64-bits CPUs: Minimizing Errors in Floating-point Arithmetic

Cristiana Manuela Guimarães de Freitas

Departamento de Informática, Universidade do Minho
4710 – 057 Braga, Portugal
Cris_tiana@iol.pt

Abstract. Using finite systems to represent non-finite quantities may lead to serious hazards in real life. Some of these are here presented together with a short explanation. Manufacturers of 64-bit CPUs are developing newer approaches to reduce these negative factors, from increasing bit representation to interval arithmetic computation; an overview of two of these approaches – from Intel and Sun – are here analysed.

1 Introduction

Nowadays, many critical decisions are made by computational methods, depending on the accuracy of results.

Squeezing infinitely many real numbers into a finite number of bits must have its risks. And floating-point operations that introduce more rounding error definitely increase those risks.

Throughout this paper, these issues will be exploited, along with descriptive examples of numerical errors and its consequences.

Different methods to reduce the risk of numerical errors are being implemented in hardware. Intel is increasing bit representation and trying to minimize rounding error due to arithmetic operations.

An interesting approach is the one of representing real numbers as floating-point bounded intervals, guarantying 100% confidence bounds on the set of all possible result values. Sun is providing software and hardware to deal with intervals.

2 Floating-point Format: Basics

The most widely used representation of numbers in computer systems is the floating-point representation using base 2 (examples of other representations are *floating slash* and *signed logarithm*).

A normalized floating-point number x is represented as a concatenation of a sign-bit, an M-bit exponent field and an N-bit significand field. Mathematically:

$$x = \sigma . s . 2^e ;$$

$\sigma = \pm 1$; $s \in [1, 2[$ significand; e exponent.

Higher precision is obtained for large values of N and higher range is obtained for large values of M. Usually, a binary floating-point representation uses at least 32 bits: 1 bit represents the sign, 8 bits for the exponent and 23 bits for the significand (in some floating-point formats, the most significant bit of the significand is not represented, its assumed value is 1, leaving 24 bits for the significand).

Other formats permit the most significant bit to be 0. These floating-point representations are called denormalized.

3 Numerical Errors

Given any real number, every computer system is condemned to store it in a finite number of bits. Naturally, this process often requires an approximate representation.

Also, given any real numbers represented using N bits, it is always possible to operate simple calculations that produce quantities that cannot be exactly represented using N bits. Therefore, the result of a floating point calculation must often be rounded in order to fit back into its finite representation.

There are three reasons why a real number *x* might not be exactly represented as a floating-point binary number using a fixed number N of bits:

- The real number *x* is out of range, meaning that *x* is bigger than the biggest (or smaller than the smallest) number represented as a floating-point using N bits.
- The real number *x* is not out of range but its exact representation uses more than N bits.
- There is no finite exact representation using a floating point binary number, using a finite number of bits.
  Such a number *x* could be an irrational number; for example, π:
  11.0010010000111111…
  Another example is the decimal number 0.1: although it has a finite decimal representation, in binary it has an infinite repeating representation:
  0.00011001100110011…

Next, some examples are presented that intend to illustrate the risks inherent to computations involving rounding errors.

3.1 Catastrophic Cancellation:

The area of a triangle can be calculated as

$$A = \sqrt{s(s-a)(s-b)(s-c)}\,, \tag{1}$$

where *a*, *b* and *c* are the lengths of sides of the triangle and $s = (a + b + c)/2$, s is the semi perimeter of the triangle.

Now, suppose the triangle is very flat, being $a \approx b + c$. Therefore, $s \approx a$ and the calculation of $(s - a)$ may lead to a relevant error, an error that will be amplified when multiplying by $s(s - b)\,(s - c)$.

If we consider that the input values of *s* and *a* may contain rounding errors (they may be the results of inaccurate measurements or of floating point operations), the computed value of $(s - a)$ may lead to catastrophic cancellation: when subtracting two nearby quantities, the most significant digits in the operands match and cancel each other, many of the accurate digits disappear, leaving mainly digits "contaminated" by rounding error. So, the relative error committed becomes excessively large.

A formula containing catastrophic cancellation can sometimes be rearranged to eliminate or reduce the problem. In this particular case, we can reduce the consequences of cancellation by using the equivalent formula (2) instead of formula (1).

$$A = \frac{\sqrt{(a+(b+c))(c-(a-b))(c+(a-b))(a+(b-c))}}{4}, \; a \geq b \geq c \qquad (2)$$

The computed area using formula (2) will return more accurate results.

3.2 Finite Precision Can Lead to Disaster

On February 25, 1991, during the Gulf war, an american Patriot missile battery in Dharan, Saudi Arabia, failed to intercept an incoming Iraqi Scud missile, due to a software problem. The Scud struck an american army barracks and killed 28 soldiers.

An official report explained the cause of the failure. Time was measured in the internal clock of the Patriot missile in tens of seconds. The time in tens of seconds was multiplied by 1/10 to produce the time in seconds. This calculation was performed using 24 bits for the significand of 1/10. Since binary representation of 1/10 has an infinite repeating representation, as seen above, the computed binary value of 1/10 which was

0.00011001100110011001100

introduced an error of about 0.000000095 decimal.

But the Patriot battery had been up around 100 hours and simple calculations shows that the resulting time error was of about 0.34 seconds:

$$0.000000095 \times 10 \times 60 \times 60 \times 100 = 0.34$$

A Scud travels at about 1676 meters per second which means that it traveled more than half a kilometer during the 0.34 seconds that weren't computed.

3.3 Finite Range Can Lead to Disaster

On June 4, 1996, thirty seconds after lift off, an Ariane 5 rocket launched by the European Space Agency veered off its flight path, broke up and exploded. The failure of the rocket was caused by a conversion problem in the software which led to a complete loss of guidance and altitude information. A Board of inquiry explained the cause of the failure:

The internal Inertial Reference System software exception was caused during execution of a data conversion from 64-bit floating-point to 16-bit signed integer value. The floating-point number which was converted had a value greater than what could be represented by a 16-bit signed integer.

4 Binary Floating-point Arithmetic in IA-64 Architecture

Intel's IA-64 processor is an extension into a 64-bit architecture. The processor's floating-point architecture was designed in order to achieve three main goals:

- High performance
- High floating-point accuracy
- Compliance with IEEE Standards for floating-point data formats and arithmetic

In order to achieve high performance, IA-64 floating-point architecture includes some special new instructions such as the *fused floating-point multiply-add* operation along with pipelined floating-point units allowing parallelism for operations. Control and data speculation and large register files also contribute for improvement of performance.

High precision is obtained thanks to the definition of several distinct data types and new instructions (*fused floating-point multiply-add, fused floating-point multiply-subtract, fused floating-point negative multiply-add ...).*

In the IA-64, floating-point values in several formats rest in registers and can be addressed in any order. *Table* 1 lists the various formats provided by this processors in which are included the ones recommended by the IEEE Standard: single precision, double precision and double-extended precision. M stands for bits in the exponent field and N bits in the precision field.

Format	***Format parameters***	
Single precision	M = 8	N = 24
Double precision	M = 11	N = 53
Double-extended precision	M = 15	N = 64
Pair of single precision floating-point numbers	M = 8	N = 24
IA-32 register stack single precision	M = 15	N = 24
IA-32 register stack double precision	M = 15	N = 53
IA-32 double-extended precision	M = 15	N = 64
Full register file single precision	M = 17	N = 24
Full register file double precision	M = 17	N = 53
Full register file double-extended precision	M = 17	N = 64

Table1: IA-64 floating-point formats

In memory, floating-point numbers are saved as single precision, double precision or double-extended precision.

An instruction may determine a specified floating-point format. Otherwise, the 64-bit floating-point status register (FPSR) contains a status field which determines precision control (pc) and widest-range-exponent (wre), along with bits that control flushing to zero, traps signaling, rounding mode, etc.

IA-64 supports four rounding modes (complying with IEEE standard): rounding to nearest, rounding to zero, rounding to positive infinity and rounding to negative infinity. It is also able of signaling all the “IEEE traps”: invalid operation, divide by zero, overflow, underflow and inexact result.

4.1 Operations

As referred before, new instructions for operations are included in hardware of IA-64 processor. In fact, these types of instructions are actually sets of operations. One example of such an instruction is the fused floating-point multiply-add operation.

The fused multiply-add operation, depending on input parameters, is able to calculate:

$$a \times b + c \; ;$$
$$a \times b \; ;$$

$$b + c$$

The main benefit of this fusion is that, if *a* and/or *b* and/or *c* are "contaminated" with rounding errors, the computation of $a \times b + c$ is done with only one rounding error ε. The computed result will be:

$$(a \times b + c)\,(1 + \varepsilon)$$

Generally, a computation using pure add and multiply operations, leads to a biggest relative error since two errors, ε_1 and ε_2, are computed:

$$\left(a \times b\,(1 + \varepsilon_1) + c\right)\,(1 + \varepsilon_2)$$

Similarly:

- fused multiply-subtract operation computes $a \times b - c$
- fused negative multiply-add operation computes $-a \times b + c$
- fused negative multiply-subtract operation computes $-a \times b - c$

Other operations such as division, square root, remainder and conversions are implemented in software. Full register file double-extended precision is required precisely in some software operations like division and square root.

5 Interval Arithmetic

Interval arithmetic is used to evaluate arithmetic expressions over sets of numbers contained in intervals. Any interval arithmetic result is a new interval that is guaranteed to contain the set of all possible resulting values.

Nowadays, computing speed provides many opportunities and risks. Unfortunately, the existing floating-point paradigm can neither exploit all the opportunities nor avoid the risks. There are two essential reasons for that:

- Floating-point numbers are disconnected from the mathematical continuum of real numbers since most real numbers cannot be represented exactly using computed floating-point numbers.
- Floating-point numbers are disconnected from inaccurate measurements in science and engineering. In fact, floating-point numbers in a computer do not contain any information about their accuracy. Normally, no error analysis is automatically done by computers using floating-point numbers and, because programming error analysis is difficult, it is rarely done.
  If computer results are used to make critical decisions, computing values without interval bounds is synonymous with risk.

On the contrary, using intervals instead of floating-point numbers:

- A single pair of interval endpoints represents an infinite number of values, a continuum, which gives intervals a logically rigorous connection to mathematics that does not exist for computed floating-point numbers.
- The width of an interval can be used to bind the error of an approximate value. The resulting interval errors bounds are mathematically rigorous, they contain the set of all possible results.

5.1 Solving Problems Using Interval Arithmetic

Computing with intervals leads to a tight logical connection between computing and reality. In fact, computing with Interval arithmetic produces numerical proofs, or 100% confidence intervals, which permits solving nonlinear problems, including design optimization, that were previously thought to be impossible to solve.

Interval computations have already been successfully applied to in solving many problems in science and engineering:

- Chemical process engineering
- Computing guaranteed parameter bounds from fallible data
- Optimal design of quantitative feedback control systems

Many more applications exist, along with published papers and reports and some are being quietly used in order to keep competitive advantage.

One of the interval computing success stories was the solution of the long-standing geometric problem of *determining the least area surface enclosing two given equal volumes*. This problem was proposed by the Belgian physicist J. Plateaux more than hundred years ago.

Until 1995, there was no proved solution for this problem, although physical experiments suggested that the desired surface should be a “double bubble”, a surface formed by two spheres separated by a flat disk (that meet along a circle at an angle of 120 degrees). However, several other surfaces (“torus bubbles”) have been proposed.

The solution of the problem was recently proven by Joel Hass from Department of Mathematics, University of California at Davis and Roger Schlafly from the Real Software Co. They manage to prove that double bubble is the minimizing surface by proving first that the solution could only be a double bubble or a torus bubble and then, using interval computations, they proved that, for all possible values of parameters, the area of the torus bubble exceeds the area of the double bubble.

5.2 Sun’s Support for Interval Arithmetic

UltraSPARC processors from Sun Microsystems, Inc. in its VIS instruction set breaks with the traditional view of “one-register-one-variable”. Instead, the VIS instruction set views registers as virtually partitioned containing a number of smaller variables.

VIS instruction set utilizes several different register partitioning formats operating on 8-, 16- or 32-bit packed integers. These formats, with the processor’s 64-bit registers, facilitate two-way parallelism and four-way parallelism, improving performance.

The VIS 2.0 instruction set in Sun’s UltraSPARC III processor is an improvement of VIS 1.0 available in all UltraSPARC processors.

VIS 1.0 instruction set is comprised of over 80 instructions which facilitate integer *Single Instruction Multiple Data* (SIMD) operations in the floating-point unit.

One of the categories of instructions more relevant in the context of this paper is the one of arithmetical instructions. The VIS instruction set provides the ability to add, subtract and multiply up to four different variables in one operation. The operands continue to reside in separate registers interacting with each other (identically positioned elements) to produce the result.

VIS 1.0 instruction set also provides conversion instructions that convert data from one type to another which improves flexibility on desired precision and level of parallelism. It provides 32 to 8, 32 to 16 and 16 to 8 bit conversion operations.

Sun Microsystems, Inc. currently offers both language and hardware support for computing with intervals. Hardware is provided in UltraSPARC III processor's VIS instruction set with the "Set Interval Arithmetic Mode" (SIAM) Rounding Mode instructions.

SIAM Rounding Mode enables interval specific hardware instructions for the basic arithmetic operations in single, double and quadruple (128-bit) precision floating-point. Rounding mode bits in the floating-point status register (FPSR) can be annulled without overhead to the pipeline flush. These instructions improve the efficiency of Interval arithmetic: performance improvement from the SIAM instructions has been measured to be approximately 30%.

Software support for Interval arithmetic is available in the Sun ONE Studio 7, Compiler Collection Fortran 95 compiler and in a C++ class library.

6 Conclusions

Clearly, it is important that hardware manufacturers care about improving accuracy in computational systems. Nowadays, computers are so fast that the work they do is changing and must change even more in the proximate future. Increasing "safe" computational results should be a concern.

Interval arithmetic may be very important in the future for science and engineering in solving nonlinear problems, such as commercial, industrial, financial and scientific problems, although nowadays there is still few hardware and software support for intervals.

References

[1] "What every Computer scientist should know about Floating-point arithmetic", Sun Microsystems, Inc (1994)

[2] J. Hass, M. Hutchings and R. Schlafly "The Double bubble conjecture", Electronic research announcements of the American Math. Society, Vol. 1 (1995)

[3] M. Cornea-Hasegan and B.Norin, "IA-64 Floating-point operations and the IEEE Standard for binary Floating-point arithmetic", Intel Technology Journal (1999)

[4] B. Parhami "Computer arithmetic algorithms and hardware designs", Oxford University Press (2001)

[5] "Interval arithmetic in high performance technical computing", Sun Microsystems, Inc (September 2002)

[6] "The VIS Instruction set", Sun Microsystems, Inc (June 2002)