Performance Analysis of GYRO: Timers, Counters, and Traces [*]

P. H. Worley [†]
Oak Ridge National Laboratory

J. Candy [‡]
General Atomics

G. Mahinthakumar [§]
North Carolina State University

P.C. Roth [¶]
Oak Ridge National Laboratory

H. Shan [‖]
Lawrence Berkeley National Laboratory

S. Sreepathi [**]
North Carolina State University

Abstract

The performance of the Eulerian gyrokinetic-Maxwell solver code GYRO is analyzed on five high performance computing systems. The analysis is based on 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. Custom scripts were used to extract results from these data. The paper discusses what was discovered via this manual analysis process, at what cost, and what aspects of the performance were not easily examined by this approach.

1 Introduction

Performance tools are 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 the first in a series with the goal of quantifying, where possible, or commenting on the benefit of using modern performance tools. The approach that we are taking is to perform detailed performance analyses of a number of production application codes. The first code we have examined is GYRO [3]. GYRO is an Eulerian gyrokinetic-Maxwell solver developed by J. Candy and

[*]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-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.

[†]Computer Science and Mathematics Division, Oak Ridge National Laboratory, P.O. Box 2008, Bldg. 5600, Oak Ridge, TN 37831-6016 (worleyph@ornl.gov)

[‡]General Atomics, P.O. Box 85608, San Diego, CA 92186-5608 (candy@fusion.gat.com)

[§]Dept. of Civil Engineering, North Carolina State University, Raleigh, NC 27695-7908 (gmkumar@ncsu.edu)

[¶]Computer Science and Mathematics Division, Oak Ridge National Laboratory, P.O. Box 2008, Bldg. 5600, Oak Ridge, TN 37831-6016 (rothpc@ornl.gov)

[‖]Lawrence Berkeley National Laboratory, Berkeley, CA 94720 (hshan@lbl.gov)

[**]Dept. of Computer Science, North Carolina State University, Raleigh, NC 27695-7908 (saratr_s@ncsu.edu)

R.E. Waltz at General Atomics. It is used by researchers worldwide to study plasma microinstabilities and turbulence relevant to controlled fusion research. The first step in our evaluation methodology is to establish a baseline, collecting and analyzing performance data using only the most basic tools. In subsequent steps we repeat aspects of the baseline analysis using more sophisticated tools, identifying what analysis activities can be accelerated and what additional insights can be gained. This paper describes the initial results from the baseline analysis of GYRO. While the primary focus of this work is on the generation of the baseline study against which the various tools will be compared, the performance analysis described herein is also of interest as a description of what can be done without sophisticated tools, and at what cost, The exposition also indicates a number of issues that should be considered in any performance analysis.

The GYRO baseline studies are of three types. First, GYRO comes with embedded wallclock timers (using MPI_WTIME) and both cumulative and sampled runtime data are collected automatically. The timers surround events that characterize the developer's view of the code. We ran the code for two different benchmark problems on five different parallel systems for a large number of different processor counts on each system. Analysis of the timing data was performed using custom `PERL` scripts and results were plotted with `gnuplot`. For the second baseline study we instrumented the code with calls to HPMLIB [7] `f_hpmstart` and `f_hpmstop` routines at the same locations as the embedded timers. Runs on an IBM SMP cluster were used to collect floating point operation counts for each user event for each benchmark problem and for a number of different processor counts. These data were combined with timing data to determine computational rates and to examine operation count scaling, again using `PERL` scripts and `gnuplot`. For the third study, we instrumented the code with calls to the MPICL [5, 10] `traceevent` routine at the same locations as the embedded timers. Runs on a Cray X1 and an IBM p690 cluster were used to collect both profile and trace data for both MPI calls and the user-defined events. These data were used to verify the developer's understanding of the embedded timers, especially with regard to MPI overheads. The trace data were also used to determine event-specific communication overhead. Visualization using ParaGraph [6] was used to look for performance bottlenecks.

The baseline studies reflect a standard methodology used by the authors. It does not include common first steps such as using compiler-based profiling. The assumption was that this type of information was already used by the developer when deciding where to insert the embedded timers. HPMLIB (and the other components of HPM) are sophisticated tools, and their use in the baseline studies is not meant to imply otherwise. However, we used them in an "unsophisticated" way, simply recording operation counts in order to determine computation rates. And, while tools for tracing and visualization have never been simple, MPICL and ParaGraph are quite old now. The basic property of all of the baseline studies is their manual, brute force, approach, reflecting what a user could have done ten years ago. As we will show, performance analysis can be accomplished in this way, but it is time consuming.

The outline of the paper is as follows. Section 2 is a description of the GYRO code. Section 3 is a brief description of the systems on which performance data were collected. Sections 4 and 5 describe the first two baseline studies. To satisfy the page limit for the extended abstracts, the exposition has been shortened. In particular, the analysis of the MPICL trace data has been eliminated. However, section 6 briefly describes some of additional results that will be presented in the final paper. Section 7 contains concluding remarks.

2 GYRO

The most promising and aggressively studied concept for power production by fusion reactions is the tokamak. Advances in the understanding and control of tokamak plasmas are continuously being realized, although uncertainties remain in predicting containment properties and performance of larger reactor-scale devices. The coupled gyrokinetic-Maxwell (GKM) equations provide a foundation for the first-principles calculation of turbulent tokamak transport. For years, the numerical solution of the nonlinear GKM equations has been a computational physics "Grand Challenge".

GYRO [3] is a code that simulates tokamak microinstabilities and turbulence by solving the time-dependent, nonlinear gyrokinetic-Maxwell equations for both ions and electrons. It uses a five-dimensional grid (three spatial and two velocity coordinates) and advances the system in time using a second-order, implicit-explicit Runge-Kutta integrator. An explicit fourth-order intergrator is also available for problems with reduced physics. GYRO is the only GKM code worldwide with proven global and electromagnetic operational capabilities. It has been ported to a variety of modern MPP platforms including Cray, IBM, and SGI high performance computing (HPC) systems and also to a number of commodity clusters.

GYRO is a parallel code written in Fortran 90 and using MPI for interprocess communication. Parallelism is based on a domain decomposition of the computational grid. A simplified view of the GYRO control flow is as follows.

```
initialization
do step=1,nstep

   1) distributed matrix redistribution (type a)
   2) collision term evaluation
   3) distributed matrix reverse redistribution (type a)

   4) implicit solve of fields and electron distribution #1
   5) implicit solve of fields and electron distribution #2
   6) explicit evaluation of right hand sides (RHS) of electron
      and ion Gyrokinetic equations (GKEs) #1
      6a) distributed matrix redistribution (type b)
      6b) nonlinear term evaluation
      6c) distributed matrix reverse redistribution (type b)
      6d) linear terms evaluation
   7) implicit solve of fields and electron distribution #3
   8) explicit evaluation of RHS of electron and ion GKEs #2
      8a) distributed matrix redistribution (type b)
      8b) nonlinear term evaluation
      8c) matrix reverse redistribution (type b)
      8d) linear terms evaluation

   9) time-advance of all distributions (adding partial results)

   10) explicit field updates

   11) diagnostic computations and field outputs

enddo
```

Note that we typically refer to the distributed matrix redistributions as *transposes*. These are implemented in GYRO using a series of calls to the MPI collective `MPI_Alltoall`. An input parameter `time_skip` determines during which timesteps the output of large arrays occur:

```
  if (modulo(step,time_skip) == 0) call write_big
```

We will refer to these as *output timesteps*.

The GYRO source code is also instrumented with timers that collect runtime data for a number of logical events. Per event runtimes are accumulated between each output timestep, and runtime data is output each output timestep. Example timer output is described below.

```
    NL        NL_tr     Coll      Coll_tr   lin_RHS    field     extras      I/O       STEP     ELAPSED   RUNTIME
 --------- --------- --------- --------- --------- --------- --------- --------- --------- --------- ---------
 0.000E+00 0.000E+00 0.000E+00 0.000E+00 0.000E+00 0.000E+00 0.000E+00 0.000E+00 0.000E+00
 0.000E+00 0.000E+00 0.000E+00 0.000E+00 0.000E+00 0.000E+00 0.000E+00 0.000E+00 0.000E+00 0.000E+00 0.000E+00

 1.374E-01 1.795E-01 2.947E-02 2.167E-02 2.731E-02 1.759E-01 9.091E-03 3.283E-01 9.086E-01
 7.180E+00 8.676E+00 1.468E+00 1.145E+00 1.387E+00 9.005E+00 1.405E+00 3.284E-01 3.059E+01 3.062E+01 3.062E+01

 1.400E-01 1.681E-01 2.938E-02 2.319E-02 2.710E-02 1.765E-01 8.675E-03 2.921E-01 8.651E-01
 7.194E+00 8.808E+00 1.470E+00 1.143E+00 1.383E+00 9.118E+00 1.390E+00 2.958E-01 3.080E+01 3.082E+01 6.145E+01
```

Two lines are written at every output timestep. The first of the two lines is the time (in seconds) spent in each event during the most recent timestep, as measured by process 0. The second line is the accumulated time in each event since the last output timestep. The second to last column is the total runtime since the last output. The last column is the cumulative runtime. Here

```
NL:       calculation of nonlinear terms in RHS of GKEs [steps 6b,8b]
NL_tr:    redistribution (transposes) before and after nonlinear
          calculations [6a,6c,8a,8c]
Coll:     collision calculation [2]
Coll_tr: redistribution (transposes) before and after collision
          calculation [1,3]
lin_RHS: calculation of linear terms in RHS of GKEs [6d,8d]
field:    explicit or implicit advance of fields (from Maxwell
          equations) [4,5,7]
extras:   diagnostic calculations and other miscellaneous events [11]
I/O:      simulation output [11]
step:     summation of other event timers
```

While these timers do not cover all of the code, there is not much left out. Note that almost all of the time spent in I/O between output timesteps occurs in the output timestep itself.

For this paper we analyze the performance of GYRO when running two benchmark problems, both provided by the GYRO authors:

- B1-std. B1-std is the Waltz standard case benchmark [9]. This is a simulation of electrostatic turbulence using parameters characteristic of the DIII-D tokamak at mid-radius. Both electrons and ions are kinetic, and electron collisions (pitch-angle scattering) are included. The grid is $16 \times 140 \times 8 \times 8 \times 20$. Since 16 toroidal modes are used, a multiple of 16 processors must be used to run the simulation. This test case is run for 500 timesteps (which represents a physical time $t = 10a/c_s$, where a is the plasma minor radius, and c_s is the ion sound speed). Results and timing data are output every 50 timesteps.

  The numerical grid resolution used for this benchmark is the same as that used in production runs [4, 8, 2]. It represents, roughly, the minimum grid size required to obtain physically accurate results. Note that approximately 40000 timesteps would be used in a typical production run.

- B3-gtc. B3-gtc is a high-toroidal-resolution electrostatic simulation with simplified electron dynamics (only ions are kinetic). The grid is $64 \times 400 \times 8 \times 8 \times 20$. This case uses 64 toroidal modes, and so must be run on multiples of 64 processors. The test case is run for 100 timesteps, representing a simulation time of $t = 3a/c_s$. Results and timing data are output every 50 timesteps. The 400-point radial domain with 64 toroidal modes gives extremely high spatial resolution, but electron physics is ignored, allowing a simple field solve and large timesteps.

GYRO supports two different methods for evaluating the nonlinear terms in the right hand side of the GKEs: direct and FFT-based. Using FFTs is slower on small grids, but faster on large grids. Unless otherwise indicated, the direct method is used for all B1-std results and the FFT-based method is used for all B3-gtc results.

3 Experimental Platforms

GYRO performance data were collected on the following systems. Processor details for these systems are summarized in Table 1.

Cray X1 at Oak Ridge National Laboratory (ORNL): 128 4-processor X1 SMP nodes and a Cray interconnect. Each processor is a Multi-Streaming Processor (MSP) comprised of 8 32-stage vector units running at 800 MHz, 128 64-bit wide, 64-element deep vector registers, and 4 scalar units running at 400 MHz.

	Processor	MHz	L1 cache	L2 cache (per proc. if shared).	L3 cache (per proc. if shared)	Peak Processor Performance (GFlop/sec)
Intel	Itanium 2	1500	32KB	.25MB	6MB	6.0
NH2	POWER3-II	375	64KB	8MB	–	1.5
p690	POWER4	1300	32KB	0.72MB	16MB	5.2
X1	Cray MSP	800	16KB per scalar unit	2MB	–	12.8

Table 1: Test Platforms

IBM p690 cluster at ORNL: 27 32-processor p690 SMP nodes and an HPS interconnect. Each node has two 2-link network adapters. Each processor is a 1.3 GHz POWER4.

IBM SP at the National Energy Research Scientific Computing Center (NERSC): 416 16-processor Nighthawk II (NH2) SMP nodes and an SP Switch2 interconnect. Each node has two network adapter cards. Each processor is a 375 MHz POWER3-II.

SGI Altix at ORNL: 128 2-processor SMP nodes and a NUMAlink interconnect. The Altix is a Non-Uniform Memory Access (NUMA) cache coherent shared memory system in which the two 2-processor SMPs are connected to form a *C-brick*, and some number of C-Bricks are then connected to form the larger system. Each processor is a 1.5 GHz Intel Itanium 2.

TeraGrid Linux cluster at the National Center for Supercomputing Applications (NCSA): 631 2-processor SMP nodes and a Myrinet 2000 interconnect. Each processor is a 1.5 GHz Itanium 2.

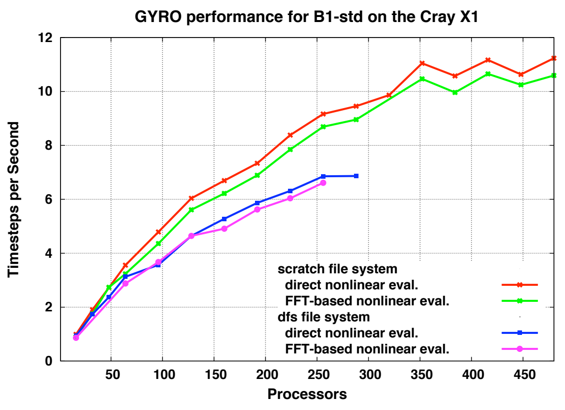
4 Embedded Timers

The runtime data generated by the embedded timers can be used in a variety of performance studies. Here we focus on platform-specific analysis and tuning, and platform intercomparisons and scalability studies. The timers, as implemented, have some deficiencies. Since only process 0 timings are reported, load imbalances cannot be observed. However, the calls to `MPI_Alltoall` used in the `NL_tr` and `Coll_tr` act as synchronization points, and the `step` and `elapsed` runtimes are accurate. Also, reporting only cumulative time between output timesteps and runtime for the output timestep itself may hide runtime variability between timesteps. Note that these comments are not meant to imply that the embedded timer logic should be modified. Measuring load imbalances or runtime variability would require the collection of significantly more performance data, data that serves no purpose in a typical production run. The current timers provide useful, if limited, data that is appropos as a default. Other tools or approaches can be applied if the embedded timers indicate that there is a potential performance problem.

The approach we took to analyzing the timer data was to generate 2D plots and manually examine (and interpret) the results. Six different types of graphs were generated:

- timesteps per second,
- percentage of total time spent in each user event (which we will refer to a *phase diagram*),
- percentage of total time spent in transpose events,
- efficiency (relative to smallest processor count),
- ratio of maximum runtime between output timesteps and minimum runtime between output timesteps,
- ratio of maximum runtime for an output timestep and minimum runtime for an output timestep,

all as functions of processor count. Here we present a sampling of the results to indicate performance issues that will be addressed again in later sections.



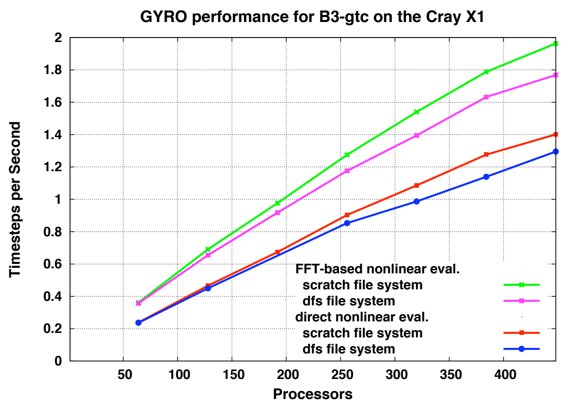
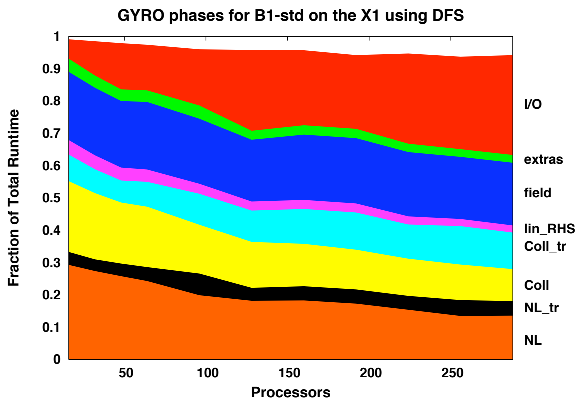


Figure 1: GYRO performance on the Cray X1.



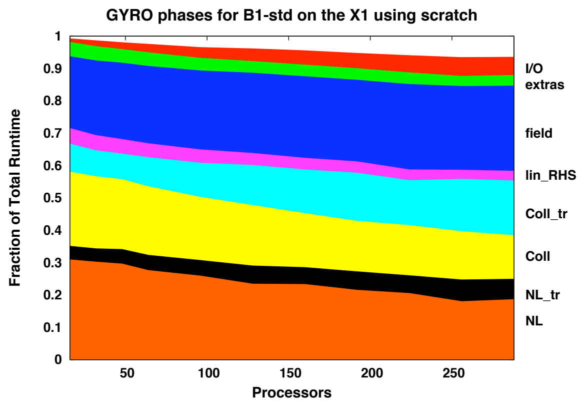


Figure 2: B1-std phase diagram for two different file systems on the Cray X1.

4.1 Platform-specific analyses

We begin with simple platform-specific analyses and tuning, empirically examining the impact of a number of performance tuning options. *For this long abstract we limit our discussion to the Cray X1. The final paper will also include platform-specific analyses for the systems.*

Cray X1. For the X1, we examined two issues: performance of direct and FFT-based evaluation of non-linear terms, and performance when running out of two different file systems, for a total of four different configurations. One of the file systems is a DFS/DCE file system used for user home directories. The other is a local "scratch" RAID file system. Figure 1 contains graphs of timesteps per second for each of the four configurations for the two benchmark problems. The FFT-based evaluation method provides a large performance boost for B3-gtc, but is somewhat worse than the direct method for B1-std. The scratch file system provides signficantly better performance than the DFS file system (as supported by the Cray X1) for both problems. Figure 2 shows the percentage of time spent in each of the user-defined events for the B1-std problem when using the direct evaluation method for the two file systems. This confirms that the performance difference is primarily difference in I/O performance.

4.2 Platform intercomparisons

The classic application of benchmark (runtime) data is in comparing performance between platforms. Interplatform comparisons can also be enlightening in performance analysis in that it helps identify which performance characteristics are common across all platforms, i.e. are primarily application characteristics, and which are platform-specific.

Figure 3 contains graphs of timesteps per second for all of the experimental platforms for B1-std, using the direct method for evaluating the nonlinear terms, and for B3-gtc, using the FFT-based method. *As will be discussed in the final version of the paper, the FFT-based evaluation method is not faster on the Teragrid. To simplify (and shorten) the presentation here, we have excluded the TeraGrid from B3-gtc discussions.* From these data, GYRO runs well on the vector system, so the code must vectorize reasonably well. For the Altix and X1, B1-std performance appears to peak before running out of parallelism. The same is not true for the other platforms.

Figure 4 contains graphs of the relative efficiency for the same problems and platforms. The data for B1-std is somewhat difficult to decipher, but one important fact is that the relative efficiency for the X1 is not that much worse than that for the SP, which continues to show performance improvement out to 1024 processors (even though relative efficiency has dropped to 25% for that processor count). If we look only at power-of-two processor counts for the X1, then the X1 scaling curve is qualitatively similar to that of the SP. Thus the performance for the other processor counts may or may not indicate that performance has peaked on the X1. It may just indicate that GYRO performance is sensitive to the domain decomposition details. The TeraGrid and p690 cluster data also show some of this processor count sensitivity. The data for B3-gtc is easier to interpret. This large problem has excellent scaling properties on all of the platforms, though efficiency is decreasing faster for the X1 than for the other platforms.

Other than pointing out similar trends, the relative efficiency does not explain much about the performance. Figure 5 contains graphs of the fraction of total runtime spent in the transpose communication events, allowing us to look at the role of communication costs in determining scaling behavior. Again, note that the communication fraction is defined by the time spent in the transpose events on process 0, and could also include load imbalance. However, comparing performance and scaling between platforms and taking into account the relative speeds of the processors and interconnects, it does appear that the majority of the time in the transpose events is due to the cost of transposing the arrays. From these data we see that, even though scalability is excellent for B3-gtc, communication overhead makes up a significant fraction of the runtime. For B1-std, with the exception of the Altix, the high fraction of communication does not prevent continued performance improvement as processor counts increase. In particular, the X1 communication fraction is the smallest of all the platforms, so any scaling problem must be due to decreased serial performance, perhaps decreasing vector lengths.

Our final performance study using the embedded timers is to examine runtime variability within a run. Figure 6 contains graphs of the ratio of the maxmimum and minimum runtimes between output timesteps. For B1-std, there are 10 data for each experiment, and each datum is the accumulated runtime for 50 timesteps. For B3-gtc, there are only 2 data for each experiment, and each datum is the accumulated runtime for 100 timesteps. Note the different y-axis scale for the B1-std and the B3-gtc results. There does not appear to be any application source of runtime variability for either benchmark (though the B3-gtc data is not sufficient for establishing this). In particular, the Cray X1 and IBM SP results show minimal jitter. Given this, the runtime variability for B1-std on the p690 cluster for large processor counts is striking (and repeatable). This will have a significant impact when scaling to even higher processor counts in the p690 cluster. (*A similar examination of the performance variability in the output timesteps will be given in the final version of the paper.*)

4.3 Summary

The embedded timers allowed us to identify and correct a few performance problems and to tune the code with respect to the existing application-supported tuning options. Overall, scaling behavior is qualitatively similar. It is clear that performance is sensitive to communication performance, and there does not appear to be any significant application source for runtime variability. These results were generated by running many experiments and manually “mining” the data, with frequent returns to the experimental platforms to run additional experiments as gaps in the data were identified.

However, even with all this data, there is no way to tell whether the code is running well on any of the platforms. And, with the exception of attempting to decrease communication overhead, there is also no

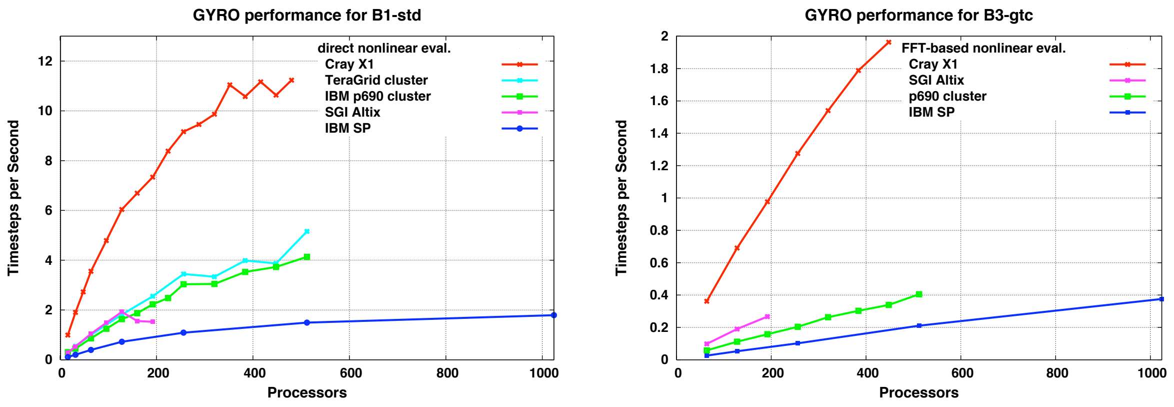


Figure 3: GYRO performance on experimental platforms.

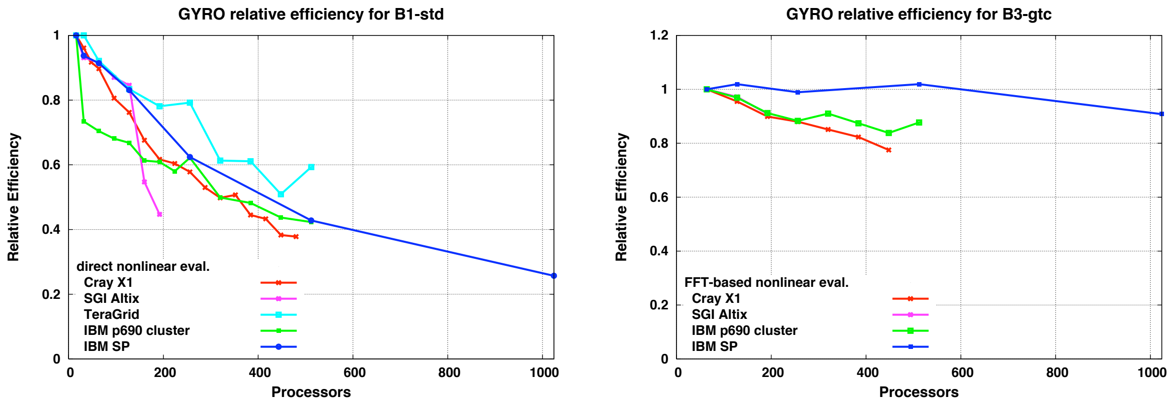


Figure 4: GYRO relative effeciency on experimental platforms.

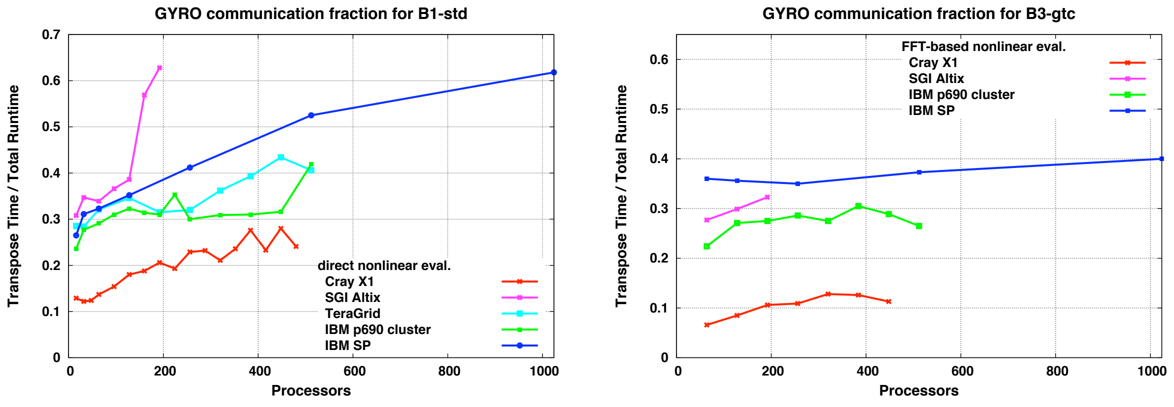
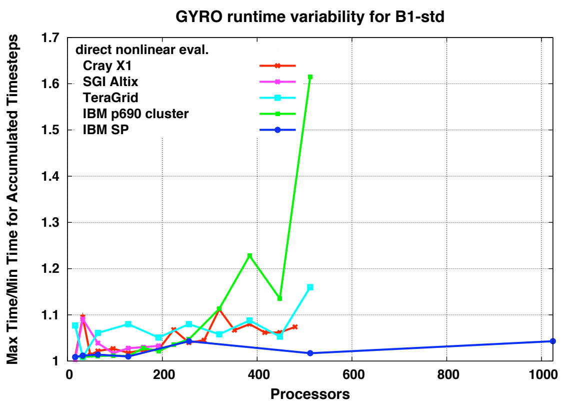


Figure 5: GYRO communication fraction on experimental platforms.



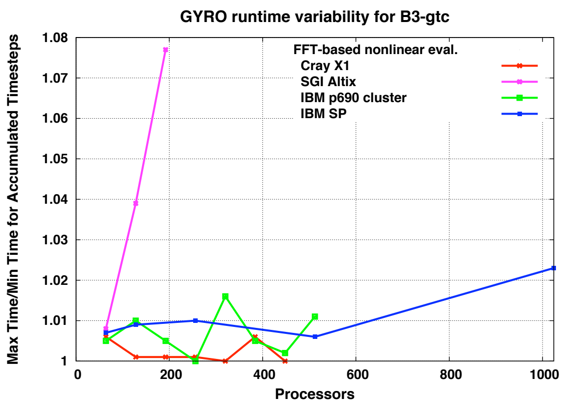
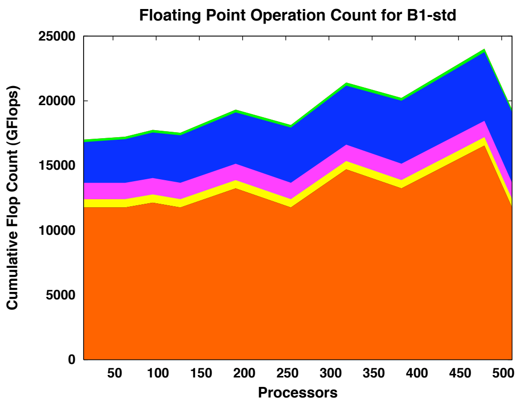


Figure 6: GYRO runtime variability on experimental platforms.



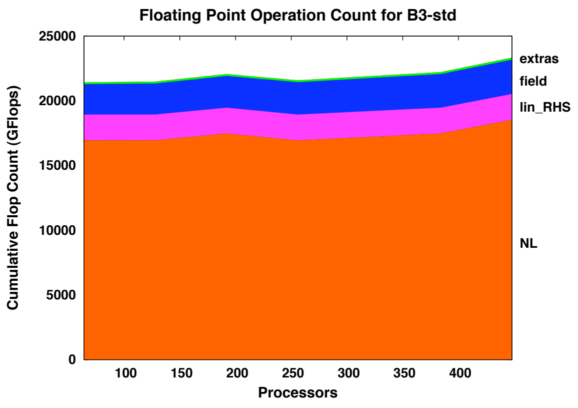


Figure 7: GYRO floating point operation counts on the IBM p690 cluster.

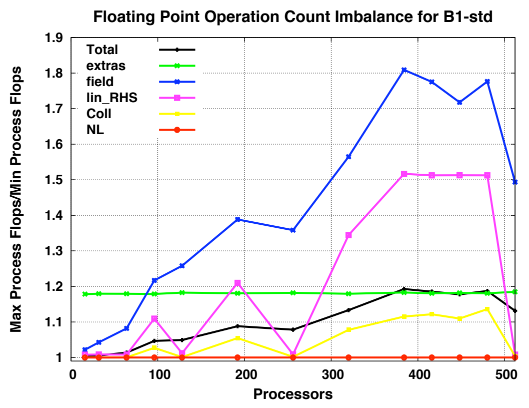
indication where to look for improved performance.

5 HPM

While the embedded timers were able to indicate relative efficiency, there was no direct information on processor performance. The HPM library (or PAPI [1] or other techniques for reading hardware performance counters) can be used to determine this information. In this case, we collected floating point operation (*flop*) counts on the p690 cluster and used this to calculate computation rates in terms of GFlop/second/processor on all of the systems. This is inaccurate on any system other that the p690 cluster, but it does provide a consistent normalization and allows direct interplatform comparisons. In our experience, it is also reasonably accurate, independent of platform.

The first step was to examine the flop count for each phase. Figure 7 contains phase diagrams that describe percentage of total flop count for each user event as a function of processor count. From these data we can see that the flop count is somewhat sensitive to processor count, especially for B1-std. The B1-std data for processor counts between 384 and 480 also at least partially explains the performance of the X1 for these same processor counts. Note that the majority of the flop count variability occurs in `NL`.

We next looked at load imbalance in the flop counts across the different processors counts. Figure 8 contains graphs of the ratio of the maximum flop count to the minimum over all processes for each computational user event. The load imbalance in total flop count is less than 22% for B1-std and less than 3% for B3-gtc. This is primarily due to a lack of load imbalance in `NL`, the most computationally expensive user event. Note that this implies that most of the flop count fluctuations as a function of processor count occur



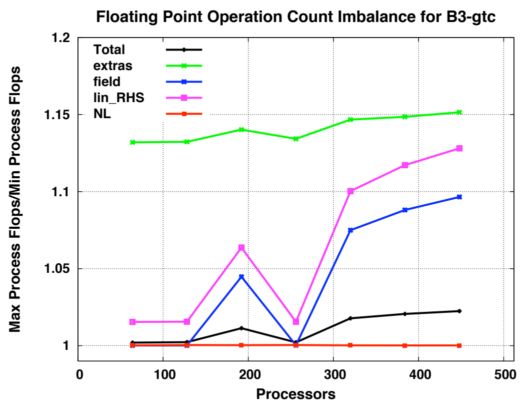
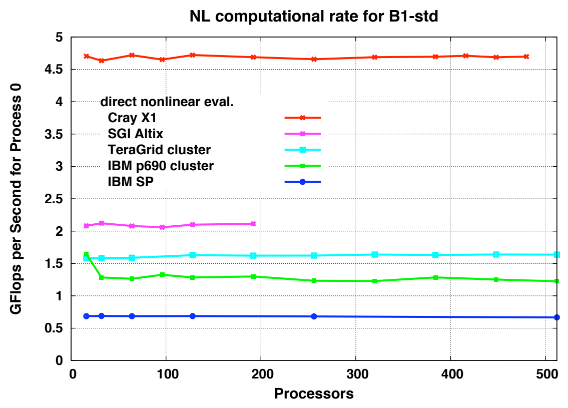


Figure 8: GYRO floating point operation count imbalances on the IBM p690 cluster.



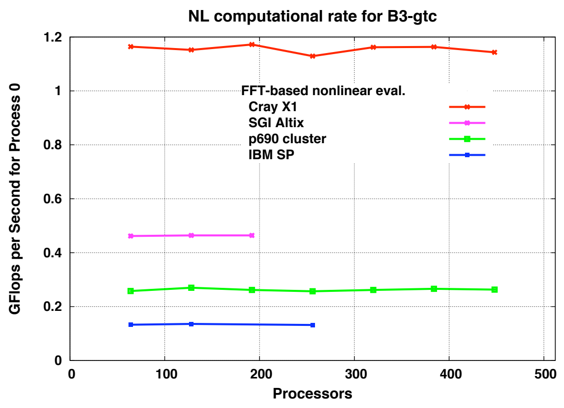


Figure 9: NL computation rates.

uniformly across all processes.

Both `field` and `lin_RHS` are nontrival events with significant imbalances for B1-std that may effect the measured overhead in the first communication event that follows them, the first transpose in `NL_tr`. The raw data also indicates that the flop count distribution in `lin_RHS` is bimodal while that in `field` has three modes, with essentially two and three distinct values, respectively.

Figure 9 uses the flop counts for process 0 on the p690 cluster to calculate computational rates for process 0 for `NL` for each platform. These data show that the relative computational performance for `NL` between the platforms is the same for both the direct method in B1-std and the FFT-based method in B3-gtc, despite the computational rate of the FFT-based method being approximately one fourth that of the direct method. The same comparison holds between the direct and FFT-based methods for B3-gtc, with the exception of the p690 cluster where the computational rate of the direct method is a little less than 3 times that of the FFT-based method. The advantage of the FFT-based method is that its floating point operation counts are approximately one sixth that of the direct method, giving it a 50% advantage. The reason for the lower computational rate for the FFT-based method is not clear from these data. However, it is at least partially due to the computation rate of the vendor FFT routines and the significant amount of low intensity computations (and simple memory copies) before and after the FFT routine calls.

The computational rates for the other computational events (`Coll`, `lin_RHS`, `field`) are also relatively insensitive to the processor count. The average rates on process 0 (averaged over all process counts) for each event, benchmark, and platform are given in Table 2. The noteworthy result in this data is that `Coll` performance is poor on the X1, and is a candidate for additional optimization. The difference in the TeraGrid and Altix performance could also be examined, given that they use the same processor. Finally,

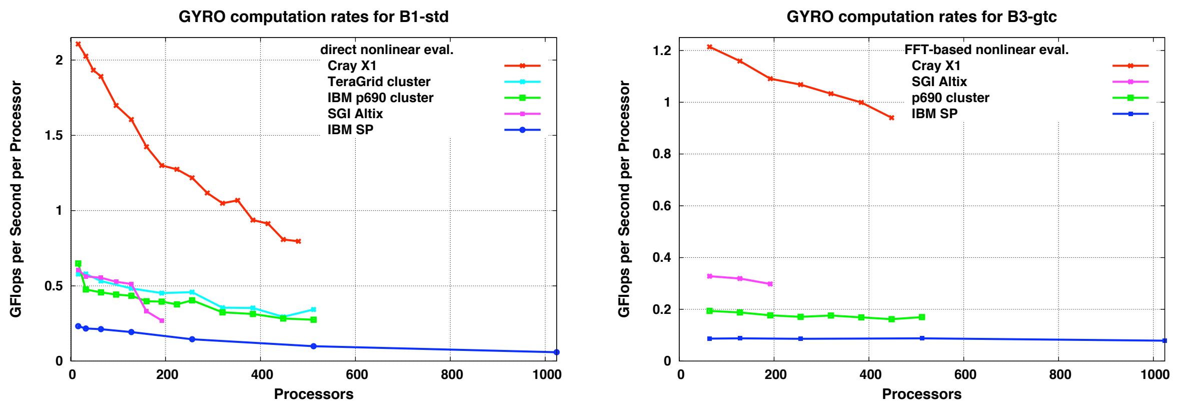


Figure 10: GYRO computation rate on experimental platforms.

the p690 performance is not as much greater than the SP performance as would be expected given the peak performances of the two processors.

B1-std computation rates (GFlop/sec)

	NL	Coll	lin_RHS	field
Cray X1	4.7	.32	3.2	1.6
IBM p690 cluster	1.3	.34	.71	.35
IBM SP	.69	.20	.26	.14
SGI Altix	2.1	.68	.75	.37
TeraGrid	1.6	.47	.57	.37

B3-gtc computation rates (GFlop/sec)

	NL	Coll	lin_RHS	field
Cray X1	1.2	-	3.7	3.1
IBM p690 cluster	.26	-	.48	.18
IBM SP	.13	-	.22	.15
SGI Altix	.46	-	.75	.39
TeraGrid	-	-	-	-

Table 2: Computation rates for user-defined events on process 0

Figure 10 contains graphs of GFlops per second per processor using the total floating point operation count for B1-std and B3-gtc as measured on 16 and 64 processors, respectively. This is simply a normalized version of efficiency that allows direct performance comparisons between the platforms. The one thing to note here is that the relatively poor computational rate from using the FFT-based method belies the fact that this significantly decreased the runtime compared to the direct method, even though the direct method achieves significantly higher computational rates. Use of HPM to determine computational complexity allowed us to quantify the realized performance. From this we determined that while load imbalances exist and the complexity is growing slowly as function of processor count, it is unlikely that either is limiting the performance scalability. We also observed that the computational rate of the FFT-based evaluation method is somewhat low, and that `Coll` performs poorly on the Cray X1. Both are candidates for additional performance optimization. We also note that the computation rate achieved on the Altix is extremely good, which makes it even more important to identify the source of the communication problem on that system.

Again, the use of the embedded timers and the hardware counter data collected via HPM instrumentation led to progress in analyzing GYRO performance and identifying opportunities for continued optimization. And, again, the process is very time and resource consuming, requiring numerous expensive runs.

6 Rest of the Paper

The final version of the paper will also use user and communication event trace data to further augment the performance analysis. These data will be used to confirm the developer's view of the flow of execution, examining variability between timesteps and between processes. We will also determine the type, number, and data volume for each communication event in each user event, and use this and the embedded timer data to determine communication rates. The trace data is then used to estimate the degree and source of load imbalance. Visualizations are used selectively to verify the quantitive results. Finally, J. Candy will comment, as the developer, on the utility of the performance analysis achieved via this manual methodology.

7 Conclusions

When an application code or computer system provides a small number of compile- or runtime options that can influence performance, empirical studies are adequate for optimizing over the small search space. This strategy breaks down when the search space is large, because of the significant increase in required computer and human resources. This "black box" empirical approach is also problematic when the options are not obvious, such as using a different file system to improve I/O performance on the Cray X1. Common knowledge, as provided by the community of users or technical support staff, and embedded timers can lead the user to discover such performance issues. However, when taking the next step of determining whether performance can be further improved, and how, embedded timers often provide only hints. Also, the cost of collecting and extracting results from the embedded timer data grew significantly as the number of experiments increased.

Simply adding information on the computational complexity increased the utility of the timer data significantly. Collecting hardware performance counter data from actual runs was not free, requiring instrumentation of the code and duplication of many of the benchmark runs. Note that a source code analysis might be sufficient to determine the computational complexity for some applications, but GYRO's variability with respect to processor count would have made it difficult for a purely automatic analysis. Even empirical studies can become problematic when there are solution-dependent variations in complexity.

As will be shown in the final version of the paper, adding user and communication event trace data again improved our ability to interpret the timer data, for example, determining achieved communication rates. The cost of these experiments was even greater than that of collecting the hardware performance counter data, generating GBytes of performance data. The timing data can also be suspect as the instrumentation can perturb the performance. However, the ability to visualize the execution is still a powerful, if not necessarily quantitative, means for identifying and determining the impact of load imbalances and communication overhead.

Our approach to generating baseline data for the tool evaluation exercise is far from exhaustive, but it was exhausting. While users often complain that performance tools are complicated and require valuable time to learn to use effectively, manual approaches can also be extremely time consuming. Ultimately, the strategy employed here is equivalent to building a primitive performance database and writing scripts to "mine" the data, looking for insight. At the very least, there exist tools that can upload these performance data into a real database and provide mechanisms for more easily looking at the data in different ways. To also decrease the cost of collecting the performance data, more intelligent experimental design is needed. For example, this could be achieved by decreasing the number of experiments and the amount of performance data required to identify performance problems. One part of this might be a facility for determining which aspects of the performance are primarily platform independent, and which are more strongly tied to the performance characteristics of a particular system. Another part of this more intelligent design could also involve access to data that is more difficult to collect (or interpret) manually. These are all areas in which performance tools should be able to prove their worth. *In the final version of the paper we will briefly mention a number of performance tools that claim to address some of these issues.*

References

[1] S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci, *A portable programming interface for performance evaluation on modern processors*, International Journal of High Performance Computing Applications, 14 (2000), pp. 189–204.

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

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

[4] C. Estrada-Mila, J. Candy, and R. Waltz, *Gyrokinetic simulations of ion and impurity transport*, Phys. Plasmas, 12 (2005), p. 022305.

[5] G. A. Geist, M. T. Heath, B. W. Peyton, and P. H. Worley, *A users' guide to PICL: a portable instrumented communication library*, Tech. Rep. ORNL/TM-11616, Oak Ridge National Laboratory, Oak Ridge, TN, August 1990.

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

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

[8] J. Kinsey, R. Waltz, and J. Candy, *Nonlinear gyrokinetic turbulence simulations of $E \times B$ shear quenching of transport*, Submitted to Phys. Plasmas.

[9] 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.

[10] P. H. Worley, *MPICL*. http://www.csm.ornl.gov/picl/.