Scalable, Automated Performance Analysis with TAU and PerfExplorer

Kevin A. Huck, Allen D. Malony, Sameer Shende and Alan Morris

Performance Research Laboratory
Computer and Information Science Department
University of Oregon, Eugene, OR, USA
E-mail: {*khuck, malony, sameer, amorris*}*@cs.uoregon.edu*

Scalable performance analysis is a challenge for parallel development tools. The potential size of data sets and the need to compare results from multiple experiments presents a challenge to manage and process the information, and to characterize the performance of parallel applications running on potentially hundreds of thousands of processor cores. In addition, many exploratory analysis processes represent potentially repeatable processes which can and should be automated. In this paper, we will discuss the current version of PerfExplorer, a performance analysis framework which provides dimension reduction, clustering and correlation analysis of individual trails of large dimensions, and can perform relative performance analysis between multiple application executions. PerfExplorer analysis processes can be captured in the form of Python scripts, automating what would otherwise be time-consuming tasks. We will give examples of large-scale analysis results, and discuss the future development of the framework, including the encoding and processing of expert performance rules, and the increasing use of performance metadata.

1 Introduction

Parallel applications running on high-end computer systems manifest a complexity of performance phenomena. Tools to observe parallel performance attempt to capture these phenomena in measurement datasets rich with information relating multiple performance metrics to execution dynamics and parameters specific to the application-system experiment. However, the potential size of datasets and the need to assimilate results from multiple experiments makes it a daunting challenge to not only process the information, but discover and understand performance insights. In order to perform analysis on these large collections of performance experiment data, we developed PerfExplorer[1], a framework for parallel performance data mining and knowledge discovery. The framework architecture enables the development and integration of data mining operations that can be applied to large-scale parallel performance profiles. PerfExplorer is built on a performance data management framework called PerfDMF[2], which provides a library to access the parallel profiles and save analysis results in a relational database.

A performance data mining framework should support both advanced analysis techniques as well as extensible *meta analysis* of performance results. The use of *process control* for analysis scripting, *persistence* and *provenance* mechanisms for retaining analysis results and history, *metadata* for encoding experiment context, and support for *reasoning* about relationships between performance characteristics and behavior all are important for productive performance analytics. However, the framework must also be concerned about how to interface with application developers in the performance discovery process. The ability to engage in process programming, knowledge engineering (metadata and inference rules), and results management opens the framework toolset for creating data mining environments specific to the developer's concerns.

We have re-engineered our integrated framework for performing meta analysis to incorporate parallel performance data, performance context metadata, expert knowledge, and intermediate analysis results. New methods were needed for correlating context metadata with the performance data and the analysis results, in order to provide the capability to generate desired empirical performance results from accurate suggestions on how to improve performance. Constructing this framework also required methods for encoding expert knowledge to be included in the analysis of performance data from parametric experiments. Knowledge about subjects such as hardware configurations, libraries, components, input data, algorithmic choices, runtime configurations, compiler choices, and code changes will augment direct performance measurements to make additional analyses possible.

The remainder of the paper is as follows. We discuss our analysis approach for the framework and our prototype implementation in Sec. 2. We will present some recent analysis examples which demonstrate some new PerfExplorer features in Sec. 3 and present future work and concluding remarks in Sec. 4.

2 PerfExplorer Design

PerfExplorer[2] is a Java application developed for performing data mining analyses on multi-experiment parallel performance profiles. Its capabilities include general statistical analysis of performance data, dimension reduction, clustering, and correlation of performance data, and multi-experiment data query and management. These functions were provided, in part, by the existing analysis toolkits (R[3] and Weka[4]), and our profile database system PerfDMF[2].

While PerfExplorer is a step forward in the ability to automatically process complex statistical functions on large amounts of multi-dimensional parallel performance data, its functionality was limited in two respects. First, the tool only allows a user to select single analysis operations via a graphical user interface. Multi-step analysis processes are not possible. Second, PerfExplorer only provides new descriptions of the data – it does not *explain* the performance characteristics or behavior observed (i.e., meta analysis). Scripting and support for retaining intermediate results can help to address the first shortcoming. The second is more challenging.

For example, an analyst can determine that on average, application X spent 30% of its total execution time in function `foo()`, and that when the number of processors is increased, the percentage of time may go up or down, and so on. However, PerfExplorer did not have the capability to explain *why* the change happened. The explanation may be as simple as the fact that the input problem doubled in size, but without that contextual knowledge, no analysis tool could be expected to come to any conclusions about the cause of the performance change without resulting to speculation.

As we discussed our enhancements to PerfExplorer, we will consider two analysis cases: 1) we have collected parallel performance data from multiple experiments, and we wish to compare their performance, or 2) we have collected performance data from one experiment, and would like to compare the performance between processes or threads of execution. Like other tools, PerfExplorer can provide the means for an analyst to determine which execution is the "best" and which is the "worst", and can even help the analyst investigate further into which regions of code are most affected, and due to which met-

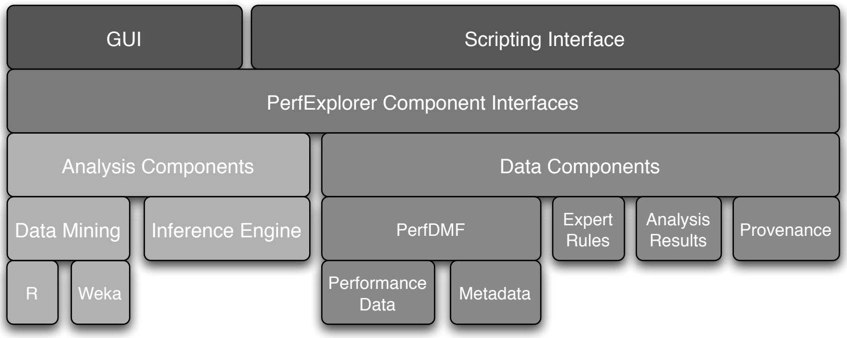


Figure 1. The redesigned PerfExplorer components.

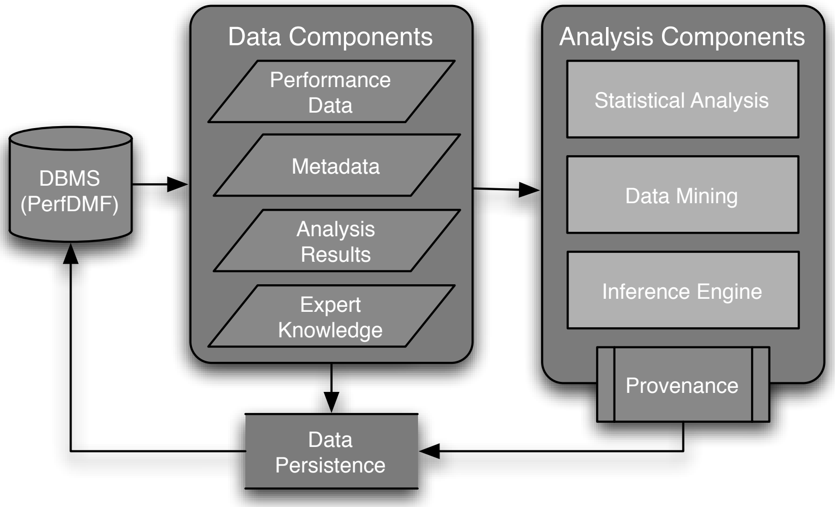


Figure 2. PerfExplorer components and their interactions.

rics. However, there is no explicit *process control*, nor is there *higher-level* reasoning or analysis of the performance result to explain what caused the performance differences. In addition, process control is required in order to perform repeated analysis procedures or non-interactive analysis automation. In order to perform this type of meta-analysis, several components are necessary to meet the desired goals.

Figure 1 shows the PerfExplorer components, and Figure 2 shows the interaction between components in the new PerfExplorer design. The performance data and accompanying metadata are stored in the PerfDMF database. Performance data is used as input for statistical analysis and data mining operations, as was the case in the original version of PerfExplorer. The new design adds the ability to make all intermediate analysis data persistent, not just the final summarization. Expert knowledge is incorporated into the analysis, and these new inputs allow for higher-level analysis. An inference engine is also added to combine the performance data, analysis results, expert knowledge and execution metadata into a performance characterization. The provenance of the analysis result is stored with the result, along with all intermediary data, using object persistence. The whole process is contained within a process control framework, which provides user control over the performance characterization process.

2.1 Process Control

One of the key aspects of the new PerfExplorer design is the requirement for process control. While user interfaces and data visualization are useful for interactive data exploration, the user will need the ability to control the analysis process as a discrete set of operations. In order to synthesize analysis results, expert knowledge and metadata into higher-level meta-analysis process, PerfExplorer needs an extension mechanism for creating higher-order analysis procedures. One way of doing this is through a scripting interface, such as Jython[a]. With such an interface, the analysis process is under the control of the analyst, who is able to reliably produce the characterization with minimal effort. This new scripting support was used to generate the results in Sec. 3.2.

2.2 Collecting and Integrating Metadata

Performance instrumentation and measurement tools such as TAU[5] collect context metadata along with the application performance data. This metadata potentially contains information relating to useful analysis input such as the build environment, runtime environment, configuration settings, input and output data, and hardware configuration. Metadata examples which are automatically collected by the profiling provided by TAU include fields such as processor speed, node hostname, and cache size. It should be easy to see how fields such as *CPU MHz*, *Cache Size* or *Memory Size* would be useful in explaining the differences between executions. By integrating these fields into the analysis process, the analysis framework can reason about potential causes for performance failures. This new metadata support was used to generate the results in Sec. 3.1.

2.3 Inference Engine

Because the needs of the meta-analysis are dynamic, why should a performance analysis tool hard-code the complex rules that guide the decision making process of a performance analyst? Is it even possible to translate the subtleties of analysis reasoning to source code? In order to provide the type of higher-level reasoning and meta-analysis we require in our design, we have integrated a JSR-94[b] compliant rule engine, JBoss Rules[c]. The strategic selection of an inference engine and processing rules allows another method of flexible control of the process, and also provides the possibility of developing a domain specific language to express the analysis.

2.4 Provenance and Data Persistence

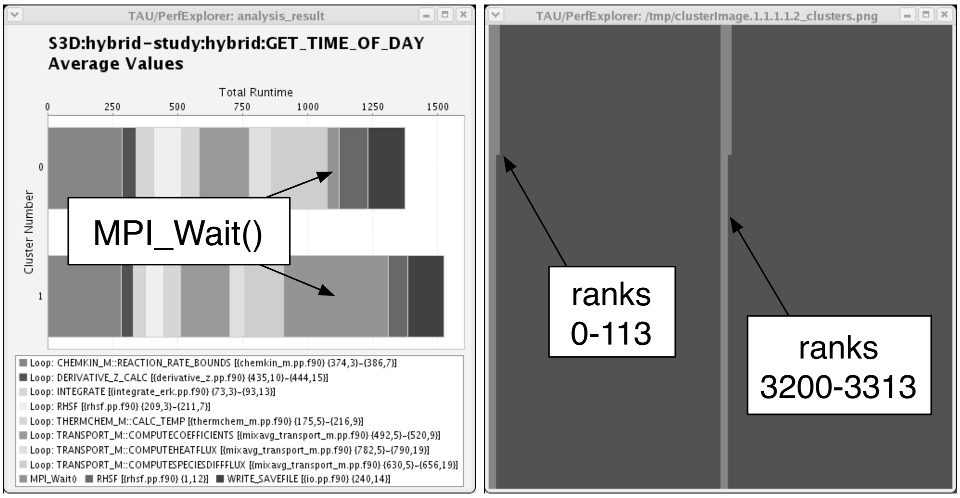
In order to rationalize analysis decisions, any explanation needs to include the data provenance, or the full chain of evidence and handling from raw data to synthesized analysis result. The new design will include the ability to make all intermediate analysis data persistent, not just the final summarization. The provenance of the analysis result is stored with the results and all intermediary data, using object persistence[d]. Any scientific endeavor is

[a] Jython: `http://www.jython.org/`

[b] Java Rule Engine API: `http://jcp.org/en/jsr/detail?id=94`

[c] JBoss Rules: `http://www.jboss.com/products/rules`

[d] Relational Persistence for Java and .NET: `http://www.hibernate.org/`



TAU/PerfExplorer: analysis_result
S3D:hybrid-study:hybrid:GET_TIME_OF_DAY
Average Values
Total Runtime
0
250
500
750
1000
1250
1500
Cluster Number
0
1
MPI_Wait()
Loop: CHEMKIN_M::REACTION_RATE_BOUNDS [{chemkin_m.pp.f90} {374,3}-{386,7}]
Loop: DERIVATIVE_Z_CALC [{derivative_z.pp.f90} {435,10}-{444,15}]
Loop: INTEGRATE [{integrate_erk.pp.f90} {73,3}-{93,13}]
Loop: RHSF [{rhsf.pp.f90} {209,3}-{211,7}]
Loop: THERMCHEM_M::CALC_TEMP [{thermchem_m.pp.f90} {175,5}-{216,9}]
Loop: TRANSPORT_M::COMPUTECOEFFICIENTS [{mixavg_transport_m.pp.f90} {492,5}-{520,9}]
Loop: TRANSPORT_M::COMPUTEHEATFLUX [{mixavg_transport_m.pp.f90} {782,5}-{790,19}]
Loop: TRANSPORT_M::COMPUTESPECIESDIFFFLUX [{mixavg_transport_m.pp.f90} {630,5}-{656,19}]
MPI_Wait()
RHSF [{rhsf.pp.f90} {1,12}]
WRITE_SAVEFILE [{io.pp.f90} {240,14}]
TAU/PerfExplorer: /tmp/clusterImage.1.1.1.1.2_clusters.png
ranks 0-113
ranks 3200-3313

Figure 3. S3D cluster analysis. The figure on the left shows the difference in (averaged mean) execution behavior between the two clusters of processes. The figure on the right shows a virtual topology of the MPI processes, showing the locations of the clustered processes. The lighter processes ran on XT3 nodes, and the darker processes ran on XT4 nodes.

considered to be of “good provenance” when it is adequately documented in order to allow reproducibility. For parallel performance analysis, this includes all raw data, analysis methods and parameters, intermediate results, and inferred conclusions.

3 Analysis Examples

3.1 S3D

S3D[6] is a multi-institution collaborative effort with the goal of creating a terascale parallel implementation of a turbulent reacting flow solver. S3D uses direct numerical simulation (DNS) to model combustion science which produces high-fidelity observations of the micro-physics found in turbulent reacting flows as well as the reduced model descriptions needed in macro-scale simulations of engineering-level systems. The examples described here were run on the hybrid Cray XT3/XT4 system at Oak Ridge National Laboratory (ORNL).

During scalability tests of S3D with TAU, it was observed that as the number of processors exceeded 1728, the amount of time spent in communication began to grow significantly, and `MPI_Wait()` in particular composed a significant portion of the overall run time (approximately 20%). By clustering the performance data in PerfExplorer, it was then observed that there were two natural clusters in the data. The first cluster consisted of a majority of the processes, and these nodes spent less time in main computation loops, but a long time in `MPI_Wait()`. The other cluster of processes spent slightly more time in main computation loops, and far less time in `MPI_Wait()`.

By adding Cray-specific `TAU_METADATA()` instrumentation calls, we were able to determine the names of the nodes on which the processes ran. In the case of a 6400 process run, as shown in Figure 3, there were again two clusters, with 228 processes in one cluster having very low `MPI_Wait()` times (about 40 seconds), and the remainder of the processes in one cluster having very high MPI_Wait() times (over 400 seconds). The metadata collected, manually correlated with information about the hardware characteristics of each node, identified the slower nodes as XT3 nodes, and the faster nodes as XT4 nodes. There are two primary differences between the XT3 and XT4 partitions. The XT3 nodes have slower DDR-400 memory (5986 MB/s) than the XT4 nodes' DDR2-667 memory (7147 MB/s), and the XT3 partition has a slower interconnection network (1109 MB/s v. 2022 MB/s). Because the application is memory intensive, the slower memory modules have a greater effect on the overall runtime, causing the XT3 nodes to take longer to process, and subsequently causing the XT4 nodes to spend more time in MPI_Wait().

Running S3D on an XT4-only configuration yielded roughly a 12% time to solution reduction over the hybrid configuration, primarily by reducing MPI_Wait() times from an average of 390 seconds down to 104 seconds. If this application is to be run on a heterogeneous configuration of this machine or any other, load balancing should be integrated which takes into consideration the computational capacity of each respective processor. The use of metadata will also be important for this optimization.

3.2 GTC

The Gyrokinetic Toroidal Code (GTC) is a particle-in-cell physics simulation which has the primary goal of modeling the turbulence between particles in the high energy plasma of a fusion reactor. Scalability studies of the original large-scale parallel implementation of GTC (there are now a small number of parallel implementations, as the development has fragmented) show that the application scales very well - in fact, it scales at a better than linear rate. However, discussions with the application developers revealed that it had been observed that the application gradually slows down as it executes[7] - each successive iteration of the simulation takes more time that the previous iteration.

In order to measure this behavior, the application was auto-instrumented with TAU, and manual instrumentation was added to the main iteration loop to capture dynamic phase information. The application was executed on 64 processors of the Cray XT3/XT4 system at ORNL for 100 iterations, and the performance data was loaded into PerfExplorer. A PerfExplorer analysis script was constructed in order to examine the dynamic phases in the execution. The script was used to load the performance data, extract the dynamic phases from the profile, calculate derived metrics (i.e. cache hit ratios, FLOPS), calculate basic statistics for each phases, and graph the resulting data as a time series showing average, minimum and maximum values for each iteration.

As shown in Figure 4, over a 100 iteration simulation, each successive iteration takes slightly more time than the previous iteration. Over the course of the test simulation, the last iteration takes nearly one second longer than the first iteration. As a minor observation, every fourth iteration results in a significant increase in execution time. Hardware counters revealed that the L2 (and to a lesser extent, L1) cache hit-to-access ratio decreases from 0.92 to 0.86. Subsequently, the GFLOPS per processor rate decreases from 1.120 to 0.979. Further analysis of the two main routines in the iterations, CHARGEI and PUSHI, show that the decrease in execution is limited to these two routines. In the CHARGEI routine, each particle in a region of the physical subdomain applies a charge to up to four cells, and in the PUSHI routines, the particle locations are updated by the respective cells after the forces are calculated. The increase in time every fourth iteration was discovered to be due to a diagnostic call, which happens every ndiag iterations, an input parameter captured as metadata.

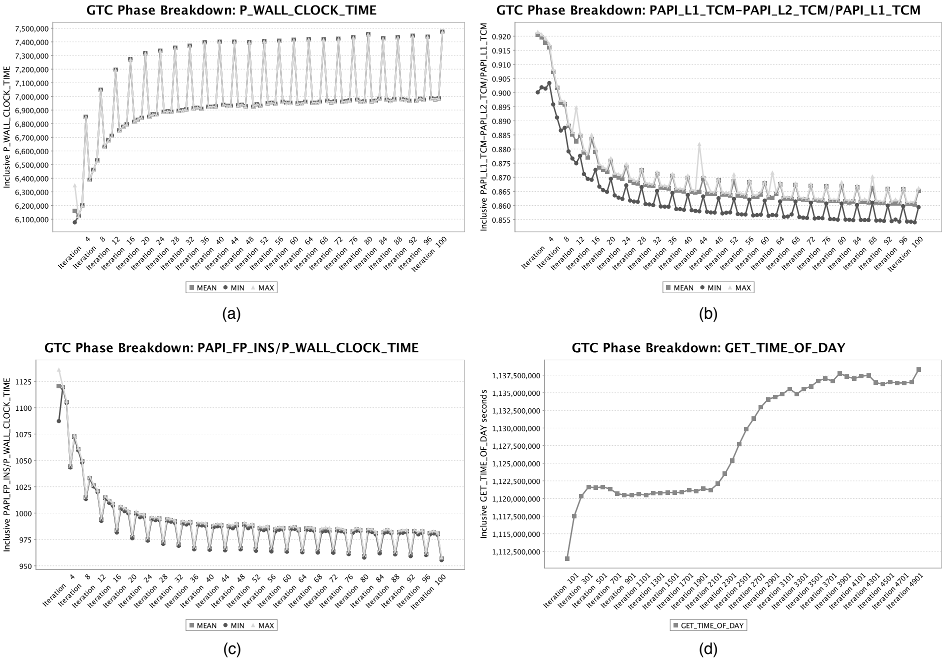


Figure 4. GTC phase analysis. (a) shows the increase in runtime for each successive iteration, over 100 iterations. (b) shows the decrease in L2 hit ratio, from 0.92 to 0.86, and (c) shows the decrease in GFLOPS from 1.120 to 0.979. (d) shows the larger trend when GTC is run for 5000 iterations, each data point representing an aggregation of 100 iterations.

Discussions with other performance analysis experts on the project revealed that the `CHARGEI` and `PUSHI` routines have good spatial locality when referencing the particles, however over time, they have poor temporal locality when referencing the grid cells. As a result, access to the grid cells becomes random over time. Further analysis is necessary to determine whether the expense of re-ordering the particles at the beginning of an iteration could be amortized over a number of iterations, and whether this added cost would yield a benefit in the execution time. While it appears that the performance degradation levels out after roughly 30 iterations, it should be pointed out that a full run of this simulation is at least 10,000 iterations, and as the 5,000 iteration execution shows in Figure 4 (d), the performance continues to degrade. Assuming a 10,000 iteration execution would take an estimated 20 hours to complete on the Cray XT3/XT4, potentially 2.5 hours of computation time per processor could be saved by improving the cache hit ratios. Further analysis is ongoing.

4 Future Work and Concluding Remarks

In this paper, we have discussed the new design of PerfExplorer, including components for scripting, metadata encoding, expert rules, provenance and data persistence. In our examples, we have discussed how features such as metadata encoding and scripting aid in the analysis process. However, we have significant work to do in extending the capabilities of PerfExplorer. While the rudimentary metadata support in PerfExplorer allows for some manual correlation between contextual information and performance results, sophisticated analysis rules to interpret the results with respect to the contextual information would aid us in our long term goal of a performance tool which would summarize performance results and link them back to the actual causes, which are essentially the context metadata relating to the application, platform, algorithm, and known related parallel performance problems. Encoding this knowledge in some form that our performance tool can use would be instrumental in developing new analysis techniques that capture more information about the experiment than simply the raw performance data. While full automation may prove difficult, we feel that a useful amount of automatic performance correlation is possible.

Acknowledgments

The research was funded by the Department of Energy, Office of Science under grant DE-FG02-05ER25680 and DE-FG02-07ER25826, and the National Science Foundation, High-End Computing program under grant NSF CCF 0444475. The authors would like to thank PERI, SciDAC, ORNL, NERSC and RENCI for including us in the PERI SciDAC project, and a special thanks to John Mellor-Crummey for a better understanding of the locality issues in GTC.

References

1. Kevin A. Huck and Allen D. Malony. PerfExplorer: A Performance Data Mining Framework for Large-Scale Parallel Computing. In *Conference on High Performance Networking and Computing (SC'05)*, Washington, DC, USA, 2005. IEEE Computer Society.
2. K. Huck, A. Malony, R. Bell, and A. Morris. Design and Implementation of a Parallel Performance Data Management Framework. In *Proceedings of the International Conference on Parallel Computing, 2005 (ICPP2005)*, pages 473–482, 2005.
3. The R Foundation for Statistical Computing. R Project for Statistical Computing. http://www.r-project.org, 2007.
4. Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, San Francisco, 2nd edition, 2005. http://www.cs.waikato.ac.nz/ ml/weka/.
5. Sameer Shende and Allen D. Malony. The TAU Parallel Performance System. *The International Journal of High Performance Computing Applications*, 20(2):287–331, Summer 2006.
6. J. H. Chen, A. Choudhary, B. de Supinski, M. DeVries, E. R. Hawkes, S. Klasky, W. K. Liao, K. L. Ma, J. Mellor-Crummy, N. Podhorski, R. Sankaran, S. Shende, and C. S. Yoo. Terascale Direct Numerical Simulations of Turbulent Combustion Using S3D. (to appear) *Institute of Physics (IOP) Journal*, 2007.
7. Nathan Wichmann, Mark Adams, and Stephane Ethier. New Advances in the Gyrokinetic Toroidal Code and Their Impact on Performance on the Cray XT Series. In *Cray Users Group*, 2007.