Benchmarking Java Against C/C++ for Interactive Scientific Visualization

Sudhir Sangappa
AVS Express Technical Team
Advanced Visual Systems Inc
Waltham, MA 02451 USA
sudhir@avs.com

K. Palaniappan
Dept. Comp. Eng & Comp. Science
University of Missouri - Columbia
Columbia, MO 65211 USA
palani@meru.cecs.missouri.edu

Richard Tollerton
Dept. Comp. Eng & Comp. Science
University of Missouri - Columbia
Columbia, MO 65211 USA
tollert@meru.cecs.missouri.edu

ABSTRACT Interactive scientific visualization applications require both high performance graphics and numerical computation capabilities. C/C++ programs are able to take direct advantage of specialized graphics rendering hardware (referred to as a Graphics Processing Unit or GPU) for display and special SIMD instructions within the CPU for computation. The former using for example the OpenGL library and latter using compiler extensions. Software developed in Java offers portability, robustness and security. However, the graphics rendering performance, memory efficiency, and computational speed of Java for visualization of large scientific data sets has not been well characterized. Benchmarking experiments were done using the JLoop (Java) and Loop (C/C++) visualization software for animation of 2D datasets. Preliminary results for six different hardware platforms showed that compared to the C/C++ version, the Java implementation was four to five times slower for graphics image animation, used on average 2.5 to three times more memory, and was up to 3.6 times slower using a pure numerical benchmark that simulated image pixel operations. Such benchmarks should be used to improve the Java VM performance across platforms, similar to JIT-compiler design.

General Terms: Performance, Languages, Experimentation
Keywords: Cross-platform visualization, benchmarking, high performance, Java, C/C++, Java advanced imaging

SUMMARY

Scientific visualization applications that require interactivity pose significant software challenges in algorithm development and optimization for handling large datasets, efficient cache management, and parallel execution, have traditionally been developed in C or C++. Software developed in Java offers compelling advantages such as platform independence, architecture neutrality, built-in support for multithreading and distributed programming, automatic garbage collection, a simpler object-oriented framework and compiled or interpreted mode of execution. However, the performance differences between Java and C/C++ implementations of the same visualization functionality needs more thorough investigation in order to quantify software engineering tradeoffs for future projects and to improve the capabilities of the Java Virtual Machine (VM). The relative performance difference of the SUN JDK v1.0 or 1.1 interpreter was 10 to 50 times slower than compiled C code on the same target processor [1]. This deficiency led to the development of Just-In-Time (JIT) compiler technology, for increasing performance so that execution times became only 1.5 to 4 times slower [1]. A JIT-compiler is a code generator that converts Java bytecode into reusable native machine code and is now a standard tool in the current SUN Java 2 SDK Standard Edition v1.4 [2]. The IISS [3] and DISS [4] visualization environments co-developed with NASA Goddard Space Flight Center has been an important tool for understanding correlations between geophysical datasets and coupled processes in the earth-ocean-atmosphere-biosphere system. Complex phenomena are studied by scientists using NASA Earth Observing System satellite imagery, timevarying multidimensional volumes from numerical models, synoptic observations, etc. The Loop module of the DISS, implemented in C is used for interactive animation, roaming, zooming, and probing of high resolution time sequence datasets. JLoop is a high performance platform independent implementation of Loop in Java and QtLoop is a standalone version in C++. JLoop is used to benchmark portability, speed, memory usage, multiprocessing, etc. of visualization software. The performance of JLoop and QtLoop/Loop was compared for interactive animation (display) and numerical computation.

JGI'02, November 3–5, 2002, Seattle, Washington, USA.
ACM 1-58113-599-8/02/0011.

RESULTS

Graphics rendering (and reading) in JLoop of a large data set such as Hurricane George GOES imagery, 3600 x 3600 x 24 bits/frame x 500 frames or 39MB each, was 4 to 5 times slower than Qt/Loop. Animation of an image sequence can be implemented utilizing the MediaTracker class with one or many portable (green) threads. Ensuring smooth flicker-free animation and loading is a challenging task in JLoop. Memory usage was up to five times higher. Computational performance using Java 2 v1.4 for a hundred million evaluations of different quadratics was up to 2.8 times slower than compiled C, both with optimization flags (see Table); SGI IRIX used Java 1.2. Penalty for disabling JIT was 44.41 sec for Linux or 16X slower than C. Other limitations include 2GB-program memory limit (except on Solaris), and the present lack of portability of Java Advanced Imaging (JAI) classes.

	WinP3	***WinP4***	***Linux***	***SUN***	***Alpha***	***SGI***
Java	11.03	7.19	4.36	28.42	18.98	36.06
C	8.55	2.59	2.78	23.23	7.17	14.38
Ratio	1.290	2.781	1.566	1.223	2.647	2.508

Comparison of execution speeds for polynomial evaluation

REFERENCES

[1] I. H. Kazi, H. H. Chen, B. Stanley, and D. J. Lilja, "Techniques for obtaining high performance in Java programs", *ACM Computing Surveys*, 32(3):213-240, 2000.
[2] Austin, C. and M. Pawlan, *Advanced Programming for the Java 2 Platform,* Chap. 8, Addison-Wesley Professional, 2002.
[3] A.F. Hasler, K. Palaniappan, M. Manyin, J. Dodge, "A high performance interactive image spreadsheet (IISS)", *Computers in Physics*, Vol. 8, No. 4, 1994, pp. 325-342.
[4] K. Palaniappan, A. F. Hasler, J. Fraser, M. Manyin "Network-based visualization using the Distributed Image SpreadSheet (DISS)", *17th Int. Conf. on Interactive Information and Processing Systems (IIPS), 81st AMS,* American Meteorological Society, Albuquerque, NM, 2001, pp. 399-403.