Evaluation of IEEE 754 Floating-Point Arithmetic Compliance Across a Wide Range of Heterogeneous Computers

Guillermo A. Lopez
The University of Texas at El Paso
Department of Computer Science
galopez1@miners.utep.edu

Michela Taufer
University of Delaware
Computer & Inf. Sciences Dept.
taufer@acm.org

Patricia J. Teller
The University of Texas at El Paso
Department of Computer Science
pteller@utep.edu

ABSTRACT
Scientific applications rely heavily on floating-point arithmetic and, therefore, are affected by the precision and implementation of floating-point operations. Although the computers we use are IEEE compliant, this only assures the same representation of floating-point numbers; it does not guarantee that floating-point operations will be performed in the same way on all computers. As a result the same program run on different computers may yield different results. This paper is a first step in understanding the reason for this, in particular, different results for the execution of the application Charmm on different computers. We report on our use of a well-known test suite, IeeeCC754, to evaluate IEEE 754 compliance across a wide range of heterogeneous computers with different architectures, operating systems, precisions, and compilers.

Categories and Subject Descriptors
G.1.0 [**Numerical Analysis**]: General – *Computer arithmetic.*

General Terms
Measurement, Performance, Experimentation.

Keywords
Heterogeneous computing, scientific computation.

1. INTRODUCTION

Scientific applications rely heavily on floating-point arithmetic [1] and, therefore, are affected by the precision and implementation of floating-point operations. Although the computers we use are IEEE compliant, this only assures the same representation of floating-point numbers; it does not guarantee that floating-point operations will be performed in the same way on all computers [4]. This may result in divergences in intermediate results that can be significantly amplified by the chaotic nature of some applications [2], where loops are iterated a large number of times, leading to significantly different final results when a computation, e.g., a simulation, is performed on different computers [6]. Since such behavior can constrain the employment of multiple computers in scientific research, this problem is an important one. Accordingly, this paper reports on our findings of running a well-known test suite, IeeeCC754, that is used to evaluate IEEE 754 compliance across a wide range of heterogeneous computers with different architectures, operating systems, precisions, and compilers. The rest of the paper is organized as follows. In Section 2 we give a short overview of the IEEE 754 floating-point arithmetic standard. Sections 3 and 4 describe the IeeeCC754 test suite and the way we use it to evaluate IEEE 754 compliance across a set of heterogeneous computer systems. Our test results are presented in Section 5. Sections 6 and 7

discuss future work and summarize the contribution of this paper.

2. IEEE 754 FLOATING-POINT ARITHMETIC STANDARD: SHORT OVERVIEW

The IEEE Standard for Binary Floating-Point Arithmetic (IEEE 754) Standard is the most widely used standard for floating-point representation and arithmetic [5]; it is implemented to some degree by most CPUs and FPUs. This standard defines basic and extended floating-point number formats, different operations such as add, multiply, and square root, among others. It specifies different conversions such as those between integer and floating-point formats, between different floating-point formats, and between basic format floating-point numbers and decimal strings. Floating-point exceptions and their handling, including non-numbers (NaNs), are also specified by IEEE 754. Despite the fact that IEEE 754 is a standard, still several aspects of floating-point operations are left to the discretion of the system designer and this may ultimately result in different results for the same computation on two different computers that follow the IEEE 754 standard. It is also important to note that the standard does not specify formats of decimal strings and integers, interpretation of NaNs, and binary-to-decimal conversions to and from extended formats.

3. BENCHMARK FOR TESTING IEEE 754 COMPLIANCE: THE IEEECC754 SUITE

To test IEEE 754 compliance we use the IeeeCC754 Test Suite [7,3], which is a precision- and range- independent set of programs that is used to test whether an implementation of floating-point arithmetic (in hardware or software) is compliant with the principles of the IEEE 754 floating-point standard [5]. IeeeCC754 is the work of the Computer Arithmetic & Numerical Techniques (CANT) research group at the University of Antwerp. We have chosen this test suite for several reasons: (1) It is a complete set of programs in terms of types of tests, i.e., it includes tests for simple operations and for conversions. (2) It extends well-known benchmarks such as Paranoia, UCBTEST, and the NAG Floating-Point Validation package. (3) The driver program included with the suite facilitates the repetition of compilation with different compilers and testing on multiple machines. (4) Its independent test vectors permit us to run the tests in different precisions that may lead to different divergences. The test suite comes with two main groups of test sets:

1. *Basic Operations*: add, divide, multiply, remainder, and sqrt (square root).
2. *Conversions:* b2dconvd, b2dconvm, b2dconvs, cint64, cuint32, d2bconvd, d2bconvm, d2bconvs, rint32, rintegral, ruint32, sqrt, b2dconvl, b2dconvq, cint32, copyto, cuint64, d2bconvl, d2bconvq, remainder, rint64, roundto, and ruint64; where b2d means binary-to-decimal, d2b means decimal-to-binary; r means rounding; c means copy; and a terminating s, d, and l stand for the precision used for the conversion, i.e., single, double, or long.

4. METHODOLOGY FOR TESTING IEEE 754 COMPLIANCE

We use the IeeeCC754 benchmark in a rigorous and systematic way in order to find errors in a computer's IEEE754 implementation. The following are considered to be errors: (1) if the result of a test does not match the expected result and (2) if the flags that an operation raises are different than those expected. Our analysis is based on four computer characteristics that we call factors: architecture, operating system, precision, and compiler. These factors have the following values:

- Precision: 32-bit and 64-bit
- Architecture: Intel and AMD
- Compiler: Intel and gcc
- Operating system: SuSE 9.3, SuSE 10.0, SuSE 10.2, and Fedora Core 4

In order to identify what factor is responsible for an error, we run the test suite changing only one

factor at a time. All the compilation and execution of the test suite is performed automatically via a script. The script prompts for factor values and tests to perform (basic operations or conversion tests). Since the test suite generates several files, it is important to establish a file organization that facilitates the comparison of results from different tests and the archiving of test results for further analysis. The file structure we use to store test output first groups the data by the addressing precision of the computers, then by machine name, test type, architecture, and the compiler. In order to compare these results we use bit-to-bit comparison on the IeeeCC754 output files. Because no divergences were observed with respect to different Linux distributions , the OS is not considered in the structure.

5. TEST RESULTS

No divergences were observed between the same tests run on different vendors' processors, i.e., AMD and Intel processors, and among tests run under different distributions and versions of Linux. In contrast, differences in results were observed for tests run with different precisions and compilers. Tables 1 and 2 present results of tests that produce divergences. Highlighted in lighter gray are divergences due to different precisions; highlighted in darker gray are divergences due to different compilers. The number of tests that exhibited differences is compared with the total number of tests (last column of the tables).

6. CONCLUSIONS AND FUTURE WORK

In this paper we used the IeeeCC754 test suite to evaluate the results of floating-point operations performed on different computers, i.e., with different architectures, operating systems, precisions, and compilers. We observed that the way floating-point operations are performed at run time can be affected by the compilation, in particular, the compiler flags and different compiler versions. Also, different precisions can result in different results. Therefore the user, when working with floating-point operations, must pay special attention to the defined compilation and the precision used to run the code. These two aspects of the computation particularly affect the results of scientific applications that make heavy use of floating-point arithmetic. The next step in our research is to analyze the discrepancies that were identified by the tests discussed in this paper and to continue this testing on other computers. Given different results for a test run on two different computers, we must understand the reason for the divergence.

7. ACKNOWLEDGMENTS

This work was supported by the National Science Foundation, grants SCI-0506429, DAPLDS – a Dynamically Adaptive Protein-Ligand Docking System based on multi-scale modeling and DUE-0631168, SHiPPER: Spreading High-Performance computing Participation in undergraduate Education and Research.

8. REFERENCES

[1] Bailey, D. H. High-precision Floating-point Arithmetic in Scientific Computation. *Computing in Science and Engineering,* May-June 2005, 7(3):54-61.

[2] Braxenthaler, M., Unger, R., Auerbach, D., Given, J. A. and Moult, J. Chaos in Protein Dynamics, December 2005.

[3] Cuyt , A., Verdonk, B., and Verschaeren, D. A Precision- and Range-independent Tool for Testing Floating-point Arithmetic II: Conversions. *ACM Transactions on Mathematical Software,* 27(1):119-140, May 2001.

[4] Goldberg, D. What Every Computer Scientist Should Know About Floating-Point Arithmetic. *ACM Computing Surveys*, March 1991, 23(1):5-48.

[5] IEEE Std 754-1985, "IEEE Standard for Binary Floating-Point Arithmetic."

[6] Taufer , M., Anderson , D.P., Cicotti , P., and Brooks III , C.L. Homogeneous Redundancy: a Technique to Ensure Integrity of Molecular

Simulation Results Using Public Computing. *Proceedings of the 14th Heterogeneous Computing Workshop HCW (2005), in conjunction with IPDPS 2005*. Denver, Colorado, April 2005.

[7] Verdonk , B., Cuyt , A., and Verschaeren, D. A Precision- and Range-independent Tool for Testing Floating-point Arithmetic I: Basic Operations, Square Root, and Remainder. *ACM Transactions on Mathematical Software,* 27(1):92-118, March 2001.

Table 1. For each type of test, i.e., mults, multd, divs, and divd, the table presents the number of unexpected results on each precision-compiler combination. The total number of tests for each operation is on the rightmost column for reference.

Test	64bit/gcc	64bit/Intel	32bit/gcc	32bit/Intel	Total Number of Tests per Platform Type
mults	0	16	16	16	5,408
multd	0	16	16	16	5,408
divs	0	2	2	2	2,153
divd	0	2	2	2	2,153

Table 2. For each type of test, i.e., remd, rint32l, d2bconvd.up , the table presents the number of unexpected results on each precision-compiler combination. The total number of tests for each operation is on the rightmost column for reference.

Test	64bit/gcc	64bit/intel	32bit/gcc	32bit/intel	Total Number of Tests per Group
d2bconvd.down	514	514	513	513	1,018
d2bconvd.nearest	3	3	12	12	1,018
d2bconvd.up	488	488	489	489	1,018
d2bconvd.zero	508	508	517	517	1,018
Remd	93	0	8	0	1,364
Reml	48	0	48	0	1,364
Rems	25	0	8	0	1,364
rint32d	20	20	0	0	184
rint32l	20	20	0	0	184
rint32s	20	20	0	0	184
rintegrald	0	21	0	0	350
rintegrals	0	21	0	0	350
sqrtl	84	0	84	0	1,784