ACPP IRG note: ACPP-N-1
KEK-CP-164
May 5th 2005

High precision numerical computations

A case for an HAPPY design

J. Fujimoto[*], *T. Ishikawa*[*], *D. Perret-Gallix*[+]

[*]*KEK, Tsukuba, Japan,* [+]*Lapp IN2P3-CNRS, France*

Summary

- Although routinely available, double precision floating point accuracy is good enough for most applications, quadruple precision capabilities are becoming more and more important for many research fields including high energy physics, non-linear process simulation or number theory and also for commercial applications like finite element modeling CAD, 3-D real-time graphic, statistics or security cryptography.
- **High precision computation not only gives more precise results, but when large number cancellation or rounding errors play an important role it leads more surely to the correct results. In addition, it improves the convergence of many iterative algorithms leading to an overall computational speed increase (as long as higher precision calculations can be made (by hardware) as fast as the lower precision equivalent).**
- Conventional Libraries implementing quadruple precision floating-point in C++/F90 shows performance drop by a factor 10-20 compared to double precision for basic operations. The ratio of Linpack execution time in quadruple precision over double is 36.
- Dedicated libraries extending the accuracy (not the range) of double precision to quadruple (Bailey's QD [11]) show a 2-3 acceleration factor (compared to plain quadruple precision) for basic operations, but a drop by a factor 10 for log and sin functions. A direct interface (not using operator overloading) may improve the figures by an additional 30-40% to the cost of more involved programming.

- Octuple accuracy also available in the QD library is “only” 3 times slower than the F90 quadruple precisions basic operations, but x50-200 slower for sin and log.
- On a specific test sensitive to large numbers cancellation, quadruple precision gives incorrect results for a x36 slower execution time. Only octuple or quad-double precision provides the correct result but for an x100 increase in CPU time as compared to double precision.
- Interval arithmetic that computes the range in which lies the result is a worth noting intermediate solution. Performed in double precision it is 50% faster than quadruple precision. However for calculation sensitive to large number cancellations the interval given is too large to be meaningful.
- Arbitrary precision libraries deliver results at a pre-selected precision at the cost of a large reduction in computing power (x10 slower than quad-double for the same precision)
- Exact Arithmetics lies in between numerical and symbolic calculations using only rationals or hypergeometric series. Although no a priori precision needs to be given, the computation is faster than Arbitrary precision by almost a factor 2, but slower than the Quad-double for octuple precision.
- Quad-double precision (256 bits) library running on array of integer ALU’s like the “Velocity engine” of the PowerPC brings an additional improvement-factor 3 for the QD library on a Pentium.
- The new line of 64-bit processor only marginally improves the performances for quadruple precision intensive calculation as the FPU is similar to the one available in the 32-bit processors line.
- **In summary, software approaches to quadruple precision (QD library) using existing most advanced hardware (PowerPc) is at least one order of magnitude slower than double precision for basic operations (up to 20 times slower for log or sine). Under the same conditions octuple precision (QD library) is x40 slower than double precision. This makes a case for the design of a dedicated co-processor named HAPPY for “High Arithmetic Precision Processor Yoke”.**
- At least two approaches can be investigated:
    - The design of a 128-bit FPU array based co-processor that would offer quadruple precision for the speed of double precision, namely an overall x30-40 improvement in performance (for Linpack in quadruple precision). Provision for additional 128/256 integer ALU should be made.
    - An large (16-32) integer (128 or 256 bits) multiplier-accumulator and floating point array that can be operated in parallel.
    - In addition to the 4 basic operations, trigonometric and log functions would be implemented as a downloaded micro-code. Additional memory would be spared for user specific functions (dilogs,…). and possibly to implement Arbitrary precision, exact arithmetic or Quad-double libraries.
- The improvement for many numerical calculations would be dramatic and the co-processor would become an important asset in the Teraflops race and for security related internet applications.

Introduction

The computation of particle interactions in high energy physics [1] requires nowadays the use of high precision floating point arithmetic. Multi-dimensional integration of singular functions often leads to a fine cancellation of large positive and negative numbers. The mandatory check of gauge invariance implies a very high accuracy comparison, usually more than what double precision allows. The generation of processes with strong forward divergences like ee->eeee also requires a very high numerical precision.
Quadruple precision are also important for some experimental algorithms like detector precise alignment as in LHC experiments (ATLAS[2]).
Therefore quadruple precision is often required and sometimes octuple precision would be quite desirable.
Higher precision improves also the convergence of some class of algorithms and leads to a reduction in the number of iterations. This can be seen as trading precision for MFlops.
High-energy physics is not the only research activity concerned with high precision calculations. Non-linear process modeling like in fluid dynamics, finite element modeling or computational number theory needs higher and higher precision. In the last case, studies are driven both by the symbolic and numerical approach where high precision numerical computation help finding out or checking analytical solutions.

But the most important markets are probably signal processing (compression/decompression), high quality 3-D graphical applications, streaming video and cryptographic security systems. They all require large computing power with high precision or/and a large number of digits (up to 1024 bits for cryptography). Some companies (IBM cryptographic coprocessor), have developed specific hardware for this purpose. STATISTICA[3] a commercial product providing data analysis, data management, data visualization and data mining procedures tools is now available in quadruple precision.
In this short note, we will compare four approaches to high precision. They are based on: interval arithmetic [6][7][8], arbitrary precision [13][14], exact arithmetic and double-double and quad-double precision library [11] in F90. Hardware developments will also be addressed.

Higher precision and algorithm convergence

Some integration or minimization procedures do not converge, not because the data, or the model are not precise enough, but because the convergence algorithm based on the difference of close numbers is hampered by discontinuities and rounding errors. At higher precision these difficulties can be alleviated.
Integration algorithm based on the quadrature techniques are limited by rounding error. If h is the discretization length, the smaller h the better the sampling of the function. However, the smaller h the larger the number of additions and one see that bellow $h= 10^{-6}$ the rounding error take over.

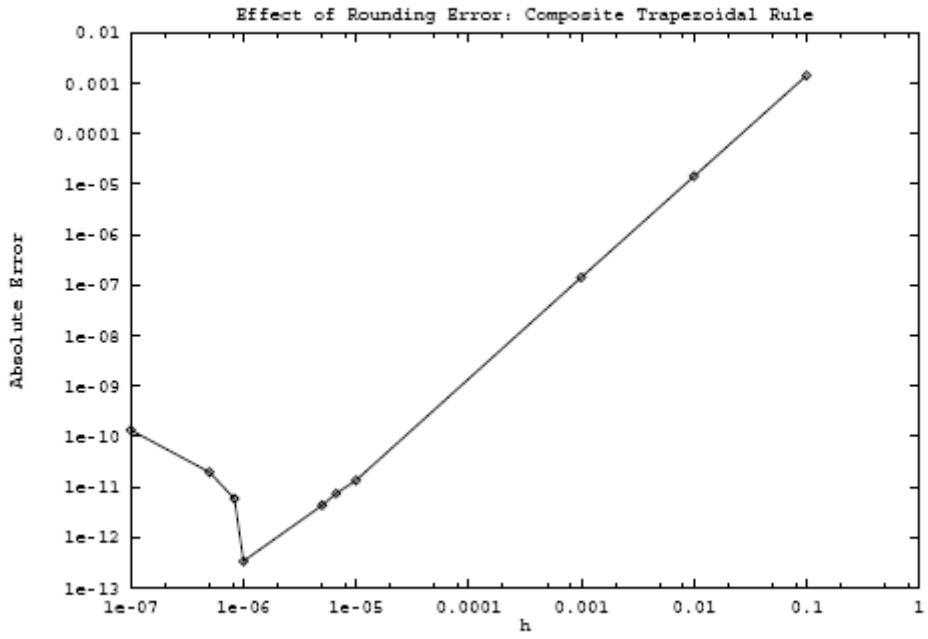


Figure 1: Precision vs discretization size

Even, for acceptable convergence rate, higher precision calculations lead the a reduction of the number of iterations needed to reach a given precision. This has been observed in several classes of problems like minimization, large linear equations system solving, ODE or PDE systems… [4][5]

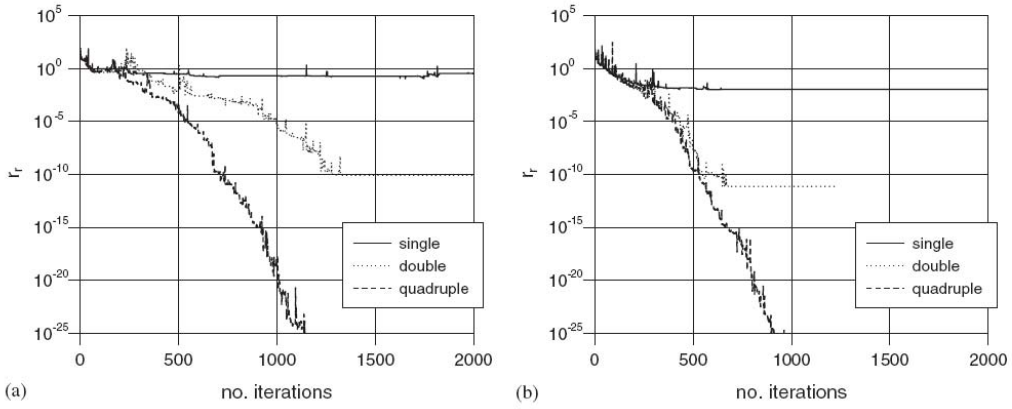


Figure 2. Quadruple precision leads to faster convergence [5].

This is a quite important aspect of high precision computing as it shows that high precision calculation buy MFlops. High precision introduces a kind of parallelization in floating point calculations as more precise knowledge of the problem is provided by each floating-point number, the convergence is accelerated. Faster convergence means shorter execution time (if high precision operation takes the same time as a low precision calculation (by hardware)) and therefore is equivalent to a higher overall rate in MFlops.

Arithmetic operations and basic functions

Addition, multiplication, division, log, sqrt and sin operations have been timed for various precisions using the Intel F90 compiler on a Pentium M 2.8 GHz running Fedora core 3. The Linpack Benchmark (1000) gives for a double precision calculation 28.29 Mflops and for quadruple 0.784 Mflops.

Single, double and quadruple precision computations are intrinsically available through the F90 library. In addition, one has measured the performance of the "QD" library [11] providing double-double (quadruple accuracy (not range)) and quad-double (octuple accuracy (not range)) precision. This library is written in C++ and a F90 wrapping is also provided. A direct C style interface would have shown an improvement of 30-40% at the cost of a less easy and intuitive expression coding (sim-

Format Name	Min Normal	Max Normal	Min Subnormal	Machine Epsilon	Sig. Decimal Digit Range
Single	1.175e-38	3.403e+38	1.401e-45	1.192e-07	6-9
Double	2.2e-308	1.8e+308	4.9e-3246	2.220e-16	15-17
Extended	<3.4e-4932	>3.4e+4932	<3.6e-4951	<1.084e-19	≥ 18-21
Quadruple	3.4e-4932	3.4e+4932	6.5e-4966	1.926e-34	33-36
Octuple	~e-2147483648	~e+2147483648		~1e-67	~67

ple octuple precision addition coded: `c=a+b` must be replaced by c_qd_add(a,b,c)). Figure.3 shows the absolute performance in MFlops of several operations in single precision (r4), double precision (r8), quadruple precision (r16), quadruple accuracy from QD library [11] (dd) and octuple accuracy from the same library (qd).
The timing measurements performed for this note are very simple and naïve. They are only given as indication. The code is similar to what follows:

```
! do-loop processing time
t1=secnds(0.0)
Do i=1,Imax
enddo
t_do=secnds(t1)

! the addition processing time
t1=secnds(0.0)
do i=1, Imax
   s=a+b
end do
t2=secnds(t1)

! megaflops
R=float(imax)/(t2-t_do)/1000000.
```

The function "secnds(t)" return the cpu-time in second minus t. The second call gives therefore the elapsed time. The optimization compiler directive –O0 is used to insure non-null empty loop timing. This approach is only approximate due to the parallelization of floating point operation tested and the integer computation of the loop mechanism.

Figure.4 shows the relative drop in performance compared to single precision.

Conclusions:

- Simple and double point calculation have the same performance due to the 64-bit floating point unit (80-bit internally)
- Basic operations (addition, multiplication) in quadruple precision are 40 times slower than in double or single precision, although, division, log, sqrt and sin are between 7-18 time slower
- QD library in quadruple accuracy gives up to 3 times better performances than the quadruple precision from the F90 library, but for log and sin the figures are much worst
- Octuple accuracy is 3 times to 200 times slower than F90 quadruple library.

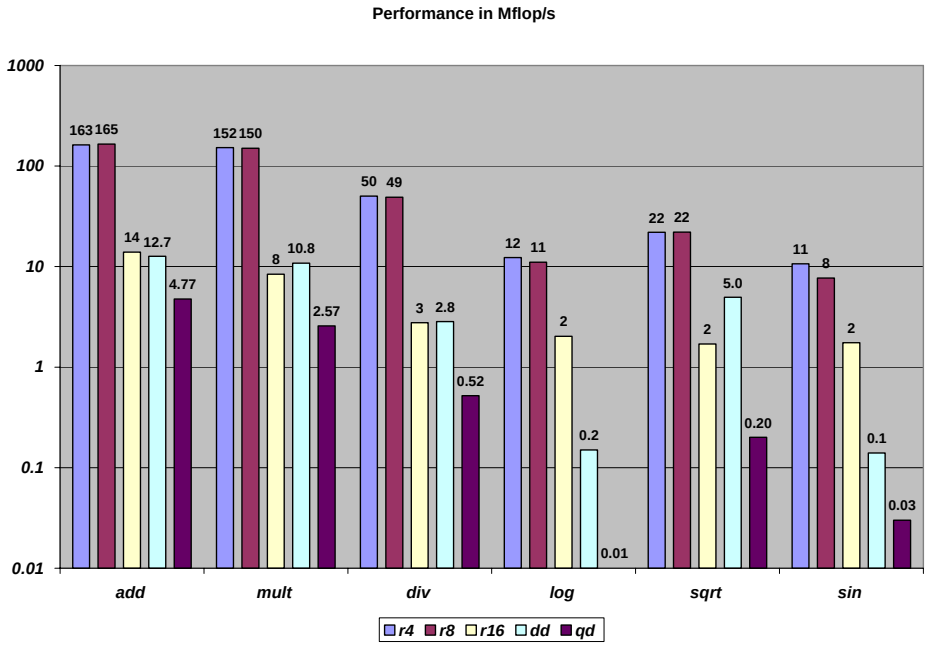


Figure 3. Performance of various operations and functions

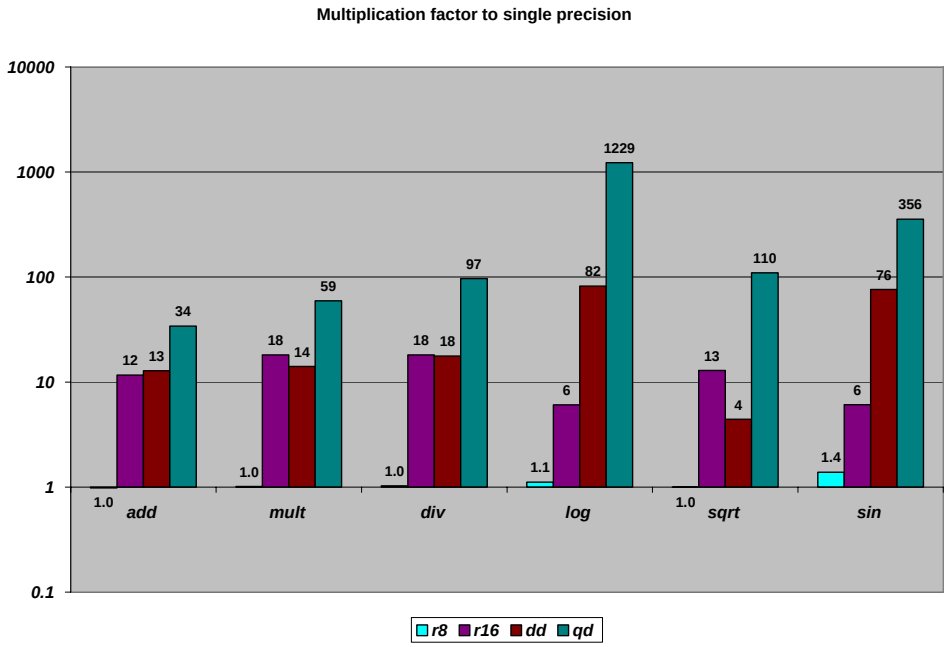


Figure 4. Drop in performance relative to single precision

Similar measurements have been performed on an Opteron 64-bit processor. In that case the Portland Compiler has been used. Unfortunately there no quadruple precision library available (Figure.5).

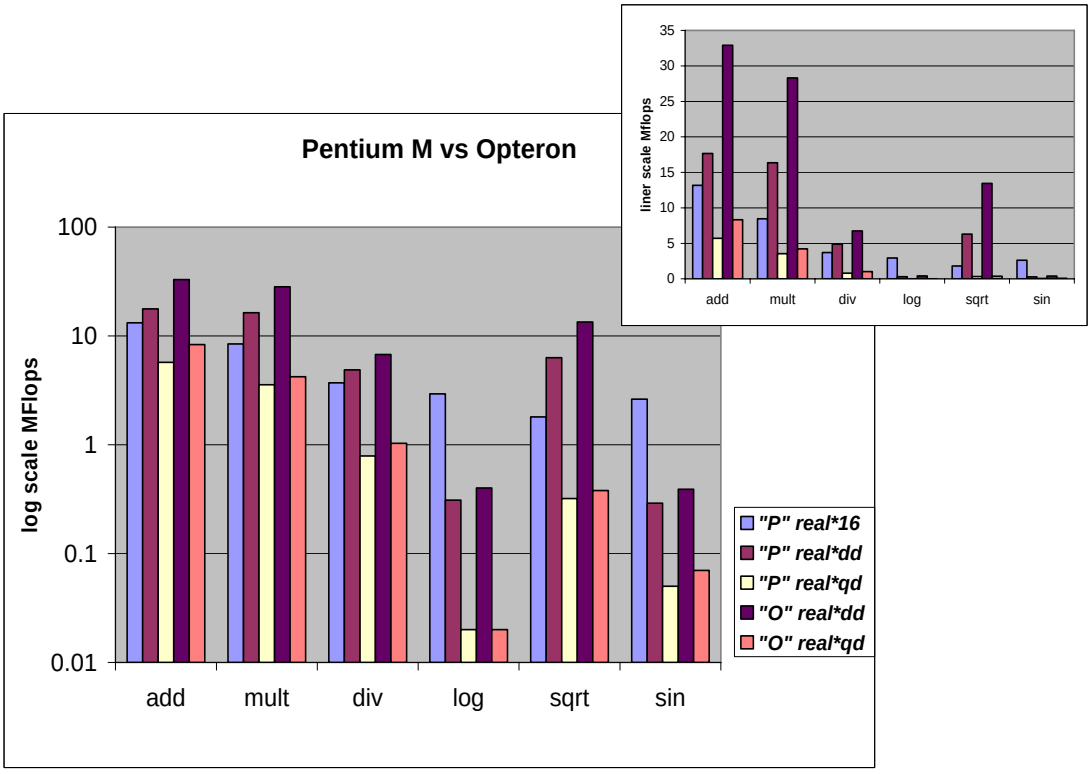


Figure 5: Pentium (P) and Opteron (O) performances for quadruple, double-double and quad-double in log scale. (small picture is in linear scale)

The Opteron is essentially twice as fast as the Pentium for the simple operation (addition, multiplication) for the double-double and quad-double QD library, but only 10-20% faster for (div, log, sqrt and sine)
The Linpack benchmark on the Opteron gives 35.2 MFlops as compared to 28.29 MFlops for the Pentium (25% faster)

A precision sensitive test

As a matter of test, we have selected the example 1 of [10].

$\{1\}\ y=333.75b^6 + a^6(11a^2b^2-b^6-121b^4-2) + 5.5b^8 + a/2b$

For a=77617, b=33096

It is a simple formula where cancellation of large numbers occurs, but each number remains within the range of accuracy of the compiler/run-time system, in such a way that computations do not trigger overflow or other exceptions.
As presented in [10], the computation gives a consistent result of 1.1760 when going from single to double and then to quadruple precision. However this result is not correct. The exact result given by Form or Mathematica is -54767/66192, namely -0.827… So in this specific case quadruple precision gives a blatantly wrong result with no exception warning.
Using Intel F90 v.8.1 one has obtained a result slightly different from [10]. For single and double precision the results are very large numbers (may be triggering some doubt in the validity of the result) but for quadruple precision one gets 1.1760… like in [10], which is the incorrect result. Here, too, one could have concluded that finally quadruple precision was needed but sufficient to obtain a sensible result… but one would have been wrong as we know that the correct result is -0.827…

Interval arithmetic

Interval arithmetic has been introduced by R.E. Moore in the late 50's (for a review see [10]). The principle is simple. For each numerical value instead of declaring a single number, an uncertainty range is defined by a maximum and a minimum value. The true value is supposed to be within this range. The computations are then carried out based on this range. For example:

Let's assume: a=[amin, amax], b=[bmin, bmax],
a+b= [amin+bmin,amax+bmax], a*b=[amin*bmin,amax*bmax], …

This arithmetic has been implemented in a F90 library INTLIB [8].
The formula {1} has been coded using this library. The value a and b are given an interval null such that amin=amax=a. The result shows a huge range clearly indicating a huge uncertainty. This result is much better than a consistent but wrong number. The interval arithmetic therefore can be used to pinpoint sensitive calculation. The computing time is 40-50% faster than quadruple precision. However, in this case the interval (-*O*(1.e22, +*O*(1.e22) is too large to be meaningful. Results can be seen in figure.6

High precision libraries

In multi-precision or arbitrary precision computations, the required number of decimal digit of accuracy is selected at the beginning of a computation. This number can usually go up to 1000 digits. Essentially 2 types of algorithms are being used, based

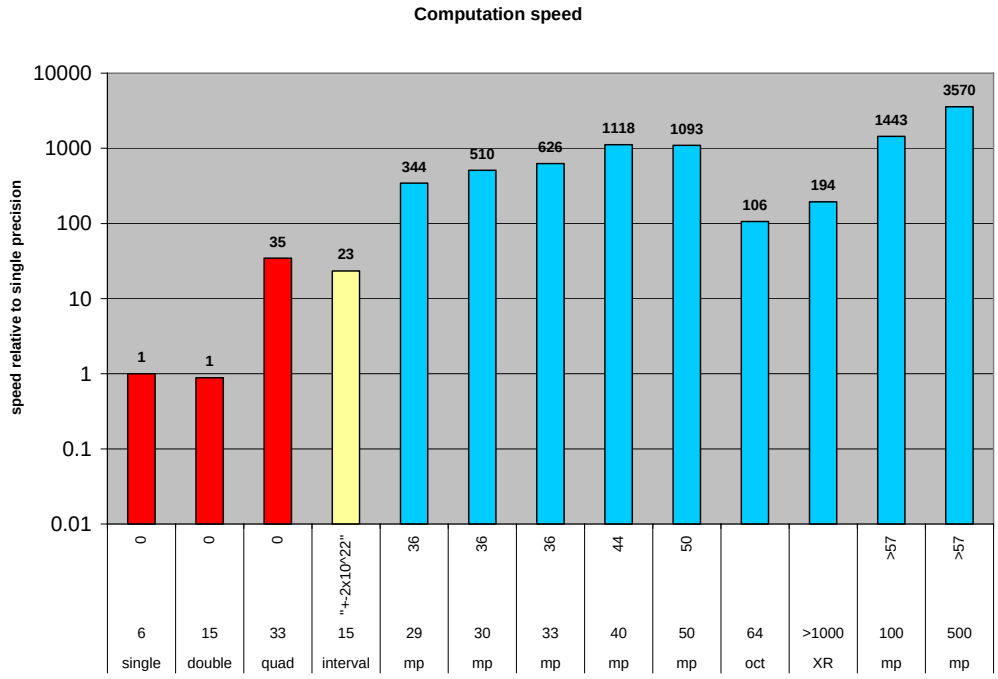
on the way the floating point numbers are represented: integer [12] and multi-digit format like symbolic language (Mathematica, Maple, Form) and MPFUN [14].
Another type of dedicated library targeting quadruple and octuple precision uses the multiple-component format, where a number is expressed as an unevaluated sum of ordinary double precision floating point numbers [11]. This last approach does not provide the full range a true quadruple or octuple precision number would. Only the mantissa complies with these precision. But computations are faster than using the multi-digit multi-precision format.

Multi-precision libraries

Several multi-precisions libraries exist. GMP [15] is the most popular but written in C, it has not been tested here. We have selected the MPFUN (ARPREC) packages from D.H. Bailey [14] for this test. It allows the selection of up to 1000 digits precision. Extended precision can also be obtained by special request to the author. Most of the basic functions are available: arithmetic, trigonometric, power, log. It treats integer, real and complex numbers.
Figure.6 shows that from 29 digits of precision the correct result is obtained. Going higher in requested number of digits just improves the accuracy of the result.
Obviously the penalty is the CPU time needed for such operations. The comparison with single-double/quadruple precision from the F90 library which gives the incorrect results shows a 340/10 time drop in performance. A price one may want to pay to get the correct result.
The implementation is relatively easy thanks to the possibility of operators and functions overload allowed by C++ and F90.
Figure.6 shows that for this example the best choice is the octuple precision calculation based on the QD library. It is 100 times slower than double precision and gives the correct results. Run on a PowerPC using the “Altivec” FPU and the Octuple library [12] one could expect only a 25-33 slower performance than double precision.

Double-double and Quad-double precision library

D.H. Bailey et al. have developed a double-double (range like in double with mantissa with quadruple accuracy) and a quad-double (double with octuple accuracy) precision library in C/C++ with a F90 wrapping. The formula {1} in quadruple accuracy gives also the incorrect results but it is twice as fast as the quadruple precision library associated to the Intel compiler. The octuple precision gives the correct value with a CPU-time penalty of only a factor 3 slower as compared to the quadruple precision timing.
The integer approach described in [12] assumes that octuple precision numbers are of the form $m.2^e$ where e is a signed 32 bit exponent (unbiased) and m is a 224-bit twos-complement fraction with 222 bits to the right of the binary point. The implementation heavily relies on the hardware of the “Altivec engine” provided in the PowerPc and uses extensively the multi-register and operation parallelism available. The benchmark shows an improvement by a factor 3-4 compared to the QD library.



(In red wrong results, in yellow correct, but meaningless results and in blue correct results are shown)

Figure 6 Relative performance drop as compared to single precision

Exact Real Arithmetic

"Exact real arithmetic is a numerical approach to real number computation based on potentially infinite data structures such as streams. Its main feature is that it solves the problems of inaccuracy and uncertainty in floating point and interval arithmetic, and is applicable in some cases in which a symbolic approach would not be appropriate.

Although exact real arithmetic can be used to calculate results which can be guaranteed to be completely accurate, it does so at the expense of the efficiency afforded by more conventional methods. One reason for this is that in practice, infinite data structures such as streams are inherently expensive to manage when compared with the type of fixed sized data structure used to represent floating point numbers. In addition, during exact computations it is often found that a small portion of the result of a computation may depend upon a large amount of the input. These inefficiencies associated with exact real arithmetic may be an acceptable trade-off against the benefits. For example which do not consider a sufficient number of digits to determine the correct result will be inherently inaccurate. "[17]

Hardware approaches to higher numerical precision

The new generation of 64-bit processors by Intel, AMD and others will not bring dramatic improvement for high precision computation as the floating point register bit size was already of 64 bit in general even 80-bit for the Intel Pentium processor. The possibility to load a floating point register in one cycle instead of 2 as well as the larger number of registers will probably be the most important source of improvement. Amazingly enough, 128-bit architectures developed for the entertainment market may well become a powerful support for research activities. The Playstation 2 from Sony with the velocity engine, a 128-bit processor, provides floating-point computation

reaching 6.2 GFlops (single precision) thanks to a parallel (x4) architecture based on 32 128-bit floating point registers. Watch out the PS3 (32 Gflops, 25 Gflops double precision) based on the "Cell" ship from IBM, Toshiba and Sony whose architecture has a lot in common with the PowerPc G4/5 velocity engine. But today there are no straight 128-bit floating point unit providing quadruple precision available.

The fastest hardware/software combination today is the multi-precision register array and multi-operational units running in parallel as available in PowerPc Altivec running the QD Bailey's library.
However, a real progress in high precision numerical computation implies the development of a specific numerical accelerator that could handle high precision arithmetic at high speed.
This project has taken the name of *HAPPY* which stands for "High Arithmetic Precision Processor Yoke".
At least two approaches are considered:

1. The simple design of a 128-bit FPU. That would allow quadruple precision computations at the speed of double precision today. In terms of silicon real estate this is a 2-3 increase as compared to the current Pentium 80-bits FPU.
2. The design of a supped up version of the PowerPc Altivec with larger rgister and more ALU's providing for example:
    a. Array of 256-bit registers.
    b. Array of 256-bit Integer multiplier-accumulator running in parallel
    c. floating point units running in parallel
    d. A reduced instruction set processor for high speed micro-code
    e. A fast access memory

    It would provide the possibility to micro code sin, log and other power functions. Octuple precision computations could also be efficiently micro-coded. Additional micro-code memory could also be made available to implement problem specific functions like dilog, gamma, random numbers…. Finally if enough memory is made available multi-precision algorithm could also be implemented in micro-code.
    It should be flexible enough to carry encryption algorithm

Ganged with 64-bit processors, it would lead a 1-2 order of magnitude jump in the teraflops race for high precision computations.

Acknowledgements

This work has been made possible thanks to KEK computer center support in the framework of the "International Research Group" on "Automatic Computational Particle Physics" (IRG ACPP) funded by CNRS/Universities in France, KEK/MEXT in Japan and MSU/RAS/MFBR in Russia.

References

Most of the non-published references can be found through a web search on the title and/or the name of the authors

[1] "*Automatic Calculation in Particle Physics*" ACPP IRG https://uimon.cern.ch/twiki/bin/view/ACPP/WebHome

[2] "*Conclusion on the Marseille alignement activities* K. bernardet, A. Bonnissent, D. Fouchez, A. Tilquin http://atlas.web.cern.ch/Atlas/GROUPS/INNER_DETECTOR/PERFORMANCE/ALIGN/docs/marseille_mar04.ps

[3] *"Statistica"* from StatSoft http://www.statsoft.com/products/products.htm

[4] *"Utilizing the quadruple-precision floating-point arithmetic operation for the Krylov Subspace Methods*" Hidehiko Hasegawa University of Tsukuba

[5] "*Scaling improves stability of preconditioned CG-like solvers for FE consolidation equations*" Giuseppe Gambolatin, Giorgio Pini and Massimiliano Ferronato Dip. di Metodi e Modelli Matematici per le Scienze Applicate, Universita degli Studi, Via Belzoni 7, 35131 Padova, Italy

[6] "*Interval Analysis I*", R. Moore and C. Yang, technical document, Lockheed Missiles and space division, number LMSD-285875, 1959

[7] "*Methods and Applications of Interval Analysis*", R. Moore, Society of Industrial and Applied Mathematics, 1979

[8] Algorithm 737: "INTLIB: *a portable Fortran-77 Interval Standard Function Library*", ACM, Trans on Math. Software Vol.20, No.4, pp 447-459, 1995.

[9] "*Interval Arithmetic: A Fortran 90 Module for an Interval Data Type*" R. B. Kearfott. Department of Mathematics University of Southwestern Louisiana

[10] "*A review on Interval Computation: Software and Applications*", C. Hu, S. Xu and X. Yang

[11] "*Quad-Double Arithmetic: Algorithms, Implementation, and Application*", Y. Hida, X. S. Li, D. H. Bailey, Oct. 30, 200 Technical Report LBNL-46996

[12] "*Octuple-precision floating point on Apple G4*", R. Crandall and J. Papadopoulos May 8, 2002 Apple computer, Inc

[13] "*Vector Implementation of multi-precision arithmetic*" R. Crandall and J. Klivington, Advanced Computation Group, Apple Computer October 25, 1999

[14] "*A fortran-90 based multi-precision system*" D. H. Bailey, RNR technical Report RNR-94-013, Jan. 6, 1995

[15] "*GMP: GNU Multi-precision library*" http://www.swox.com/gmp/

[16] "*A Quadruple Precision and Dual Double Precision Floating-Point Multiplier*", A. Akkas (University of Istanbul) M. J. Schulte, University of Wisconsin-Madison

[17] *A Calculator for Exact Real Number*, Computation Departments of Computer Science and Artificial Intelligence, University of Edinburgh, Dave Plume http://www.dcs.ed.ac.uk/home/mhe/plume/report.html

[18] *XR – Exact Real Arithmetic* K. Briggs http://members.lycos.co.uk/keithmbriggs/xrc.html

[19] *A Quadruple Precision and Dual Double Precision Floating-Point Multiplier*, Ahmet Akkas, Koç University Michael J. Schulte, University of Wisconsin-Madison