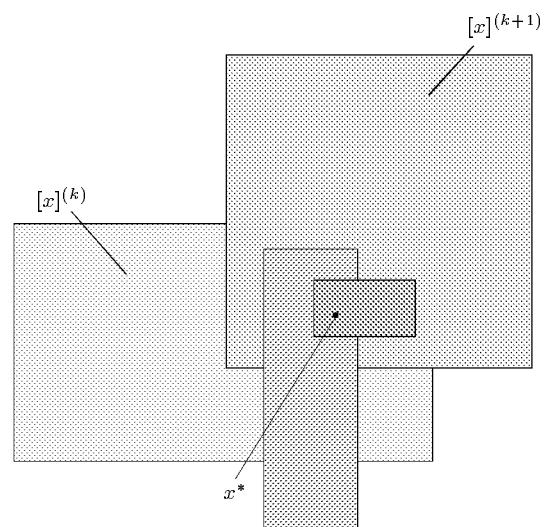


**The Fifth Floating-Point Operation for
Top-Performance Computers**
or
**Accumulation of Floating-Point Numbers and
Products in Fixed-Point Arithmetic**

Ulrich Kulisch

Forschungsschwerpunkt
Computerarithmetik,
Intervallrechnung und
Numerische Algorithmen mit
Ergebnisverifikation



Bericht 4/1997

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`

**The Fifth Floating-Point Operation for
Top-Performance Computers**
or
**Accumulation of Floating-Point Numbers and
Products in Fixed-Point Arithmetic**

Ulrich Kulisch

Contents

1	A Brief History	4
2	State of the Art	6
3	Definition of Terms	6
4	A Frequently Quoted Remark by C. F. Gauss	7
5	Problems with Inaccurate and with Accurate Data	8
6	Accumulation of Floating-Point Numbers and Products in Quadruple-Precision Floating-Point Arithmetic	11
7	How Much Local Memory is Needed on the Arithmetic Unit?	13
8	Optimal Scalar Product for Multi-User Systems	13
9	Does the Long Accumulator Mean a Return to Fixed-Point Arithmetic?	13
10	Is the Double Precision Format a Natural Format?	14
	References	14

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 A Brief History

Floating-point arithmetic is used since the early forties and fifties (Zuse Z3, 1941). Technology in those days was poor (electro-magnetic relays, electron tubes). It was complex and expensive. Thus, it was considered to be sufficient to yield or get a somewhat correct floating-point number as a result of an operation of two floating-point operands. For more complicated expressions an error analysis was left to and put on the shoulder of the user.

About 25 years ago a general mathematical theory of computer arithmetic was developed. It was realized in this context that it is appropriate to provide, apart from the four basic floating-point operations, a fifth floating-point operation for the computation of scalar products with maximum accuracy according to the principle of semimorphism [19], [20]. The first solution methods that have been proposed were of an algorithmic nature. About 20 years ago it was discovered that the problem can be solved in software and in hardware in an elegant manner if the products of the vector components are accumulated into a fixed-point register that covers the full floating-point range to the double floating-point exponent. Since that time it was desirable to develop and build a coprocessor in VLSI-technology.

In a floating-point system the number of mantissa digits as well as the exponent range are finite. Therefore, the fixed-point register is finite as well and it is relatively small, consisting of about 1000 to 4000 bits depending on the data format in use. So we have the seemingly paradox and striking situation that dot products of floating-point vectors with even millions of components can be computed to a fully accurate result in infinite precision arithmetic using a relatively small finite local register memory on the arithmetic unit. The dot product is a fundamental operation in linear algebra. A fully accurate dot product eliminates many rounding errors in numerical computations. It stabilizes these computations and speeds them up as well.

In the following years the functionality of the optimal scalar product with several roundings was integrated into various computer languages (ACRITH-XSC [15], PASCAL-XSC [16], C-XSC [17]) by means of a software simulation via integer arithmetic. In combination with interval arithmetic, algorithms for numerous numerical problems were developed in short time by which computers could prove both existence and uniqueness of the computed solution, as well as deliver bounds for the solution with high accuracy [9], [10], [18]. It turns out that many iterative methods reach the desired accuracy faster if all dot products are evaluated in infinite precision arithmetic to full accuracy or with only one final rounding [23], [25], [26]. Nevertheless, till today there is no commercial VLSI coprocessor available for the computation of the optimal scalar product.

One reason for this certainly is the fact that 20 years ago the technology as well as the design tools were not powerful enough to solve the problem. An enforced solution would have been very expensive (tens of millions of dollars). A well balanced design does not only depend on the mathematical solution but also on many boundary conditions such as bandwidth of the data bus, connection to the main processor, coordination with the arithmetic unit of the main processor, architecture and speed of the main processor. Moreover, development tools and production technology play an important role. These multiple dependencies kept many competent and interested scientists and manufacturers from tackling the problem in VLSI technology.

In 1987 GAMM¹ published a resolution that criticized the mathematically not adequate execution of matrix and vector operations on vector processors and demanded an amendment. In 1993 GAMM and IMACS² approved and published a *Proposal for Accurate Floating-Point Vector Arithmetic* [6] where a mathematically correct execution of matrix and vector operations, in particular of scalar products, is demanded and required for any computer. The proposal indicates solution methods (see also [1], [5], [12], [28]). In 1995 the IFIP-Working Group 2.5 on Numerical Software endorsed this proposal. Meanwhile it became a *EU-Guideline*.

¹GAMM = Gesellschaft für Angewandte Mathematik und Mechanik

²IMACS = International Association for Mathematics and Computers in Simulation

2 State of the Art

In 1992 three institutes at Karlsruhe, Hamburg and Stuttgart³ got funding (approximately 900 000 DM) from the VW-Stiftung for a project, that aimed at the construction of a VLSI vector arithmetic coprocessor for the PC. The project was supposed to be done within a two-year term (January 1993 to December 1994). At Karlsruhe 3, at Hamburg 1 and at Stuttgart 2 scientists were working on the project. A VLSI vector arithmetic coprocessor XPA 3233 was developed and manufactured in the 0.8 μm CMOS gate array technology of the IMS. The processor can be connected with PCs via the currently available 33 MHz 32-bit wide PCI-bus. It is the first vector arithmetic coprocessor on a chip. For instance in PASCAL-XSC [16] the coprocessor can be addressed directly by the operator symbols for vector and matrix operations which are provided by the language. On the computer the chip has two properties. It accelerates the computation and eliminates all rounding errors that occur in a conventional accumulation. In a pipeline the three steps: read the two factors for the next product, perform a multiplication and accumulate a product are executed simultaneously. In comparison with conventional computation of the dot product in floating-point arithmetic the execution speed thus is at least doubled. By the time of completion the XPA computed scalar products faster than any commercially available microprocessors with conventional floating-point arithmetic, although those were normally manufactured in a far more refined and advanced technology. In contrast to these, the XPA 3233 always provides the correct result with full accuracy or with only one single rounding at the end. Compared to software implementations of the optimal scalar product in PASCAL-XSC, ACRITH-XSC and C-XSC the XPA yields a speedup by an order of two magnitudes [5].

3 Definition of Terms

In the following the term *precision* refers to the number of digits used during a computation while the term *accuracy* refers to the accuracy of the final result. By *double* or more complete *double precision* we understand a 64 bit floating-point word for instance according to the IEEE-arithmetic standard 754 [3]. *Quadruple* or *quadruple precision* describes a 128 bit floating-point format. *Long accumulator* denotes a fixed-point format that suffices to sum up products of double precision floating-point numbers without loss of information. The corresponding accumulation process is simply called fixed-point accumulation. Actually in case of the IEEE arithmetic standard 754 the *long accumulator* consists of local memory on the arithmetic unit of 67 words of 64 bits. *Quadruple accumulator* denotes a computer register, that suffices to sum up quadruple precision floating-point numbers in floating-point arithmetic with a rounding error of at most $1/2$ or 1 ulp (unit in the last place) after each addition.

In the XSC-languages [15], [16], [17] in addition to the numerical data type *real* and many additional numerical data types an array type *staggered* (*staggered precision*)

³Institut 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)

is available. A variable of the type *staggered* consists of an array of variables of the type of its components. Components of the *staggered* type can be of type *real* or of type *interval*. The value of a variable of type *staggered* is the sum of its components. Sums of variables of the type *staggered* can be computed easily in the long accumulator and products of two variables of this type can be computed by means of the optimal dot product. Quotients are computed iteratively. The *staggered* type is controlled by a global variable *stagprec*. If *stagprec* = 1, the *staggered* type is identical to its component type. If, for instance, *stagprec* = 4 each variable of this type consists of an array of four variables of its component type. Again its value is the value of the sum of its components. In this case the variable has, in general, four times the precision of its components. The global variable *stagprec* can be incremented or decremented at arbitrary places in a program. The standard functions for the type *staggered* are already available in Karlsruhe for the component types *real* and *interval* (see the papers of K. Braune and W. Krämer in [22]). In the case *stagprec* = 2 a data type is encountered that is occasionally denoted as *double-double* or *quadruple* precision.

4 A Frequently Quoted Remark by C. F. Gauss

To C. F. Gauss we owe the remark: *Der Mangel an mathematischer Bildung gibt sich durch nichts so auffallend zu erkennen, wie durch maßlose Schärfe im Zahlenrechnen (There is nothing that reveals the lack of mathematical scholarship more conspicuously than the use of excessive precision when computing with numbers).*

Many a numerical analyst feels urged by this remark to compute as many problems as possible in a short precision. He is proud to solve *his problems* satisfactorily executing the entire computation including all scalar products in double, perhaps even in single precision. If this leads to success, this method is, of course, legitimate and there is no objection against it.

It is also often concluded from the above remark that a correct scalar product is not needed on the computer and, consequently, should not be provided. The usual way to compute scalar products in floating-point arithmetic is considered to be easier since *superfluous digits* do not have to be carried along. This conclusion is completely wrong. It is due to an insufficient knowledge and familiarity with the technology and implementation techniques which are available today, on the side of mathematicians. The engineer on the other hand, who is familiar with these techniques is not aware of the consequences for mathematics.

The remark by C. F. Gauss is now more than 150 years old. It fits into the time of Gauss where any computation had to be executed manually and each digit that needed not to be considered meant an enormous ease of work. The slow manual computation allowed, as a rule, to perform a simultaneous error analysis or at least an error estimate.

Today we have a completely different situation which could not at all be foreseen or anticipated at the time of Gauss. Fast processors today are capable to execute a billion arithmetic operations in each second. This number exceeds the imagination of any user. The available technology (millions of transistors on a single chip) is extremely powerful. It allows solutions which even an experienced computer user is totally unaware of.

For all kinds of computers such as PCs, workstations, mainframes and supercomputers there are available circuitries for the computation of the optimal dot product where virtually no computation time for the arithmetic is needed. In a pipeline, the arithmetic can be done within the time the processor needs to read the data into the arithmetic unit. This means, among others, that no other method to compute scalar products can be faster. This includes, in particular, floating-point arithmetic. Technically, the process of an always correct accumulation of scalar products in a fixed-point register, though different, is by no means more complicated than a hardware accumulation in floating-point arithmetic. The basic building blocks for both kinds of arithmetic such as shifter, multiplier, adder and rounding unit, are very similar. Fixed-point accumulation only needs an additional small local memory on the arithmetic unit. Many intermediate steps that are necessary when doing floating-point accumulations such as normalization, rounding, composition into a floating-point number and decomposition into mantissa and exponent for the next operation do not occur when doing fixed-point accumulations.

A relatively complex arithmetic operation such as the scalar product of floating-point numbers which can be executed always correctly with moderate technical effort in today's technology should indeed always be executed correctly. In any case, this is the way C. F. Gauss would do it today. An error analysis thus becomes superfluous for this additional basic operation once and for all. In addition to that, always the same results are obtained for this operation on different platforms. Computer applications nowadays are of immense variety. Any discussion of the question where this is favorable or not is useless and superfluous. Expanding the arithmetic unit of the computer by the optimal scalar product creates a far more powerful and capable computer. This new and always absolutely reliable operation can be applied favorably in many algorithms and places.

5 Problems with Inaccurate and with Accurate Data

In numerical analysis for more than fifty years floating-point arithmetic is used almost exclusively. This has formed the way of our thinking. Now, it turns out to be difficult to shake off usual and well trained methods, skills and schemes of thinking. During that time the so-called backward error analysis was often the only practicable way to perform an error analysis of numerical algorithms. There, a computed solution is viewed as the correct solution to a problem with modified input data. If the latter are close to the given input data of the problem, an algorithm is called well behaved or a problem is called well conditioned.

This scheme of thinking implies the imagination that, in fact, never the given problem is solved but always a problem in its neighborhood or surrounding. This suggests the assumption that absolute accuracy in the data is not significant and that, in principle, one always has to deal with problems with inexact data. For well behaved problems this assumption is quite legitimate. In fact, computations are often based on uncertain assumptions. For instance, measured data are used in a computation, the accuracy of which is not known, or the mathematical model has to be simplified drastically in order to make the computation possible at all. When treating differential equations

numerically, discretization errors can be far greater than computational inaccuracies. For many applications a great number of similar computations are performed. Based on experience or experimental measurements one can trust simple floating-point computation.

For such problems, one can prove often mathematically by means of the backward error analysis that simple floating-point arithmetic suffices to compute a satisfactory solution. Because of the high frequency of the occurrence of such well behaved problems, many users are not aware of the necessity of highly accurate computations. Any improvement of simple floating-point arithmetic is considered to be superfluous. An exact scalar product has not been employed so far. Consequently, one believes that it is not needed and it is not demanded. Due to this ignorance, it is overlooked that an optimal scalar product is simpler and that it runs faster than the traditionally used summation loop in floating-point arithmetic, so that even well conditioned computations can profit from it.

However, not all numerical problems are of the above described *harmless* nature. Actually this is only one side of the medal. On the other side is the large class of problems with exact data. Also problems with small but safe intervals in the data belong to this class. In mathematical modeling, for instance, one usually assumes exact input data. The model only can be improved systematically if the computational error can be virtually excluded. Moreover, many inherent mathematical problems have exact input data. This is true e.g. for weights and nodes of numerical quadrature and cubature formulas, for coefficients of discretization formulas, for coefficients that arise, when computing zeros of polynomials, or when computing eigenvalues of matrices or of differential operators. In computer algebra, as a rule, problems with exact input data are considered and treated. There the computation is performed whenever possible, exactly i.e. with a very large number of digits. This can involve a very high overhead of computation time. A numerical computation of these problems with floating-point arithmetic that yields certain and sufficiently exact bounds for the solution will run, in general, much faster.

Computers are getting ever faster. With increasing speed problems to be dealt with become larger. Instead of two-dimensional problems users would like to solve three-dimensional problems. A discretization of 100 steps in each one of three coordinate directions leads to a system of equations of 1 million unknowns. The Gauss elimination method requires the magnitude of $\mathcal{O}(n^3)$ operations. If $n = 10^6$ then n^3 is so large that a GIGAFLOPS computer would need many years to solve such a system. Therefore large systems of linear equations are to be solved iteratively with less than n^3 arithmetic operations. The basic operation of iterative methods (Jacobi method, Gauss-Seidel method, overrelaxation method, conjugate gradient method, Krylov space methods, multigrid methods and others like the QR method for the computation of eigenvalues) is the matrix-vector multiplication which consists of a number of dot products. It is well-known that *finite precision* arithmetic often deteriorates the convergence of these methods [7], [8], [11], [23], [25], [26], [29]. An iterative method which converges to the solution in infinite precision arithmetic usually converges much slower or even can diverge in *finite precision* arithmetic. Fixed-point accumulation of dot products of floating-point vectors is absolutely error free and it is faster than a conventional

computation in floating-point arithmetic. An error free computation of dot products speeds up the rate of convergence for many iterative methods. In many applications the data of the problem are exact (coefficients of discretization formulas). They are fed back into the computation at each step of the iteration. So a highly accurate computation really makes sense. The components of a matrix-vector product can easily be rounded into a variable of type *staggered* to two-, three- or four-fold accuracy, and used further in the algorithm. This increases the run-time only marginally.

By interval arithmetic and the optimal dot product modern numerical analysis has got new tools which allow new solution methods for numerical problems. Examples: The new methods for global optimization and for automatic differentiation [1], [9], [10], [18]. For the reverse mode of automatic differentiation, the memory overhead and the spatial complexity can be significantly reduced by the optimal dot product. There the dot product is considered as a single basic operation in the vector spaces [28].

Often these tools allow a productive forward error analysis by the computer itself. Examples: numerical quadrature and cubature [1], numerical integration of ordinary differential equations and of integral equations [1], applications in computer geometry such as the safe computation of the intersection of two surfaces [27] or definite answers to questions of coincidence [22].

Furthermore, these tools often allow an a posteriori error analysis by means of the computer. This process is called automatic result verification or validation of results. Examples: Highly accurate inclusion of the solution of linear and non-linear systems of equations [9], [10], [18], [21], linear and non-linear optimization problems [18], [22]. The final verification step can fail which happens only very rarely. But it is detected by the computer itself. Then computation with higher precision often leads to success. The staggered data type is the appropriate tool for this purpose. Its arithmetic operations are realized by means of the optimal dot product [1], [18].

The question how many digits of a defect can be guaranteed with single, double or extended precision arithmetic has been carefully investigated. With the optimal dot product the defect can always be computed to full accuracy.

Many problem solving routines of the XSC-languages (PASCAL-XSC, ACRITH-XSC, C-XSC, FORTRAN-XSC) are available also for (narrow) interval data. Special routines allow to read data as intervals. If a user does not make use of these possibilities and enters all data as point data (computer reals) one may conclude that he/she is of the opinion that the data are represented precisely enough to full accuracy in the machine. Highly accurate problem solving routines for problems with exact point data (of infinite accuracy) therefore, are worth-while.

For the solution of problems with exact point data with guaranteed high accuracy, fixed-point accumulation in a long accumulator is often required and clearly superior to floating-point accumulation. Example: Eigenvalues of Zielke matrices can be always computed correctly by means of the long accumulator.

6 Accumulation of Floating-Point Numbers and Products in Quadruple-Precision Floating-Point Arithmetic

The scalar product is a central operation in Numerical Analysis. It occurs in complex arithmetic, matrix and vector arithmetic, defect correction methods, range reduction when computing elementary functions, multiple precision arithmetic and many other places. By means of an optimal dot product all operations of the usual product spaces of numerical analysis can be provided with maximum accuracy according to the principle of semimorphism [19], [20]. Among these spaces are the real and complex floating-point numbers, the real and complex floating-point intervals as well as the vectors and matrices over these four basic data types. By semimorphism the real and complex vector spaces and the corresponding interval spaces are mapped to their floating-point subspaces best possible with respect to their algebraic structure. The order structure can very well be preserved.

New programming languages like Fortran 90 or C++ provide the scalar product as a fundamental operation (operators *matmul* and *dotproduct*). Similarly an optimal dot-product can be embedded in and provided by table calculation programs and computer algebra packages.

In the XSC-languages and by the XPA 3233, floating-point numbers and products of floating-point numbers are accumulated into a long fixed-point word (long accumulator). Doing so, no information is lost. Since the final result is almost always rounded to a double precision floating-point number, often the question is asked whether this is really necessary, and whether it would not suffice to accumulate the double length products of floating-point numbers in quadruple precision floating-point arithmetic. For such an addition an accumulator is needed which is only slightly longer than this quadruple precision format. We are now going to compare these two accumulation methods. Weighting up the pros and cons of both methods shows definite advantages for the fixed-point accumulation.

The computation of the scalar product of two floating-point vectors in the long accumulator is the only floating-point operation that always yields a totally correct result. Compared to this, an accumulation of products of floating-point numbers in a quadruple accumulator can occasionally yield a wrong result. Therefore, the user has to ponder the computational error whenever such an operation is executed. With the long accumulator, an error estimate is never needed, the result is always correct.

The long accumulator is far easier to handle than a quadruple accumulator, because the long accumulator only operates on one data format which is double precision. The quadruple accumulation, on the other hand, has to cope with two different formats, double and quadruple precision. Only very few additional machine instructions are needed for computation of the optimal scalar product by fixed-point accumulation in the long accumulator. For the quadruple accumulation, instructions for mixed types have to be provided as well. Thus, more opcodes are needed.

So far experience with different technologies (assembler, microcode, bit-slice-technology, VLSI) has shown that fixed-point accumulation of double precision products is faster and simpler than double precision floating-point accumulation in the traditional manner. An accumulation of double length products of double precision

variables in quadruple precision is very likely to run slower than the traditional double precision floating-point accumulation. The fixed-point accumulation only needs two instructions: shift and accumulate. The exponent of the product provides the address for its addition. In case of floating-point accumulation many intermediate steps have to be performed additionally. The following steps are necessary: decomposition of operands, shift, addition, normalization, rounding, composition to a floating-point number, decomposition into mantissa and exponent for the next operation et cetera. Compared to that fixed-point accumulation is simpler. It only needs a shift and an addition. Rounding is performed only once at the very end.

When computing scalar products, the products of floating-point numbers that are to be accumulated are only of double length. Therefore the adder required for the fixed-point accumulation is of the same order as the adder for the quadruple precision accumulation. The *long accumulator* only has to be kept as local memory on the arithmetic logical unit. In case of the IEEE arithmetic data format double the long accumulator consists of 67 words of memory of 64 bits, [1], [5], [12], [30], [31].

The long accumulator that is used for the fixed-point accumulation covers the full double exponent range and some additional guard digits. Thus an overflow or underflow in a scalar product computation can never happen. For the quadruple precision accumulation the data format is not uniquely defined. It can be a double double word or a genuine quadruple format. Thus, with different manufacturers, different results can be obtained when doing accumulation with quadruple precision. When a quadruple format is realized as a double double format, only the *double precision* exponent range is available and underflow or overflow of exponents can easily occur. Thus, when computing, many plus or minus infinity or NaN results are obtained, that yield virtually no information. Since quadruple arithmetic is not precisely defined, when transferring data between machines from different manufacturers problems are very likely to occur, if a quadruple accumulation is provided. These problems do not arise with a fixed-point accumulation since it uses only one format, i.e. double precision. With the long accumulator a quadruple arithmetic is treated as a simple case of a multiple precision (staggered) arithmetic based on the double precision format. A multiple precision number (of staggered format) is represented as an array of double precision components. The value of a number of type staggered is the sum of its components. Addition of such numbers is performed within the *long accumulator*. Multiplication is simply a sum of products.

One advantage of quadruple precision accumulation of products of double precision numbers seems to be, that many users believe they immediately can imagine what this means. Yet, the idea of it can be different from user to user. Normally, a user will believe, that a full quadruple precision format with all four basic operations is provided. This imagination will finally put pressure on the manufacturers, to include far more with their hardware, than is actually necessary. That is, simply, to provide a full quadruple arithmetic including appropriate standard functions. By this, the necessary, useful and essential system features are fairly exceeded.

On the other hand, the staggered type as an array of double precision reals or intervals is far more powerful than the quadruple precision type, since it provides a dynamic precision.

7 How Much Local Memory is Needed on the Arithmetic Unit?

Are there several long accumulators needed on the arithmetical logical unit for special applications? By the same argumentation that demands one *long accumulator* for real point problems, two *long accumulators* are suitably used for complex point problems. Thus, the overhead of data transfer to the arithmetic unit is halved. Complex scalar products, then, can be always executed correctly. For the second accumulator, merely the required local memory on the arithmetic unit has to be increased. Nothing else has to be changed on the arithmetic unit. If there is enough memory available for two long accumulators on the arithmetic unit, interval scalar products can be summed up as well using always two long accumulators. In general the bounds obtained this way will only be gradually better than those obtained by accumulating the interval bounds with quadruple precision. But all the disadvantages listed above for quadruple precision accumulation apply again. Thus, in all probability, accumulation of interval scalar products with quadruple precision is slower than with two long accumulators. Furthermore, the accumulation result of interval scalar products is liable to be more accurate than a corresponding quadruple precision accumulation, since unnecessary rounding errors that occur with quadruple precision additions are not accumulated. When doing fixed-point accumulation of intervals, the bounds of the result are only rounded once at the end of the computation.

8 Optimal Scalar Product for Multi-User Systems

In connection with fixed-point accumulation usually the question arises what has to be done in case of a context-switch or an interrupt. This problem can easily be solved by providing enough local memory for several long accumulators on the arithmetic unit. The Hyperstone, a German DSP (Digital Signal Processor), provides 8 KByte of local memory on the arithmetic unit. This is enough for about seven long accumulators for the IEEE arithmetic format double precision. If a context switch requires another long accumulator simply a new accumulator is allocated. Thereby, swapping of intermediate results is avoided. In [5], refined techniques for handling optimal scalar products in multi-user systems are dealt with and presented.

9 Does the Long Accumulator Mean a Return to Fixed-Point Arithmetic?

The answer to this question is strictly *no*. Simple floating-point computations are still performed by floating-point arithmetic. All data are still stored in the floating-point format. The advantages of floating-point arithmetic such as the abending of tiresome (problem) scalings, i.e. their automatization, still apply. Only accumulation, the most sensitive floating-point operation, is performed by fixed-point arithmetic. Fixed-point accumulation of floating-point numbers and products is absolutely error free. Extending traditional floating-point arithmetic by the fifth floating-point operation, which is

accumulation of floating-point numbers and products in fixed-point arithmetic, means an ideal combination of fixed- and floating-point arithmetic, combining the advantages of both. Providing the fifth floating-point operation on computers, expands essentially their traditional arithmetical repertoire.

10 Is the Double Precision Format a Natural Format?

There still remains the question whether the double precision format of our days is a natural format for the problems that are to be solved today, or whether it is imposed by today's technology. Data used in engineering problems, in general, can be represented precisely enough by about four to six decimal digits. Nevertheless, every engineer computes his problems virtually always with double precision, which corresponds to about 16 decimal digits. If a complete and equally fast quadruple arithmetic were available, it can be expected that all users would immediately employ and use this quadruple arithmetic. If someone would invent a packed format that allows to compute with hundred decimal digits as fast as with double precision, everybody would immediately use this packed format. And this is justified, since correct computation in the space of real numbers would be carried out with infinite precision. In interval arithmetic it is shown that increasing the precision also increases the accuracy of the result or at least does not decrease it. In this sense, today's double arithmetic is neither naturally nor optimally adjusted to our problems. By means of the long accumulator, in the form of the staggered format a dynamic precision is provided to the user. Within certain bounds, the arithmetic can be adjusted to the needs of the problem. The staggered type allows to increase or decrease precision within certain bounds at arbitrary places of a program. The user or the computer itself can choose the precision which optimally fits to his problem. On the computer, there is only one (standardized) floating-point format that is double precision. Thus, a natural (dynamic) arithmetic is built upon only one existing standardized floating-point arithmetic. This, in return, strongly emphasizes the meaning of a double precision floating-point arithmetic standard.

References

- [1] Adams, E., Kulisch, U. (eds.): *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 — and 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*, 3rd ed., GC33-6163-02, IBM, 1986.
- [15] IBM, *High Accuracy Arithmetic-Extended Scientific Computation (ACRITH-XSC), General Information*, GC33-6461-01, IBM, 1990.
- [16] Klätte, R., Kulisch, U., Neaga, M., Ratz, D., Ullrich, Ch.: *PASCAL-XSC-Sprachbeschreibung mit Beispielen*, Springer, Berlin, 1991; engl. transl., Springer, Berlin, 1992; russ. transl., Springer, Berlin, 1996.
- [17] Klätte, 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. (eds.): *A New Approach to Scientific Computation*, Academic Press, 1983.
- [22] Kulisch, U., Stetter, H.J. (eds.): *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 Gleitkommprozessor*, Dissertation, Universität Karlsruhe, 1984.
- [31] Zeitschrift Elektronik 26, 1994, *Genauer und trotzdem schneller: Neuer Coprozessor für hochgenaue Matrix- und Vektoroperationen*, titlestory.

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.*