Interval Arithmetic in High Performance Technical Computing

Version 1.0

September 2002

A White Paper

4150 Network Circle
Santa Clara, CA 95054
800-681-8845
www.sun.com/processors/whitepapers

Table of Contents

CHAPTER 1

Introduction

Interval Arithmetic is a computing system that makes it possible to:

- automatically perform rigorous error analysis by computing mathematical bounds on the set of all possible problem solutions, and
- solve nonlinear problems that were previously thought to be impossible to solve.

Computing with interval arithmetic produces numerical proofs, or 100% confidence intervals. Intervals have been used in just this way to prove mathematical theorems that were not possible to prove otherwise[1]. This claim can be made for no other known numerical computing system. By using interval algorithms to solve nonlinear problems, more accurate mathematical models of physical phenomena become practical. Interval arithmetic is arguably the best and most efficient way to safely translate ever-increasing computer speed into mission-critical problem solutions that are otherwise impractical or impossible to obtain. From interval bounds on uncertain values, interval arithmetic computes guaranteed bounds on the set of all possible result values.

Sun Microsystems, Inc. is one of the few computer companies to currently offer both language and hardware support for computing with intervals. With this support, it is practical to compute mathematical bounds on solutions to difficult linear and nonlinear problems. Interval arithmetic is a feature intrinsically supported with a rich set of features and extensions in the Sun™ ONE (Open Network Environment) Studio 7, Compiler Collection Fortran 95 compiler, and in a C++ class library. Sun ONE Fortran 95 is a language component of the Sun ONE High Performance Technical Computing 6 Update 2 product line.

Interval Arithmetic is also supported in Sun's UltraSPARC® III processor's VIS™ Instruction Set[2]. The SIAM (Set Interval Arithmetic Mode) rounding mode, is available to support interval arithmetic. Interval-specific hardware instructions for the basic arithmetic operations (+, –, ×, ÷) in single, double, and quadruple precision floating-point will eliminate the existing performance deficit in the time required to compute interval versus floating-point expressions.

1. http://mathworld.wolfram.com/news/2002-02-13_smale14th
2. VIS Whitepaper

A generic nonlinear interval solver library is the key to raising the level of abstraction at which High Performance Technical Computing (HPTC) software application developers and end users can work. Sun is perfectly positioned to efficiently supply both the needed hardware support and the generic interval solver library:

- Sun is committed to deliver the highest quality floating-point mathematical function library. This is a key requirement for interval versions of these basic functions to return narrow guaranteed bounds.
- Sun's latest developer suites include intrinsic compiler support for interval data types. In addition to Fortran, intrinsic interval support can be added to C, C++, and the Java™ programming language.
- By leveraging compiler support to compute narrow intervals and increase speed, Sun can exploit its position as a computer systems company that provides the entire hardware and software stack to perform the most demanding mission critical computing.
- Sun owns the intellectual property for basic interval algorithms that can be implemented in hardware and for new state-of-the-art algorithms to solve nonlinear systems of equations and global optimization. By implementing these algorithms, Sun has the opportunity to establish itself as the leading provider of this key, "generic[3]" technology[4].

3. Interval algorithms for solving nonlinear equations and global optimizations are purely mathematical. They are not specific to any discipline, or vertical market. In this sense, they are "generic."

4. Sun has applied for 26 arithmetic interval related patents on fundamental technology that is needed to efficiently and completely implement interval computing systems using digital computers.

CHAPTER 2

Fallible Measures

Except when counting, almost all measurements contain uncertainty. The most accurate (to about 10 decimal digits), frequently-used, measured physical constants are provided by the National Institute of Science and Technology[5]. The implication is that in almost any calculation, inputs should be intervals to bound input uncertainty.

To illustrate the fact that interval data need not be precise to be useful, suppose one wants to calculate how long it will take to double a sum of money if invested at an annual interest rate (compounded monthly) in the interval [3, 6]%. The minimum value in months is the interval [11 yr. 7 mo., 23 yr. 2mo.].

Alternatively, to illustrate that the interval evaluation of an expression does not always increase interval result width, evaluate the expression:

$f(x) = \ln(x)/x$ over the interval $X = [2.716, 2.718]$.

The answer is [0.3678793, 0.3678795].

Using the simple convention that displayed numbers are accurate to ±1 unit in the last printed digit, with $X = 2.717$, the value of $f(X)$ is 0.3678794.

While representing fallible values, using the familiar "$x \pm \varepsilon$" notation is convenient, computing with this representation is cumbersome. Using assumed statistical distributions for input uncertainty is even more cumbersome and risky. If input uncertainty and finite precision arithmetic errors are simultaneously considered, the complexity of nonlinear expression error analysis quickly becomes overwhelming. Rigorous error analysis of even modestly small algorithms is so labor-intensive it is rarely done. At the same time, there is an ever increasing requirement to use computers for mission-critical computations in which errors must be bounded because the consequences of these errors can be catastrophic[6].

5. http://physics.nist.gov/cuu/Constants/index.html

6. http://www.nytimes.com/2002/06/25/science/physical/25COMP.html?pagewanted=print&position=bottom

A single number written in scientific notation (such as 2.35×10^{-3}) contains implicit accuracy information in the number of digits of the fractional part – in this case the three digits 2.35. A frequently used convention is to let the last displayed digit be incorrect by at most ±1 unit. Unfortunately, no accuracy information exists in the format used to store such numbers in computers. Using floating-point computer notation, when the number 2.35E-03 is entered, it is incorrectly stored as if it were much more accurate.

Unfortunately, two problems arise:

- First, the actual uncertainty in input values has no convenient way to be either represented or rigorously propagated through a floating-point computation.
- Second, most real numbers cannot be represented as finite precision floating-point values.

Without intervals, validated arithmetic (with guaranteed bounds) has been almost unknown in numerical computing. To the exclusion of any concern for accuracy, focus has been only on speed as measured by floating-point operations per second (FLOPS). Evidence of this fact is not difficult to find. For example, the Fortran language standard does not contain a single word about accuracy of floating-point or even integer results[7].

As more critical decisions are based on unvalidated computing, the resulting risks becomes essentially unbounded. The “Patriot Missile Failure,” which was traced to an inaccurate calculation of the time since boot due to computer arithmetic errors. The “Explosion of the Ariane 5” forty seconds after its lift-off was caused by an arithmetic conversion in software. Specifically a 64-bit floating-point number was converted to a 16-bit signed integer. The initial number was greater than 32,767 and conversion failed. Clearly, there is a growing requirement to compute with guaranteed accuracy the first time and to solve increasingly difficult nonlinear problems[8].

7. http://www.j3-Fortran.org/

8. http://www.nytimes.com/2002/06/25/science/physical/25COMP.html?pagewanted=print&position=bottom

CHAPTER 3

Interval Arithmetic

The requirements for rigorous error analysis are logically satisfied by interval arithmetic. An interval is simply a range of numbers bounded by the interval's endpoints. For example, the range from 950 to 1050 is written as [950, 1050] and contains all the numbers between and including 950 and 1050. Hence, in planning a trip, assume the following: number of miles traveled will be bounded by the interval [950, 1050] miles; the number of miles traveled per gallon of gasoline is bounded by the interval [20, 23] miles; and the price of gasoline per gallon is bounded by the interval $[1.15, 2.00]. Performing arithmetic on the given intervals produces the interval $[47.50, 105.00] that must contain the set of all possible costs of gasoline for the planned trip.

The width of any interval is a natural accuracy measure. Computing with intervals makes accuracy as visible as program execution time. A direct consequence of seeing numerical accuracy is the motivation to develop interval procedures and algorithms that quickly return ever narrower (more accurate) intervals.

The two common ways to improve numerical accuracy and reduce interval width are the following:

- Reduce the width of input values.
- Increase the number of interval data values used to compute interval bounds on a given parameter.

Although the first is obvious, the second is more subtle because working with intervals leads to new unfamiliar ways to compute parameter bounds from fallible interval data.

For example, given n interval observations X_i (i = 1, … ,n) of the same true value, say t, the best way to compute a narrow interval bound T on t is not to compute the average value (or arithmetic mean) of the interval observations. Rather, the intersection of the observations is used. The logic for this is simple: Given that each and every observation X_i must contain t, so must their intersection. If all the

observations do not contain at least one common value, then their intersection is empty[9]. The beauty of this interval procedure is that the empty outcome is undeniable proof that something is wrong. The assumed error in the observations might be too small or, because of uncontrolled factors in the measurement procedure, there might really be no single value t in the given observations. That is, the observation model (or theory) is wrong.

Another simplicity of the interval parameter estimation procedure is that it requires only interval observation error bounds. No assumption about statistical error distributions must be made. Nevertheless, if available in any of a variety of forms, statistical distribution information can also be rigorously used with interval algorithms, even when computing complicated nonlinear expressions.

Three properties of intervals and interval arithmetic precisely link the fallible observations of real engineering and scientific observations to mathematics and floating-point numbers:

- Any contiguous set of real numbers (or a continuum) is represented by a containing interval.
- Intervals provide a convenient and mechanical way to represent and compute error bounds from fallible data.
- With the use of "directed rounding," the most important properties of infinite precision intervals are preserved in finite precision floating-point interval arithmetic. Directed rounding is commonly available on all computers supporting the IEEE 754 floating-point standard. Indeed, the reason for directed rounding in the floating-point standard is to support rigorously computing interval bounds.

3.1. Interval Arithmetic Operation Example

Let [a, b] and [c, d] denote two intervals in which a and c are the lower bounds and b and d are the upper bounds, respectively. Further, let the interval endpoints be finite precision floating-point numbers. Then interval addition is defined:

$$[a, b] + [c, d] = [\downarrow(a + c), \uparrow(b + d)]$$

where the arrows indicate the rounding direction for the computed sum. All the basic interval operations for finite real and complex interval arithmetic have been developed starting with R. E. Moore in 1957[10]. Sun's compiler has been extended to support infinite intervals, meaning one can divide by intervals containing zero. There is also support for computing other "indeterminate forms," such as $\infty \div \infty$ and $0 \div 0$.

The field of interval analysis has stimulated the creation of interval algorithms to solve nonlinear problems, including nonlinear systems of equations and global nonlinear optimization. With these algorithms, otherwise numerically unsolvable problems can be numerically solved while bounding errors from all sources,

9. An empty interval is the same thing as the empty set
10. http://www.interval.louisiana.edu/Moores_early_papers/bibliography.html

including: input data uncertainty, modeling errors, machine rounding errors, and all their interactions. In the future, safely solving nonlinear problems will become increasingly important as the speed of computers and parallel processing are used to replace physical experiments that are too costly or impractical to conduct.

There are indications that intervals are currently being used by both the automotive and aerospace industries to validate on-board processor optimal control software. Unfortunately, because these applications are perceived to be such a competitive advantage, no public statements about them have yet been made. The number of significant real interval applications is quietly increasingly every year.

When the interval computing "tsunami" arrives, Sun is ideally positioned to ride this wave into the future of High Performance Technical Computing (HPTC). Once interval solutions to engineering design problems become commercially available, product liability and safety will force manufacturers to use interval-based design tools. They will simply have no other choice[11].

3.2. Interval Arithmetic Advantages

In many cases, interval algorithms are actually faster than conventional algorithms. Moreover, for important interval algorithms called "contracting maps," widths decrease as algorithms proceed. The interval version of Newton's method for finding (and bounding) roots of nonlinear equations is an example of a "contracting map."

Important interval algorithms are naturally parallel, because they progress by deleting regions where solutions are proved not to exist. Because these problems are also "generic" (neither vertical-market specific nor problem specific), intervals provide the only known general algorithms that achieve linear speedup as the number of processors increases in parallel computing systems. These profoundly parallel interval algorithms can even run on low-power "green[12]" yet massively parallel systems.

---

11. It is the law in the field of product liability, that if a producer of a product does not use the best commercially available design techniques, then the producer is liable for the consequence of a flawed product design. Current practice in the computing industry is to ship every system and piece of software with a disclaimer stating that these systems are not designed for mission critical applications in which there could be serious consequences from erroneous results. When computing with floating-point numbers, nothing else is logically possible, as it is generally impossible to guarantee the accuracy of floating-point computations. With intervals, on the other hand, it is logically and technically possible to "warrant" that the results of interval computations are bounds on the set of all possible answers to the problem defined by the user's code. Of course, the code will have to have passed a stringent "lint" certification. However, such a test and such a warranty are logically possible using intervals. They are not using floating-point computing.

12. http://www.nytimes.com/2002/06/25/science/physical/25COMP.html?pagewanted=print&positon=bottom

Additional interval advantages include:

- A closed mathematical system that eliminates the need for complicated exception handling.
- Otherwise numerically unstable algorithms can be safely used, thereby allowing substantial speedups for most applications.
- Sums and dot products can be performed without need for sorting to guard against rounding errors (although narrower width, if needed, can be achieved by sorting).
- Precise argument reduction for periodic functions like sin(x) and cos(x) is only required for degenerate (zero width) interval arguments.
- For increased speed, interval algorithms can dynamically reduce the precision of interval endpoints when computing with wide intervals.
- Embedded differential equations need only be implicitly solved in the process of using interval algorithms to find and bound the global minimum of a nonlinear function.

CHAPTER 4

Real World Examples

Almost every computational problem can benefit from computing with intervals. Intervals even provide a consistent framework within which to evaluate the total quality (speed and accuracy) of different interval algorithms on different machines.

Suppose the result of some complicated computation, or even a computer simulation, is the single quantity (*f*) that is a function of a number of measured values, such as space (x, y, z) and time (t). Suppose further that from physics, it is impossible for *f*(x, y, z, t) to be outside the interval F(x,y,z,t), where F is an interval enclosure of *f*. If true values of x, y, z, and t are assumed to be contained respectively in the intervals X, Y, Z, and T, an interval bound on the true value of *f* can be simply computed from the interval evaluation of F(X, Y, Z, T). This fact was first proved by Moore and later dubbed "The Fundamental Theorem of Interval Arithmetic." The total absolute error E_a is bounded by the width of the interval evaluation of *f*: $E_a = w(F(X, Y, Z, T))$, where $w([a,b]) = b - a$ is the width of the interval [a, b].

Even if the interval bound on *f* contains zero, an extended interval (including infinite endpoints) bound on relative error can be computed:

$$E_r = E_a / \text{abs}(F(X,Y,Z,T))$$

where "abs" is the interval extension of the absolute value function.

These two definitions permit valid comparison of interval algorithms and a way to rigorously assess performance including both speed and accuracy.

For example, two very different computers using very different interval approaches to solve a given problem can be compared using the absolute or relative error bound achieved in a given amount of time. Alternatively, the time required to achieve a given accuracy can be measured. The trick is to compute the interval bound F in practical real world situations.

Examples of problems to which interval methods have been successfully applied include:

- Chemical Process Engineering[a13]
- Computing Guaranteed Parameter Bounds from Fallible Data[b]
- Optimal design of Quantitative Feedback Control Systems[c]
- Many more exist in published papers and reports and some are quietly being used commercially.

13. Notes a, b, and c appear at the end of this document.

CHAPTER 5

Sun Support for Interval Arithmetic

Hardware support for interval instructions is provided in UltraSPARC III processors with the “Set Interval Arithmetic Mode” (SIAM) instructions. These instructions improve the efficiency of interval arithmetic by enabling the rounding mode bits in the floating-point status register (FSR) to be overridden without the overhead of modifying the RD field of the FSR and the resulting pipeline flush. Typical interval performance improvement from the SIAM instructions has been measured to be approximately 30%[d14].

Software support for interval arithmetic is featured intrinsically in Sun’s Fortran compiler and in a class library for C++. Interval support in both Fortran and C++ include three interval data types, one each for: single, double, and quadruple (128-bit) precision[e]. Additional hardware support is an obvious next step because interval multiplication in software requires up to nine floating-point operations. Clearly, a performance enhancement will be experienced when interval basic arithmetic operations (+, –, ×, ÷) are included in next generation processors.

With a shipping library of interval nonlinear solvers, and with compiler and hardware support, Sun is positioning itself to be unassailable in HPTC.

14. Notes d and e appear at the end of this document.

CHAPTER 6

Conclusion

Moore's Law (reducing chip feature size by a factor of 2 every 18 months) and the market for "virtual experimentation"[15] are increasing the pressure to find verifiable ways for safe computation. The National Science Foundation has recently announced a $[4, 6]M grant program to fund "trusted computing" research[16]. Sun is the only commercial computer manufacturer that is currently supplying compiler support for interval computing – the only known practical way to compute mathematical bounds on solution sets.

The principle barrier to mainstream use of interval arithmetic is the effort required to successfully apply interval algorithms to solve commercial, industrial, financial, and scientific problems. Numerous successful interval applications already exist, but have required more time, effort, and skill, than typical commercial software manufacturers and users can afford. An important first step in making the use of intervals commercially practical has been taken with Sun's Fortran intrinsic compiler support. Extending intrinsic support to other languages and easy-to-access fast solver libraries are the next steps.

With more interval support in education, hardware, and software, intervals will become the natural way to think about and compute numerical solutions to physical problems.

15. http://www.nytimes.com/2002/06/25/science/physical/25COMP.html?pagewanted=print&position=bottom

16. http://www.nsf.gov/pubs/2002/nsf01160/nsf01160.html

Work in Progress: The Dependence "Problem" or "Opportunity"

Even elementary school children learn that $2 - 2 = 0$. In middle school, they learn that $x - x = 0$, where x can be any real number. For a real interval, such as [2, 3] then $[2, 3] - [2, 3] = [-1, +1]$, which is not zero. Why? The reason is that the two occurrences of the interval [2, 3] are not necessarily dependent. They could be two separate measurements that just happen to have the same interval endpoints. Because interval arithmetic guarantees to contain the set of all possible results, the worst case assumption is implicitly made: namely that all intervals are independent.

However, if it is known that two intervals are dependent, this information can be used to compute narrower intervals that are still bounds on the set of all possible results. As a simple example, given the interval variable $X = [x_l, x_u]$, then $X - X = 0$, which is also the degenerate interval [0, 0].

Failing to take dependence into account has been a "difficulty" that has caused many interval results to be unnecessarily wide. The dependence opportunity is to provide tools to rigorously and automatically take dependence information into account. This is "work in progress" that can be integrated into compiler support for interval data types.

While it is easy to compute interval bounds, it is often unnecessarily time-consuming to compute usefully narrow (or tight) bounds without tools to automate the use of dependence information. Bounds that are as narrow as possible are called "sharp." Computing sharp bounds is a member of the class of extremely difficult computing problems, called NP-hard. The "dependence opportunity" is to develop fast algorithms to compute approximate interval bounds that, while still bounds, are narrow enough to be practically useful. In practice, sharp intervals are not required.

Notes

a - There are many important problems in the design of chemical process controls and oil refineries. Without intervals, totally incorrect solutions are not hard to find in published papers and reports. Using intervals, correct results can be guaranteed. Specific problems that intervals have successfully solved include the computation of phase stability using excess Gibbs energy models, mixture critical points, solid-fluid equilibrium points, and parameter bounds in vapor-liquid equilibrium models. (For an example, visit www.nd.edu/~markst/Interv.html.)

b - Newton's gravitational constant G is difficult to measure. Interval methods were used a few years ago at the University of Karlsruhe to compute an interval bound on G. The computed bound was sufficiently different from the accepted approximate value that other experiments were designed to estimate G. If they all compute valid interval bounds, then the intersection of these intervals must contain the true value of G. If any intervals do not overlap, this proves there is a flaw in at least one of the following or some other part of the method used to compute bounds on G: the theoretical model used to estimate G, the experimental apparatus, or the analysis of the data. The point is that computing with intervals is the only known way to prove some combination of theory and/or experiments is measurably flawed (for more information, visit http:// www.npl.washington.edu/eotwash/gconst.html).

c - The goal of optimal feedback control system design is to construct simple, low-order, minimum-bandwidth controllers that satisfy given performance constraints in the presence of uncertainties in the form of plant changes and/or external disturbances (for more information, visit http://www.ee.iitb.ac.in/~nataraj/projects.htm).

d - Test runs were executed against the library libsunimath built with and without SIAM support with performance increments of 26.44%, 35.59% and 37.70%. Forte™ Developer 7 Fortran 95 6.1, Sun Workshop™ 6 update 2 C 5.3, and Forte Developer 7 C 5.4.

e - The interval classes include; operations and mathematics functions that form a closed mathematical system.This means that valid results are produced for any possible operator-operand combination, including division by zero and other indeterminate forms involving zero and infinities. Three types of interval relational functions are supported; certainty, possibility and set. The classes also support interval-specific functions such as intersection and interval_hull.