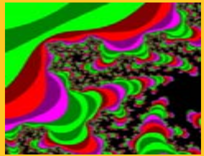


Extended-Precision Floating-Point Numbers for GPU Computation

Andrew Thall, Department of Computer Science, Allegheny College
athall@allegheny.edu

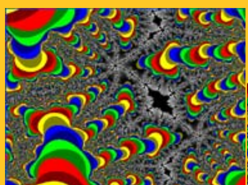
SUMMARY

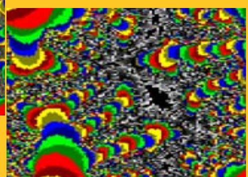
Using unevaluated sums of paired or quadrupled single-precision (f_{32}) values, *double-float* (df_{64}) and *quad-float* (qf_{128}) numeric types can be implemented on current GPUs and used efficiently and effectively for extended-precision computation for real and complex arithmetic. These numeric types provide 48 and 96 bits of precision respectively at f_{32} exponent ranges for computer graphics and general purpose (GPGPU) programming.

Rationale for and Prior Work on Extended Precision

Modern GPUs have wide data-buses allowing extremely high throughput, effecting a stream-computing model and allowing SIMD/MIMD computation at the fragment (pixel) level. Machine precision is limited, however, to 32-bit *nearly* IEEE 754 compliant floating-point. This limited precision of fragment-program computation presents a drawback for many GPGPU applications. GPU hardware will improve its IEEE compliance, but it is unlikely to support double or extended-precision arithmetic in the near future.

Techniques for performing extended-precision arithmetic in software using pairs of machine-precision numbers have a long history: [Dekker 1971], [Wyatt 1976], and [Brent 1978]; the *doubledoubles* of [Briggs 1998] and *quad-doubles* of [Hida et al 2001]. The use and hazards of these numerical types is discussed in [Li 2000].





The images in this example are areas of the Mandelbrot set in a region of the complex plane only **2.6×10^{-6}** across. The right image was computed on the GPU using f_{32} precision floats; the left image used complex cdf_{64} floats. The cdf_{64} code ran approximately 3x slower, averaging 8 frames/second for a 768x512 image; this includes time to compute an extended-precision square-root, unnecessary except to compare accuracy with d_{64} routines on the CPU.

Double-Float and Quad-Float Computation on GPUs

The hardware support for vector operations in fragment programs makes them well-suited for double- and quad-precision computation. Preliminary exploration of the feasibility and error-characteristics of GPU double-floats has been done using the Cg language [Meredith and Bremer 2004] and the Brook language [Da Graça and Defour 2006]. The current research was undertaken independently and uses Cg, as did Meredith's. We chose Cg, despite some drawbacks, in order to support other projects in interactive graphics, image-analysis and visualization.

This discussion will limit itself to df_{64} methods; techniques for qf_{128} generalize similarly from the methods described by Hida [5]. A df_{64} number A is an unevaluated sum of two f_{32} numbers represented in Cg as a 2-vector of single-precision floats:

$$A = a_{hi} + a_{lo}$$

`float2(a_hi, a_lo);`

Arithmetic operations on these numbers are done exactly using the a_{lo} term to give the error on the a_{hi} term. By allowing the a_{lo} term to be negative, we make use of the full 48-bits of precision available in the pair of values (23 bits + 1 hidden per mantissa).

```
float2 df64_add(float2 a, float2 b) {
   // use parallelism to simultaneously compute
   //   [(s, err), (t, err)] = [(a.x + b.x), (a.y, b.y)]
   float4 st = twoSumHiLo(a, b);
   st.y += st.z;
   st = quickTwoSum(st.x, st.y); // assumes |st.x| >= |st.y|
   st.y += st.w;
   return quickTwoSum(st.x, st.y);
}
```

```
float2 df64_mult(float2 a, float2 b) {
   // Multiply the hi-order terms at df64 precision,
   //   then add the low-order products
   float2 p = twoProd(a.x, b.x);
   p.y += dot(a, b.yx);
   return quickTwoSum(p.x, p.y); // renormalize product
}
```

```
float2 df64_exp(float2 a) {
   float2 outVal, r, rem, df_z;
   if (!specialCase(outVal)) {
       rem = df64_rem(a, df64M.LOG2, df_z);
       int z = (int) df_z;
       r = rem/64;
       r = df64_expTAYLOR(r);
       outVal = df64_mult(r, df64_npow(2.0*ONE, z));
   }
   return outVal;
}
```

Achievements of the Current Project

- Creating df_{64} data structures, basic arithmetic operations, and code for functions such as sqrt(x), ln(x), exp(x), and trigonometric functions taking df_{64} and f_{32} operands.
- Creating qf_{128} data structures and basic arithmetic operations and example functions;
- Creating a C++ **df₆₄** class for CPU-side interface with df_{64} fragment code, with tables of constants at df_{64} precision for π, e, ln(2).
- Creating test-applications to demonstrate the effectiveness and efficiency of extended-precision fragment code for graphics and GPGPU programming;
- Creating fast bitonic merging of addends for qf_{128}.
- Creating complex cdf_{64} numbers and demonstrating their efficiency.
- Creating test-applications to study error-generation and propagation by basic and library methods;

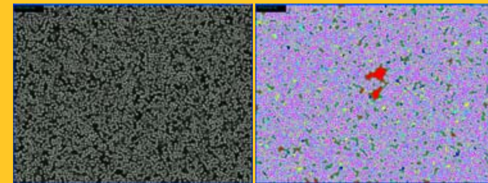
A sample routine showing the use of cdf_{64} complex variables to generate a the Mandelbrot set images used in the examples.

Improvements to current implementation:

- Full implementation of df_{64}, cdf_{64}, qf_{128} and cqf_{128} as first-order primitives in Cg using preprocessor or compiler modifications.
- More in-depth study of error behavior given the non-strict IEEE 754 compliance of current GPU hardware [Hillesland and Lastra 2004].
- Porting of these methods to Brook or CGiS or other non-graphics-oriented stream languages for GPGPU.
- Careful and correct generation of NaN and ±Inf values.
- Improvements to code for range-reduction in numerical routines.

Future work!

- GPU-based fractal- and non-fractal data compression.
- Porting of GPU FFT methods to df_{64} and qf_{128} precision routines for applications in image generation and analysis, and in computational number theory (e.g. Lucas-Lehmer testing for Mersenne primes).
- Application of df_{64} and qf_{128} to solving linear systems, PDEs, and physically-based graphics applications.
- Application to primitives to shading and illumination effects
- Modification of code to create extended precision integer class for numerical algorithms.



Solution of PDEs for simulation, such as the above reaction-diffusion textures, can benefit from extended precision: on the left, a Grey-Scott RD-texture (original code by Mark Harris); on the right, a similar simulation at df_{64} precision, showing relative concentrations of reactants.

DISCUSSION

Some points of interest:

1. Care must be taken to prevent compiler optimizations from altering or eliminated necessary computation. Defining **ONE** as a uniform parameter-multiplier for constants was only one of the necessary kludges to sidestep the aggressive **cgc** compiler.
2. GPUs have a MAD operator, equivalent to the numerical MAC (multiply-and-accumulate), allowing

   $$d = a*b + c$$

   to be evaluated in a single command; preliminary tests seem to indicate that this is not an FMAC (fused multiply-and-accumulate), which has a single roundoff error and would allow more efficient versions of extended-precision routines.
3. Because GPU operations can act on the four elements of a **float4** in a single operation, the above routines can be used for cdf_{64} complex values with no additional runtime overhead. In terms of these basic ops, we get complex numbers for free.

Thanks to Dr. David Bremer (LLNL) and Dr. Jeremy Meredith (ORNL) for discussions of their experiments, and to Dr. Mark Harris for his inestimable help on GPU floating-point performance and other GPGPU issues.

References

Brent, R. (1978) A Fortran multiple-precision arithmetic package. *ACM Trans. Math. Softw.* **4**(1), 57-70.

Briggs, K. (1998) Doubledouble floating point arithmetic. http://keithbriggs.info/software.html.

Dekker, T. (1971) A floating-point technique for extending the available precision. In *Numerische Mathematik* **18***,* 224-242.

G. Da Graca, D. Defour. Implementing of float-float operators on graphics hardware. *7th Conference on Real Numbers and Computers* (*RNC7*), Nancy, France, July 2006.

Hida, Y., Li, X., Bailey, D. (2001) Quad-double arithmetic: algorithms, implementation, and application. Lawrence Berkeley National Laboratory, Technical Report LBNL-46996.

Hillesland, K., Lastra, A. (2004) GPU floating-point paranoia. *Proceedings of GPU²*

Li, X., Demmel, J., Bailey, D., Henry, G., Hida, Y., Iskandar, J., Kahan, W., Kapur, M., Martin, M., Tung, T., Yoo, D. (2000). Design, implementation and testing of extended and mixed precision BLAS. Lawrence Berkeley National Laboratory. Technical Report LBNL-45991

Meredith, J., Bremer D., et. al., "The GAIA Project: Evaluation of GPU-Based Programming Environments for Knowledge Discovery," HPEC 2004, Boston.

Wyatt, W.T., Lozier, D.W., Orser, D.J. (1976) A portable extended precision arithmetic package and library with Fortran precompiler. *ACM Trans. Math. Softw.* **2**(3), 209-231.

ACM SIGGRAPH 06