GPU Floating-Point Paranoia

Karl E. Hillesland
University of North Carolina at Chapel Hill *

Anselmo Lastra
University of North Carolina at Chapel Hill *

1 Introduction

Up until the late eighties, each computer vendor was left to develop their own conventions for floating-point computation as they saw fit. As a result, programmers needed to familiarize themselves with the peculiarities of each system in order to write effective software and evaluate numerical error. In 1987, a standard was established for floating-point computation to alleviate this problem, and CPU vendors now design to this standard [IEEE 1987].

Today there is an interest in the use of graphics processing units, or GPUs, for non-graphics applications such as scientific computing. GPUs have floating-point representations similar to, and sometimes matching, the IEEE standard. However, we have found that GPUs do not adhere to IEEE standards for floating-point operations, nor do they give the information necessary to establish bounds on error for these operations. Another complication is that this behavior seems to be in a constant state of flux due to the dependence on the hardware, drivers, and compilers of a rapidly changing industry.

Our goal is to determine the error bounds on floating-point operation results for quickly evolving graphics systems. We have created a tool to measure the error for four basic floating-point operations: addition, subtraction, multiplication and division.

2 IEEE Standard Floating Point

Ideally, GPUs would follow the IEEE standard for floating-point operations. The IEEE standard gives us a guarantee on error bounds for certain operations, including addition, subtraction, multiplication and division. It does so by requiring that these operations follow the *exact rounding* convention. Under this convention, the result of an operation must be the same as a result computed exactly, and then rounded to the nearest representable number. This means a bound of [-0.5 , 0.5] in units of the last bit of the significand.

3 Paranoia

Paranoia [Karpinski 1985], originally written by William Kahan in the 1980's, explores a number of aspects of floating-point operation. We have adopted the guard bit and rounding mode tests for subtraction, multiplication, and division. All of these operations were found to have guard bits.

Paranoia looks for two kinds of rounding: exact rounding, and chopping. The GPUs we tested did not follow either of these models. In order to obtain a bound on error, we turned to a more empirical approach, which we describe next.

4 Measuring Floating-Point Error

To ensure we have complete bounds requires exhaustive tests of all combinations of all floating-point numbers. Since this is fairly impractical, we chose a subset of floating-point numbers that we believe does a reasonable job of characterizing the entire set. This is an approach used by others for testing correct operation of floating-point hardware. We used a superset of significands suggested by Schryer [Schryer 1981]. By testing all combinations of these numbers, we include all the test cases in Paranoia, as well as cases that push the limits of round-off error and cases where the most work must be performed, such as extensive carry propagation. Table 1 gives results for some example systems.

Operation	R300/arbfp	NV30/fp30
Addition	[-1.000, 0.000]	[-1.000, 0.000]
Subtraction	[-1.000, 1.000]	[-0.750, 0.750]
Multiplication	[-0.989, 0.125]	[-0.782, 0.625]
Division	[-2.869, 0.094]	[-1.199, 1.375]

Table 1: Floating-Point Error in ULPs (Units in Last Place). Note that the R300 has a 16 bit significand, whereas the NV30 has 23 bits. Therefore one ULP on an R300 is equivalent to 2^7 ULPs on an NV30. Division is implemented by a combination of reciprical and multiply on these systems. Cg version 1.2.1. ATI driver 6.14.10.6444. NVIDIA driver 56.72.

5 System Considerations

Results are for specific configurations of graphics card, driver, operating system, CPU, chipset, compiler version, and other factors. The tool we developed is intended to be run each time any of these items change.

Semantics for programming GPUs currently allow for considerable leeway in how a program is implemented. Instructions can be re-ordered. Subexpressions involving constants or "uniform" parameters may be evaluated on the CPU. Associative and distributive properties, which do not hold for floating-point operations, may be applied in optimization. Our tool does not take into consideration the kinds of optimizations possible in larger program contexts.

6 Conclusion

Our goal is to give the developer a tool to characterize GPU floating-point error in order to aid them in developing compute-intensive applications. We use an empirical approach to establish error bounds for addition, subtraction, multiplication and division. Watch *http://www.gpgpu.org* for more information and tool availability.

References

IEEE. 1987. IEEE standard for binary floating-point arithmetic. *ACM SIGPLAN Notices 22*, 2 (Feb.), 9–25.

KARPINSKI, R. 1985. Paranoia: A floating-point benchmark. *Byte Magazine 10*, 2 (Feb.), 223–235.

SCHRYER, N. L. 1981. A test of a computer's floating-point arithmetic unit. Tech. Rep. Computer Science Technical Report 89, AT&T Bell Laboratories, Feb.

*Email: [khillesl,lastra]@cs.unc.edu