Performance Analysis of GYRO: A Tool Evaluation

P Worley[1], J Candy[2], L Carrington[3], K Huck[4], T Kaiser[3], G Mahinthakumar[5], A Malony[4], S Moore[6], D Reed[7], P Roth[1], H Shan[8], S Shende[4], A Snavely[3], S Sreepathi[9], F Wolf[6] and Y Zhang[7]

[1] Oak Ridge National Laboratory, P.O. Box 2008, Oak Ridge, TN 37831-6016
[2] General Atomics, P.O. Box 85608, San Diego, CA 92186-5608
[3] San Diego Supercomputer Center, University of California, San Diego, 9500 Gilman Drive, La Jolla, California 92093-0505
[4] Computer and Information Science Dept., 1202 University of Oregon, Eugene, OR 97403-1202
[5] Dept. of Civil Engineering, North Carolina State University, Raleigh, NC 27695-7908
[6] Innovative Computing Laboratory, University of Tennessee, 1122 Volunteer Blvd., Suite 413, Knoxville, TN 37996-3450
[7] Renaissance Computing Institute, University of North Carolina at Chapel Hill, CB 7583, Carr Building, Chapel Hill, NC 27599-7583
[8] Lawrence Berkeley National Laboratory, Berkeley, CA 94720
[9] Dept. of Computer Science, North Carolina State University, Raleigh, NC 27695-7908

E-mail: `worleyph@ornl.gov`

Abstract.
The performance of the Eulerian gyrokinetic-Maxwell solver code GYRO is analyzed on five high performance computing systems. First, a manual approach is taken, using custom scripts to analyze the output of embedded wallclock timers, floating point operation counts collected using hardware performance counters, and traces of user and communication events collected using the profiling interface to Message Passing Interface (MPI) libraries. Parts of the analysis are then repeated or extended using a number of sophisticated performance analysis tools: IPM, KOJAK, SvPablo, TAU, and the PMaC modeling tool suite. The paper briefly discusses what has been discovered via this manual analysis process, what performance analyses are inconvenient or infeasible to attempt manually, and to what extent the tools show promise in accelerating or significantly extending the manual performance analyses.

1. Introduction

Performance tool development is an active research area, driven by changing processor, memory, and network technologies, increasing system size, increasing application code complexity, evolving programming languages and paradigms, new messaging layers, etc. This paper is an overview of an ongoing study on the benefits of using modern performance tools. The approach taken is to perform detailed performance analyses of a number of scientific application codes. The first code being examined is GYRO [2], an Eulerian gyrokinetic-Maxwell solver developed by J. Candy and R.E. Waltz at General Atomics. GYRO is used by researchers worldwide to study plasma microinstabilities and turbulence relevant to controlled fusion research. The first step in the study was to establish a baseline, collecting and analyzing performance data using only the most basic tools. In subsequent steps, aspects of the baseline analysis were repeated

using more sophisticated tools, identifying what analysis activities could be accelerated and what additional insights could be gained. As there are many tools, we currently focus on performance tools developed by or used in the Performance Evaluation Research Center (PERC) project [12].

In this paper we use data collected on the Cray X1 at Oak Ridge National Laboratory (ORNL) [8], the IBM p690 cluster at ORNL [8], the IBM SP at the National Energy Research Scientific Computing Center (NERSC) [10], the SGI Altix 3700 at ORNL [8], and the TeraGrid Linux cluster at the National Center for Supercomputing Applications (NCSA) [9] for the Waltz standard case benchmark [15], which we refer to as B1-std. The B1-std grid is $16 \times 140 \times 8 \times 8 \times 20$, which is the same resolution used in many production runs, e.g.[1].

The GYRO baseline studies are of three types. First, GYRO comes with embedded wallclock timers and both cumulative and sampled runtime data are collected automatically. The timers surround events that characterize the developers' view of the code. We analyzed these timing data using custom `PERL` scripts and results were plotted with `gnuplot`. For the second baseline study we instrumented the code with calls to HPMLIB [4] `f_hpmstart` and `f_hpmstop` routines at the same locations as the embedded timers. Runs on the p690 cluster were used to collect floating point operation counts for each user event for a number of different processor counts. These data were combined with timing data to determine computational rates and to examine operation count scaling. For the third study, we instrumented the code with calls to the MPICL [7] `traceevent` routine at the same locations as the embedded timers. Runs on the X1 and the p690 cluster were used to collect trace data for both MPI calls and the user-defined events that were used to determine event-specific communication overhead. Visualizations using ParaGraph [3] were used to look for performance bottlenecks.

After the baseline studies were complete, we began applying the following tools and techniques to GYRO.

- Integrated Performance Monitor (IPM) [5]. IPM is a lightweight profiling tool for parallel applications, automatically reporting runtime, communication time, computation rate, and memory requirements, both aggregate and per process, as well as detailed profile data on MPI routine calls and data from system-supported hardware performance counters.
- KOJAK [6, 16]. KOJAK is an automatic trace-analysis toolkit for parallel applications using MPI and/or OpenMP, generating event traces during execution and searching them offline for execution patterns indicating inefficient performance behavior.
- Performance Modeling and Characterization (PMaC) [13]. PMaC is a suite of tools for characterizing system and application performance and for using these characterizations to build performance models suitable for performance optimization and extrapolation.
- SvPablo [11]. SvPablo is a graphical environment for instrumenting application source code and browsing and analyzing the resulting performance data.
- Tuning and Analysis Utilities (TAU) [14]. TAU is framework and toolkit for performance instrumentation, measurement, and analysis of parallel applications.

It is beyond the scope of this paper to describe the full capabilities of any of these tools. Rather we briefly relate what aspects of the baseline studies could be improved or superceded by using these tools.

2. Results

The remainder of the paper is concerned with the following question. *For which performance analysis and optimization activities are the examined tools and methodologies (a) not needed, (b) useful but not required, and (c) difficult to do without.*

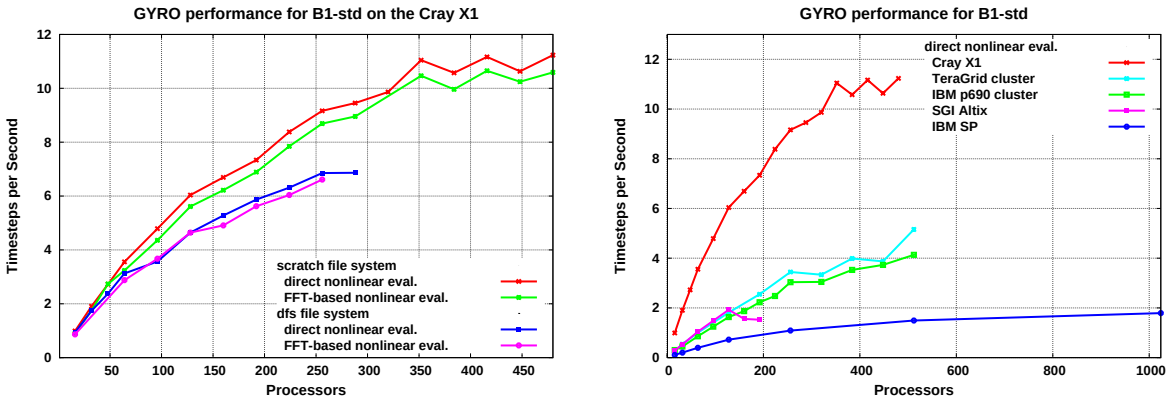

Figure 1. GYRO runtime performance.

2.1. Analyses for which tools are not needed.
There are a number of standard analyses for which an in-depth understanding of the performance is not needed. We mention two here.

Many codes have embedded tuning options that allow the algorithms or implementation to be modified at compile- or runtime. The optimal choice is often a function of the computer system, problem specification, or runtime configuration (e.g., number of processors). If the search space is small, it is simplest to determine the optimum by measuring the performance of each option directly. For example, GYRO supports two methods for evaluating nonlinear terms in the underlying equations: direct and FFT-based. The FFT-based method is slower on small grids, but faster on large grids. However, the direct method achieves higher computational rates than the FFT-based method, and the crossover point varies from system to system. The left graph in Figure 1 shows the performance differences in terms of timesteps per second on the Cray X1 for the two methods, and for running out of two different file systems. Here the direct method is approximately 10% faster than the FFT method, and the choice of file system makes a 30% difference in performance. Other options found to be important to GYRO include settings for system-specific environment variables that impact memory, OpenMP, or MPI performance.

Another standard activity is benchmarking. The benchmark timings should represent what would be observed in a production run, i.e., without performance tools. The right graph in Figure 1 is an inter-platform comparison of GYRO performance. Note that even these "simple" analyses were not inexpensive. Over 175 experiments were run, on processor counts up to 1024, and the number of experiments on a given system was constrained by resource availability. We were not able to collect all of the data that we would have liked on any of the target systems.

2.2. Analyses for which tools are useful.
A number of common performance analyses did not require sophisticated performance tools, but were time consuming without them. Examples include (in order of increasing difficulty) (a) user event profiling, (b) computational rate calculations (both whole code and per user event), and (c) communication rates calculations (both whole code and per MPI command). To calculate computational rates we ran 13 additional experiments, while to calculate communication rates we ran 31 additional experiments and collected over 700 MB of trace data (compressed). Both required writing scripts to combine timing data with the operation count and MPI data.

An example of user event profiling appears in the left graph of Figure 2, where the percentage of runtime spent in each user event is plotted as a function of processor count ("phase diagram") on the Altix. These data indicate that performance scales poorly in `Coll_tr`, a user event

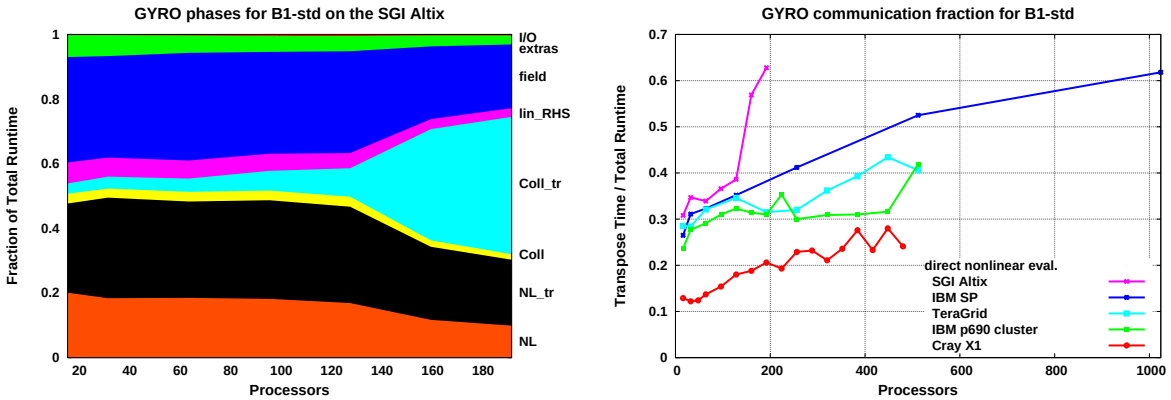

Figure 2. Phase diagram for B1-std on the SGI Altix and inter-platform comparison of time spent in tranpose events

dominated by MPI communication, when increasing the number of processors from 128 to 192. The analysis does not indicate why, however, and additional data are needed to determine the reason for the poor performance. The right figure compares the fraction of time spent in the two user events most dominated by MPI communication (`Coll_tr` and `NL_tr`) across the platforms. Because the catastrophic scaling behavior is unique to the Altix, additional data on the application characteristics are unlikely to be sufficient to diagnose the problem.

It is in the collection, analysis, and presentation of multiple related measurements that performance tools begin to show their worth. For example, SvPablo and TAU both support calculating communication rates while collecting profile data, not requiring the postprocessing of trace data. Manually collected data can also be loaded into the performance database component of TAU, called PerfDMF, facilitating subsequent multi-experiment, multi-data-type, analyses.

2.3. Analyses for which tools are important.

The previous analyses address only implicitly the real question, i.e., whether performance is acceptable, and, if not, why not. However, the tools KOJAK, SvPablo and TAU can be used for (d) identifying critical paths (potential performance bottleneck) and (e) global view analysis, e.g. examining load balancing or the impact of system noise. For example, KOJAK, by comparing event traces for different runs, identified particular MPI_AlltoAll calls as the location of the Altix performance problem, though it has not yet led to a resolution. Comparative analysis of trace files is clearly not a manual activity.

SvPablo and TAU can also be used for the iterative process of (f) detailed performance debugging, i.e., identifying and tracking performance problems down to individual routines and lines of code. When performed by hand, detailed performance debugging is time consuming and fraught with problems due to instrumentation perturbation and global effects (e.g., load imbalances) masquerading as local performance problems.

The performance questions mentioned previously were all concerned with understanding and optimizing current performance. Another class of questions include (g) estimating performance when changing the problem size, number of processors, or moving to a different system and (h) finding the optimal tuning parameters within a large search space. Both of these questions can be addressed by performance models, i.e., parameterized representations of application runtime. Depending on the form of the model, it may be easily manipulated "manually". The difficulty with the model is its generation. There are a number of modeling methodologies described in the literature, including the PMaC tools and methodology examined in these studies.

3. Conclusions

Our study indicates that there are a number of common performance analyses for which sophisticated performance tools are not necessary. However, many of these analyses are expensive, in both system resources and labor, and a number of useful analyses are simply not practical to perform manually, thus requiring tool support. The contribution of this work is in characterizing some of the costs for a real application code on current parallel systems. One issue that has become obvious during the course of the study is that performance tools need to be in the hands of developers. Too much time was spent by the performance experts in discovering performance characteristics that the developer already knew about. Because most developers do not do performance analyses every day, it is difficult for them to be comfortable with any performance tool, much less a suite of them. There is clearly a tradeoff between tool functionality and usability. Tools such as KOJAK, SvPablo, and TAU require considerable effort to install and set up for use with an application in order to collect the desired performance metrics at an appropriate level of granularity. Similarly, while models are wonderful tools that a developer could use for many activities, generating the model is something few people are willing to do, and efficient ways of updating and maintaining models are still open questions. In conclusion, there is still more to do in performance tool development, but tools make performance analysis and optimization feasible in instances when it would not be otherwise, especially when running with many processors and working with complex applications.

Acknowledgments

This research was sponsored by the Office of Mathematical, Information, and Computational Sciences, Office of Science, U.S. Department of Energy under Contract No. DE-FG03-95ER54309 with General Atomics, No. DE-FC02-04ER25612 with the University of North Carolina, No. DE-AC03-76SF00098 and No. DE-FC02-01ER25491 with the University of California, No. DE-FG02-05ER23680 and No. DE-FG03-01ER25501 with the University of Oregon, No. DE-FC02-01ER25490 with the University of Tennessee, and No. DE-AC05-00OR22725 with UT-Batelle, LLC. Accordingly, the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

References

[1] J. Candy, *Beta scaling of transport in microturbulence simulations*, Submitted to Phys. Plasmas.

[2] J. Candy and R. Waltz, *An eulerian gyrokinetic-maxwell solver*, J. Comput. Phys., 186 (2003), p. 545.

[3] M. T. Heath and J. A. Etheridge, *Visualizing the performance of parallel programs*, IEEE Software, 8 (1991), pp. 29–39.

[4] IBM Advanced Computing Technology Center, *Hardware Performance Monitor*. http://www.research.ibm.com/actc/projects/hardwareperf.shtml.

[5] Integrated Performance Monitor. http://www.nersc.gov/nusers/resources/SP/ipm/.

[6] KOJAK. http://www.fz-jeulick.de/zam/kojak/.

[7] MPICL. http://www.csm.ornl.gov/picl/.

[8] National Center for Computational Sciences. http://www.ccs.ornl.gov/.

[9] National Center for Supercomputing Applications. http://www.ncsa.edu/.

[10] National Energy Research Scientific Computing Center. http://www.nersc.gov/.

[11] Pablo Research Projects. http://www.renci.unc.edu/Project/ResearchProjects.htm.

[12] Performance Evaluation Research Center. http://perc.nersc.org/.

[13] Performance Modeling and Characterization. http://www.sdsc.edu/PMaC.

[14] Tuning and Analysis Utilities. http://www.cs.uoregon.edu/research/paracomp/tau/tautools.

[15] R. Waltz, G. Kerbel, and J. Milovich, *Toroidal gyro-landau fluid model turbulence simulations in a nonlinear ballooning mode representation with radial modes*, Phys. Plasmas, 1 (1994), p. 2229.

[16] F. Wolf and B. Mohr, *Automatic performance analysis of hybrid MPI/OpenMP applications*, Journal of Systems Architecture, 49 (2003), pp. 421–439.