Institut für
Angewandte
Mathematik

Universität Karlsruhe (TH)
D-76128 Karlsruhe

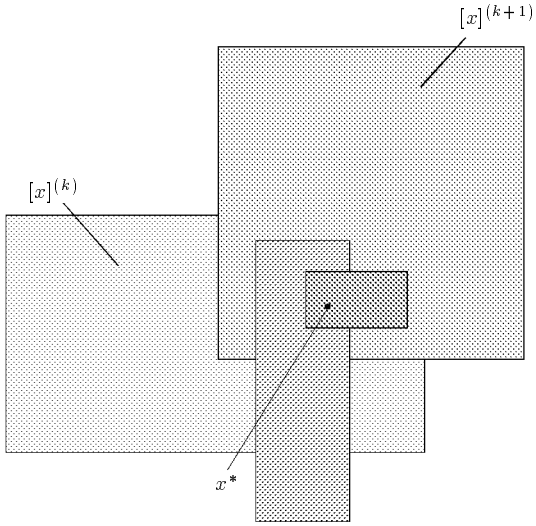
Die fünfte Gleitkommaoperation für top-performance Computer

oder

Akkumulation von Gleitkommazahlen und -produkten in Festkommaarithmetik

Ulrich Kulisch

**Forschungsschwerpunkt
Computerarithmetik,
Intervallrechnung und
Numerische Algorithmen mit
Ergebnisverifikation**



Bericht 3/1997

Impressum

Herausgeber:	Institut für Angewandte Mathematik Lehrstuhl Prof. Dr. Ulrich Kulisch Universität Karlsruhe (TH) D-76128 Karlsruhe
Redaktion:	Dr. Dietmar Ratz

Internet-Zugriff

Die Berichte sind in elektronischer Form erhältlich über

`ftp://iamk4515.mathematik.uni-karlsruhe.de`

im Verzeichnis: `/pub/documents/reports`

oder über die World Wide Web Seiten des Instituts

`http://www.uni-karlsruhe.de/~iam`

Autoren-Kontaktadresse

Rückfragen zum Inhalt dieses Berichts bitte an

Prof. Dr. Ulrich Kulisch
Institut für Angewandte Mathematik
Universität Karlsruhe (TH)
D-76128 Karlsruhe

E-Mail: ae42@rz.uni-karlsruhe.de

Die fünfte Gleitkommaoperation für top-performance Computer

oder

Akkumulation von Gleitkommazahlen und -produkten in Festkommaarithmetik

Ulrich Kulisch

Inhaltsverzeichnis

Zusammenfassung

Die fünfte Gleitkommaoperation: Seit Ende der 70er Jahre sind an verschiedenen Institutionen Programmiersprachen, zugehörige Compiler, aber auch Hardwareprozessoren entwickelt worden, welche außer den vier Gleitkommaoperationen noch eine fünfte Gleitkommaoperation bereitstellen, und zwar die Akkumulation von Gleitkommazahlen und -produkten in Festkommaarithmetik. In Gleitkommaarithmetik ist die Akkumulation eine sehr empfindliche Operation. Mit dieser neuen Operation können Skalarprodukte, Matrizenprodukte usw. in infinite precision Arithmetik immer völlig fehlerfrei berechnet werden, was eine Fehleranalyse bei diesen Operationen überflüssig macht. Es sind viele Algorithmen entwickelt worden, welche diese neue Operation systematisch anwenden. Bei anderen wird die Grenze der Anwendbarkeit durch Gebrauch dieser Operation hinausgeschoben. Darüber hinaus beschleunigt das optimale Skalarprodukt die Konvergenz von Iterationsverfahren. Dennoch wird immer wieder die Frage gestellt, ob diese Operation überhaupt notwendig ist. Die folgende Abhandlung versucht diese und damit zusammenhängende Fragen zu beantworten.

Abstract

The Fifth Floating-Point Operation: Programming languages, accompanying compilers and even hardware which, apart from the four floating-point operations, provide an additional fifth floating-point operation, namely the accumulation of floating-point numbers and products in fixed-point arithmetic, have been developed at many institutions since the end of the seventies. Accumulation of numbers is the most sensitive operation in floating-point arithmetic. By that operation scalar products of floating-point vectors, matrix products etc. can be computed without any error in infinite precision arithmetic, making an error analysis for those operations superfluous. Many algorithms applying that operation systematically have been developed. For others the limits of applicability are extended by using that fifth operation. Furthermore, the optimal dot product speeds up the convergence of iterative methods. Nevertheless, the question whether that operation is really necessary is often asked. The following treatise aims at answering that and related questions.

1 Vorgeschichte

Gleitkommaarithmetik wird seit den 40er und 50er Jahren verwendet (Zuse Z3, 1941). Die damalige Technologie (elektromechanische Relais, Elektronenröhren) war kompliziert und teuer. Man war daher damit zufrieden, als Ergebnis der Verknüpfung zweier Gleitkommazahlen wieder eine einigermaßen korrekte Gleitkommazahl zu erhalten bzw. abzuliefern. Eine Fehleranalyse für kompliziertere Ausdrücke wurde dem Benutzer überlassen.

Vor etwa 25 Jahren schälte sich im Rahmen der Entwicklung einer allgemeinen mathematischen Theorie der Rechnerarithmetik [19], [20] die Einsicht heraus, daß es zweckmäßig ist, neben den vier Grundrechenoperationen im Rechner auch eine Operation zur Berechnung von Skalarprodukten mit maximaler Genauigkeit nach dem Prinzip des Semimorphismus bereitzustellen. Die ersten Lösungsvorschläge waren algorithmischer Natur. Vor etwa 20 Jahren erkannte man dann, daß sich das Problem sowohl in Software als auch in Hardware elegant lösen läßt, indem man die Komponentenprodukte in einem Festkommaregister aufsummiert, welches den gesamten Gleitkommabereich zum doppelten Exponentenbereich abdeckt. Fast ebenso alt ist die Forderung, hierfür einen Koprozessor in VLSI-Technik zu entwickeln und zu bauen.

In einem Gleitkommasystem ist sowohl die Anzahl der Mantissenziffern als auch der Exponentenbereich endlich. Das Festkommaregister ist daher ebenfalls endlich. In Abhängigkeit vom verwendeten Datenformat benötigt man dafür etwa 1000 bis 4000 bits. Es besteht also die paradox anmutende Situation, daß sich Skalarprodukte von Gleitkommavektoren selbst mit Millionen von Komponenten in einem relativ kleinen Registerspeicher auf der Arithmetikeinheit auf volle Genauigkeit in infinite precision Arithmetik berechnen lassen. Das Skalarprodukt ist eine ganz fundamentale Operation in der linearen Algebra. Ein immer korrektes Skalarprodukt eliminiert viele Rundungsfehler in numerischen Rechnungen. Es stabilisiert diese Rechnungen und beschleunigt sie gleichzeitig.

In der Folgezeit wurde die Funktionalität des optimalen Skalarproduktes mit verschiedenen Rundungen mittels einer Softwaresimulation via *integer* Arithmetik in mehrere Programmiersprachen (ACRITH-XSC [15], PASCAL-XSC [16], C-XSC [17]) eingebaut. Zusammen mit der Intervallarithmetik gelang es dann in kurzer Zeit, für eine große Anzahl von Problemen der numerischen Mathematik Algorithmen zu entwickeln, bei denen der Rechner sowohl die Existenz und Eindeutigkeit einer Lösung nachweist als auch hochgenaue Schranken für die Lösung berechnet [9], [10], [18]. Bei iterativen Verfahren erreicht man eine gewünschte Ergebnisgenauigkeit häufig mit weniger Iterationsschritten, wenn man alle Skalarprodukte in infinite precision Arithmetik mit voller Genauigkeit oder mit nur einer einzigen Rundung am Schluß ausführt [23], [25], [26]. Dennoch gibt es bis heute keinen kommerziellen VLSI-Koprozessor zur Berechnung des optimalen Skalarproduktes.

Ein Grund hierfür besteht darin, daß vor 20 Jahren die Technologie wie auch die Entwurfswerkzeuge der Komplexität des Problems noch nicht gewachsen waren und eine erzwungene Lösung sehr teuer geworden wäre (einige zig-Mio. DM). Ein ausgewogener Entwurf hängt nicht nur von dem mathematischen Lösungsverfahren für das Problem ab, sondern auch von vielen Randbedingungen wie etwa der Busbreite für die Datenversorgung, der Ankopplung an den Hauptrechner, der Abstimmung mit der Arithmetikeinheit des Hauptrechners und der Architektur und Geschwindigkeit des Hauptprozessors. Auch die Entwurfswerkzeuge und die Fertigungstechnologie spielen eine große Rolle. Diese vielfachen Abhängigkeiten haben auch kompetente und interessierte Wissenschaftler und Hersteller immer wieder davon abgehalten, eine Lösung in VLSI-Technik anzugehen.

Im Jahre 1987 hat die GAMM[1] eine Resolution veröffentlicht, welche die mathematisch nicht korrekte Ausführung von Matrix- und Vektoroperationen in Vektorrechnern kritisiert und Besserung verlangt. Im Jahre 1993 haben dann die GAMM und die IMACS[2] ein *Proposal for Accurate Floating-Point Vector Arithmetic* [6] verabschiedet, in dem u. a. eine mathematisch korrekte Ausführung von Matrix- und Vektoroperationen, insbesondere des Skalarproduktes, in allen Rechnern verlangt wird. Das Proposal enthält auch Lösungsvorschläge (siehe dazu auch [1], [5], [12], [28]). Im Jahre 1995 hat auch die IFIP-Working Group 2.5 on Numerical Software diesem Proposal zugestimmt. Inzwischen wurde es zu einer *EU-Guideline*.

[1] GAMM = Gesellschaft für Angewandte Mathematik und Mechanik

[2] IMACS = International Association for Mathematics and Computers in Simulation

2 Stand der Entwicklung

Im Jahre 1992 gelang es drei Instituten aus Karlsruhe, Hamburg und Stuttgart[3], von der VW-Stiftung Fördermittel (ca. 900 TDM) für ein Projekt zu bekommen, welches den Bau eines VLSI-Vektorarithmetik-Koprozessors für den PC vorsah. Das Projekt hatte eine Laufzeit von zwei Jahren (01.01.1993 bis 31.12.1994). In Karlsruhe arbeiteten drei, in Stuttgart zwei und in Hamburg ein Mitarbeiter an dem Projekt. Es wurde ein VLSI-Vektorarithmetik-Koprozessor XPA 3233 entwickelt und in der 0,8 μm CMOS Gate Array Technologie des IMS gefertigt. Der Prozessor kann über den derzeit verfügbaren 33 MHz 32 bit-breiten PCI-Bus an Personal Computer angeschlossen werden. Er ist der erste Vektorarithmetik-Koprozessor auf einem Chip. Der Prozessor kann beispielsweise unter PASCAL-XSC [16] von den dort vorhandenen Operatorsymbolen direkt angesprochen werden. Im Rechner wirkt er wie ein Katalysator im doppelten Sinne. Er beschleunigt die Rechnung und eliminiert viele unnötige Rundungsfehler. Die folgenden drei Schritte werden in einer Pipeline (Fließbandverarbeitung) stets gleichzeitig ausgeführt: Lesen zweier Faktoren für das nächste Produkt, Ausführung einer Multiplikation und Akkumulation eines Produktes. Im Vergleich mit herkömmlicher Gleitkommaarithmetik erzielt man deshalb bei gleicher Taktfrequenz mindestens die doppelte Ausführungsgeschwindigkeit. Zum Zeitpunkt der Fertigstellung hat der XPA Skalarprodukte schneller berechnet als alle anderen auf dem Markt erhältlichen Mikroprozessoren in herkömmlicher Gleitkommaarithmetik, obwohl diese in der Regel in einer wesentlich feineren Technologie gefertigt waren. Im Gegensatz zu diesen liefert der XPA 3233 immer das richtige Ergebnis mit voller Genauigkeit oder mit nur einer einzigen Rundung am Schluß. Gegenüber den Softwareimplementierungen des optimalen Skalarproduktes in PASCAL-XSC, ACRITH-XSC und C-XSC bringt er eine Geschwindigkeitssteigerung um einen zweistelligen Faktor [5].

3 Begriffsbestimmungen

Der Begriff *precision* bezieht sich im folgenden auf die in einer Rechnung mitgeführte Stellenzahl, *accuracy* auf die Genauigkeit im Endergebnis. Unter *double* oder ausführlicher *double precision* wird im folgenden ein 64 bit Gleitkommawort, beispielsweise im IEEE-Datenformat im Sinne des IEEE-Arithmetik-Standards 754 [3], verstanden. *Quadruple* oder ausführlicher *quadruple precision* bezeichnet ein 128 bit Gleitkommawort. *Langer Akkumulator* bezeichnet ein Festkommawort, welches ausreicht, um Produkte von double precision Gleitkommazahlen immer korrekt, d. h. ohne Informationsverlust, aufzusummieren. Der zugehörige Akkumulationsprozeß wird auch einfach als Festkommaakkumulation bezeichnet. *Quadruple Akkumulator* bezeichnet ein Computerregister, welches ausreicht, um quadruple precision Gleitkommazahlen in Gleitkommaarithmetik mit einem Rundungsfehler von 1/2 oder 1 ulp (unit in the last place) aufzusummieren.

[3] Institut für Angewandte Mathematik der Universität Karlsruhe (IAM) (Prof. U. Kulisch)
Arbeitsbereich Technische Informatik, TU Hamburg-Harburg (TUHH) (Prof. T. Teufel)
Institut für Mikroelektronik Stuttgart (IMS) (Prof. B. Höfflinger)

In den XSC-Sprachen [15], [16], [17] steht außer dem numerischen Datentyp *real* und vielen weiteren numerischen Datentypen noch ein Feldtyp *staggered (staggered precision)* zur Verfügung. Als Komponententyp kann der Typ *real (double precision)* oder *interval (double precision interval)* auftreten. Eine Variable vom Typ *staggered* besteht aus einem Feld (array) von Variablen des Komponententyps. Sie erlaubt die Speicherung von Daten mit mehrfacher Genauigkeit. Ihr Wert ist die Summe der Komponenten. Summen von Variablen des Typs *staggered* lassen sich mit dem langen Akkumulator — Produkte über das optimale Skalarprodukt — leicht berechnen. Quotienten werden iterativ berechnet. Der Typ *staggered* wird durch eine globale Variable *stagprec* gesteuert. Im Falle $stagprec = 1$ ist der Typ *staggered* identisch mit dem Komponententyp. Ist beispielsweise aber $stagprec = 4$, so besteht jede Variable dieses Typs aus einem Feld von vier Variablen des Komponententyps. Ihr Wert ist die Summe der Komponenten. Die Variable ist i. allg. von vierfacher Genauigkeit des Komponententyps. Die globale Variable *stagprec* kann an beliebigen Stellen im Programm vergrößert oder verkleinert werden. Die Standardfunktionen für den Typ *staggered* stehen in Karlsruhe für die Komponententypen *real* und *interval* bereits zur Verfügung (siehe die Artikel von K. Braune und W. Krämer in [22]). Ist $stagprec = 2$, so hat man es im Falle des Komponententyps *real* mit einem Datentyp zu tun, welcher gelegentlich auch als *double-double* oder als *quadruple* bezeichnet wird.

4 Eine oft zitierte Bemerkung von C. F. Gauß

Von C. F. Gauß stammt die Bemerkung: *Der Mangel an mathematischer Bildung gibt sich durch nichts so auffallend zu erkennen, wie durch maßlose Schärfe im Zahlenrechnen.*

Mancher Numeriker sieht in dieser Bemerkung eine Aufforderung, möglichst viele Probleme mit einer möglichst kurzen Wortlänge (precision) zu rechnen. Er ist stolz darauf, daß er *seine Probleme* zufriedenstellend lösen kann, indem er die gesamte Rechnung einschließlich aller Skalarprodukte in double, womöglich sogar in single precision Gleitkommaarithmetik ausführt. Wenn dies zum Erfolg führt, ist diese Vorgehensweise natürlich berechtigt, und es ist überhaupt nichts dagegen einzuwenden.

Häufig wird aus der Bemerkung aber auch der Schluß gezogen, man bräuchte ein immer korrektes Skalarprodukt im Rechner gar nicht und sollte es folglich auch nicht bereitstellen; die herkömmliche Art, Skalarprodukte in Gleitkommaarithmetik zu berechnen, wäre einfacher, da dabei *unnütze Ziffern* gar nicht erst mitgeführt würden. Diese Folgerung ist völlig falsch. Sie beruht auf einer unzureichenden Vertrautheit und Kenntnis der heute verfügbaren Technologie und Implementierungstechniken seitens der Mathematiker. Der Ingenieur andererseits, welcher diese Techniken beherrscht, überblickt in der Regel nicht die Konsequenzen für die Mathematik.

Die Bemerkung von C. F. Gauß ist heute mehr als 150 Jahre alt. Sie paßt in die damalige Zeit, als jede Rechnung von Hand ausgeführt werden mußte und jede Ziffer, die man nicht mitführte, eine enorme Arbeitserleichterung bedeutete. Das langsame Rechnen von Hand erlaubte es in der Regel, eine die Rechnung begleitende Fehleranalyse oder wenigstens eine Fehlerschätzung durchzuführen.

Heute liegen ganz andere Verhältnisse vor, welche man damals in keiner Weise übersehen oder auch nur ahnen konnte. Schnelle Prozessoren sind heute in der Lage, eine Milliarde arithmetische Operationen in der Sekunde auszuführen. Diese Zahl übersteigt das Vorstellungsvermögen eines Benutzers. Die zur Verfügung stehende Technologie (mehrere Millionen Transistoren auf einem Chip) ist extrem leistungsfähig. Sie erlaubt Lösungen, von denen selbst ein geübter Benutzer von Rechnern nichts ahnt.

Für alle Arten von Rechnern wie Personal Computer, Workstations, Großrechner und Supercomputer gibt es heute einfache Schaltungen für die Berechnung des **optimalen** Skalarproduktes, bei denen für die Arithmetik selbst praktisch keine Rechenzeit mehr benötigt wird. In einer Pipeline läßt sich die Arithmetik in der Zeit ausführen, welche benötigt wird, um die Daten in die Arithmetikeinheit zu lesen. Das bedeutet u. a., daß keine andere Art, Skalarprodukte zu berechnen, schneller sein kann — auch nicht diejenige in reiner Gleitkommaarithmetik. Technisch ist der Prozeß der immer korrekten Akkumulation von Skalarprodukten in einem Festkommaregister zwar anders, aber keineswegs aufwendiger oder komplizierter als eine Hardwareakkumulation in Gleitkommaarithmetik. Beide Arten der Akkumulation arbeiten mit sehr ähnlichen Grundbausteinen der Arithmetik wie Shifter, Multiplizierer, Addierer und Rundungseinheit — allerdings in unterschiedlicher Anordnung. Die Festkommaakkumulation benötigt zudem lediglich noch einen kleinen lokalen Speicher auf der Arithmetikeinheit. Viele bei der Gleitkommaakkumulation anfallende Zwischenschritte wie Normalisieren, Runden, Zusammensetzen zu einer Gleitkommazahl, Speichern, Wiederzerlegen in Mantisse und Exponent für die nächste Operation fallen bei der Festkommaakkumulation gar nicht an.

Eine relativ komplexe arithmetische Operation wie das Skalarprodukt mit Gleitkommazahlen, welche aufgrund der heutigen Technologie mit mäßigem technischem Aufwand immer korrekt ausgeführt werden kann, sollte man tatsächlich auch immer korrekt ausführen. C. F. Gauß würde dies heute jedenfalls so machen. Eine Fehleranalyse erübrigt sich für diese zusätzliche Grundoperation dann ein für allemal, und auf verschiedenen Rechnerplattformen erhält man bei ihrer Ausführung stets identische Ergebnisse. Rechneranwendungen sind heute von unübersehbarer Vielfalt. Jede Diskussion darüber, wo diese zusätzliche Operation Vorteile bringt und wo nicht, ist nutzlos und müßig. Durch Ergänzung der herkömmlichen Arithmetikeinheit um das optimale Skalarprodukt entsteht ein wesentlich leistungsfähigerer und mächtigerer Rechner. Die neue, immer absolut zuverlässige Operation läßt sich bei der Entwicklung von Algorithmen an vielen Stellen vorteilhaft verwenden.

5 Probleme mit ungenauen und mit genauen Daten

Seit mehr als 50 Jahren wird in der Numerik fast ausschließlich Gleitkommaarithmetik verwendet. Dies hat die Denkweise geprägt. Häufig fällt es daher schwer, sich von gewohnten und geübten Denkschemata zu lösen. Während dieser Zeit war die sogenannte Rückwärtsfehleranalyse häufig die einzig praktikable Art, eine Fehlerbetrachtung bei numerischen Algorithmen durchzuführen. Dabei interpretiert man die berechnete Lösung eines Problems als die korrekte Lösung zu einem Problem mit geänderten

Ausgangsdaten. Liegen die letzteren nahe bei den Daten des gegebenen Problems, so spricht man von einem gutartigen Algorithmus oder einem gutkonditionierten Problem.

Dieses Denkschema impliziert die Vorstellung, daß man eigentlich nie das gegebene Problem löst, sondern immer eines aus dessen Nachbarschaft oder Umgebung. Dies führt zu der Annahme, es komme auf die absolute Genauigkeit in den Daten gar nicht an, man habe es im Grunde immer mit ungenauen Daten zu tun. Bei gutartigen Problemen ist diese Annahme durchaus gerechtfertigt. Tatsächlich stützen sich Berechnungen auch nicht selten auf unsichere Annahmen. In die Berechnung gehen etwa Meßwerte ein, deren Genauigkeit sich nicht angeben läßt, oder das mathematische Modell mußte, um eine Berechnung überhaupt zu ermöglichen, stark vereinfacht werden. Bei der numerischen Behandlung von Differentialgleichungen ist der Diskretisierungsfehler häufig wesentlich größer als die Rechenungenauigkeiten. Bei vielen Anwendungen werden sehr viele Berechnungen ähnlicher Art durchgeführt. Aufgrund der Erfahrung oder von experimentellen Messungen hat man Vertrauen zur einfachen Gleitkommarechnung.

Bei solchen Problemen läßt sich sehr häufig mit Hilfe der Rückwärtsfehleranalyse mathematisch zeigen, daß einfache Gleitkommaarithmetik zur Berechnung der Lösung ausreicht. Wegen der großen Häufigkeit des Auftretens derart gutartiger Probleme fehlt daher bei vielen Anwendern das Problembewußtsein für eine hochpräzise Rechnung. Jede Verbesserung der bisherigen Arithmetik wird als überflüssig betrachtet: ein genaues Skalarprodukt wurde bisher nicht verwendet und wird folglich auch nicht benötigt und nicht gefordert. Aus Unkenntnis wird dabei übersehen, daß das optimale Skalarprodukt in Hardware schneller als die üblicherweise verwendete Summationsschleife in Gleitkommaarithmetik abläuft, so daß auch gutartige Berechnungen davon profitieren können.

Nun sind aber keineswegs alle numerischen Probleme von der beschriebenen *harmlosen* Art. Tatsächlich ist dies nur eine Seite der Medaille. Auf der anderen befindet sich die große Klasse der Probleme mit exakten Daten. Hierher gehören auch Probleme mit kleinen, aber sicheren Schranken für die Daten. In der mathematischen Modellbildung geht man beispielsweise in der Regel von exakten Eingabedaten aus. Das Modell kann nur dann systematisch weiter verbessert werden, wenn der Rechenfehler weitgehend ausgeschlossen werden kann. Auch bei vielen innermathematischen Problemen hat man es mit exakten Eingabedaten zu tun, z. B. für die Gewichte und Stützstellen von Quadratur- und Kubaturformeln, für die Koeffizienten von Diskretisierungsformeln, für die Koeffizienten bei der Berechnung von Polynomnullstellen, von Eigenwerten von Matrizen oder von Differentialoperatoren. In der Computeralgebra behandelt man in der Regel Probleme mit exakten Eingabedaten. Die Rechnung wird nach Möglichkeit exakt, d. h. mit sehr langen Wortlängen, ausgeführt. Dies kann sehr große Rechenzeiten erforderlich machen. Eine numerische Rechnung mit Gleitkommaarithmetik, welche sichere und hinreichend genaue Schranken für die Lösung des Problems liefert, wird i. allg. wesentlich schneller ablaufen.

Die Rechner werden immer schneller. Mit steigender Rechenleistung wachsen auch die zu behandelnden Probleme. An Stelle zweidimensionaler möchten die Benutzer gerne dreidimensionale Probleme lösen. Eine Diskretisierung von 100 Schritten in jede von drei Koordinatenrichtungen führt auf ein Gleichungssystem mit einer Million Unbekannten. Der Gauß-Algorithmus benötigt $\mathcal{O}(n^3)$ Operationen. Ist z. B. $n = 10^6$, so

ist n^3 so groß, daß selbst ein GIGAFLOPS Computer viele Jahre benötigt, um ein solches System zu lösen. Bei großen Gleichungssystemen ist man daher auf iterative Löser angewiesen. Man hofft, mit weniger als n^3 arithmetischen Operationen eine geeignete Approximation der Lösung zu erhalten. Bei iterativen Methoden (Jacobi-Verfahren, Gauß-Seidel-Verfahren, Relaxationsverfahren, cg-Verfahren, Krylow-Raum-Methoden, Multigrid-Verfahren und anderen wie etwa das QR-Verfahren zur Berechnung von Eigenwerten) ist die Matrix-Vektormultiplikation die zentrale Operation. Sie besteht aus einer Anzahl von Skalarprodukten. Es ist wohlbekannt, daß *finite precision* Arithmetik die Konvergenz von Iterationsverfahren verschlechtert [7], [8], [11], [23], [25], [26], [29]. Ein Iterationsverfahren, welches im Reellen gegen die Lösung konvergiert, konvergiert bei Rechnung mit *finite precision* Arithmetik i. allg. langsamer. Es kann sogar divergieren. Die Festkommaakkumulation von Skalarprodukten von Gleitkommavektoren ist absolut fehlerfrei. Sie ist darüber hinaus sogar noch schneller als eine konventionelle Berechnung in Gleitkommaarithmetik. Eine fehlerfreie Berechnung von Skalarprodukten beschleunigt zudem die Konvergenz bei vielen Iterationsverfahren. Bei vielen Anwendungen liegen die Problemdaten exakt vor (Koeffizienten von Diskretisierungsformeln). Sie werden in jedem Iterationsschritt aufs neue in die Rechnung hineingefüttert. Eine hochgenaue Rechnung erscheint daher als besonders sinnvoll. Die Komponenten eines Matrix-Vektorproduktes werden dabei in eine Variable vom Typ *staggered* auf zwei-, drei- oder vierfache Genauigkeit gerundet und so im Algorithmus weiterverwendet. Die Laufzeit des Algorithmus wird dadurch nur unwesentlich erhöht.

Der modernen Numerik stehen mit der Intervallrechnung und dem optimalen Skalarprodukt neue Werkzeuge zur Verfügung, welche neuartige Lösungsverfahren für numerische Fragestellungen ermöglichen, z. B. die neuen Methoden der globalen Optimierung und die automatische Differentiation [1], [9], [10], [18]. Bei der Rückwärtsmethode der automatischen Differentiation läßt sich mit dem optimalen Skalarprodukt der Speicheraufwand drastisch reduzieren. Dabei wird das Skalarprodukt als Grundoperation in den Vektorräumen aufgefaßt [28].

Häufig erlauben diese Werkzeuge zudem eine produktive Vorwärtsfehleranalyse. Beispiele: numerische Quadratur und Kubatur [1], numerische Integration gewöhnlicher Differentialgleichungen [1], Anwendungen in der Computergeometrie wie die sichere Berechnung des Schnittes zweier Flächen [27] oder die sichere Beantwortung von Koinzidenzfragen [22].

Ferner ermöglichen es diese neuen Werkzeuge häufig, eine à posteriori Fehleranalyse mit dem Rechner durchzuführen. Dies wird gelegentlich auch als automatische Ergebnisverifikation oder als Ergebnisvalidierung bezeichnet. Beispiele: hochgenaue Einschließung der Lösung linearer und nichtlinearer Gleichungssysteme [9], [10], [18], [21], lineare und nichtlineare Optimierungsprobleme [18], [22]. Schlägt der abschließende Verifikationsschritt fehl — was nur sehr selten vorkommt, dann aber auch vom Rechner selbst festgestellt wird —, so führt eine erneute Berechnung mit höherer Rechengenauigkeit (precision) häufig zum Ziel. Der Datentyp staggered, dessen arithmetische Operationen über das optimale Skalarprodukt realisiert werden, ist das geeignete Werkzeug hierzu [1], [18].

Die Frage, wie viele Stellen eines Defektes bei Rechnung in single, double oder extended precision garantiert werden können, ist in der Vergangenheit sorgfältig untersucht worden. Mit dem optimalen Skalarprodukt läßt sich der Defekt immer auf volle Genauigkeit und zudem schneller berechnen.

Viele Problemlöseroutinen der XSC-Sprachen (PASCAL-XSC, ACRITH-XSC, C-XSC, FORTRAN-XSC) sind auch verfügbar für (kleine) Intervalle als Daten. Es stehen auch Routinen zur Verfügung, welche es erlauben, Daten als Intervalle einzulesen. Falls ein Benutzer davon keinen Gebrauch macht und alle Daten als Punktdaten (computer reals) einliest, kann man davon ausgehen, daß er der Meinung ist, daß seine Daten in der Maschine damit genau genug beschrieben sind. Hochgenaue Problemlöser für Probleme mit exakten Punktdaten (von unendlicher Genauigkeit) sind daher durchaus sinnvoll. Zur Lösung von Punktproblemen mit garantierter hoher Genauigkeit ist die Festkommaakkumulation im langen Akkumulator häufig erforderlich und der Gleitkommaakkumulation deutlich überlegen. Beispiel: Eigenwerte von Zielke-Matrizen lassen sich mit dem langen Akkumulator immer richtig berechnen.

6 Akkumulation von Gleitkommazahlen und -produkten in quadruple precision Gleitkommaarithmetik

Das Skalarprodukt ist eine ganz zentrale Operation in der Numerik. Es tritt auf in der komplexen Arithmetik, in der Matrix- und Vektorarithmetik, bei Defektkorrekturmethoden, in der Bereichsreduktion bei der Berechnung von Standardfunktionen, in mehrfach genauer Arithmetik und an vielen anderen Stellen. Mit dem optimalen Skalarprodukt lassen sich alle Operationen in den üblichen Produkträumen der Numerik — insbesondere für Vektoren und Matrizen über den reellen bzw. komplexen Gleitkommazahlen oder -intervallen — nach dem Prinzip des Semimorphismus [19], [20] und damit immer maximal genau bereitstellen. Die entsprechenden reellen und komplexen Vektorräume und die zugehörigen Intervallräume werden dadurch bezüglich der algebraischen Struktur bestmöglich und bezüglich der Ordnungsstruktur weitestgehend strukturerhaltend auf die entsprechenden Räume über Gleitkommazahlen abgebildet.

Neuere Programmiersprachen wie Fortran 90 oder C++ stellen das Skalarprodukt als Grundoperation (Operatoren *matmul* und *dotproduct*) zur Verfügung. Ähnlich läßt sich das optimale Skalarprodukt beispielsweise auch in Tabellenkalkulationsprogramme, Computeralgebrapakete usw. einarbeiten.

In den XSC-Sprachen und beim XPA 3233 werden Zahlen und Produkte von Gleitkommazahlen in ein langes Festkommawort (langer Akkumulator) akkumuliert. Bei dieser Vorgehensweise geht keinerlei Information verloren. Oft wird die Frage gestellt, ob dies wirklich notwendig ist oder ob es nicht ausreicht, die doppelt langen Produkte zweier Gleitkommazahlen als quadruple precision Zahlen in einem für dieses Datenformat bereitgestellten Addierer, welcher nur geringfügig breiter ist als dieses Datenformat, zu akkumulieren, da man das Endergebnis letztlich fast immer in eine double precision Variable rundet. Es sollen jetzt diese beiden Arten der Akkumulation miteinander verglichen werden. Die Waage neigt sich eindeutig auf die Seite der Festkommaakkumulation.

Die Berechnung des Skalarproduktes im langen Akkumulator ist die einzige Gleitkommaoperation, welche immer ein richtiges Ergebnis liefert. Eine Akkumulation von Produkten von Gleitkommazahlen im quadruple Akkumulator kann demgegenüber gelegentlich ein falsches Ergebnis liefern: der Benutzer muß sich bei jeder Ausführung einer solchen Operation Gedanken über den Rechenfehler machen. Bei der Berechnung des Skalarproduktes im langen Akkumulator braucht man nie eine Fehlerabschätzung durchzuführen — das Ergebnis ist immer richtig.

Der lange Akkumulator ist wesentlich einfacher zu handhaben als ein quadruple Akkumulator, weil der lange Akkumulator mit nur einem Datenformat, nämlich double, arbeitet. Bei einer quadruple Akkumulation hat man es hingegen mit zwei verschiedenen Datenformaten zu tun. Für die Berechnung des optimalen Skalarproduktes in Festkommaakkumulation im langen Akkumulator braucht man nur ganz wenige zusätzliche Befehle in der Maschine. Bei einer quadruple Akkumulation muß man auch Befehle für gemischte Operationen vorsehen. Man braucht daher wesentlich mehr OP Codes.

Nach allen bisherigen Erfahrungen (Assembler, Mikrocode, Bit-Slice-Technologie, VLSI) ist die Festkommaakkumulation von Produkten von double precision Variablen schneller und einfacher als die Gleitkommaakkumulation in double precision. Eine Akkumulation von doppelt langen Produkten von double precision Variablen in quadruple precision wird aller Voraussicht nach noch einmal langsamer ablaufen. Bei der Festkommaakkumulation wird nur ein shift ausgeführt, und es wird akkumuliert. Der Exponent stellt die Adresse für die Addition des Produktes dar. Bei der Gleitkommaakkumulation müssen viele Zwischenschritte zusätzlich ausgeführt werden. Es fällt an: Dekomposition der Operanden, Schieben, Addieren, Normalisieren, Runden, Zusammensetzen zu einer Gleitkommazahl, Dekomposition in Mantisse und Exponent für die nächste Operation und vieles andere mehr. Bei der Festkommaakkumulation fällt demgegenüber nur an: Schieben und Addieren. Gerundet wird nur einmal am Schluß.

Da bei der Berechnung von Skalarprodukten die zu akkumulierenden Produkte von Gleitkommazahlen nur doppelt lang sind, ist der für die Festkommaakkumulation benötigte Addierer von der gleichen Größenordnung wie derjenige für die Akkumulation in quadruple precision. Der *lange Akkumulator* braucht nur als lokaler Speicher auf der Arithmetikeinheit gehalten zu werden. Im Falle des Datenformates double precision des IEEE Standards 754 besteht er aus 67 Worten à 64 bit [1], [5], [12], [30], [31].

Bei der Festkommaakkumulation im langen Akkumulator hat man den doppelten Exponentenbereich und zusätzlich einige Schutzziffern zur Verfügung. Ein Überlauf oder Unterlauf ist im Skalarprodukt ausgeschlossen. Bei quadruple precision ist zunächst das Datenformat nicht eindeutig definiert. Es kann ein double-double Wort sein, aber auch ein echtes quadruple Format ist möglich. Bei verschiedenen Herstellern können daher bei Akkumulation in quadruple precision durchaus verschiedene Ergebnisse anfallen. Wenn quadruple nur als double-double aufgefaßt wird, steht ausschließlich der einfache Exponentenbereich zur Verfügung, und man kann leicht Über- oder Unterlauf im Exponenten bekommen. In der Rechnung treten dann viele plus oder minus infinity oder NaNs auf, welche praktisch keine Information enthalten. Da quadruple nicht genau definiert ist, handelt man sich, wenn man eine quadruple Arithmetik bereitstellt, Übertragungsprobleme für Daten zwischen Maschinen verschiedener Hersteller ein. Bei der Festkommalösung treten solche Probleme nicht auf, da es nur ein

Datenformat, nämlich double precision, gibt. Eine quadruple Arithmetik wird damit als einfacher Fall einer mehrfach genauen Arithmetik realisiert, und zwar auf der Basis des Datenformates double precision. Eine mehrfach genaue Zahl (vom Typ staggered) wird als array von double precision Variablen dargestellt. Der Wert einer Variablen vom Typ staggered ist die Summe der Komponenten. Die Addition solcher Zahlen erfolgt im *langen Akkumulator*. Die Multiplikation ist nichts anderes als ein Skalarprodukt.

Ein scheinbarer Vorteil der Akkumulation von double precision Produkten in quadruple precision besteht darin, daß viele Benutzer glauben, sich sofort etwas darunter vorstellen zu können. Diese Vorstellung kann und wird allerdings von Benutzer zu Benutzer verschieden ausfallen. Normalerweise wird ein Benutzer zunächst glauben, daß ein volles Datenformat quadruple precision mit allen vier Grundoperationen zur Verfügung steht. Dadurch entsteht ein Druck auf die Hersteller, wesentlich mehr als unbedingt notwendig in die Hardware hineinzupacken — eben eine volle quadruple Arithmetik einschließlich der zugehörigen Standardfunktionen. Damit wird weit über das Notwendige, Nützliche und Wesentliche hinausgegangen.

Der Typ staggered als ein array von double precision reals ist wesentlich leistungsfähiger als der Typ quadruple precision, weil er praktisch eine dynamische precision bereitstellt.

7 Wieviel lokalen Speicher braucht man auf der Arithmetikeinheit?

Braucht man für spezielle Anwendungen mehrere lange Akkumulatoren auf der Arithmetikeinheit? Mit der gleichen Argumentation, welche bei reellen Punktproblemen einen *langen Akkumulator* fordert, verwendet man bei komplexen Punktproblemen zweckmäßigerweise zwei *lange Akkumulatoren*. Dies halbiert den Aufwand bei der Datenübertragung in die Arithmetikeinheit. Dann lassen sich komplexe Skalarprodukte immer korrekt ausführen. Der zweite Akkumulator vergrößert nur den lokalen Speicher auf der Arithmetikeinheit. Die Steuerung und der Befehlssatz brauchen nur geringfügig ergänzt zu werden. Wenn auf der Arithmetikeinheit genügend Speicher für zwei lange Akkumulatoren zur Verfügung steht, kann man auch Intervallskalarprodukte immer unter Verwendung von zwei langen Akkumulatoren aufsummieren. Das Ergebnis wird dabei in der Regel nicht wesentlich besser ausfallen, als wenn man die beiden Schranken des Ergebnisses in quadruple precision aufsummiert. Es treffen aber wieder alle Nachteile zu, welche bereits oben für quadruple precision aufgelistet wurden. Die Akkumulation von Intervallskalarprodukten in quadruple precision wäre daher aller Voraussicht nach langsamer als mit zwei langen Akkumulatoren; außerdem werden sich die bei den Additionen in quadruple precision anfallenden Rundungsfehler in den Schranken akkumulieren, während bei der Akkumulation von Intervallskalarprodukten im langen Akkumulator keine Rundungsfehler auftreten und somit das Ergebnis in der Regel genauer ausfällt als eine entsprechende Akkumulation in quadruple precision. Bei Festkommaakkumulation werden die Schranken des Ergebnisses nur ganz am Schluß der Rechnung ein einziges Mal gerundet.

8 Optimales Skalarprodukt bei Multi-User-Systemen

Bei der Festkommaakkumulation im langen Akkumulator tritt stets die Frage auf, wie Kontextwechsel behandelt werden sollen. Diese Frage läßt sich dadurch beantworten, daß man hinreichend viel lokalen Speicher für mehrere lange Akkumulatoren auf der Arithmetikeinheit bereitstellt. Der hyperstone, ein deutscher DSP (Digital Signal Prozessor), stellt auf der Arithmetikeinheit 8 KByte lokalen Speicher zur Verfügung. Dies reicht aus für etwa sieben lange Akkumulatoren für das IEEE-Datenformat double precision. Bei Kontextwechsel, welcher einen langen Akkumulator benötigt, wird einfach ein neuer Akkumulator eröffnet. Dadurch erledigt sich das Auslagern von Zwischenergebnissen. In [5] werden verfeinerte Techniken zur Handhabung des optimalen Skalarproduktes in Multi-User-Systemen behandelt.

9 Bedeutet der lange Akkumulator eine Rückkehr zur Festkommaarithmetik?

Die Antwort auf diese Frage lautet *nein*. Einfache Gleitkommaoperationen werden nach wie vor in Gleitkommaarithmetik ausgeführt. Auch die Datenhaltung erfolgt nach wie vor in Gleitkomma. Die Vorteile der Gleitkommaarithmetik, wie der Wegfall lästiger Skalierungen bzw. deren Automatisierung, bleiben daher erhalten. Nur die Akkumulation, die empfindlichste Operation in Gleitkomma, wird in Festkommaarithmetik ausgeführt. Die Festkommaakkumulation von Gleitkommazahlen und -produkten verläuft immer absolut fehlerfrei. Die Erweiterung der herkömmlichen Gleitkommaarithmetik um die fünfte Gleitkommaoperation, die Akkumulation von Gleitkommazahlen und -produkten in Festkommaarithmetik, ist eine ideale Kombination von Fest- und Gleitkommaarithmetik, welche die Vorteile beider verbindet. Durch die Bereitstellung der fünften Gleitkommaoperation in Rechenanlagen erfährt das bisherige arithmetische Repertoire eine wesentliche **Ergänzung**.

10 Ist das Datenformat double precision ein natürliches Datenformat?

Es stellt sich noch die Frage, ob das Datenformat double precision für heutigentags zu lösende Probleme ein natürliches Datenformat ist oder ob es uns durch die heutige Technologie aufoktroyiert wird. Die Daten können bei vielen Ingenieurproblemen mit etwa vier bis sechs dezimalen Ziffern hinreichend genau beschrieben werden. Trotzdem rechnen alle Ingenieure ihre Probleme praktisch immer in double precision, was etwa 16 dezimalen Ziffern entspricht. Man kann wohl davon ausgehen, daß bei Verfügbarkeit einer gleichschnellen, vollständigen quadruple Arithmetik alle Anwender sofort diese quadruple Arithmetik einsetzen und verwenden würden. Wenn es jemandem gelänge, ein gepacktes Datenformat zu erfinden, welches es erlaubt, mit 100 dezimalen Ziffern genauso schnell zu rechnen wie mit double precision, würde jeder sofort dieses gepackte Datenformat verwenden. Dies ist auch gerechtfertigt; denn die korrekte Rechnung im Raume der reellen Zahlen würde mit infinite precision ablaufen. In der Intervall-

rechnung wird gezeigt, daß bei Erhöhung der precision die Ergebnisgenauigkeit i. allg. zu-, keinesfalls aber abnimmt. In diesem Sinne ist die heutige double Arithmetik nicht natürlich und nicht optimal den heutigen Problemen angepaßt. Über den langen Akkumulator wird dem Benutzer in Form des Datenformates staggered in gewissen Grenzen auf einfache Weise eine dem Problem anpaßbare Arithmetik mit dynamischer Genauigkeit bereitgestellt, welche auf der double Arithmetik aufbaut. Der Datentyp staggered ermöglicht es, an beliebigen, ausgewählten Stellen im Programm die precision in gewissen Grenzen zu erhöhen oder zu erniedrigen. Der Benutzer kann sich die Genauigkeit aussuchen, welche seinem Problem angepaßt ist. Im Rechner gibt es nur ein (standardisiertes) Gleitkommadatenformat, nämlich double precision. Auf diese Weise baut eine natürliche (dynamische) Arithmetik auf nur einer vorhandenen standardisierten Gleitkommaarithmetik auf, was seinerseits auch die Bedeutung eines derartigen Standards unterstreicht.

Literatur

[1] Adams, E., Kulisch, U. (Hrsg.): *Scientific Computing with Automatic Result Verification*, Academic Press, New York, 1993.

[2] Alefeld, G., Herzberger, J.: *An Introduction to Interval Computations*, Academic Press, New York, 1983.

[3] ANSI/IEEE: *IEEE Standard for Binary Floating-Point Arithmetic*, ANSI/IEEE Std 754, New York, 1985.

[4] Baumhof, Ch., Kernhof, J., Höfflinger, B., Kulisch, U., Kwee, S., Schramm, P., Selzer, M., Teufel, T.: *A CMOS Floating-Point Processing Chip for Verified Exact Vector Arithmetic*, 20th European Solid State Circuits Conference, Ulm, 1994.

[5] Baumhof, Ch.: *Ein Vektorarithmetik-Koprozessor in VLSI-Technik zur Unterstützung des wissenschaftlichen Rechnens*, Dissertation, Universität Karlsruhe, 1996.

[6] GAMM-IMACS *Proposal for Accurate Floating-Point Vector Arithmetic*, in Rundbrief der GAMM, 1993, Brief 2 — und in Mathematics and Computers in Simulation, Vol. 35, No. 4, IMACS, 1993.

[7] Greenbaum, A.: *Iterative Methods for Solving Linear Systems*, SIAM, 1997.

[8] Greenbaum, A.: *Recent Advances and Open Problems in Iterative Methods for Solving Linear Systems*, SIAM-Meeting, Stanford University, July 1997.

[9] Hammer, R., Hocks, M., Kulisch, U., Ratz, D.: *Numerical Toolbox for Verified Computing I: Theory, Algorithms and PASCAL-XSC Programs*, Springer, Berlin, 1993.

[10] Hammer, R., Hocks, M., Kulisch, U., Ratz, D.: *C++ Toolbox for Verified Computing I*, Springer, Berlin, 1995.

[11] Hestenes, M.R., Stiefel, E.: *Methods of Conjugate Gradients for Solving Linear Systems*, J. Res. Nat. Bur. Stand. 49, 409 – 436, 1952.

[12] Höfflinger, B.: *Next-Generation Floating-Point Arithmetic for Top-Performance PCs*, The 1995 Silicon Valley Personal Computer Design Conference and Exposition, Conference Proceedings, 319 – 325.

[13] IBM, *System / 370 RPQ, High Accuracy Arithmetic,* SA22-7093-0, IBM Corp., 1984.

[14] IBM, *High Accuracy Arithmetic Subroutine Library (ACRITH), General Information Manual*, 3. Aufl., GC33-6163-02, IBM, 1986.

[15] IBM, *High Accuracy Arithmetic-Extended Scientific Computation (ACRITH-XSC), General Information*, GC33-6461-01, IBM, 1990.

[16] Klatte, R., Kulisch, U., Neaga, M., Ratz, D., Ullrich, Ch.: *PASCAL-XSC-Sprachbeschreibung mit Beispielen*, Springer, Berlin, 1991; engl. Übers., Springer, Berlin, 1992; russ. Übers., Springer, Berlin, 1996.

[17] Klatte, R., Kulisch, U., Lawo, Ch., Rauch, M., Wiethoff, A.: *C-XSC: A C++ Class Library for Extended Scientific Computing*, Springer, Berlin, 1993.

[18] Krämer, W., Kulisch, U., Lohner, R.: *Numerical Toolbox for Verified Computing II: Theory, Algorithms and PASCAL-XSC Programs*, Springer, Berlin, 1996.

[19] Kulisch, U.: *Grundlagen des Numerischen Rechnens*, BI Wissenschaftsverlag, Mannheim, 1976.

[20] Kulisch, U., Miranker, W.L.: *Computer Arithmetic in Theory and Practice*, Academic Press, New York, 1981.

[21] Kulisch, U., Miranker, W.L. (Hrsg.): *A New Approach to Scientific Computation*, Academic Press, 1983.

[22] Kulisch, U., Stetter, H.J. (Hrsg.): *Scientific Computing with Automatic Result Verification*, Computing Supplementum 6, Springer, Wien, 1988.

[23] Mascagni, M., Miranker, W.L.: *Arithmetically Improved Algorithmic Performance*, Computing 35, 153 – 175, 1985.

[24] Meis, T.: *Brauchen wir eine Hochgenauigkeitsarithmetik?* Elektronische Rechenanlagen, Carl Hanser Verlag, 19 – 23, 1987.

[25] Miranker, W.L., Rump, S.: *Case Studies for ACRITH*, IBM Research Center Report No. 10249, 1983.

[26] Miranker, W.L., Mascagni, M., Rump, S.: *Case Studies for Augmented Floating-Point Arithmetic*, in Accurate Scientific Computations, Lecture Notes in Computer Science No. 235, Springer, 86 – 118, 1986.

[27] Schramm, P.: *Sichere Verschneidung von Kurven und Flächen im CAGD*, Dissertation, Universität Karlsruhe, 1995.

[28] Shiriaev, D.: *Fast Automatic Differentiation for Vector Processors and Reduction of the Spatial Complexity in a Source Translation Environment*, Dissertation, Universität Karlsruhe, 1993.

[29] Steihaug, T., Yalçinkaya, Y.: *Asynchronous Methods and Least Squares: An Example of Deteriorating Convergence*, Report No. 131, Department of Informatics, University of Bergen, 1997.

[30] Teufel, T.: *Ein optimaler Gleitkommaprozessor*, Dissertation, Universität Karlsruhe, 1984.

[31] Zeitschrift Elektronik 26, 1994, *Genauer und trotzdem schneller: Neuer Coprozessor für hochgenaue Matrix- und Vektoroperationen*, Titelgeschichte.

In dieser Reihe sind bisher die folgenden Arbeiten erschienen:

1/1996 Ulrich Kulisch: *Memorandum über Computer, Arithmetik und Numerik.*

2/1996 Andreas Wiethoff: *C–XSC — A C++ Class Library for Extended Scientific Computing.*

3/1996 Walter Krämer: *Sichere und genaue Abschätzung des Approximationsfehlers bei rationalen Approximationen.*

4/1996 Dietmar Ratz: *An Optimized Interval Slope Arithmetic and its Application.*

5/1996 Dietmar Ratz: *Inclusion Isotone Extended Interval Arithmetic.*

1/1997 Astrid Goos, Dietmar Ratz: *Praktische Realisierung und Test eines Verifikationsverfahrens zur Lösung globaler Optimierungsprobleme mit Ungleichungsnebenbedingungen.*

2/1997 Stefan Herbort, Dietmar Ratz: *Improving the Efficiency of a Nonlinear-System-Solver Using a Componentwise Newton Method.*

3/1997 Ulrich Kulisch: *Die fünfte Gleitkommaoperation für top-performance Computer — oder — Akkumulation von Gleitkommazahlen und -produkten in Festkommaarithmetik.*

4/1997 Ulrich Kulisch: *The Fifth Floating-Point Operation for Top-Performance Computers — or — Accumulation of Floating-Point Numbers and Products in Fixed-Point Arithmetic.*

5/1997 Walter Krämer: *Eine Fehlerfaktorarithmetik für zuverlässige a priori Fehlerabschätzungen.*