

# Performance Analysis of GYRO

## *A Tool Evaluation*

P Worley, P Roth

J Candy

H Shan

G Mahinthakumar, S Sreepathi

L Carrington, T Kaiser, A Snavely

D Reed, Y Zhang

K Huck, A Malony, S Shende

S Moore, F Wolf

Oak Ridge National Laboratory

General Atomics

Lawrence Berkeley National Laboratory

North Carolina State University

San Diego Supercomputing Center

University of North Carolina

University of Oregon

University of Tennessee

2005 SciDAC Conference

June 26-30, 2005

San Francisco, California

# Acknowledgements

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-Battelle, 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.

# Problem

- Performance analysis and optimization are not trivial, and are only getting harder as application codes become more complex, as the problem sizes and number of processors used increase, as processor, memory, and network technologies evolve, etc.
- Performance tools have a reputation for being difficult to learn to use, especially for the infrequent user. However, this must be compared with the difficulty and limitations of the alternative: manual methods.
- This study looks at what performance analyses can be performed without modern tools, and at what cost. We then briefly describe a number of tools that address some of the deficiencies of the manual approach.

# Approach

The performance of the Eulerian gyrokinetic-Maxwell solver code GYRO was examined on five high performance computing systems:

- 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 and 4 scalar units running at 400 MHz.
- 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 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. Each processor is a 1.5 GHz Intel Itanium 2. The Altix is a Non-Uniform Memory Access (NUMA) cache coherent shared memory system.
- 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.

using the Waltz standard case benchmark, 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. The benchmark is run for 500 timesteps.

# Experimental Design

First, a manual “baseline” approach was taken, using custom PERL and GNUPLOT scripts to analyze the output of:

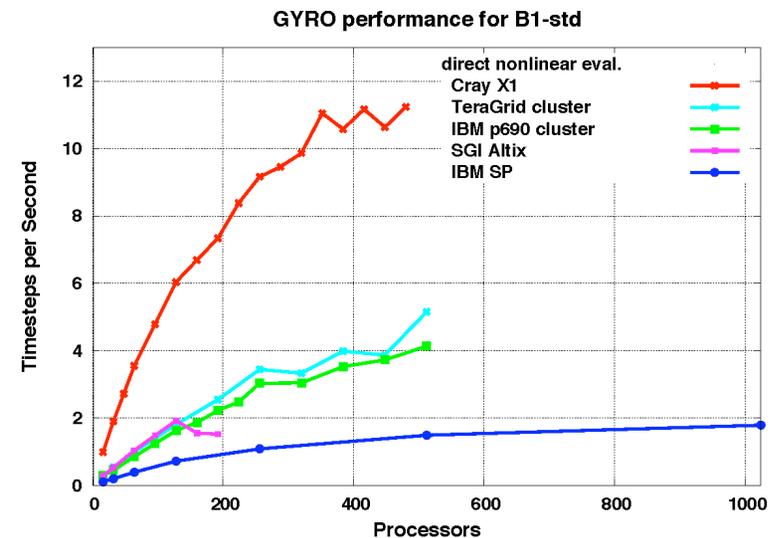
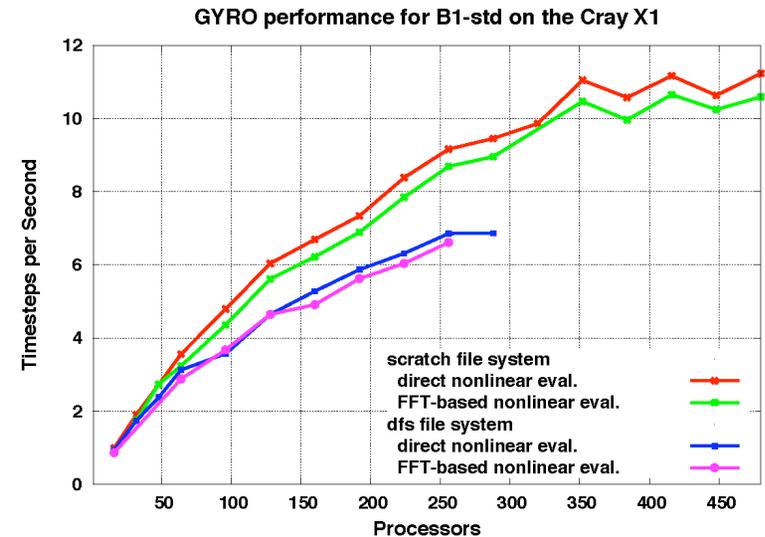
- **wallclock timers.** GYRO comes with embedded wallclock timers and both cumulative and sampled runtime data are collected automatically. The timers surround events that characterize the developers' view of the code.
- **floating point operation counts.** We instrumented the code with calls to HPMLIB `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.
- **event traces.** We instrumented the code with calls to the MPICL `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 were used to look for performance bottlenecks.

After the baseline studies were complete, we next analyzed the performance of GYRO using a number of tools developed by or used within the Performance Evaluation Research Center (PERC) project: PerfDMF, IPM, TAU, SvPablo, KOJAK, PMAc, identifying ways in which the tools simplified, accelerated or extended the manual approach.

# Analyses for which tools are not needed

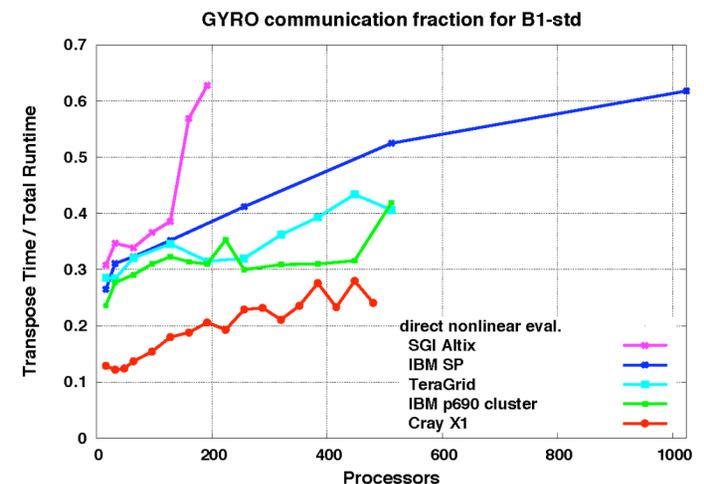
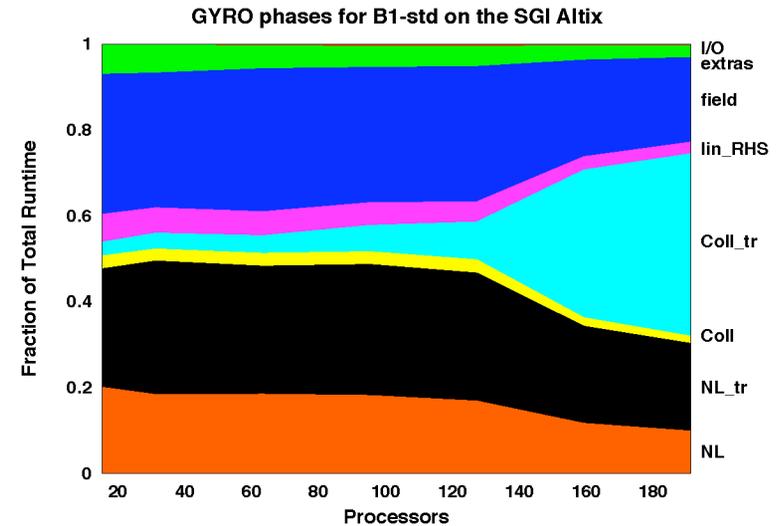
- **Optimization over small search spaces.** Many codes have embedded tuning options that allow the algorithms or implementation to be modified at compile- or runtime. The optimal choice is often a function of the computer system, problem specification, or runtime configuration (e.g., number of processors). If the search space is small, it is simplest to determine the optimum by measuring the performance of each option directly. Example here is the choice of nonlinear evaluation method to use in GYRO and the choice of filesystem to run out of on the ORNL Cray X1.
- **Benchmarking.** Benchmark timings should represent what would be observed in a production run, i.e., without performance tools.

While both of these analyses require only whole program timings, to observe scaling behavior requires many runs. For these analyses over 175 experiments were run, on processor counts up to 1024, and the number of experiments on any 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.



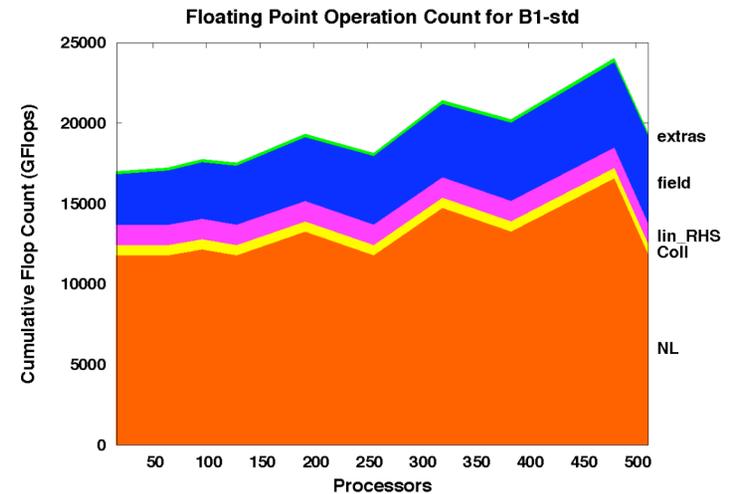
# Analyses for which tools are not necessary

- **User Event Profiling.** Profiling is a standard first step in performance analysis, to determine which events are most important to optimize. Examining scaling as well can help identify performance problems. From the benchmark experiments, Altix performance is not scaling as well as on the other platforms. A plot of the percentage of runtime per phase on the Altix indicates that the Coll\_tr phase (dominated by calls to MPI\_Alltoall) is the probable source of the poor scaling. As the timer data plotted here is for process 0 only, this could also denote a load imbalance. However, the other systems do not show similar catastrophic communication performance. Note that the embedded timers define the events that the developer expects to characterize performance. An incorrect choice can provide misleading information, and a good choice on one platform may not be a good choice on another.



# Analyses for which tools are not necessary

- Computational cost and rate metrics.** Operation counts can be used to understand computational complexity and to compute computation rates. Aggregating the floating point operation counts on the p690 cluster illustrates that the work required by the parallel implementation of GYRO is sensitive to the processor count, and that some scaling issues are unavoidable (when using this implementation). Using the operation counts to approximate computational rates indicates that rates on all systems are relatively insensitive to processor count (not shown here). The average rates are listed in the table. From these data, the Coll phase is a candidate for additional vectorization work on the X1, and the difference between computational rates on the Altix and TeraGrid, which use the same processor, need to be looked at more closely. Note that the operation counts were collected in separate experiments, and on only one platform, and combined with the timing data in a postprocessing step. While sufficient here, other analyses would require collecting operation counts on each system.



B1-std computation rates (GFlop/sec)

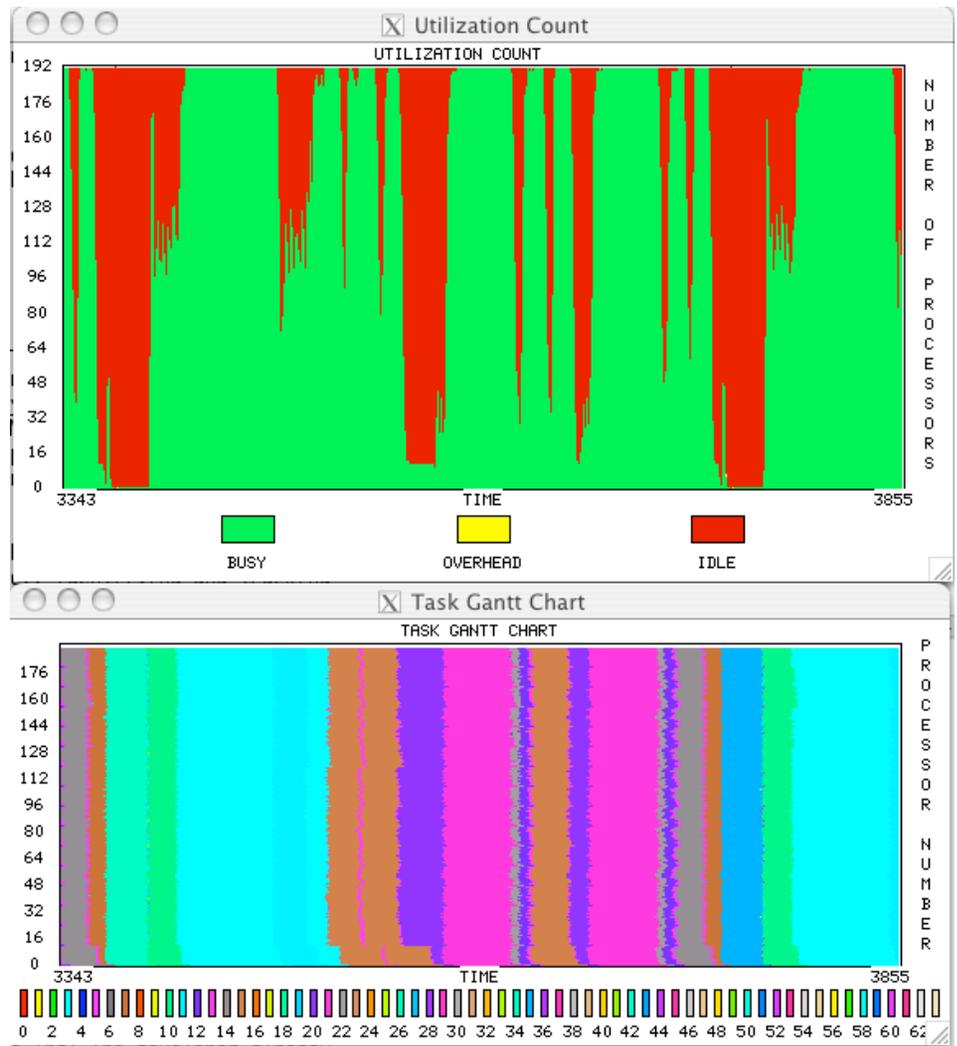
	NL	Coll	lin_RHS	field
Cray X1	4.7	.32	3.2	1.6
IBM p690 cluster	1.3	.34	.71	.35
IBM SP	.69	.20	.26	.14
SGI Altix	2.1	.68	.75	.37
TeraGrid	1.6	.47	.57	.37

Table 1: Computation rates for user-defined events on process 0

# Analyses for which tools are not necessary

- **Communication cost and rate metrics.** MPI command profiles and traces can be used to understand communication complexity and to compute communication rates. Example data are not shown here, but the advantages, costs, and drawbacks are similar to those for collecting data on operation counts, excepting that trace data is much higher volume. For example, in these (limited) studies we collected over 700MB of trace data (compressed).
- **Visualization.** MPI-aware performance visualization tools have been around for over 10 years. We include one of the original ones, ParaGraph, in the baseline studies to help identify whether more modern tools have improved on this basic functionality. The combination of the utilization graph and the task Gantt chart indicate that load imbalance contributes to some of the communication overhead, but is not the dominant source. These data are for 192 processors on the X1, so do not necessarily indicate anything about performance on, for example, the Altix.

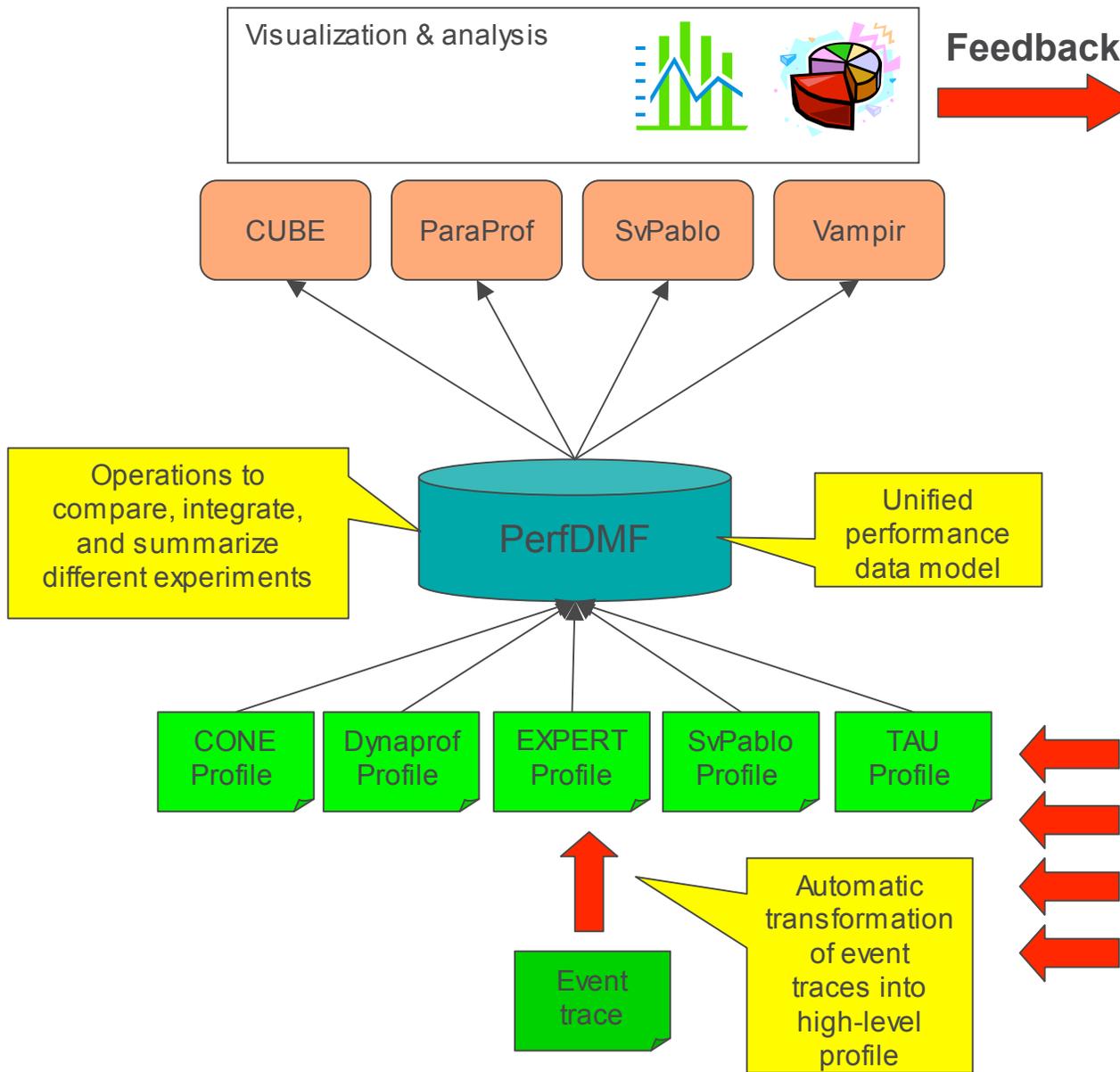
**It is in the collection, analysis, and presentation of multiple related measurements where manual methods become onerous, especially when requiring frequent additional experiments to fill “holes” in the experimental database.**



# Analyses for which tools are important

There are a number of analyses where the manual approach we took was either too expensive or was only able to approximate the analysis indirectly, including:

- **Identifying critical paths.** Critical paths indicate potential performance bottlenecks. Trace analyzers provide one approach to this, but are best linked to more than just timing data to provide context and guidance.
- **Global view analysis.** Global view analysis allows direct examination of load imbalances, system bottlenecks, and the impact of system noise. While user and MPI event visualizations are useful, they require the user to recognize and interpret the data correctly.
- **Detailed performance debugging.** Debugging is an iterative process of 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.



- Selection of the best version
  - Refinement of existing optimization strategies
  - Creation of new optimization strategies
  - Scalability analysis
- 

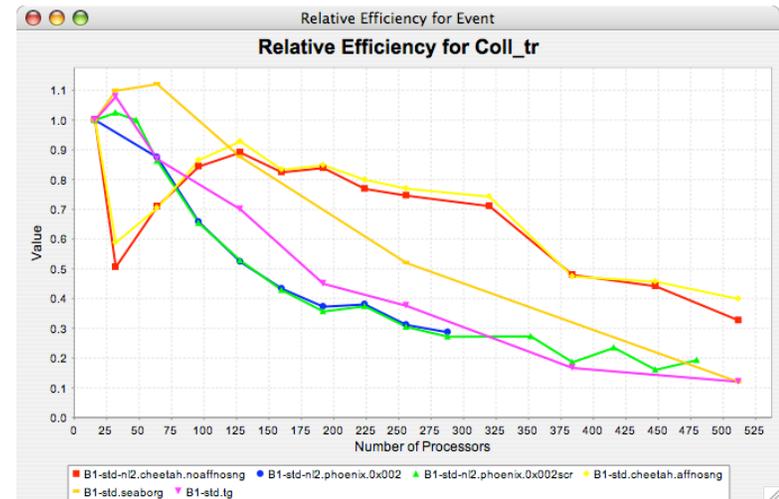
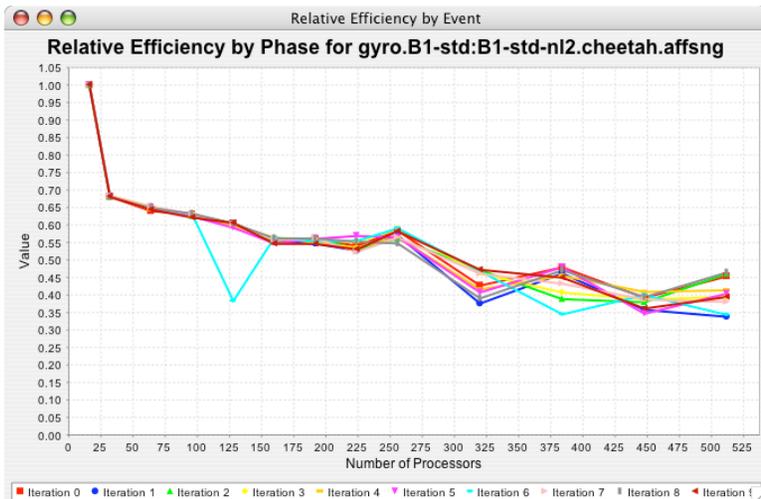
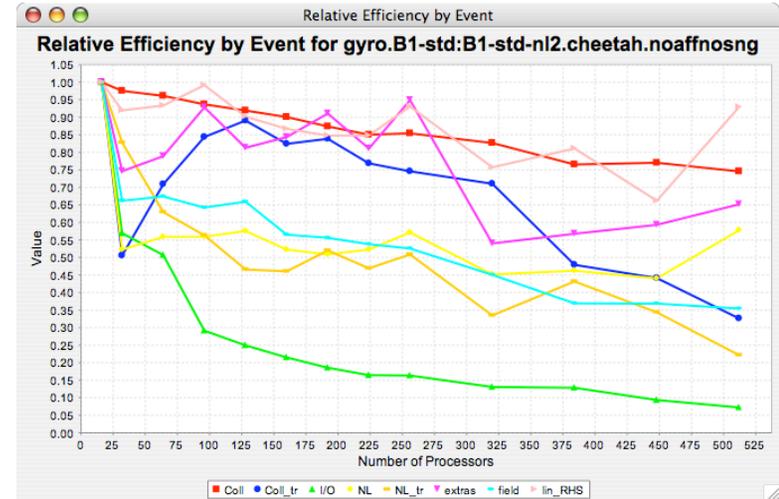
Multiple code versions

Automatic collection of performance data



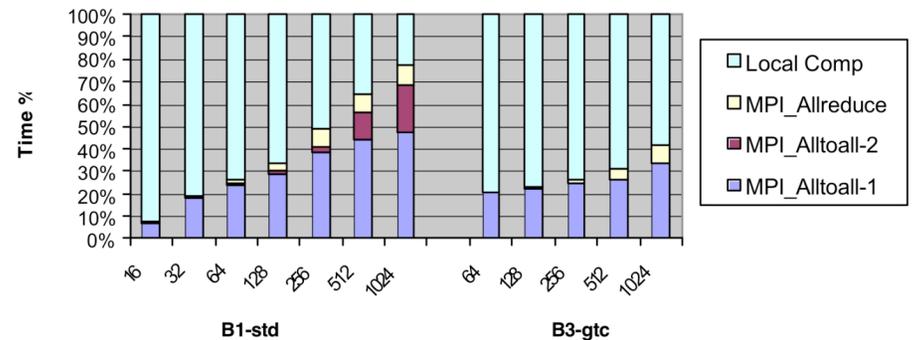
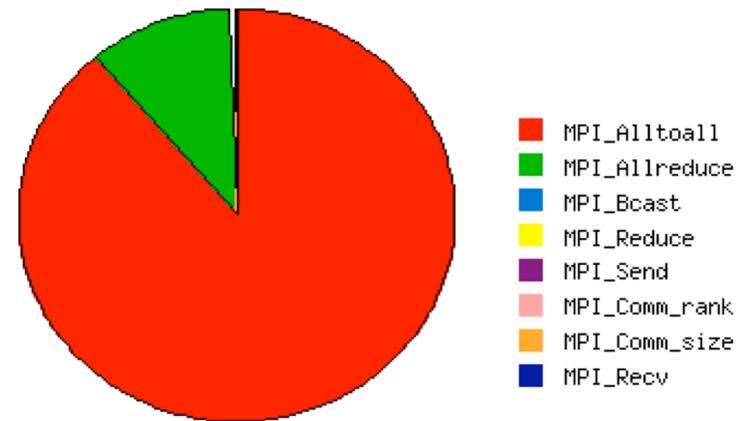
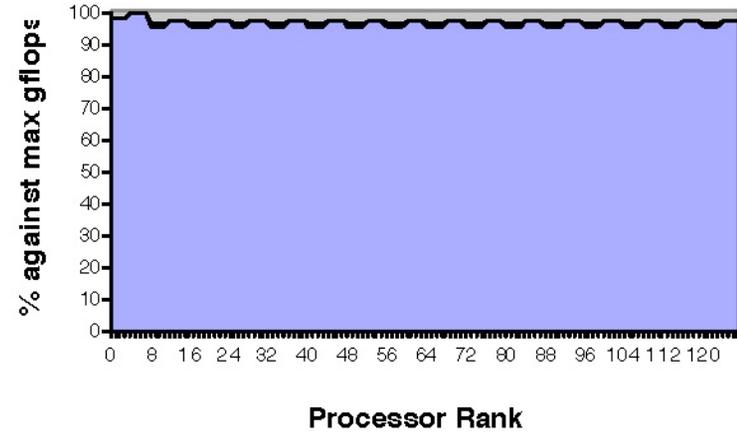
# PerfDMF

PerfDMF (Performance Data Management Framework) is a database schema and toolkit layered on top of an SQL database for organizing multi-experiment profile data. A data mining framework called PerfExplorer is also built on top of PerfDMF. Loading the manually collected baseline data into PerfDMF facilitates subsequent multi-experiment analyses. For example, the PerfExplorer user interface was used to quickly and easily generate all of the manually-generated plots described previously. It was also used to examine other performance issues, .e.g, (1) relative efficiency for different simulation timesteps, (2) relative efficiency for different user events, and (3) relative efficiency for a communication event for different platforms.



# IPM

IPM (Integrated Performance Monitor) 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. Running IPM does not require any source code modification unless the user wants to define special regions to monitor, in which case MPI\_Pcontrol is used to define the starting and ending points of the region. While this is the same requirement as the manual approaches described previously, IPM automatically collects multiple metrics and plots both raw and derived metrics. The examples here show (1) percentage of maximum computational rate as a function of processor, (2) percentage of communication time spent in each MPI command, and (3) percentage of total time spent in various phases as a function of processor count. While not designed for multi-experiment analyses, IPM quickly generates a number of common views of performance for a given run.



# SvPablo – a Graphical Performance Analysis Environment

Automatic instrumentation

Performance data correlating to source code

Easy bottleneck discovery

The screenshot displays the SvPablo interface with several key components:

- Source File:** A code editor showing C++ source code with line numbers 43 to 66. A callout window shows 'HW Statistics by Line' for line 51, indicating 329.79250 MFLOPS.
- Performance Metric Selection Dialog:** A dialog box with sections for 'Call Statistics', 'Loop Statistics', and 'HW Statistics by Line'. Each section has checkboxes for 'Count' and 'Inclusive Duration'. Under 'HW Statistics by Line', there are checkboxes for 'Floating Point Instructions', 'Cycles', and 'MFLOPS'.
- Detailed Performance Data:** A table titled 'Detailed Performance Data: HW Statistics by Line' with columns for Task Number, Count, Seconds, Exclusive Seconds, and FP Instructions. The data shows performance metrics for 15 tasks.
- Line Level Scalability Graph:** A line graph showing 'Efficiency' on the y-axis (0 to 1) versus 'Processor Count' on the x-axis (0 to 250). The efficiency starts at 1.0 for 1 processor and decreases as the number of processors increases, reaching approximately 0.2 at 250 processors.

Software & hardware statistics

Load imbalance detection

Line level scalability analysis

# SvPablo

SvPablo is a graphical environment for instrumenting application source code and browsing dynamic performance data. It is a sophisticated tool supporting many performance analyses that are difficult to do manually. For example, SvPablo can calculate computation and communication rates while collecting profile data, not requiring the merge and postprocessing of multiple experiments or of trace data. The examples here show the output of a performance debugging session, attempting to identify and characterize the performance of performance sensitive routines for further investigation, and analyses of multi-experiment results at a subroutine and at a loop level.

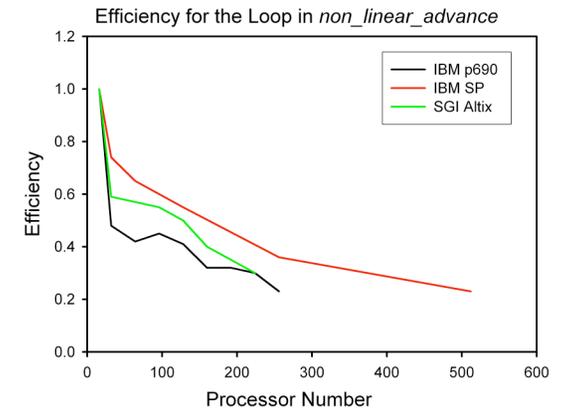
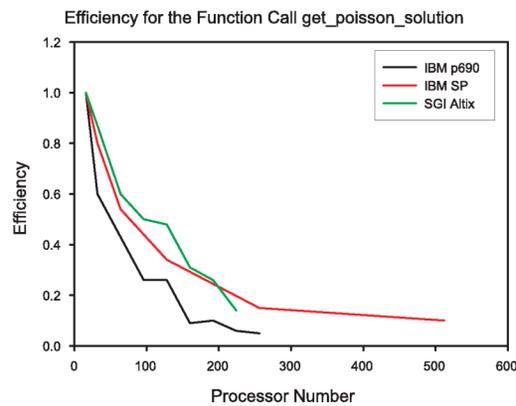
```

tg-login.ncsa.teragrid.org - Teragrid - SSH Secure Shell
File Edit View Window Help
Quick Connect Profiles

Reading Profile files in profile.*

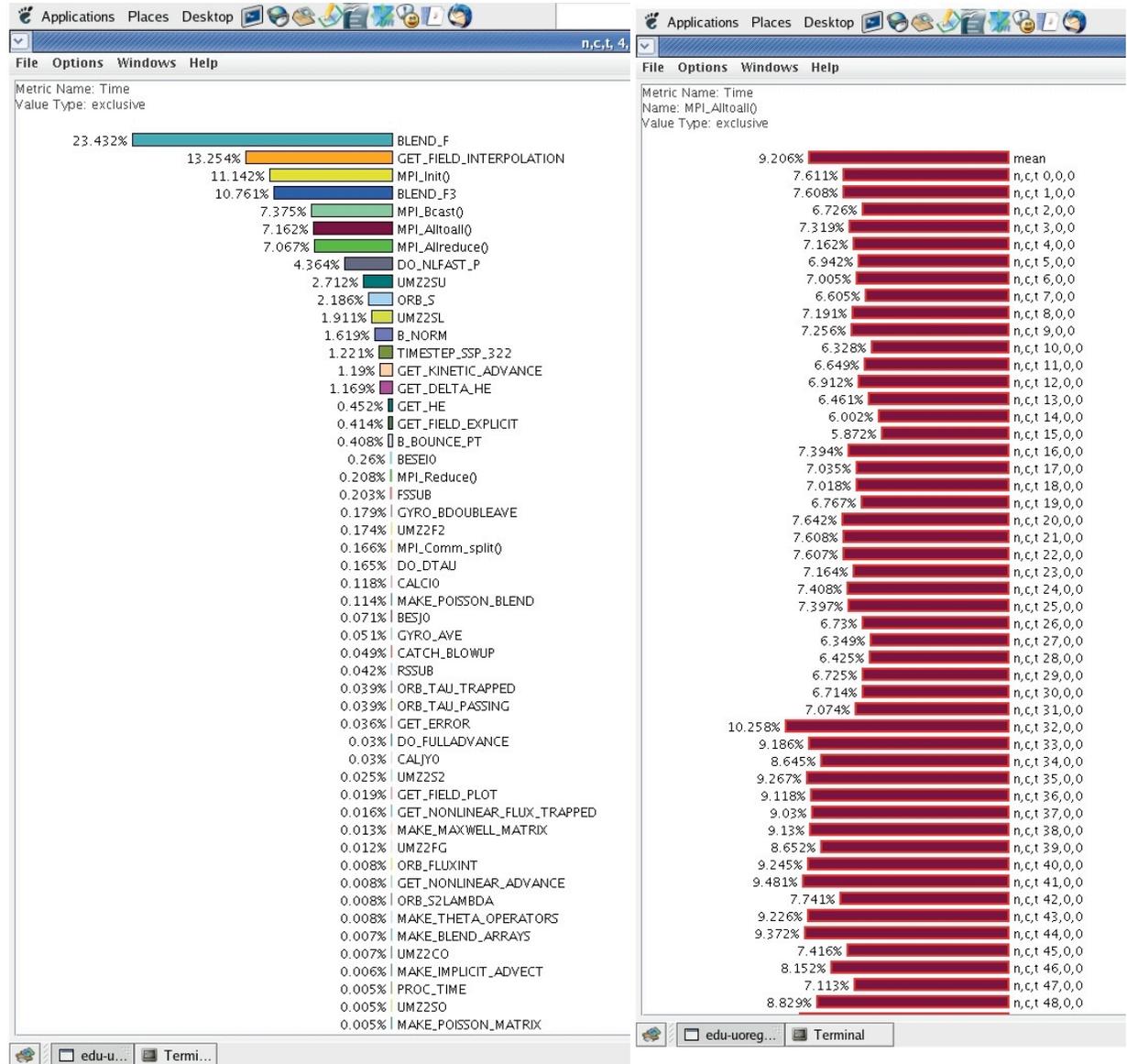
FUNCTION SUMMARY (total):
-----
$Time Exclusive Inclusive #Call #Subrs Count/Call Name
      counts total counts
-----
63.2 1.162E+13 1.162E+13 32000 0 363184642 DO_NLFASST_P
91.8 1.129E+12 1.687E+13 16000 192000 1054106938 TIMESTEP_SSP_322
5.6 1.038E+12 1.038E+12 64064 0 16206989 GET_FIELD_INTERPOLATION
14.5 9.634E+11 2.667E+12 48000 336000 55570775 GET_KINETIC_ADVANCE
3.9 7.083E+11 7.083E+11 16000 0 44265892 DO_PITCH_ANGLE_SCATTER
3.4 6.207E+11 6.207E+11 1.088E+06 0 570513 MPI_Alltoall()
2.8 5.161E+11 5.161E+11 48000 0 10752027 GET_DELTA_HE
1.7 3.183E+11 3.183E+11 48000 0 6630427 GET_HE
1.5 2.814E+11 2.814E+11 32000 0 8794971 DO_DTAU
3.0 2.557E+11 5.476E+11 16032 80160 34155209 GET_FIELD_EXPLICIT
1.2 2.231E+11 2.233E+11 64 9088 3488589420 MAKE_POISSON_BLEND
1.1 1.958E+11 2.012E+11 16000 16000 12575223 GET_ERROR
0.9 1.562E+11 1.563E+11 16000 16000 9768373 DO_COLLISION_CORRECT
0.5 9.829E+10 9.829E+10 2.19968E+06 0 44684 ORB_S
0.3 6.219E+10 6.219E+10 64064 0 970731 UMZ2SU
0.3 5.404E+10 5.404E+10 64064 0 843467 UMZ2SL
0.2 4.493E+10 4.493E+10 32 0 1404018210 MAKE_GYRO
0.1 2.035E+10 2.035E+10 120608 0 168761 MPI_Allreduce()
0.1 1.355E+10 1.356E+10 32 192 423615278 MAKE_IMPLICIT_ADVECT
0.1 1.203E+10 1.203E+10 140 312 85931994 UMZ2F2
0.0 6.917E+09 6.931E+09 416 832 16661282 GET_NONLINEAR_FLUX_TRAPPED
0.0 6.917E+09 6.936E+09 416 832 16679282 GET_NONLINEAR_FLUX_PASSING
97.7 5.767E+09 1.796E+13 16000 960320 1122361135 DO_FULLADVANCE
0.0 3.691E+09 3.691E+09 19904 0 185459 MPI_Bcast()
0.0 3.45E+09 3.485E+09 32 9024 108891947 MAKE_BLEND_ARRAYS
0.0 2.496E+09 2.496E+09 17920 0 139306 ORB_TAU_TRAPPED
0.0 2.452E+09 2.452E+09 17920 0 136824 ORB_TAU_PASSING
0.0 2.365E+09 2.439E+09 16000 96000 152446 RTRANSF_INIT
0.0 2.327E+09 2.732E+09 16000 96000 170720 RTRANSF_INIT
0.0 2.009E+09 7.17E+09 32 44800 224060447 MAKE_THETA_OPERATORS
0.0 1.09E+09 1.095E+09 32 64 34225359 WRITE_PREC
0.0 6.835E+08 6.835E+08 352 0 1941686 GET_FIELD_SPECTRUM
0.0 6.423E+08 1.221E+09 32 128 38171371 MAKE_INITIAL_H
0.0 5.485E+08 5.485E+08 32 0 17140940 MAKE_COLLISION_STENCIL

lines 1-41
    
```



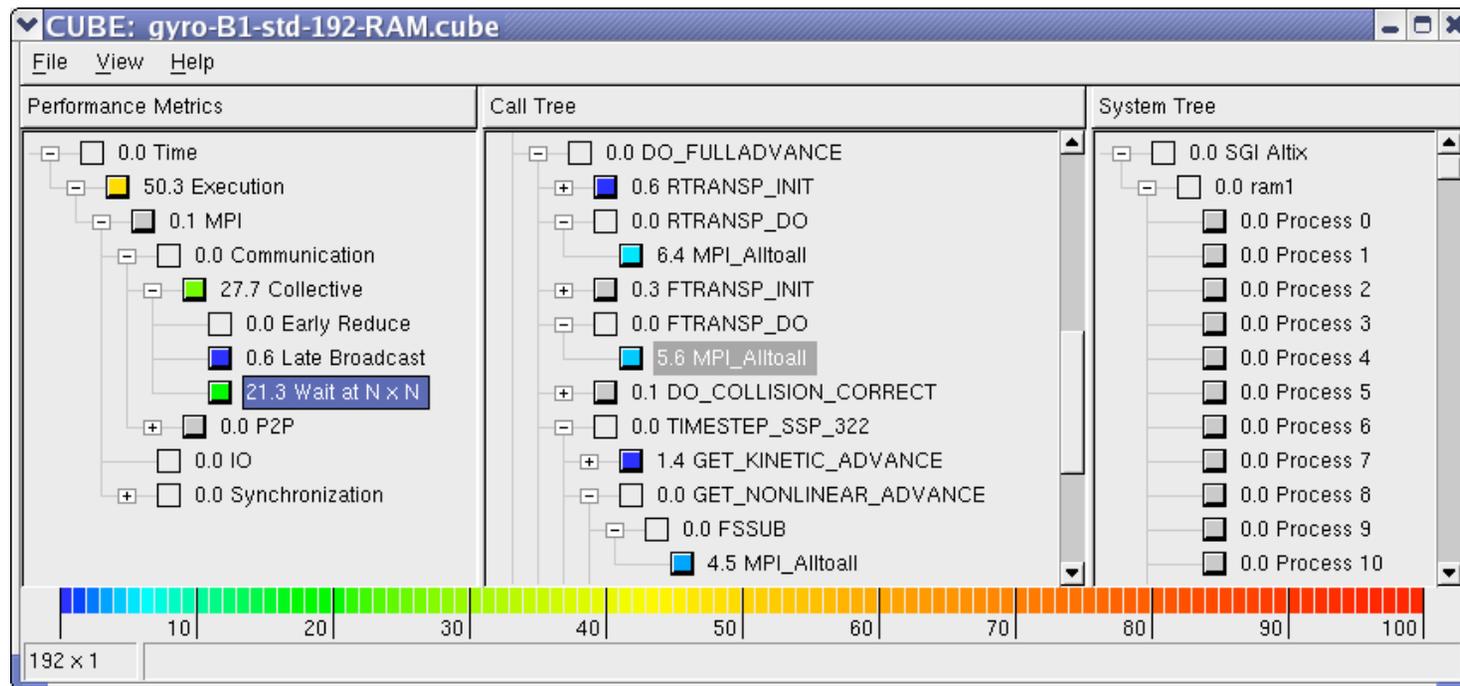
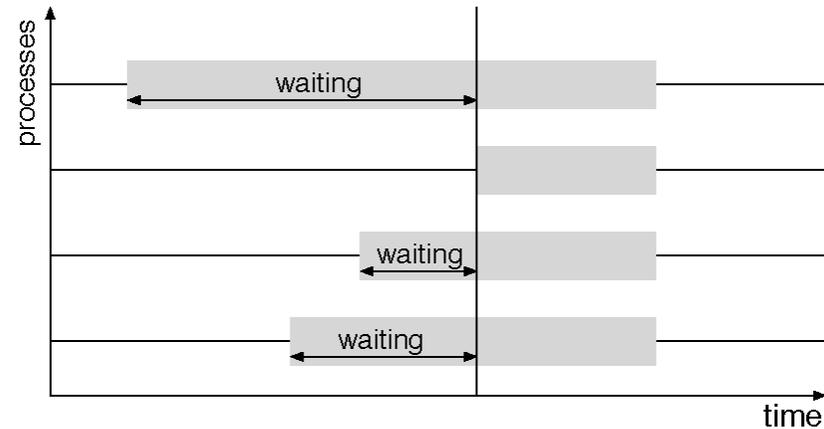
# TAU

TAU (Tuning and Analysis Utilities) is a framework and toolkit for performance instrumentation, measurement, and analysis of parallel applications. Like SvPablo, TAU is a sophisticated tool supporting many performance analyses that are difficult to do manually. PerfDMF is also a component of TAU, and TAU analysis routines can be used to analyze data collected with other tools. The first example here is another performance debugging application, identifying where time is spent in the code. The second example is a view of the imbalance in the time spent in the MPI\_Alltoall, representing either computational load imbalances or hotspots in the communication logic or network. While this latter task could be achieved by a manual analysis of a trace file, TAU supports many common analyses and a mechanism for defining and saving custom analyses.



# KOJAK

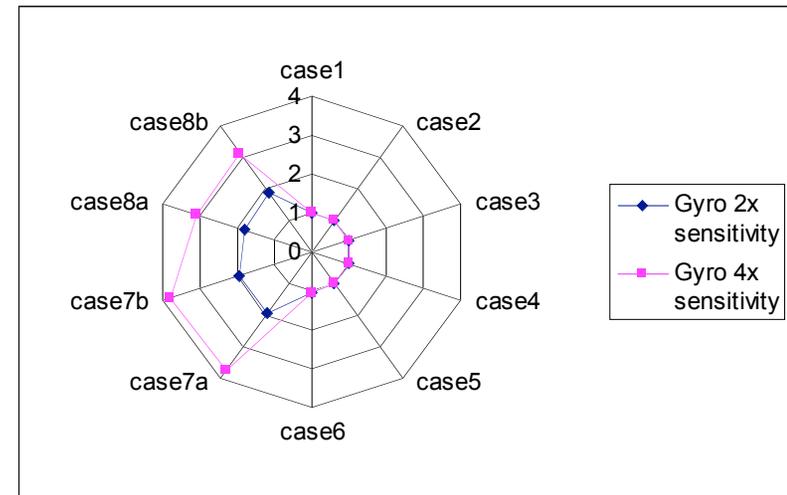
KOJAK is an automatic trace-analysis toolkit for parallel applications using MPI and/or OpenMP, generating event traces during execution and searching them offline for execution patterns indicating inefficient performance behavior. By comparing event traces for different runs, KOJAK identified particular 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.



# PMaC

PMaC (Performance Modeling and Characterization) 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. The performance questions mentioned previously were all concerned with understanding and optimizing current performance. Another class of questions include (1) estimating performance when changing the problem size, number of processors, or moving to a different system and (2) 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. Here PMaC is used to examine the performance impact of changing a number of different machine characteristics, relative to performance on 16 processors of the IBM p655+ cluster at NAVO. From these results the code is having difficulty staying within the mid-tier (L2) and outer-tier (L3) cache, as it greatly benefits from L3 and MM BW increases.

Scenario	Description
Case 1	Reduced interconnect latency by 2
Case 2	Increased interconnect bandwidth (BW) by 2
Case 3	Increased FLOP rate by 2
Case 4	Increased L1 BW by 2
Case 5	Increased L1 and L2 BWs by 2
Case 6	Increased L1, L2, and L3 BWs by 2
Case 7a	Increased L1, L2, L3, and main memory (MM) BWs by 2
Case 7b	Increased L1, L2, L3, MM, and on-node BWs by 2
Case 8a	Increased MM BW by 2
Case 8b	Increased MM and on-node BW by 2



# Conclusions

This 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. There is 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.

# References

- B1-std.** 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.
- GYRO.** J. Candy and R. Waltz, *An eulerian gyrokinetic-maxwell solver*, J. Comput. Phys., 186 (2003), p. 545.
- HPM.** <http://www.research.ibm.com/actc/projects/hardwareperf.shtml> .
- IPM.** <http://www.nersc.gov/nusers/resources/SP/ipm/> .
- KOJAK.** <http://icl.cs.utk.edu/kojak/> .
- MPICL.** <http://www.csm.ornl.gov/picl/> .
- ORNL Computer Systems.** <http://www.ccs.ornl.gov/> .
- NCSA Computer Systems.** <http://www.necsa.gov/> .
- NERSC Computer Systems.** <http://www.nersc.gov/> .
- ParaGraph.** M. T. Heath and J. A. Etheridge, *Visualizing the performance of parallel programs*, IEEE Software, 8 (1991), pp. 29-39.
- PERC.** <http://perc.nersc.org/> .
- PMaC.** <http://www.sdsc.edu/PMaC/> .
- SvPablo.** *Pablo Research Projects.* <http://www.renci.unc.edu/Project/ResearchProjects.htm>
- TAU.** <http://www.cs.uoregon/reseach/paracomp/tau/tautools/> .