threshold} - T_{ambient}}, \frac{\gamma}{\hat{\gamma}} \in [0, 1] \tag{5.15}$$

where T is the measured temperature and $\hat{\gamma}$ is the learning rate.

Table 5.2.: Influence Γ of state transitions on learning

from \ to	good	neutral	bad
good	▨	$(-\gamma, 0)$	$(-\gamma, -2\gamma)$
neutral	$(\gamma, 0)$	▨	$(0, -\gamma)$
bad	$(\gamma, 2\gamma)$	$(0, \gamma)$	▨

Every time a state transition occurs in a PE, Equation 5.14 is applied for each task class in order to adapt the state transition values S_1 and S_2. The income is then adjusted depending on the current state, analog to Figure 5.17. If the temperature of the PE is in the *neutral* state, the income of the corresponding LA becomes

$$\text{LA.income} = \text{income}_{max} \cdot (1 - \frac{0.4}{S_2 - S_1}(T - S_1))$$

and in the *bad* state it becomes

$$\text{LA.income} = \text{income}_{max} \cdot (0.6 - \frac{0.6}{T_{threshold} - S_{2,0}}(T - S_2))$$

where $S_{2,0}$ is the initial state transition value at time $t = 0$. The initial state transition is used as opposed to the current state transition to allow the income to become *negative* before the threshold is reached.

5.13. 3D Architectures

In the following, we generalize the above approach in order to make it applicable to 3D architectures. The approach in 2D becomes a special case of the approach in 3D using only one layer.

5.13.1. Considerations When Adapting 2D Algorithms for 3D

A number of changes to the 2D approach must be taken into consideration for it to be applicable to a 3D multi-core architecture. In particular, the thermal characteristics of each layer must be considered. Since the heat sink is only directly connected to the upper layer, heat is dissipated from tiles in that layer much more efficiently. In contrast, heat generated by tiles on the lower layers must dissipate through all layers above it to the heat sink. Additionally, the connection area between two PEs in the vertical layer is considerably higher then the area connecting two PEs vertically. As such it is not sufficient to apply EcoLe as is to a 3D architecture, instead it must be augmented.

5.13.2. Extending EcoLe for 3D

In order to consider the thermal behavior of the layered architecture, we consider a vertical stack of processing elements as an additional thermal domain. This domain is referred to as a *column*, $\mathcal{C}_{xy} = \sum_z PE_{xy}$. Thus the horizontal distribution of power budget remains the same as with a 2D architecture, however, the power budget is distributed to columns instead of tiles. To accomplish this, we must add an additional type of agent.

Column Agents (CA) are agents in a 3D multi-core architecture which are situated in and responsible for a vertical stack of PEs. These agents take the place of local agents in bidding with market agents. As such, CAs require an overall state classification and an income for their column. Functionally, it is an augmented local agent which gathers the information of all other local agents in its column and bargains/bids with the market agent on behalf of its column.

Additionally, the state classification/income system described above must be adapted to the 3D architecture.

5.13.3. Income and State Classification in 3D Columns

In each column, income is generated from the income of the local agents of the individual PEs according to their state. However, this total income must take into consideration and compensate for distance from the heat sink. In practice, this is done by adjusting the state transitions (through reinforcement/penalties) for all the local agents in a column as soon as any local agent undergoes a transition. However, unlike in the two-dimensional case, the amount the state transitions are shifted depends on the layer on which the local agent is located, with transitions in lower agents (i.e. further away form the heat sink) undergoing a greater shift.

In the implementation, this is realized by adapting the learning rate γ depending on the layer. If the layers are running at different voltages (see below), then this must be factored into the learning rate as well.

Since the layers in each column are thermally correlated, it is highly unlikely that one layer will undergo a transition into a worse state while another in the same column undergoes one to a better state. However, if it does occur, these two transitions will cancel each other out, leaving the overall column state and income unchanged. Like the income, the savings are also gathered by the CA for the entire column. Thus, when a new task is assigned after bidding, it is initially assigned to the column and not to a specific PE. The column agent is then responsible for actual assignment. Similar to the assignment based

Table 5.3.: Overview of assumed periodic SPEC2006 tasks

Task	deadline (ms)	f_{min} (GHz)
soplex	150	0.5
h264ref	42 (per frame)	0.51
bzip2	10000	0.788
calculix	500	0.198
xalancbmk	2000	0.399

on bids using the savings, the CA assigns an incoming task to the PE with the highest contributing factor of the total excess income.

Voltage scaling in the 3D architecture must also be addressed. • In the simplest case, voltage islands are the same on each layer and are stacked on top of each other, creating voltage boxes. These boxes are all set to one voltage, and thus voltage scaling can be done by the MA as in the 2D case. • More generally though, each layer can have a different voltage. In this case, the *market agent* must set the voltage level for each layer individually. • In the most general case, the voltage islands of individual layers are not aligned, meaning that the market agents must negotiate when setting the voltage. Here, we assume that we do have voltage boxes, but each layer can be run on different voltages. This limits the decisions to the domain of one market agent. However, the problem arises that the market agent does not have the necessary information for each layer's voltage island, since it only communicates with the column agent. As such, the CAs must send additional feedback to the market agents, after they have assigned a task to a PE. This then allows the MA to determine how much of the total power budget is assigned to which layer of the voltage box, and to then scale the voltage accordingly.

5.14. Experimental Setup

To evaluate EcoLe, an 8 × 8 mesh of Alpha processors is examined – partitioned into four 4 × 4 voltage islands. Simulation is performed using the m5 [23] architecture simulator paired with the MCPAT [80] simulator for power estimation. These have been modified to provide a continuous power trace as an input for the HotSpot [64] thermal simulator. As a benchmark, we have taken the applications of SPEC2006 benchmark suite [6]. For the 3D evaluation, we assume there to three mesh layers on top of each other. Power estimation is done for the 45nm technology node running at 1 GHz with a supply voltage of 1.1 V, resulting in leakage power of 8.95 W and a peak dynamic power consumption of 14 W. The power values are adjusted in our simulation based on Equation 5.12 depending on the chosen voltage and frequency during runtime.

Simulation Summary:

Ambient temperature	45°C
Tile size	4.9 mm × 4.9 mm
Die height	150 μ m
Conductivity:	
Si	100 W/mK
Cu	400 W/mK

For the simulation, we assume that selected benchmark tasks are periodic. An overview of these tasks is given in Table 5.3 showing their required minimum operating frequency with respect to their deadlines. For *h264ref*, this deadline is determined to allow 25 frames per second. For the rest of the applications, the deadline is assumed in order to evaluate a range of deadlines. The minimum frequency f_{min} is calculated using the number of execution cycles for each application. The remaining tasks are run as best-effort tasks and all tasks have their own unique task class.

5.15. Results

Regarding 2D architectures, we compare our approach and implementations based on the principles of [36, 43], enabling a comparison to state-of-the-art proactive global centralized and fully distributed approaches which have shown the best results regarding thermal management. Additionally, we also compare our approach to a central reactive one.

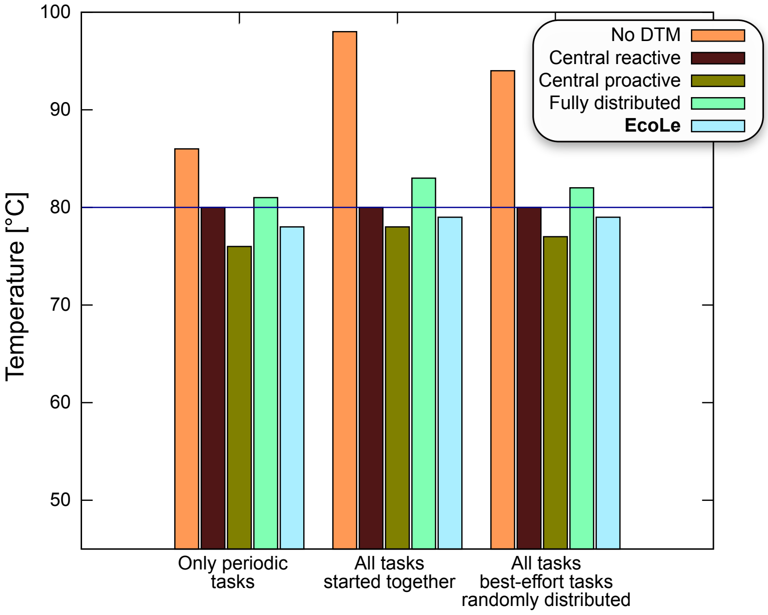


Figure 5.18.: Peak temperatures

Figure 5.18 shows the results regarding peak temperatures. As seen, our approach is able to successfully keep the temperature below a threshold of 80°C. The reduction achieved is independent of the uncertainties of the thermal simulations and instead relies on the accuracy of temperature measurements performed during runtime. Simulation is done in three runs. First, only the periodic tasks are considered. Next, the best-effort tasks are added and all tasks start at the beginning of the simulation, resulting in the worst-case power consumption when considering one instance of each task. Lastly, the start of the best-effort tasks is randomly and evenly distributed throughout the simulation, representing a more realistic scenario where tasks are dispatched over time. The periodic tasks execute continuously throughout all runs. A reduction of 15.1% compared to the case without a specific thermal management can be observed. Also, there is a 4% reduction compared to the fully distributed approach (i.e. [43]). This 4% can be critical at near-threshold temperatures in terms of transient errors caused by increased interconnect delays. Our temperatures do, however, exceed the peak temperatures achieved by the central proactive approach, as used by [36] by 2.16% on average. This was to be expected, as central approaches have a global view of the system and thus more optimization potential. They are, however, limited by their scalability [43].

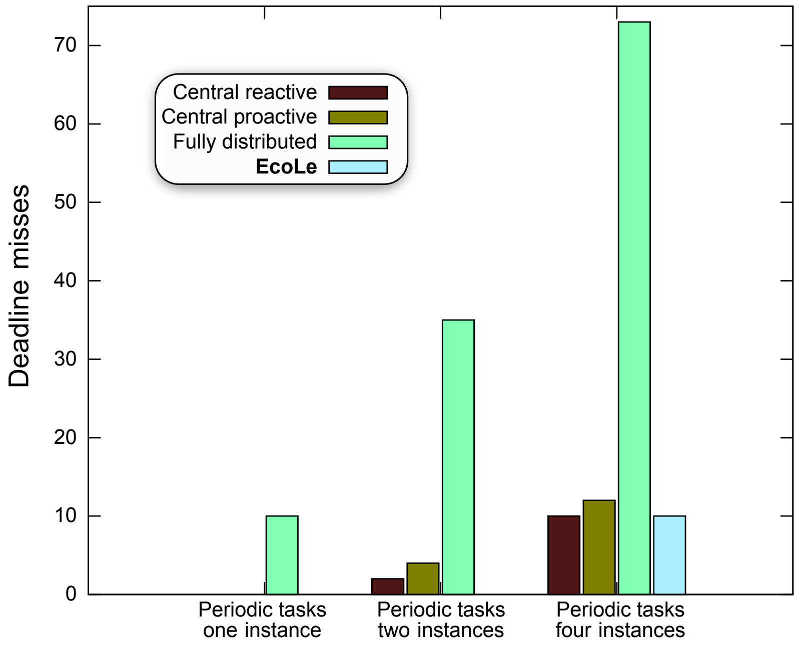


Figure 5.19.: Missed deadlines in periodic SPEC2006 tasks

In Figure 5.19, the effect of the different thermal management techniques with respect to meeting task deadlines is shown. Since best-effort tasks have no deadlines, only the periodic tasks are considered. Three simulation runs were performed having one, two, and four instances of each periodic task, respectively. The fully distributed approach has the most deadline misses in all runs. This is mainly due to the h264ref task, which has a short deadline compared to the trading interval. In this specific case, the power budget is too small locally at the time of task mapping and additional power budget must be propagated from adjacent PEs. This process can take multiple iterations. Our approach avoids this as long as there is a power budget available in the MA. The deadline misses in the third run are due to the MA requesting additional power budget from the global power source. This additional power budget is not available to the MA until the next iteration. The deadline misses of the central approaches are also caused by the h264ref task. However, in this case, the overhead of the central computation of the task mapping results in delayed task execution and missed deadlines.

Simulation Summary:

Ambient temperature	45°C
Column size	4.9 mm × 4.9 mm
Layer height	150 μm
Layer interface	20 μm
TSV diameter	7.5 μm
TSV area / total area	10%
TSV distribution	uniform
Conductivity:	
Si	100 W/mK
Cu	400 W/mK
Layer interface[5]	4 W/mK

When considering 3D architectures, the overall peak temperature is, of course, high. So that we can effectively use the additionally PEs, we increase the number of tasks and compare the results to the state-of-the-

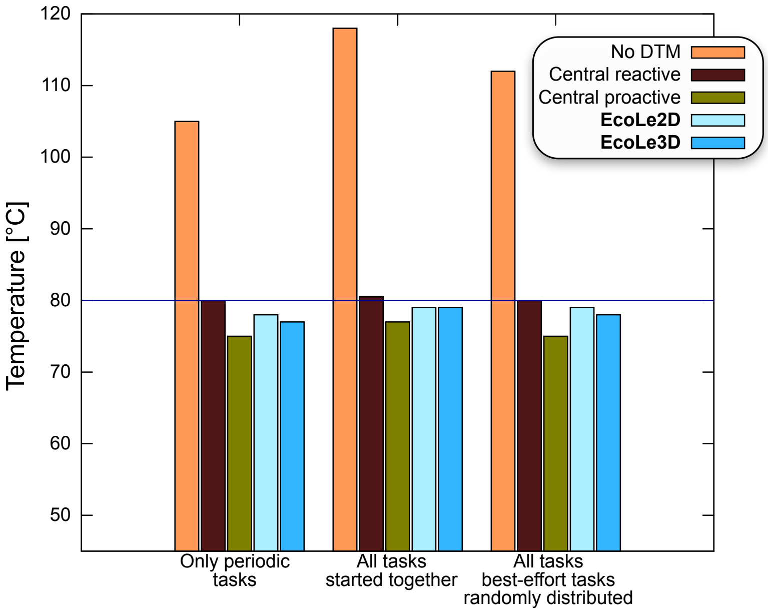


Figure 5.20.: Peak temperatures in 3D multi-core architecture

art 3D thermal management task scheduling approach presented in [127]. Additionally to our approach extended for 3D architectures (*EcoLe3D*, we also compare to EcoLe without the 3D architecture extensions (*EcoLe2D*), i.e. where all *local agents* obtain power budget from their *market agent* directly, regardless of which layer they are situated on.

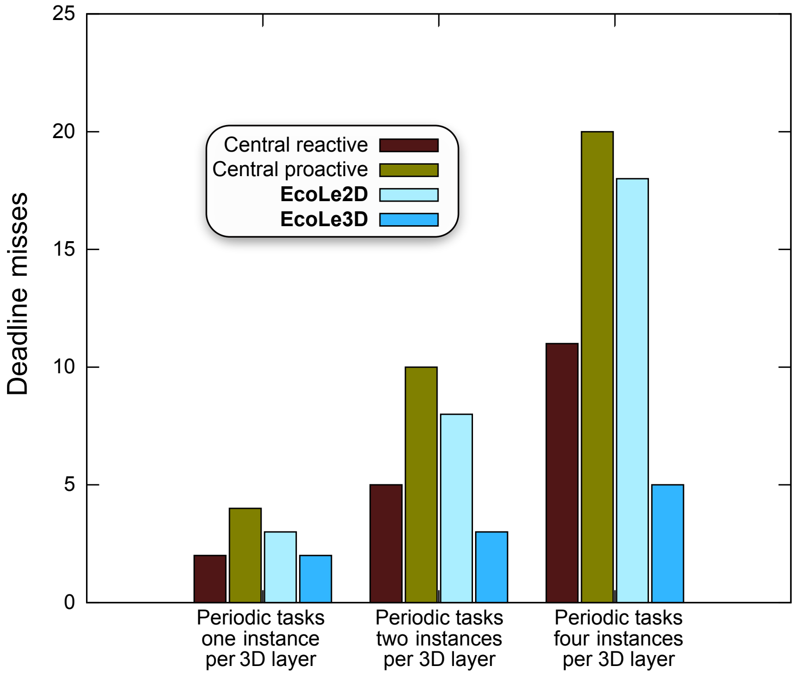


Figure 5.21.: Missed deadlines of periodic SPEC2006 tasks in 3D many-core architecure

As seen in Figure 5.20 the three proactive thermal management approaches all achieve a high reduction of peak temperature compared to the case of no DTM as well as a reduction compared to the reactive approach. Of the three, the central proactive approach has the highest reduction of peak temperature. However, this comes with a high overhead in both computation as well as very frequent task migrations. As a result, the largest number of deadline misses occur using in the central proactive approach (see Figure 5.21). While EcoLe2D is still able to reduce peak temperatures in a 3D architecture, without the CA it maps tasks to PEs without considering the temperature distribution inside a column. As a result, temperatures of the PEs in a column rise faster than in EcoLe3D and more frequent task migrations are required. This manifests itself through more deadline misses in EcoLe2D. In summary, EcoLe3D reduces the peak temperatures by an average 34°C compared to having no DTM and is 3.3°C higher than the central proactive approach. However, it increases performance by reducing the number of deadline misses by 70% compared to the central approach, and by 65% compared to EcoLe2D, on average.

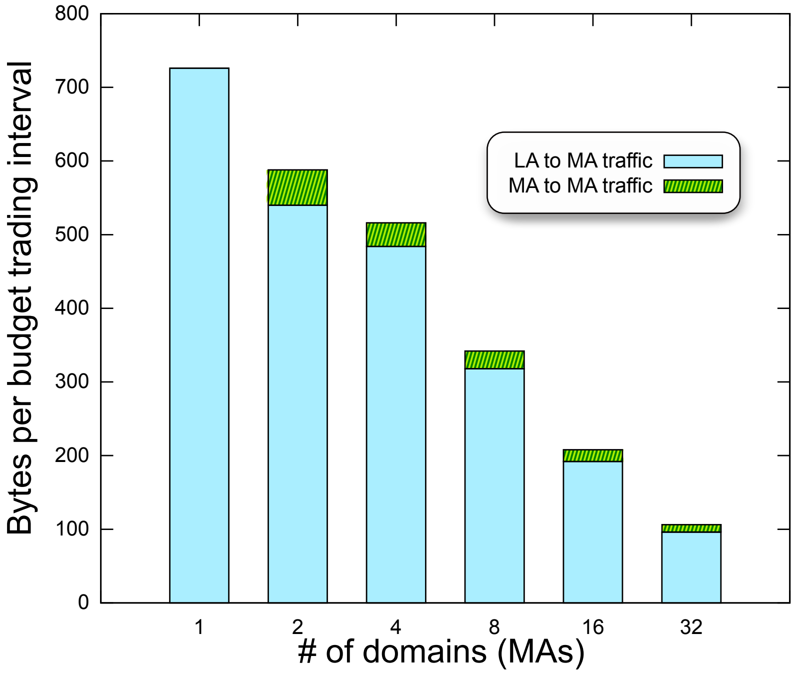


Figure 5.22.: Communication overhead per budget trading interval for 64 cores and various number of domains

The scalability of EcoLe depends largely on the number of domains (i.e. MAs). For instance, Figure 5.22 shows the communication overhead for a 64-core architecture with different numbers of MAs. Each LA to MA communication is assumed to take three Bytes: one for the LA's income, one for the LA's savings, and one for the MA's acknowledgment. The MA to MA communication is assumed to be two bytes to each neighboring MA. The result for having only one MA is equivalent to implementing the approach using a centralized management management instance. In this case, the highest communication overhead is required, even though there is no MA to MA communication. The lowest communication overhead is observed with the smallest domains, i.e. 32 domains consisting of two cores each. It should be noted that, when the number of domains is high, it takes more intervals for the MAs to propagate the maximum LA savings to all MAs. This is mostly non-critical because the LA that has the highest savings generally does not change in short time spans. However, to limit the propagation time of the LA savings values and keep the communication between the MA and the global power source (which grows linearly with the number of MAs) efficient, a lesser number of MAs is preferable, i.e. 8. Alternatively, more MAs may propagate the savings values at once or to more neighbors. This, however needs to be limited in order to retain scalability.

Additional overhead that must be considered is the memory requirement of our approach. In order to utilize the economic learning, it is necessary to store the state transition values X for each task class. This requires two bytes of memory for each task. Thus the required memory is larger than the two bytes needed for the fully distributed approach in [43]. However, this overhead can be seen as marginal, as even executing 100 tasks will only require 200 bytes of memory for state transition values.

5.16. Summary

This chapter addresses thermal management in multi-core architectures with two approaches. The first, TAPE [43], minimizes the possibility of *thermal hotspots* through pro-active power distribution. It utilizes a classical *supply/demand* model from *agent-based economics* [117] where neighboring agents trade power units locally. By limiting power trading to immediate neighbors, the communicational overhead is reduced by 11.9 times compared to a central approach considering a 96-core architecture. Agents can be implemented in hardware as well as in software with hardware agents requiring chip area (143 slices) and software agents requiring execution time on a processor. The approach is simulated using power traces obtained from the MiBench benchmark suite. When running multiple applications simultaneously, TAPE exploits a dynamic thermal threshold to keep deadlines. While PRTM [124] achieves 4.65% less peak temperature compared to our approach, it still achieves an 11.23% reduction compared to having no *DTM*. It reduces the execution time by up to 44.2% and energy consumption up to 44.4% compared to PRTM [124].

The second approach, EcoLe [45], presents a novel power budget economy for performing thermal management in both 2D and 3D multi-core architectures. It deals with the shortcomings of TAPE arising from a limited system view. Through economic learning it is able to reduce peak temperatures compared to [43] and keep them below a thermal threshold. Since our distributed approach is based on an agent hierarchy, it is also able to reduce deadline misses compared to a fully distributed approach. It can be viewed as a hybrid approach which aims to trade-off the effectiveness of a central approach using global knowledge with the scalability of a fully distributed one.

6. Outlook and Conclusion

Recently, the *International Technology Roadmap for Semiconductors* (ITRS) introduced aging-induced reliability concerns (e.g. NBTI, TDDB) as one of the difficult challenges faced by industry. These deterioration processes are highly temperature-dependent, as was shown in Chapter 3. Fortunately, the exponential growth of power densities predicted a decade ago has ceased. This means that chip temperatures – at least in planar 2D architectures – are not expected to rise beyond current ones. At the same time, however, decreasing feature sizes increase dependability issues since aging-induced degradation is accelerated. As a result, the following conclusions can be made:

- In 2D architectures, performing dynamic thermal management explicitly to target dependability will become more prominent.
- In 3D architectures, where temperature increases remain a challenge, DTM targeting both dependability and the growing temperatures is a necessity.

The dynamic of aging effects also shifts with the introduction of high-κ gate dielectrics. In a general sense, since the increased permittivity decreases the electric field between gate and bulk, aging mechanisms dominated by this field are weakened – that is, particularly NBTI and TDDB. PBTI, however, is expected to increase as additional electron traps are introduced with the high-κ insulator. As a result, the significance of HCI and PBTI are expected to grow.

The thermal management approaches presented in this thesis target different levels: from the microarchitecture to cores in a multi-core architecture (which is also extended to cylinder stacks in 3D layered architectures). First, COOL employs an extremum-seeking controller to optimize the temperature output from individual system components. TAPE and EcoLe, on the other hand, perform thermal management on a larger scale in multi-core architectures. Both operate on the principles behind distributed economies, i.e. they are based on supply-and-demand. EcoLe is also extended to manage 3D stacked layered multi-core architectures by extending its clustering hierarchy to account for the thermal properties of 3D architectures (i.e. with increased heating in the vertical direction).

The presented DTM approaches lead to the following conclusions:

- In the microarchitecture, temperature-related dependability can be maximized by optimizing temperature through a controller. This is only possible when there is a limited overhead of activity migration, and therefore cannot be expanded to the multi-core level.
- As these multi-core architectures continue to grow in size and complexity, the main limiting factor for DTM is scalability. Scalability can be explicitly targeted by employing distributed agent-based systems.

Bibliography

[1] "Bsim6.1 manual." [Online]. Available: http://www-device.eecs.berkeley.edu/bsim/Files/BSIM6/BSIM6.1.0/BSIM6.1.0_Technical_Note.pdf

[2] "Chipworks (cmos sram analysis)." [Online]. Available: https://www.chipworks.com/TOC/CMOS_Module_4_SRAM_REV1.pdf

[3] "Dias pyroview infrared camera." [Online]. Available: http://www.dias-infrared.com/pdf/pyroview380lcompacteng.pdf

[4] "Ring Oscillator, http://www-unix.ecs.umass.edu/~bdatta/lab4_report.htm."

[5] "Salome: open source integration platform for numerical simulation." [Online]. Available: http://www.salome-platform.org/

[6] "SPEC2006 benchmark suite." [Online]. Available: http://www.spec.org/cpu2006/

[7] "Stanford CPU DB." [Online]. Available: http://cpudb.stanford.edu/

[8] "Transistor aging." [Online]. Available: http://spectrum.ieee.org/semiconductors/processors/transistor-aging/0

[9] "Tsmc: Taiwan semiconductor manufacturing company." [Online]. Available: http://www.tsmc.com/english/default.htm

[10] "Intel: "intel lifts the hood on its 'single-chip cloud computer"," 2010. [Online]. Available: http://spectrum.ieee.org/semiconductors/processors/intel-lifts-the-hood-on-its-singlechip-cloud-computer

[11] "International technology roadmap for semiconductors," 2012. [Online]. Available: http://www.itrs.net/Links/2012ITRS/Home2012.htm

[12] M. Agarwal, B. Paul, M. Zhang, and S. Mitra, "Circuit failure prediction and its application to transistor aging," *VLSI Test Symposium, 25th IEEE*, pp. 277–286, 2007.

[13] M. Alam and S. Mahapatra, "A comprehensive model of pmos nbti degradation," *Microelectronics Reliability*, pp. 71–81, 2005.

[14] A. Amouri, H. Amrouch, T. Ebi, J. Henkel, and M. B. Tahoori, "Accurate thermal-profile estimation and validation for fpga-mapped circuits," in *International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2013, pp. 57–60.

[15] H. Amrouch, T. Ebi, and J. Henkel, "Stress balancing to mitigate nbti effects in register files," in *Dependable Systems and Networks (DSN), 2013 43rd Annual IEEE/IFIP International Conference on*, 2013, pp. 1–10.

[16] K. Arabi, R. Saleh, and M. Xiongfei, “Power supply noise in SoCs: Metrics, management, and measurement,” *Design Test of Computers, IEEE*, vol. 24, no. 3, pp. 236–244, May-June 2007.

[17] K. B. Ariyur and M. Krstic, *Real-time optimization by extremum-seeking control.* John Wiley & Sons, 2003.

[18] T. Austin, T. Mudge, and D. Grunwald, “Sim-panalyzer,” *http://www.eecs.umich.edu/˜panalyzer/.*

[19] A. Bansal, R. Rao, J.-J. Kim, S. Zafar, J. H. Stathis, and C.-T. Chuang, “Impacts of nbti and pbti on sram static/dynamic noise margins and cell failure probability,” *Microelectronics reliability*, vol. 49, no. 6, pp. 642–649, 2009.

[20] A. Bartolini, M. Sadri, J. Furst, A. K. Coskun, and L. Benini, “Quantifying the impact of frequency scaling on the energy efficiency of the single-chip cloud computer,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012.* IEEE, 2012, pp. 181–186.

[21] A. Bellaouar, A. Fridi, M. Elmasry, and K. Itoh, “Supply voltage scaling for temperature insensitive cmos circuit operation,” *IEEE transactions on circuits and systems. 2, Analog and digital signal processing*, vol. 45, no. 3, pp. 415–417, 1998.

[22] A. J. Bhavnagarwala, X. Tang, and J. D. Meindl, “The impact of intrinsic device fluctuations on cmos sram cell stability,” *Solid-State Circuits, IEEE Journal of*, vol. 36, no. 4, pp. 658–665, 2001.

[23] N. L. Binkert, R. G. Dreslinski, L. R.Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt, “The m5 simulator: Modeling networked systems,” *IEEE Micro*, vol. 26, pp. 52–60, 2006.

[24] B. Black, M. Annavaram, N. Brekelbaum, J. DeVale, L. Jiang, G. H. Loh, D. McCaule, P. Morrow, D. W. Nelson, D. Pantuso, P. Reed, J. Rupley, S. Shankar, J. Shen, and C. Webb, “Die stacking (3d) microarchitecture,” in *MICRO 39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture.* Washington, DC, USA: IEEE Computer Society, 2006, pp. 469–479.

[25] J. Black, “Electromigration – a brief survey and some recent results,” *Electron Devices, IEEE Transactions on*, vol. 16, no. 4, pp. 338–347, Apr 1969.

[26] S. Borkar, “Thousand core chips: a technology perspective,” *Design Automation Conference (DAC)*, pp. 746–749, 2007.

[27] K. M. Bretthauer and B. Shetty, “The nonlinear knapsack problem–algorithms and applications,” *European Journal of Operational Research*, vol. 138, no. 3, pp. 459–472, 2002.

[28] J. Cao, D. P. Spooner, S. A. Jarvis, and G. R. Nudd, “Grid load balancing using intelligent agents,” *FUTURE GENERATION COMPUTER SYSTEMS*, vol. 21, no. 1, pp. 135–149, 2005.

[29] K. Cao, W.-C. Lee, W. Liu, X. Jin, P. Su, S. Fung, J. An, B. Yu, and C. Hu, “Bsim4 gate leakage model including source-drain partition,” in *Electron Devices Meeting, 2000. IEDM’00. Technical Digest. International.* IEEE, 2000, pp. 815–818.

[30] Y. A. Cengel, M. A. Boles, and M. Kanoglu, *Thermodynamics: an engineering approach.* McGraw-Hill New York, 2011, vol. 5.

[31] A. Chakraborty, K. Duraisami, A. Sathanur, P. Sithambaram, L. Benini, A. Macii, E. Macii, and M. Poncino, "Dynamic thermal clock skew compensation using tunable delay buffers," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 16, no. 6, pp. 639–649, 2008.

[32] J. Cong, J. Wei, and Y. Zhang, "A thermal-driven floorplanning algorithm for 3d ics," in *ICCAD '04: Proceedings of the 2004 IEEE/ACM International conference on Computer-aided design.* Washington, DC, USA: IEEE Computer Society, 2004, pp. 306–313.

[33] J. Cong and Y. Zhang, "Thermal via planning for 3-d ics," in *ICCAD '05: Proceedings of the 2005 IEEE/ACM International conference on Computer-aided design.* Washington, DC, USA: IEEE Computer Society, 2005, pp. 745–752.

[34] A. Coskun, T. Rosing, J. Ayala, D. Atienza, and Y. Leblebici, "Dynamic thermal management in 3d multicore architectures," in *Design Automation and Test in Europe (DATE)*, 2009, pp. 1410–1415.

[35] A. Coskun, T. Rosing, and K. Gross, "Temperature management in multiprocessor socs using online learning," *Design Automation Conference (DAC)*, pp. 890–893, 2008.

[36] ——, "Utilizing predictors for efficient thermal management in multiprocessor SoCs," *IEEE Transactions on CAD*, vol. 28, pp. 1503–1516, 2009.

[37] A. Coskun, T. Rosing, and K. Whisnant, "Temperature aware task scheduling in MPSoC," in *Design, Automation & Test in Europe Conference (DATE)*, 2007, pp. 1–6.

[38] A. K. Coskun, J. Ayala, D. Atienza, and T. S. Rosing, "Modeling and dynamic management of 3d multicore systems with liquid cooling," in *17th IFIP/IEEE International Conference on Very Large Scale Integration (VLSI SoC)*, 2009.

[39] F. J. DiSalvo, "Thermoelectric cooling and power generation," *Science*, pp. 703–706, 1999.

[40] R. Diversi, A. Bartolini, A. Tilli, F. Beneventi, and L. Benini, "Scc thermal model identification via advanced bias-compensated least-squares," in *DATE*, 2013, pp. 230–235.

[41] C. Dixon and E. W. Frew, "Controlling the mobility of network nodes using decentralized extremum seeking," in *Decision and Control, 2006 45th IEEE Conference on.* IEEE, 2006, pp. 1291–1296.

[42] R. D. Doug, D. Burger, S. W. Keckler, and T. Austin, "Sim-alpha: a validated, execution-driven alpha 21264 simulator," 2001.

[43] T. Ebi, M. A. A. Faruque, and J. Henkel, "TAPE: Thermal-aware agent-based power economy for multi/many-core architectures," in *27th IEEE/ACM International Conference on Computer-Aided Design (ICCAD '09)*, 2009, pp. 302–309.

[44] T. Ebi, H. Amrouch, and J. Henkel, "COOL: control-based optimization of load-balancing for thermal behavior," *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pp. 255–264, 2012.

[45] T. Ebi, D. Kramer, W. Karl, and J. Henkel, "Economic learning for thermal-aware power budgeting in many-core architectures," in *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2011 Proceedings of the 9th International Conference on.* IEEE, 2011, pp. 189–196.

[46] T. Ebi, H. Rauchfuss, A. Herkersdorf, and J. Henkel, "Agent-based thermal management using real-time i/o communication relocation for 3D many-cores," in *Integrated Circuit and System Design. Power and Timing Modeling, Optimization, and Simulation workshop (PATMOS)*, 2011, pp. 112–121.

[47] M. Eisele, J. Berthold, D. Schmitt-Landsiedel, and R. Mahnkopf, "The impact of intra-die device parameter variations on path delays and on the design for yield of low voltage digital circuits," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 5, no. 4, pp. 360 –368, dec. 1997.

[48] O. Engström, T. Gutt, and H. Przewlocki, "Energy concepts involved in mos characterization," *Journal of Telecommunication and Information Technology*, 2007.

[49] M. A. A. Faruque, R. Krist, and J. Henkel, "Adam: Run-time agent-based distributed application mapping for on-chip communication," *Design Automation Conference (DAC)*, pp. 760–765, 2008.

[50] B. Feero and P. Pande, "Performance evaluation for three-dimensional networks-on-chip," in *VLSI, 2007. ISVLSI '07. IEEE Computer Society Annual Symposium on*, March 2007, pp. 305–310.

[51] T. Fleig, O. Mattes, and W. Karl, "Evaluation of adaptive memory management techniques on the tilera tile-gx platform," in *Architecture of Computing Systems (ARCS), 2014 27th International Conference on.* VDE, 2014, pp. 1–8.

[52] B. Griffith, D. Trler, and H. Goudey, "Infraredthermography," in *Encyclopedia of Imaging Science and Technology*, 2002.

[53] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown, "Mibench: A free, commercially representative embedded benchmark suite," *IEEE International Workshop on Workload Characterization*, pp. 3–14, 2001.

[54] Y. Han, I. Koren, and C. A. Moritz, "Temperature aware floorplanning," in *Workshop on Temperature Aware Computer Systems*, 2005.

[55] U. Hansen and P. Vogl, "Hydrogen passivation of silicon surfaces: A classical molecular-dynamics study," *Physical Review B*, vol. 57, no. 20, p. 13295, 1998.

[56] J. Henkel, T. Ebi, H. Amrouch, and H. Khdr, "Thermal management for dependable on-chip systems." in *ASP-DAC*, 2013, pp. 113–118.

[57] S. Heo, K. Barr, and K. Asanović, "Reducing power density through activity migration," *International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 217–222, 2003.

[58] S. Herbert and D. Marculescu, “Analysis of dynamic voltage/frequency scaling in chip-multiprocessors,” *International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 38–43, 2007.

[59] M. D. Hill and M. R. Marty, “Amdahl’s law in the multicore era,” *Computer*, vol. 41, no. 7, pp. 33–38, Jul. 2008.

[60] G. Holzmann, “The model checker SPIN,” *IEEE Transactions on Software Engineering*, vol. 23, no. 5, pp. 279–295, 1997.

[61] H. Homayoun, A. Gupta, A. Veidenbaum, A. Sasan (M.A. Makhzan), F. Kurdahi, and N. Dutt, “Relocate: register file local access pattern redistribution mechanism for power and thermal management in out-of-order embedded processor,” *Proceedings of the 5th international conference on High Performance Embedded Architectures and Compilers*, pp. 216–231, 2010.

[62] P. Horn, “Autonomic computing: Ibm’s perspective on the state of information technology,” *IBM Corporation*, 2001.

[63] W. Huang, M. R. Stan, K. Skadron, K. Sankaranarayanan, S. Ghosh, and S. Velusam, “Compact thermal modeling for temperature-aware design,” in *Design Automation Conference (DAC)*, 2004, pp. 878–883.

[64] W. Huang, K. Sankaranarayanan, R. J. Rib, M. R. Stan, and K. Skadron, “An improved block-based thermal model in hotspot 4.0 with granularity considerations,” 2007.

[65] T. J. Hughes, *The finite element method: linear static and dynamic finite element analysis.* Courier Dover Publications, 2012.

[66] C. Ih-Chin, S. E. Holland, and C. Hu, “Electrical breakdown in thin gate and tunneling oxides,” *Solid-State Circuits, IEEE Journal of*, vol. 20, no. 1, pp. 333–342, 1985.

[67] M. Ismail, O. Hasan, T. Ebi, M. Shafique, and J. Henkel, “Formal verification of distributed dynamic thermal management,” in *Proceedings of the International Conference on Computer-Aided Design.* IEEE Press, 2013, pp. 248–255.

[68] N. Jennings, K. Sycara, and M. Woolridge, “A roadmap of agent research and development,” *Autonomous Agents and Multi-Agent Systems*, 1998.

[69] K. Joshi, S. Mukhopadhyay, N. Goel, and S. Mahapatra, “A consistent physical framework for n and p bti in hkmg mosfets,” in *Reliability Physics Symposium (IRPS), 2012 IEEE International.* IEEE, 2012, pp. 5A–3.

[70] M. Kadin, S. Reda, and A. Uht, “Central vs. distributed dynamic thermal management for multi-core processors: which one is better?” in *Great Lakes symposium on VLSI*, 2009, pp. 137–140.

[71] H. Khdr, T. Ebi, M. Shafique, H. Amrouch, and J. Henkel, “mDTM: Multi-objective dynamic thermal management for on-chip systems,” in *Design Automation and Test in Europe Conference*, 2014.

[72] J. Kim, C. Nicopoulos, D. Park, R. Das, Y. Xie, V. Narayanan, M. S. Yousif, and C. R. Das, “A novel dimensionally-decomposed router for on-chip communication

in 3d architectures," *SIGARCH Comput. Archit. News*, vol. 35, no. 2, pp. 138–149, 2007.

[73] J. Kim, S. T. Jhang, S.-J. Moon, and C. S. Jhon, "Register-relocation: a thermal-aware renaming method for reducing temperature of a register file," *ACM SIGAPP Applied Computing Review*, vol. 11, no. 1, pp. 41–51, 2010.

[74] K. K. Kim, W. Wang, and K. Choi, "On-chip aging sensor circuits for reliable nanometer mosfet digital circuits," *Circuits and Systems II: Express Briefs, IEEE Transactions on*, pp. 798–802, 2010.

[75] S. Koranne, "Vlsi cad tools," in *Handbook of Open Source Tools.* Springer US, 2011, pp. 317–347.

[76] J. Kumar and S. Vasudevan, "Verifying dynamic power management schemes using statistical model checking," in *Asia and South Pacific Design Automation Conference.* IEEE, 2012, pp. 579–584.

[77] S. V. Kumar, C. H. Kim, and S. S. Sapatnekar, "Impact of nbti on sram read stability and design for reliability," in *Quality Electronic Design, 2006. ISQED'06. 7th International Symposium on.* IEEE, 2006, pp. 6–pp.

[78] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, 1978.

[79] E. Le Sueur and G. Heiser, "Dynamic voltage and frequency scaling: The laws of diminishing returns," in *Proceedings of the 2010 international conference on Power aware computing and systems.* USENIX Association, 2010, pp. 1–8.

[80] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures," *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 469–480, 2009.

[81] G. M. Link and N. Vijaykrishnan, "Thermal trends in emerging technologies," in *Proceedings of the 7th International Symposium on Quality Electronic Design.* IEEE Computer Society, 2006, pp. 625–632.

[82] Z.-Y. Lu, C. Nicklaw, D. Fleetwood, R. Schrimpf, and S. Pantelides, "Structure, properties, and dynamics of oxygen vacancies in amorphous s i o 2," *Physical review letters*, vol. 89, no. 28, p. 285505, 2002.

[83] A. Lungu, P. Bose, D. J. Sorin, S. German, and G. Janssen, "Multicore power management: Ensuring robustness via early-stage formal verification," in *Formal Methods and Models for Co-Design, 2009. MEMOCODE '09. 7th IEEE/ACM International Conference on*, july 2009, pp. 78 –87.

[84] L. Ma, X. Ji, Z. Chen, Y. Liao, F. Yan, Y. Song, and Q. Guo, "Physical understanding of hot carrier injection variability in deeply scaled nmosfets," *Japanese Journal of Applied Physics*, vol. 53, no. 4S, p. 04EC15, 2014.

[85] S. Mahapatra, A. Islam, S. Deora, V. Maheta, K. Joshi, A. Jain, and M. Alam, "A critical re-evaluation of the usefulness of rd framework in predicting nbti stress and recovery," in *Reliability Physics Symposium (IRPS), 2011 IEEE International.* IEEE, 2011, pp. 6A–3.

[86] M. Malinen and P. Rzolback, “Elmer finite element solver for multiphysics and multiscale problems,” *Multiscale Modelling Methods for Applications in Materials Science: CECAM Tutorial, 16-20 September 2013, Forschungszentrum Jülich, Lecture Notes*, vol. 19, p. 101, 2013.

[87] S. M. Martin, K. Flautner, T. Mudge, and D. Blaauw, “Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads,” *Proceedings of the IEEE/ACM international conference on Computer-aided design*, pp. 721–725, 2002.

[88] J. F. Martínez and E. Ipek, “Dynamic multicore resource management: A machine learning approach,” *IEEE Micro*, vol. 29, no. 5, pp. 8–17, 2009.

[89] G. C. Meijer and A. v. Herwaarden, *Thermal sensors.* Taylor & Francis, 1994.

[90] G. Memik, M. T. Kandemir, and O. Ozturk, “Increasing register file immunity to transient errors,” in *Design, Automation and Test in Europe, 2005. Proceedings.* IEEE, 2005, pp. 586–591.

[91] G. E. Moore *et al.*, “Cramming more components onto integrated circuits,” 1965.

[92] F. Mulas, M. Pittau, M. Buttu, S. Carta, A. Acquaviva, L. Benini, and D. Atienza, “Thermal balancing policy for streaming computing on multiprocessor architectures,” *Design, automation and test in Europe (DATE)*, pp. 734–739, 2008.

[93] H. Nguyen, C. Salm, B. Krabbenborg, K. Weide-Zaage, J. Bisschop, A. Mouthaan, and F. Kuper, “Effect of thermal gradients on the electromigration life-time in power electronics,” *Reliability Physics Symposium Proceedings, 42nd Annual. IEEE International*, pp. 619–620, 2004.

[94] G. Norman, D. Parker, M. Kwiatkowska, S. Shukla, and R. Gupta, “Using probabilistic model checking for dynamic power management,” *Formal Aspects of Computing, Springer-Verlag*, vol. 17, no. 2, pp. 160–176, 2005.

[95] U. Y. Ogras, R. Marculescu, P. Choudhary, and D. Marculescu, “Voltage-frequency island partitioning for gals-based networks-on-chip,” *Design Automation Conference (DAC)*, pp. 110–115, 2007.

[96] K. Patel, W. Lee, and M. Pedram, “Active bank switching for temperature control of the register file in a microprocessor,” in *Proceedings of the 17th ACM Great Lakes symposium on VLSI.* ACM, 2007, pp. 231–234.

[97] M. D. Powell, M. Gomaa, and T. N. Vijaykumar, “Heat-and-run: Leveraging smt and cmp to manage power density through the system,” *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 260–270, 2004.

[98] R. Preston, R. Badeau, D. Bailey, S. Bell, L. Biro, W. Bowhill, D. Dever, S. Felix, R. Gammack, V. Germini, M. Gowan, P. Gronowski, D. Jackson, S. Mehta, S. Morton, J. Pickholtz, M. Reilly, and M. Smith, “Design of an 8-wide superscalar risc microprocessor with simultaneous multithreading,” *Solid-State Circuits Conference, Digest of Technical Papers. ISSCC. IEEE International*, pp. 334–472, 2002.

[99] K. Puttaswamy and G. H. Loh, “Thermal analysis of a 3d die-stacked high-performance microprocessor,” in *GLSVLSI ’06: Proceedings of the 16th ACM Great Lakes symposium on VLSI.* New York, NY, USA: ACM, 2006, pp. 19–24.

[100] M. T. Quddus, T. A. DeMassa, and J. J. Sanchez, "Unified model for qbd prediction for thin gate oxide mos devices with constant voltage and current stress," pp. 357–372, 2000.

[101] P. Rantala, J. Isoaho, and H. Tenhunen, "Novel agent-based management for fault-tolerance in network-on-chip," in *DSD '07: Proceedings of the 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools.* Washington, DC, USA: IEEE Computer Society, 2007, pp. 551–555.

[102] S. Reda, "Thermal and power characterization of real computing devices," *Emerging and Selected Topics in Circuits and Systems, IEEE Journal on*, vol. 1, no. 2, pp. 76–87, 2011.

[103] E. Rotem, J. Hermerding, A. Cohen, and H. Cain, "Temperature measurement in the intel® coretm duo processor," 2006.

[104] M. M. Sabry and D. Atienza, "Temperature-aware design and management for 3d multi-core architectures," *Foundations and Trends in Electronic Design Automation*, vol. 8, no. 2, pp. 117–197, 2014.

[105] M. M. Sabry, A. Sridhar, J. Meng, A. K. Coskun, and D. Atienza, "GreenCool: An energy-efficient liquid cooling design technique for 3-D MPSoCs via channel width modulation," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 32, no. 4, pp. 524–537, 2013.

[106] T. Sakurai and A. Newton, "Alpha-power law mosfet model and its applications to cmos inverter delay and other formulas," *Solid-State Circuits, IEEE Journal of*, vol. 25, no. 2, pp. 584–594, Apr 1990.

[107] S. Salivahanan, *Electronic Devices & Circuits.* Tata McGraw-Hill Education, 2012.

[108] D. Schroder and J. Babcock, "Negative bias temperature instability: Road to cross in deep submicron silicon semiconductor manufacturing," *Journal of Applied Physics*, pp. 1–18, 2003.

[109] K. F. Schuegraf and C. Hu, "Hole injection sio 2 breakdown model for very low voltage lifetime extrapolation," *Electron Devices, IEEE Transactions on*, vol. 41, no. 5, pp. 761–767, 1994.

[110] J. A. Schwarz, "Effect of temperature on the variance of the log-normal distribution of failure times due to electromigration damage," *Journal of Applied Physics*, pp. 801–803, 1987.

[111] M. Shafique, S. Garg, D. Marculescu, and J. Henkel, "The eda challenges in the dark silicon era: Temperature, reliability, and variability perspectives," pp. 185:1–185:6, 2014.

[112] S. Shukla and R. Gupta, "A model checking approach to evaluating system level dynamic power management policies for embedded systems," in *High-Level Design Validation and Test Workshop.* IEEE Computer Society, 2001, pp. 53–57.

[113] K. Skadron, M. R. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan, "Temperature-aware microarchitecture: Modeling and implementation," *ACM Trans. Archit. Code Optim., vol. 1, no. 1*, pp. 94–125, 2004.

[114] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers, "The case for lifetime reliability-aware microprocessors," *SIGARCH Comput. Archit. News*, pp. 276–287, 2004.

[115] K. Støy, W.-M. Shen, and P. M. Will, "How to make a self-reconfigurable robot run," in *AAMAS*, 2002, pp. 813–820.

[116] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction.* MIT press, 1998.

[117] L. Tesfatsion, "Agent-based computational economics: Growing economies from the bottom up," *Artificial Life*, pp. 55–82, 2002.

[118] J. B. Velamala, K. Sutaria, T. Sato, and Y. Cao, "Physics matters: statistical aging prediction under trapping/detrapping," *Design Automation Conference*, pp. 139–144, 2012.

[119] N. Weste and K. Eshraghian, *Principles of CMOS VLSI design: a systems perspective*, ser. VLSI systems series. Addison-Wesley Pub. Co., 1993. [Online]. Available: http://books.google.de/books?id=UiBTAAAAMAAJ

[120] M. White, "Microelectronics reliability: physics-of-failure based modeling and lifetime evaluation," 2008.

[121] D. Wolpert and P. Ampadu, "A sensor to detect normal or reverse temperature dependence in nanoscale cmos circuits," in *Defect and Fault Tolerance in VLSI Systems, 2009. DFT '09. 24th IEEE International Symposium on*, Oct 2009, pp. 193–201.

[122] ——, "Temperature effects in semiconductors," in *Managing Temperature Effects in Nanoscale Adaptive Systems.* Springer, 2012, pp. 15–33.

[123] C. Yang and A. Orailoglu, "Processor reliability enhancement through compiler-directed register file peak temperature reduction," in *IEEE/IFIP International Conference on Dependable Systems & Networks (DSN)*, 2009, pp. 468–477.

[124] I. Yeo, C. C. Liu, and E. J. Kim, "Predictive dynamic thermal management for multicore systems," *Design Automation Conference (DAC)*, pp. 734–739, 2008.

[125] K. Zhao, J. Stathis, A. Kerber, and E. Cartier, "Pbti relaxation dynamics after ac vs. dc stress in high-k/metal gate stacks," in *Reliability Physics Symposium (IRPS), 2010 IEEE International.* IEEE, 2010, pp. 50–54.

[126] W. Zhao and Y. Cao, "Predictive technology model for nano-cmos design exploration," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 3, no. 1, p. 1, 2007.

[127] X. Zhou, J. Yang, Y. Xu, Y. Zhang, and J. Zhao, "Thermal-aware task scheduling for 3D multicore processors," *IEEE Trans. Parallel Distrib. Syst.*, pp. 60–71, 2010.

[128] X. Zhou, Y. Xu, Y. Du, Y. Zhang, and J. Yang, "Thermal management for 3d processors via task scheduling," in *Parallel Processing, 2008. ICPP '08. 37th International Conference on*, Sept. 2008, pp. 115–122.

[129] C. Zhu, Z. Gu, L. Shang, R. Dick, and R. Joseph, "Three-dimensional chip-multiprocessor run-time thermal management," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 27, no. 8, pp. 1479–1492, Aug. 2008.

List of Figures

List of Tables

Appendices

A. SPICE model of NBTI circuit

This chapter presents an example SPICE implementation of the NBTI circuit introduced in Chapter 3.1.2.2. To prevent contributions to the timing behavior, the transistors and diodes are kept as ideal as possible. The diode controlling the current direction approximates ideal behavior by using a low emission coefficient ($n = 0.001$). Both the ideal PMOS and NMOS transistors are approximated using ideal switches, with NMOS transistors receiving inverted stress voltage. The voltage stored in `CTH` is representative of the degradation due to NBTI, i.e. $\alpha \cdot (\texttt{VINIT - V(NTOT)}) = \Delta V_{th}$.

```
NBTI equivalent RC circuit
****************************
**
**

.PARAM VINIT = 1.0

** Pulse stress source:
V1 VSTRESS GND 0 PULSE(0 1 10N 10N 10N 10m 20m)
** Alternate constant stress source:
* V2 VSTRESS GND 1
** Inverted stress condition:
BN NNONSTRESS 0 V = ’1-V(VSTRESS)’

** Capacitors representing Vth shift
CTH         NTOT GND 10U IC=VINIT       ; Capacitor representing total traps
CIT         NIT  GND 8U                 ; Cap. representing interface traps
CHT         NHT  GND 5N                 ; Cap. representing hole traps
```

```
S1          NTOT C VSTRESS 0 SM ON
DIT         C D ID
RVAR        D NIT R=’(VINIT-V(NTOT))*1E12’
SN1         NIT NPRIT NNONSTRESS 0 SM OFF
RVA2        NPRIT NRIT R=’(VINIT-V(NTOT))*1E12’
VSenseIT    NRIT 0 0

S2          NTOT B VSTRESS 0 SM ON
DHT         B NPHT ID
RRecoveryHT NPRHT NRHT 1K
RStressHT   NHT NPHT 1K
SN2         NHT NPRHT NNONSTRESS 0 SM OFF
VSenseHT    NRHT 0 0

BIR         0 NIR I=’I(VSenseIT) + I(VSenseHT)’
VSense3     NIR NTOT  0

** Near-ideal switch model:
.MODEL SM SW(VT=0, VH=0, RON=1, ROFF=1.0E+12 )
** Near-ideal diode model:
.MODEL ID D(N=0.001)
*  Analysis
.TRAN 1n 1e3 0.0001m 100m UIC

.OPTIONS NOMOD

.END
```

Figure A.1 shows a summary of the results of the SPICE simulation. Voltage stress is applied for 10 ms of each 20 ms period. To keep the figure comprehensible, only every 10 000 sample is plotted, as the full plot is too dense to differentiate. The short-term characteristics of hole trapping and detrapping can be seen in the high frequency noise in ΔV_{th}.

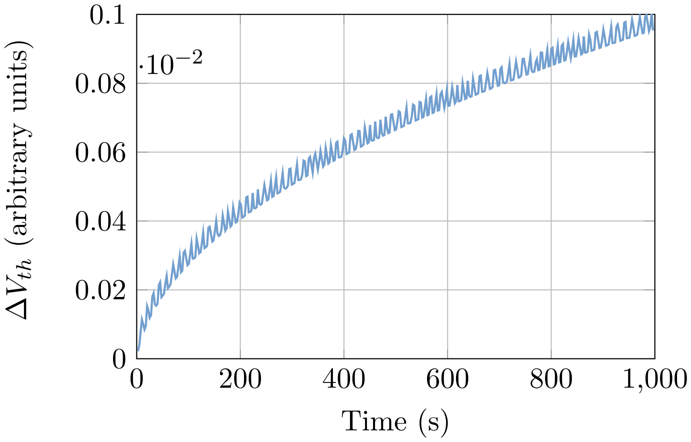


Figure A.1.: Results of circuit simulation in SPICE. Plot shows every 1e5 sample

B. OpenMPI Implementation of TAPE-Agent Communication

```
int in[4], out[4], east, rank;
MPI_Request request_reast = MPI_REQUEST_NULL;
MPI_Request request_seast = MPI_REQUEST_NULL;
MPI_Cart_create (MPI_COMM_WORLD, ... , &cart);
MPI_Cart_shift (cart, 1, 1, &west, &east);

if ( east != MPI_PROC_NULL ) {
   MPI_Irecv(data, 1, MPI_INT, east, UPDATE,
             MPI_COMM_WORLD, &request_reast);
}

while(TRUE){    // Agent Loop

   MPI_Test( &request_reast, &flag , &status);
   if ( flag && status.MPI_TAG != MPI_ANY_TAG ) {
     input = data[0];
     MPI_Irecv(data , 1, MPI_INT, east, UPDATE,
               MPI_COMM_WORLD, &request_reast);
     }


     //  ... Agent code ...
   MPI_Wait( &request_seast, &status );
   if ( east != MPI_PROC_NULL ) {
     MPI_Isend(out, 1, MPI_INT, east, UPDATE,
               MPI_COMM_WORLD, &request_seast);
   }

      sleep(INTERVAL);
 }
```

Glossary

Symbols

V_{DD}

Positive supply voltage. 12, 20

V_{GS}

Gate-to-source voltage. Equivalent to gate bias. 12

V_{SS}

Negative supply voltage, i.e. ground. 12, 20

V_{th}

Threshold voltage. 12

A

availability

Resources remain accessable, operate, and provide a response. 5, 27, 123

B

BL

Bit Line (SRAM). 20

BSIM

Berkeley Short-channel IGFET Model – Transistor models used for integrated circuit design. 35

C

CMOS

Complementary Metal-Oxide-Semiconductor. 11

D

dependability

Reliability together with availability. 5, 27

Device layer

Concerning individual transistors. 33

DTM

Dynamic Thermal Management. 1, 6, 62

DVFS

Dynamic Voltage and Frequency Scaling. 6

Dynamic Thermal Management

Methods for controlling temperature during runtime at system level. 1, 124

E

electromigration

Aging effect due to erosion of metal interconnects through ion movement. 28

F

FEA

Finite-Element Analysis. 14

feature size

Characteristic length of a given manufacturing process, correspondes to half-pitch (half the distance between) identical features. 1, 126

FET

Field-effect transistor. 125

H

HCI

Hot Carrier Injection. 33

high-κ

Having a high (i.e. compared to SiO_2) dielectric constant $\kappa = \epsilon(\omega)/\epsilon_0$, where $\epsilon(\omega)$ is the (frequency dependent) absolute permittivity of the material and ϵ_0 the vacuum permittivity, i.e. κ determines the strength of a electric field through the material, compared to an electric field through a vacuum.. 12, 33, 35, 97

Hot Carrier Injection

Aging effect due to carriers accelerated by the horizontal electric field through the transistor channel, leading to carriers entering the oxide layer (either directly, through scattering due to impact ionization, or tunneling effects) and breaking $Si-h$ bonds.. 33, 124

I

ITRS

International Technology Roadmap for Semiconductors. 3

M

MOSFET

Metal-Oxide-Semiconductor FET. 35

N

NBTI

Negative Bias Temperature Instability. 6, 9, 27, 28, 31, 110, 125

Negative Bias Temperature Instability

Aging mechanism due to vertical electric field through gate oxide in a negative biased transistor (PMOS). 6, 9, 125

P

PBTI

Positive Bias Temperature Instability. 33

Positive Bias Temperature Instability

Like NBTI, but with positive biased NMOS transistors and, as a result, electron trapping instead of hole trapping.. 33, 125

R

reliability

Correctness in operation. 5, 27, 123

S

SNM

Static Noise Margin. 36

Static Noise Margin

SRAM cell resiliency to voltage noise. 36, 125

steady state temperature

The stable temperature reached when dissipation and heat generation reach equilibrium (assuming power consumption does not change). 7

T

TDDB

Time Dependent Dielectric Breakdown. 34

TDP

Thermal Design Power. 3, 13, 109

technology node

Manufacturing process characterized by feature size of a memory cell. 2

Thermal Design Power

Power dissipated under maximum workload stress. Guideline for required temperature management. 3, 109, 126

Through Silicon Via

Vertical interconnects in 3D layered architectures. Enable communication between layers. Thermal conduction towards packaging/heat sink is considerably higher through TSVs than through silicon. 5, 126

Time Dependent Dielectric Breakdown

Aging effect due to deterioration of oxide layer through trap generation. Eventually a conducting path is created between gate and bulk, resulting in device failure.. 34, 125

TSV

Through Silicon Via. 5, 82

V

Very Large Scale Integration

Process for creating circuits with several thousands of transistors. 126

VLSI

Very Large Scale Integration. 1

W

WL

Word Line (SRAM). 20