Extended Precision Floating Point Numbers for Ill-Conditioned Problems

Daniel Davis, Advisor: Dr. Scott Sarra

Department of Mathematics, Marshall University

Floating Point Number Systems

Floating point representation is based on scientific notation, where a nonzero real decimal number, x, is expressed as $x = \pm S \times 10^E$, where $1 \leq S < 10$. The values of S and E are known as the *significand* and *exponent*, respectively. When discussing floating points, we are interested in the computer representation of numbers, so we must consider base 2, or binary, rather than base 10. Thus, a non-zero number, x, is written in the form, $x = \pm S \times 2^E$, where $1 \leq S < 2$. It follows, then, that the binary expansion of the significand is given by $S = (b_0.b_1b_2b_3...)_2$, with $b_0 = 1$. Consider $x = 7.0$. Then, in a floating point system, we might have $x = 1.75 \times 2^2$.

IEEE Double Standard

The ANSI/IEEE std 754 defines the *double* floating point format used in virtually all microprocessors.

- Double values use a 64-bit word, with one bit reserved for the sign, 11 for the exponent, and 52 for the significand.
- The exponent field is stored using *biased representation*, meaning that the value of the exponent B is stored as $E + B$, where B is the exponent bias, which is 1023 for double precision.
- The 52-bit binary significand allows for about 16 decimal places of precision.

Sample Conversion

Let us convert $x = 4.5$ into an IEEE double. The sign is positive, so the sign bit is 0. We use division to find that $x/2^2 = 1.125$, giving us an exponent of 2. Using biased representation, this value is stored as 1025, which has a binary string of 10000000001. The decimal portion of .125 is converted to binary using the *Repeated Multiply-by-2* method, appending the value left of the decimal point to the fractional part at each stage, until we reach the exact value or the limit of the length of the significand. Thus, the double precision value is given by:

Sign	Exponent	Mantissa
0	10000000001	0010000000...

Precision

- The *precision*, p, of a floating point number system is the number of bits in the significand.
- This means that any normalized floating point number with precision p can be written as:
$$x = \pm(1.b_1b_2...b_{p-2}b_{p-1})_2 \times 2^E$$
- The smallest x such that $x > 1$ is then:
$$(1.00...01)_2 = 1 + 2^{-(p-1)}$$
- The gap between this number and 1 is called *machine epsilon*, which we can write as:
$$\epsilon_m = (0.00...01)_2 = 2^{-(p-1)}$$
- Let x_c be a floating point representation of a real number, x. Then the absolute error of x_c is given by $|x_c - x|$. Its relative error is given by $\frac{|x_c - x|}{|x|}$.
- The maximum value of the rounding error for a binary floating point representation is equal to $\frac{\epsilon_m}{2}$.

Format	Precision	Machine Epsilon
Single	$p = 24$	$\epsilon_m = 2^{-23} \approx 1.2 * 10^{-7}$
Double	$p = 53$	$\epsilon_m = 2^{-52} \approx 2.2 * 10^{-16}$

Table 1: Precision of Floating Point Formats

Extended Precision

An increasing number of problems exist for which IEEE double is insufficient. These include modeling of dynamical systems such as our solar system or the climate, and numerical cryptography. Several arbitrary precision libraries have been developed, but are too slow to be practical for many complex applications. David Bailey's QD library may be used in applications where two or four times double precision is sufficient. Though much faster than arbitrary precision, the fact that these algorithms are implemented in software still forces them to be much slower than double calculations. The table below displays a time comparison among double, double-double, and quad-double calculations for the basic arithmetic operators, with the data normalized so that the times for doubles represent one time unit.

Type	Add.	Sub.	Mult.	Div.
Double	1	1	1	1
Double-Double	21.45	11.87	13.83	15.53
Quad-Double	70.22	74.10	120.21	129.11

Table 2: Times Required for Arithmetic Operations with Data Normalized so that Double Precision is 1 Unit For Each Operator

Solutions of Chaotic Lorenz Equations

Chaotic systems are those which are highly sensitive to changes in initial conditions.. We seek time-step independent solutions to the chaotic Lorenz equations:

$$\begin{aligned} x' &= \sigma(y - x), & x(0) &= x_0, \\ y' &= rx - y - xz, & y(0) &= y_0, \\ z' &= -bz + xy, & z(0) &= z_0. \end{aligned} \tag{1}$$

We performed the experiment with step sizes $\Delta t_1 = 1 \times 10^{-6}$ and $\Delta t_2 = 1 \times 10^{-7}$. Figure 1 shows the resulting x-values of the solutions calculated in double precision, with the blue line representing the solutions found using Δt_1, and the green line using Δt_2. The solutions visually diverge around $t = 42$. Figure 2 represents the same calculation performed in double-double precision. Here, the solutions do not visually diverge for $t \leq 50$.

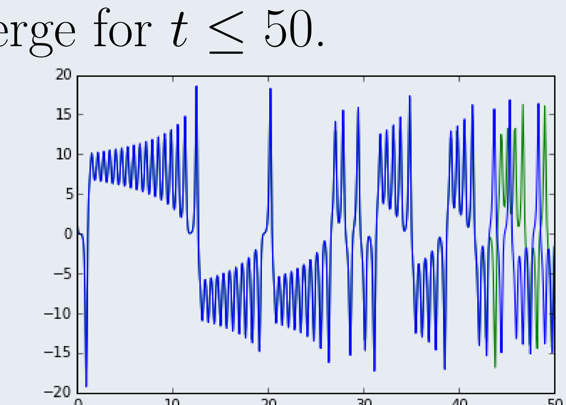

Figure 1: Solutions in double precision.

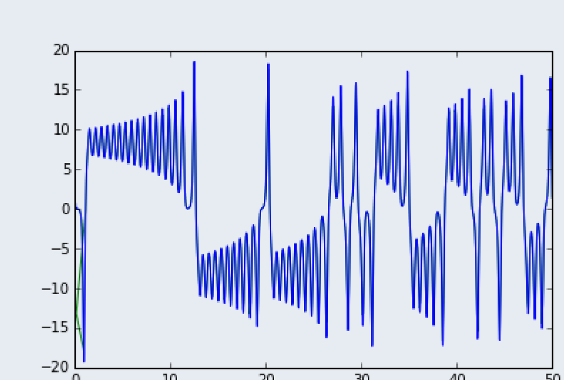

Figure 2: Solutions in double-double precision.

Patriot Missile Failure

Patriot missile defense modules have been used by the U.S. Army since the mid-1960s. On February 21, 1991, a Patriot protecting an Army barracks in Dharan, Afghanistan failed to intercept a SCUD missile, leading to the death of 28 Americans. This failure was caused by floating point rounding error. The system measured time in tenths of seconds, using only 24-bit registers. The algorithms employed by the Patriot were implemented in a way that allowed the rounding error for the time to compound over time. When the Dharan system failed, it had been left on for about 100 hours.

Conclusion/Future Work

Extended precision offers a solution for many problems in numerical mathematics that suffer from the rounding error in IEEE double. However, that solution comes at the cost of efficiency. Modern Graphics Processing Units (GPUs) offer thousands of cores which can be highly advantageous for parallelizable algorithms. A parallel implementation of the QD library known as GQD was published in 2010, but abandoned shortly after. In the future, we hope to implement the QD algorithms in OpenCL for execution across GPUs and other highly parallel platforms.

References

[1] Yozo Hida, Xiaoye S Li, and David H Bailey.
Library for double-double and quad-double arithmetic.
NERSC Division, Lawrence Berkeley National Laboratory, 2007.

[2] Scott A Sarra and Clyde Meador.
On the numerical solution of chaotic dynamical systems using extend precision floating point arithmetic and very high order numerical methods.
Nonlinear Analysis, 16(3):340–352, 2011.

Acknowledgements

Thank you to Dr. Sarra and Dr. Niese for your support and for answering so many questions during this project.