Preface

In general, a computer is understood to be an engine that deals with data. These are stored by binary digits or bits. The data can represent all kinds of objects such as whole numbers, letters of an alphabet, the books of a library, the accounts of a bank, the inhabitants of a town, the text of a law, or the features of a disease. If the computer is functioning correctly, then its processing is free of error. A computer that is programmed properly is therefore generally considered to be an infallible tool.

It is quite ironical that this view is often not justified when the digital computer is used for its original purpose – numerical or scientific computation. This book treats the arithmetic processing of real numbers in computers. It describes how the computational error can be reduced and avoided by special design of the computer's arithmetic.

A real number like π, for instance, consists of an infinite b-adic expansion, an infinite string of digits. On a computer, however, only finite parts of these expansions can be represented. Thus arithmetic for real numbers can only be approximated. For numerical or scientific computing, floating-point numbers and floating-point arithmetic have been used since the early days of electronic computers. A floating-point number takes the form $m \cdot 2^e$. Here m is the mantissa or significand and e is the exponent.

Computers designed and built by Konrad Zuse, the Z3 (1941) and the Z4 (1945), were among the first computers that used the binary number system and floating-point for number representation. In the Z4, a floating-point number was represented by a 32 bit word and arithmetic was used in a manner very similar to the data format single precision on today's computers. Over the years computer technology was permanently improved. This permitted an increase of the word size.

In 1985 a standard for floating-point arithmetic was internationally adopted. It is known as the IEEE 754 floating-point arithmetic standard. It specifies two data formats for single precision of 32 bits and double precision of 64 bits. The format double precision uses 53 bits to represent the mantissa and 11 bits for the exponent. A revision of the standard IEEE 754, published in 2008, added another word size of 128 bits.

The IEEE floating-point arithmetic standard is undoubtedly thorough, consistent, and well defined. It has been widely accepted and has been used in almost every processor developed since 1985. This has greatly improved the portability of floating-point programs. Until today the most used floating-point format is double precision. It corresponds to about 16 decimal digits. Floating-point arithmetic has been used successfully in the past. Every computer user is trained and familiar with all details of floating-point arithmetic including all its exceptions like *underflow*, *overflow*, $-\infty$, $+\infty$, NaN (not a number), -0, $+0$, and so on. Seventy years of extensive use of floating-point

arithmetic makes many users believe that this is the only reasonable way of using the computer for scientific computing.

In the 1950s computers were able to perform about 100 floating-point operations per second (flops). For comparison: With a mechanical or electrical calculator a man or woman can execute about 1 000 arithmetic operations somewhat reliably in a day. A computer now could do this in 10 seconds. So the step to the electronic computer really meant a revolution in computing speed. In order to judge the reliability of computed results, error estimates were derived for frequently used numerical methods in the 1950s and 1960s. These are built upon estimates of the error of each single arithmetic operation. However, already in 1965 computers were on the market (CDC 6600) that performed 10^5 flops. At these speeds a conventional error analysis becomes questionable. It easily can be shown that for a system of two linear equations with two unknowns computers even today deliver a result of which possibly not a single digit is correct. Such results strongly suggest trying to use the computer more for computing close bounds of the solution instead of murky approximations.

Dramatic advances in computer technology, in memory size, and in speed have been made since 1985. Arithmetic speed has gone from megaflops (10^6 flops), to gigaflops (10^9 flops), to teraflops (10^{12} flops), to petaflops (10^{15} flops), and it is already approaching the exaflops (10^{18} flops) range. This is not just a gain in speed. A qualitative difference goes with it. At the time of the megaflops computer a conventional error analysis was recommended in every numerical analysis textbook. Today the PC is a gigaflops computer. For the teraflops or petaflops computer conventional error analysis is no longer practical. An avalanche of numbers is produced when a petaflops computer runs. If the numbers a petaflops computer produces in one hour were to be printed (500 on one page, 1 000 on one sheet, 1 000 sheets 10 cm high) they would need a pile of paper that reaches from the earth to the sun[1] and back. In 2012 the fastest computers are listed to perform about 10 petaflops. With these the pile would grow from the earth to the sun in less than 3 minutes. This is hard to imagine. Light takes about 8 minutes for this distance.[2]

Computing indeed has already reached astronomical dimensions! With increasing speed, problems that are dealt with become larger and larger. Extending the word size cannot keep up with the tremendous increase in computer speed. Computing that is continuously and greatly speeded up calls conventional computing into question. Even with quadruple or extended precision arithmetic the computer somehow remains an experimental tool.

The capability of a computer should not just be judged by the number of operations it can perform in a certain amount of time without asking whether the computed result is correct. It should also be asked how fast a computer can compute correctly to 3, 5, 10 or 15 decimal places for certain problems. If the question were asked that way, it would

[1] The distance between the earth and the sun is about $150 * 10^6$ km.

[2] The speed of light is ca. 300 000 km/sec.

very soon lead to better computers. Mathematical methods that give an answer to this question are available for very many problems. Computers, however, are at present not designed in a way that allows these methods to be used effectively. Computer arithmetic must move strongly towards more reliability in computation. Instead of the computer being merely a fast calculating tool it must be developed into a scientific instrument of mathematics.

Mathematical analysis has designed algorithms that deliver highly accurate and completely verified results. These algorithms compute with intervals of real numbers. Intervals bring the continuum on the computer. An interval between two floating-point bounds represents the continuous set of real numbers between these bounds.

This book deals with computer arithmetic in a more general sense than usual, and shows how the arithmetic and mathematical capability of the digital computer can be enhanced in a quite natural way. The work is motivated by the desire and the need to improve the accuracy of numerical computing and to control the quality of computed results.

As a first step towards achieving this goal, the accuracy requirements for the elementary floating-point operations as defined by the IEEE arithmetic standard, for instance, are extended to the customary product spaces of computation: the complex numbers, the real and complex intervals, the real and complex vectors and matrices, and the real and complex interval vectors and interval matrices. All computer approximations of arithmetic operations in these spaces should deliver a result that differs from the correct result by at most one rounding. For all these product spaces this accuracy requirement leads to operations which are distinctly different from those traditionally available on computers. This expanded set of arithmetic operations is taken as a definition of what is called **advanced computer arithmetic** in the book.

Central to this treatise is the concept of semimorphism. It provides a mapping principle between the mathematical product spaces and their digitally representable subsets. The properties of a semimorphism are designed to preserve as many of the ordinary mathematical laws as possible. All computer operations of advanced computer arithmetic are defined by semimorphism.

The book has three antecedents:

(I) Kulisch, U. W., *Grundlagen des numerischen Rechnens – Mathematische Begründung der Rechnerarithmetik*, Bibliographisches Institut, Mannheim, Wien, Zürich, 1976, 467 pp., ISBN 3-411-015617-9.

(II) Kulisch, U. W. and Miranker W. L., *Computer Arithmetic in Theory and Practice*, Academic Press, New York, 1981, 249 pp., ISBN 0-12-428650-X.

(III) Kulisch, U. W., *Advanced Arithmetic for the Digital Computer – Design of Arithmetic Units*, Springer-Verlag, Wien, New York, 2002, 139 pp., ISBN 3-211-83870-8.

The need to define all computer approximations of arithmetic operations by semimorphism goes back to the first of these books. By the time the second book had been written, early microprocessors were on the market. They were made with a few thousand transistors, and ran at 1 or 2 MHz. Arithmetic was provided by an 8-bit adder. Floating-point arithmetic could only be implemented in software. In 1985 the IEEE binary floating-point arithmetic standard was internationally adopted. Floating-point arithmetic became hardware supported on microprocessors, first by coprocessors and later directly within the CPU. Of course, all operations of advanced computer arithmetic can be simulated using elementary floating-point arithmetic. This, however, is rather complicated and results in unnecessarily slow performance. A consequence of this is that for large problems the high quality operations of advanced computer arithmetic are hardly ever applied. Higher precision arithmetic suffers from the same problem if it is simulated by software.

During the course of the book it is shown that all operations of advanced computer arithmetic can be provided on the computer if, beyond conventional floating-point arithmetic, two additional features are made available by fast hardware:

I. fast hardware support for interval arithmetic and

II. a fast and exact multiply and accumulate operation or, equivalently, an exact scalar product.

Fast hardware circuitries for I. and II. are developed in Chapters 7 and 8, respectively. This additional computational capability is gained at very modest hardware cost. I. and II., in particular, are basic ingredients of what is called validated numerics or verified computing. With simple hardware circuitry, interval operations can be made as fast as floating-point operations. The simplest and fastest way for computing a scalar or dot product is to compute it exactly. By pipelining, it can be computed in the time the processor needs to read the data, i.e., it comes with utmost speed. Besides being more accurate, the new computer operations I. and II. greatly speed up the operations of advanced computer arithmetic. This would boost both the speed of a computation and the accuracy of its result. Advanced computer arithmetic opens the door to very many additional applications. Compared with conventional computer arithmetic all these applications are extremely fast.

This book has three parts. Part 1, of four chapters, deals with the theory of computer arithmetic, while Part 2, also of four chapters, treats the implementation of arithmetic on computers. Part 3, of one chapter, illustrates by a few sample applications how advanced computer arithmetic can be used to compute highly accurate and mathematically verified results.

Part 1: The different kinds of computer arithmetic for the customary product spaces of computation follow an abstract mathematical pattern and are just special realizations

of it. In the book the basic mathematical properties are extracted into an axiomatic approach to computer arithmetic. Abstract mathematical concepts are suited to focus the understanding on the essentials.

Frequently mathematics is seen as the science of structures. Analysis carries three kinds of structures: an algebraic structure, an order structure, and a topological or metric structure. These are coupled by certain compatibility properties, for instance: $a \leq b \Rightarrow a + c \leq b + c$.

It is well known that floating-point numbers and floating-point arithmetic do not obey the rules of the real numbers $\mathbb{R}$. However, rounding is a monotone function. So the changes to the order structure are minimal. This is the reason why the order structure plays a key role for an axiomatic approach to computer arithmetic.

All the product spaces under consideration carry an order structure with respect to which they are complete lattices. The computer representable subsets can be characterized as complete sublattices. A rounding is defined as a monotone mapping of the basic set into the computer representable subset. Arithmetic in the latter is defined by a monotone, antisymmetric rounding applied to the result of the operation carried out in the basic set. In case of interval spaces, the rounding is additionally upwardly directed, i.e., it maps the correct result of the operation in the powerset onto the least upper bound in the subset with respect to set inclusion as the order relation. The mapping principle just described is called a semimorphism. It is close to a homomorphism between ordered algebraic structures.

Definition of computer arithmetic by semimorphism in the product spaces under consideration does not necessarily directly lead to operations that are executable on the computer. It is, however, shown in the book that in all cases the operations can be reduced to computer executable operations via isomorphisms. These isomorphisms are to be established in the mathematical spaces in which the actual computer operations operate. This requires a careful study of the structure of these spaces. Their properties are defined as invariants with respect to semimorphisms. These concepts are developed in the first three chapters of the book.

The fourth chapter deals with interval arithmetic. In most of the preceding literature, interval arithmetic is defined and considered for closed and bounded real intervals. See, for instance the well-known books by R. E. Moore [457, 458], by G. Alefeld and J. Herzberger [28, 29], or by E. Hansen [219, 222]. This leads to well defined and well-known formulas for the arithmetic operations. Overflow and underflow, however, can occur and need special treatment. Another approach is taken by SUN's interval Fortran [725]. Here, intervals are considered over the extended real numbers $\overline{\mathbb{R}} := \mathbb{R} \cup \{-\infty, +\infty\}$ and $-\infty$ and $+\infty$ are permitted as elements of unbounded intervals. Then interval operations are to be defined for operations like $\infty - \infty, 0 \cdot \infty$, or ∞/∞ which leads to unusual and unsatisfactory operations.

This book avoids the difficulties of both approaches. Interval arithmetic just deals with sets of real numbers. A real interval is defined as a closed and connected set of real

numbers.[3] Since $-\infty$ and $+\infty$ are not real numbers, they cannot be elements of a real interval. They can only be bounds of an interval. Formulas for the arithmetic operations for bounded real intervals are well established in conventional interval arithmetic. It is shown in the book that these formulas can be extended to unbounded real intervals by a continuity principle. As a result, it is further shown in the book that for a bound $-\infty$ or $+\infty$ in an interval operand the bounds for the resulting interval can easily be obtained from the formulas for bounded intervals by applying well-established rules of real analysis for computing with $-\infty$ and $+\infty$. This new approach to real interval arithmetic leads to an algebraically closed calculus which is totally free of exceptions. It remains free of exceptions if the operations are mapped on a floating-point screen by the monotone, upwardly directed rounding.

Part 2: In Part 2 of the book, basic ideas for the implementation of advanced computer arithmetic are discussed under the assumption that the data are floating-point numbers. Algorithms and circuits are developed which realize the semimorphic operations in the various spaces mentioned above. The result is an arithmetic with many desirable properties, such as high speed, optimal accuracy, theoretical describability, closedness of the theory, and ease of use.

Chapters 5 and 6 consider the implementation of elementary floating-point arithmetic on the computer for a large class of roundings. The final section of Chapter 6 contains a brief discussion of all arithmetic operations defined in the product sets mentioned above as well as between these sets. The objective here is to summarize the definition of these operations and to point out that they all can be performed as soon as an exact scalar product is available in addition to the operations that have been discussed in Chapters 5 and 6.

Floating-point operations with directed roundings are basic ingredients of interval arithmetic. But with their isolated use in software, interval arithmetic is too slow to be widely accepted in the scientific computing community. Chapter 7 shows, in particular, that with very simple circuitry interval arithmetic can be made practically as fast as elementary floating-point arithmetic. To enable high speed, the case selections for interval multiplication (9 cases) and division (14 cases including division by an interval that includes zero) are done in hardware where they can be chosen without any time penalty. The lower bound of the result is computed with rounding downwards and the upper bound with rounding upwards by parallel units simultaneously. Also the basic comparisons for intervals together with the corresponding lattice operations and the result selection in more complicated cases of multiplication and division are done in hardware. There they are executed by parallel units simultaneously. The circuits described in this chapter show that with modest additional hardware costs, interval arithmetic can be made almost as fast as simple floating-point arithmetic. Such high

[3] In real analysis a set of real numbers is called closed if its complement is open.

speed cannot be obtained just by running many elementary floating-point arithmetic processors in parallel.

A basic requirement of advanced computer arithmetic is that all computer approximations of arithmetic in the usual product spaces should deliver a result that differs from the correct result by at most one rounding. This requires scalar products of floating-point vectors to be computed with but a single rounding.

The most natural way to accumulate numbers is fixed-point accumulation. It is simple, error free and fast. In Chapter 8 circuitry for *exact* computation of the scalar product of two floating-point vectors is developed for different kinds of computers. It is shown that an exact scalar product can be computed in the time the processor needs to read the data, i.e., it comes with utmost speed. No software simulation can compete with a simple and direct hardware solution. The hardware needed for the exact dot product is comparable to that for a fast multiplier by an adder tree, accepted years ago and now standard technology of every modern processor. The exact dot product brings the same speed up for accumulations at comparable costs.

To make the new capability conveniently available to the user a new data format called *complete* is introduced together with a few simple arithmetic operations associated with each floating-point format. *Complete arithmetic* computes all scalar products of floating-point vectors exactly. The result of complete arithmetic is always exact; it is complete, not truncated. Not a single bit is lost. A variable of type complete is a fixed-point word wide enough to allow exact accumulation (continued summation) of floating-point numbers and of simple products of such numbers.

If register space for the complete format is available, complete arithmetic is very fast. The arithmetic needed to perform complete arithmetic is not much different from what is available in a conventional CPU. In the case of the IEEE double precision format a *complete register* consists of about 500 bytes. Straightforward pipelining leads to very fast and simple circuits. The process is at least as fast as any conventional way of accumulating the products including the so-called partial sum technique on existing vector processors which alters the sequence of the summands and causes errors beyond the usual floating-point errors.

Complete arithmetic opens a large field of new applications. An exact scalar product rounded into a floating-point number or a floating-point interval serves as building block for semimorphic operations in the product spaces mentioned above.

Fast multiple precision floating-point and multiple precision interval arithmetic are other important applications. All these applications are very fast. Complete arithmetic is an instrumental addition to floating-point arithmetic. In many instances it allows recovery of information that has been lost during a preceding pure floating-point computation.

Because of the many applications of the hardware support for interval arithmetic developed in Chapter 7, and of the exact scalar product developed in Chapter 8, these two modules of advanced computer arithmetic emerge as its central components. Fast hardware support for all operations of *advanced computer arithmetic* is a fundamental

and overdue extension of elementary floating-point arithmetic. Arithmetic operations which can be performed correctly with very high speed and at low cost should never just be done approximately or simulated by slow software. The minor additional hardware cost allows their realization on every CPU.

Part 3: Mathematical analysis has provided algorithms that deliver highly accurate and completely verified results. Part 3 of the book goes over some examples. Such algorithms are not widely used in the scientific computing community because they are slow when the underlying arithmetic has to be carried out by software simulations on conventional processors.

The first section describes some basic properties of interval mathematics and shows how these can be used to compute the range of a function's values. Used with automatic differentiation these techniques lead to powerful and rigorous methods for global optimization. The following section then deals with differentiation arithmetic or automatic differentiation. Values or enclosures of derivatives are computed directly from numbers or intervals, avoiding the use of a formal expression for the derivative of the function. Evaluation of a function for an interval X delivers a superset of the function's values over X. This overestimation tends to zero with the width of the interval X. Thus for small intervals, interval evaluation of a function practically delivers the range of the function's values. Many numerical methods proceed in small steps. So this property together with differentiation arithmetic to compute enclosures of derivatives is the key technique for validated numerical computation of integrals and for solution of differential equations, and for many other applications.

Newton's method is considered in two sections of Chapter 9. It attains its ultimate elegance and power in the *extended interval Newton method*, which is globally convergent and computes all zeros of a function in a given domain. The key to achieving these fascinating properties is division by an interval that includes zero. It is used to separate diferent zeros from each other.

The basic ideas needed for verified solution of systems of linear equations are developed in Section 9.5. Highly accurate bounds for a solution can be computed in a way that proves the existence and uniqueness of the solution within these bounds. Mathematical fixed-point theorems, interval arithmetic combined with defect correction or iterative refinement techniques using complete arithmetic are basic tools for achieving these results.

In Section 9.6 a method is developed that allows highly accurate and guaranteed evaluation of polynomials and of other arithmetic expressions.

Section 9.7 shows how fast multiple precision arithmetic and multiple precision interval arithmetic can be provided using complete arithmetic and other tools developed in the book. A very impressive application for an extremely ill-conditioned problem is considered in [84], an iteration with the logistic equation (dynamical system) which shows chaotic behavior. Double precision floating-point or interval arithmetic totally

fail (no correct digit) after 30 iterations while long interval arithmetic still computes correct digits of a guaranteed answer after 2 790 iterations.

Finally, in Section 9.8 Kaucher arithmetic is briefly mentioned and a core of arithmetic operations is specified. Kaucher arithmetic extends the formulas for the arithmetic operations for closed and bounded real intervals to the entire $\mathbb{R}^2$. Specialists claim that with it important algorithms of computer geometry can be formulated in compact, closed form thus avoiding complicated analyses of arithmetic expressions by hands-on methods.

Of course, the computer may often have to work harder to produce verified results, but the mathematical certainty makes it worthwhile. After all, the step from assembler to higher programming languages or the use of convenient operating systems also consumes a lot of computing power and nobody complains about it since it greatly enlarges the safety and reliability of the computation.

Computing is being continually and greatly speeded up. Fast computers are often used for safety critical applications. Severe, expensive, and tragic accidents can occur if the eigenfrequencies of a large electricity generator, for instance, are erroneously computed, or if a nuclear explosion is incorrectly simulated. Floating-point operations are inherently inexact. It is this inexactness at very high speed that calls conventional computing, just using naïve floating-point arithmetic, into question.

As shown in the book, properly developed interval arithmetic is an exception-free calculus. With very little extra hardware interval arithmetic can be made as fast as simple floating-point arithmetic. The tremendous progress in computer technology should be accompanied by the extension of the mathematical capacity of the computer. The basic components of advanced computing, floating-point arithmetic, interval arithmetic, and an exact scalar product should be provided by the computer's hardware. This would boost both the speed of a computation and the accuracy of its result.

A bibliography of more than 700 entries terminates this book. It is recommended to everyone who is interested in how a computer works and how it could work better and faster. The detailed elaboration of a vision of future computing should be of particular interest to designers and manufacturers of computers.

This book can, of course, be used as a textbook for lectures on the subject of computer arithmetic. If one is interested only in the more practical aspects of implementing arithmetic on computers, Part 2, with acceptance *a priori* of some results of Part 1, is also suitable as a basis for lectures. Part 3 can be used as an introduction to verified computing.

The second previous book was jointly written with Willard L. Miranker. On this occasion Miranker was very busy with other studies and could not take part, so this new book has been compiled solely by the other author and he takes full responsibility for its text. However, there are contributions and formulations here which go back to Miranker without being explicitly marked as such. I deeply thank Willard for his collaboration on the earlier book as well as on other topics, and for a long friendship. Contact with him was always very inspiring for me and for my Institute.

I would like to thank all former collaborators at my Institute. Many of them have contributed to the contents of this book, have realized advanced computer arithmetic in software on different platforms and in hardware in different technologies, have embedded advanced computer arithmetic into programming languages and implemented corresponding compilers, developed problem solving routines for standard problems of numerical analysis, or applied the new arithmetic to critical problems in the sciences. Among these colleagues are: Christian Ullrich, Edgar Kaucher, Rudi Klatte, Gerd Bohlender, Dalcidio M. Claudio, Kurt Grüner, Jürgen Wolff von Gudenberg, Reinhard Kirchner, Michael Neaga, Siegfried M. Rump, Harald Böhm, Thomas Teufel, Klaus Braune, Walter Krämer, Frithjof Blomquist, Michael Metzger, Günter Schumacher, Rainer Kelch, Wolfram Klein, Wolfgang V. Walter, Hans-Christoph Fischer, Rudolf Lohner, Andreas Knöfel, Lutz Schmidt, Christian Lawo, Alexander Davidenkoff, Dietmar Ratz, Rolf Hammer, Dimitri Shiriaev, Manfred Schlett, Matthias Hocks, Peter Schramm, Ulrike Storck, Christian Baumhof, Andreas Wiethoff, Peter Januschke, Chin Yun Chen, Axel Facius, Stefan Dietrich, and Norbert Bierlox. For their contributions I refer to the bibliography.

I owe particular thanks to Axel Facius, to Gerd Bohlender, and to Klaus Braune. Axel Facius keyed in and laid out the entire manuscript of the first edition of the book in LaTeX and he did most of the drawings. Drawings were also done by Gerd Bohlender. Klaus Braune helped to prepare the final version of the text. I thank Bo Einarsson for proofreading the book. I also thank my colleagues at the institute Götz Alefeld and Willy Dörfler for their support of the book project.

I gratefully acknowledge the help of Neville Holmes who went carefully through great parts of the manuscript, sending back corrections and suggestions that led to many improvements. His help was indeed vital for the completion of the book.

Contents