Performance Forensics: Knowledge Support for Parallel Performance Data Mining

Kevin A. Huck and Allen D. Malony

Performance Research Laboratory
Department of Computer and Information Science
University of Oregon, Eugene, OR, USA
{khuck,malony}@cs.uoregon.edu
http://www.cs.uoregon.edu/research/tau

Abstract. Many parallel performance tools provide the ability to summarize and report statistical data about an application's performance on a given platform, and/or report the performance differences between two executions of a particular application. However, few tools provide an explanation of performance results in a way that includes performance data results and the integration of the experiment context. The tools that do provide some explanation of performance results do so in a confined way which only identify known problems with particular symptoms, and do not provide the ability to perform general purpose performance engineering. In this paper, we describe our design for an analysis framework to integrate context metadata and performance assumptions, considered *expert knowledge* about an experiment into the analysis process, present early results, and describe our plans for further exploration of this analysis space.

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. The application is integrated with existing analysis toolkits (R[3], Weka[4]), and provides for extensions using those toolkits.

Like many performance tools, PerfExplorer is capable of generating and displaying summary statistics for parallel performance data. However, performance tools of the future need to go beyond the display and/or visualization of summary statistics and provide in-depth analysis and explanation of performance results to the user. There is *context metadata* relating to the application, platform, algorithm, and related parallel performance problems that would be helpful in the analysis process. Encoding this knowledge in some form that a 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.

In many parallel scientific applications, intimate knowledge of the computational semantics and systems environment is necessary to correctly reason and make conclusions about the performance data. In order to develop intelligent analysis heuristics, this *expert knowledge* has to be encoded in a form that an analysis expert or engine can apply to the problem in order to identify correlations, locate causality, and otherwise make conclusions about application performance. Certainly, there are known relationships between events in the application and the events themselves carry logical semantics. Unfortunately, these relationships and semantics are not currently conveyed in any form to the performance tools. This makes it difficult to develop performance characterization support.

In addition to the integration of context metadata and expert knowledge, a successful performance analysis framework requires an extensible control mechanism beyond interactive commands through a user interface. Successful *meta analysis* of performance results requires programmable *process control*, *persistence* of analysis results as well as the *provenance* of the results, and finally, requires some mechanism for *reasoning* about the large amount of input data and the potential conclusions which lead to the performance characterization.

We are currently developing an integrated framework for performing meta-analysis using parallel performance data, performance context metadata, expert knowledge, and intermediate analysis results. New methods are 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 more accurate suggestions on how to improve performance.

Constructing this framework also requires 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.

While many successful performance tools are distributed with some type of open-source license, very few application developers will have the desire to make modifications to low-level analysis code in a performance tool with which they have little experience. This an unreasonable expectation, and there exists a need to control the analysis, while providing proven analysis heuristics. For

this reason, we see the need for process control to provide explicit control of the analysis process, and an inference mechanism to provide reasoning about performance results and the potential causes.

The remainder of the paper is as follows. We discuss our analysis approach for the knowledge-supported framework in §2. We discuss our prototype design in §2. We will present our initial findings with this prototype in §3 and present related work in §4. Conclusions and future work are discussed in §5.

2 Design Approach

As mentioned in §1, PerfExplorer[2] is a graphical user interface based Java application for performing data mining analyses on multi-experiment parallel performance profiles. It's capabilities include general statistical analysis of performance data, dimension reduction, clustering, and correlation of performance data, and multi-experiment data query and management.

While PerfExplorer is a step forward in the ability to automatically process complex statistical functions on large multi-dimensional data, its functionality was limited to providing new descriptions of the data − it does not explain the performance characteristics or the behavior observed in the data. 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.

In general, we will consider the analysis case where we have collected parallel performance data from multiple experiments, and we wish to compare their performance. 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 metrics. 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 order to perform this type of meta-analysis, several components are necessary to meet the desired goals.

Figure 1 shows the interaction between components in the proposed 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.

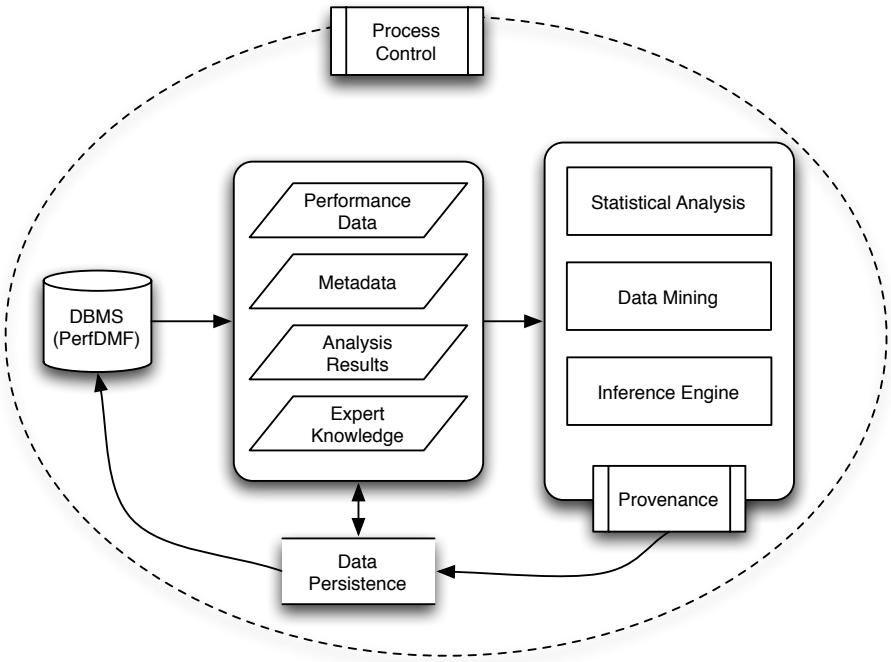


Fig. 1. PerfExplorer analysis framework.

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 New Analysis Input

Expert Knowledge There are several types of parametric study commonly seen in the parallel performance literature: application benchmarking, machine benchmarking, application performance testing and workload characterization. For each of these studies, application performance data is collected while varying one or more configuration parameters. In the example of a scalability study, the number of processors used and the input problem size is varied, and empirical performance results are compared with expected results, based on baseline comparisons. In each of these parametric studies, we have identified eight common categories of parameters, listed in Table 1, along with example parameters for each category and an example of a known assumption, or *expert knowledge*, about a parameter in that category that could be helpful in analyzing the performance of an experiment.

Category	Parameter Examples	Possible Assumptions
Machines	processor speed/type, memory size, number of cores	CPU A faster than CPU B
Components	MPI implementation, linear algebra library, runtime component	component A faster than B
Input	problem size, input data, problem decomposition	smaller problem means faster execution
Algorithms	FFT vs. DFT	algorithm A faster than B for problem > X
Configurations	number of processors, runtime parameters, number of iterations	more processors means faster execution
Compiler	compiler choice, compiler options, pre-compiler usage, code transformations	execution time: -O0 $\geq$ -O1 $\geq$ -O2 $\geq$ -O3 $\geq$ -fast
Code Relationships	call order, send-receive partners, concurrency, functionality	code region has expected concurrency of X
Code Changes	code change between revisions	newer code expected to be faster

Table 1. Parametric categories and corresponding example assumptions in those categories.

As an example, the first category includes differences between architectures, such as when porting an application, or performing an application benchmarking study on more than one architecture. Parameters such as CPU type and speed, the amount of cores per CPU, the number of CPUs per node, etc. all represent useful information when comparing two or more architectures. In order to utilize this information, performance assumptions can be made in the analysis process which will help guide the analysis. For example, take an application executed with the same configuration on two different machines. If the metadata shows that the only difference between the two machines is the speed of the CPU, then the analysis should correlate the performance differences between the two executions to the differences in speed. As another example, suppose that we can identify a region of code as inherently sequential. Therefore, any scalability analysis of this region could then assume that there will be no expected improvement by increasing the number of processors, and will not flag this section as a performance bottleneck. While these are overly simplified examples, they illustrate the potential utility that expert knowledge about an execution can provide to the performance analysis.

Collecting and Integrating Metadata Many performance instrumentation and collection utilities 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. Table 2 shows

Field	Example
CPU Cores	4
CPU MHz	2660.006
CPU Type	Intel(R) Xeon(R) CPU X5355 @ 2.66GHz
CPU Vendor	GenuineIntel
CWD	/home/joeuser/tau2/examples/NPB2.3/bin
Cache Size	4096 KB
Executable	/home/joeuser/tau2/examples/NPB2.3/bin/lu.C.16
Hostname	garuda.cs.uoregon.edu
Local Time	2007-03-29T16:06:08-07:00
Memory Size	8155912 kB
Node Name	garuda.cs.uoregon.edu
OS Machine	x86_64
OS Name	Linux
OS Release	2.6.18.1_ktau_1.7.9_pctr
OS Version	#2 SMP Mon Mar 26 17:36:14 PDT 2007
TAU Architecture	x86_64
TAU Config	-fortran=intel -cc=icc -c++=icpc -mpi ...
UTC Time	2007-03-29T23:06:08Z
username	joeuser

Table 2. Default TAU metadata fields.

examples of metadata fields which are automatically collected by the profiling provided by TAU[5]. 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.

2.2 Meta-Analysis Components

Process Control One of the key aspects of the new PerfExplorer design is the requirement for process control. While user interfaces and data visualzation 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 extention mechanism for creating higher-order analysis procedures. One way of doing this is through a scripting interface[1]. With such an interface, the analysis process is under the control of the anayst, who is able to reliably produce the characterization with minimal effort.

Inference Engine Because the needs of the meta-analysis is dynamic, why should a performance analysis tool hard-code the complex rules that guide the

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

Optimizations	Effects	Time
-O0	Very few optimizations.	0m06.124s
-O2	Optimizations without an unreasonable increase in compilation time or storage	0m28.827s
-O3	-O2 plus -qhot=level=0 -qnostrict -qmaxmem=-1	0m55.131s
-O4	-O3 plus -qarch=auto -qcache=auto -qhot -qipa -qtune=auto	3m42.912s
-O5	-O4 plus -qipa=level=2	4m29.263s

Table 3. Affect of compiler option on compiler time. The time reflects the total time to compile 27 Fortran files, 1 C file, and link the executable.

decision making process of a performance analyst? Is it 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 will integrate a JSR-94[2] compliant rule engine[3]. 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 rules.

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[4]. Any scientific endeavor is 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 Preliminary Results

3.1 Compiler options

Need to point out some key points: - process control necessary for automated tuning - one size fits all doesn't work with optimization flags - for best possible performance, need combine optimization for different files

[2] Java Rule Engine API: `http://jcp.org/en/jsr/detail?id=94`
[3] JBoss Rules: `http://www.jboss.com/products/rules`
[4] Relational Persistence for Java and .NET: `http://www.hibernate.org/`

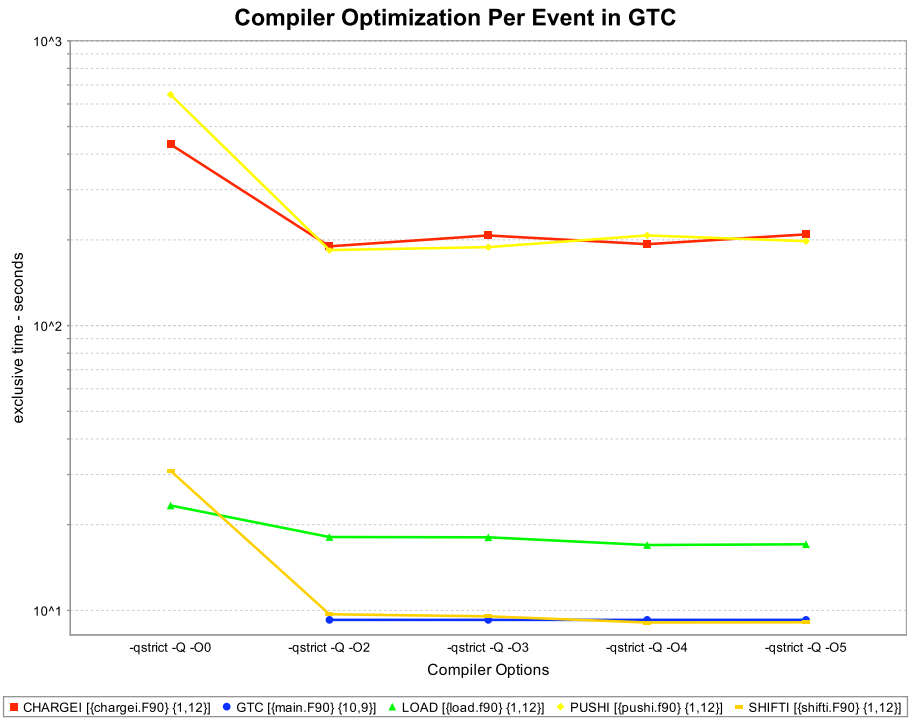


Fig. 2. Affect of compiler options on major events in GTC.

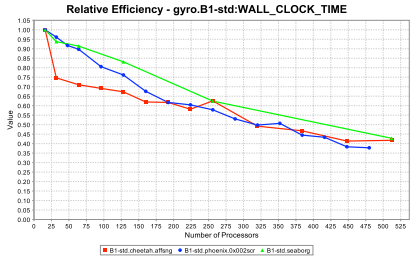
3.2 input parameters

3.3 scalability

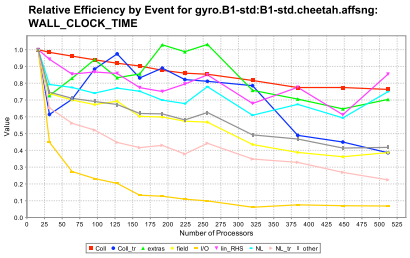
4 Related Work

Hercule[6–8] is a parallel performance diagnosis tool which uses the expert system CLIPS to process computational model-centric rules which can diagnose common performance problems. Hercule's rules define symptoms of known parallel application problems, such as *load imbalance*, *insufficient parallelization*, etc., and encodes possible solutions for correcting these known problems. Hercule takes as input the application's parallel model, and diagnoses known problems from the input data and the application model assumptions. Hercule analyzes event trace files, not profiles.

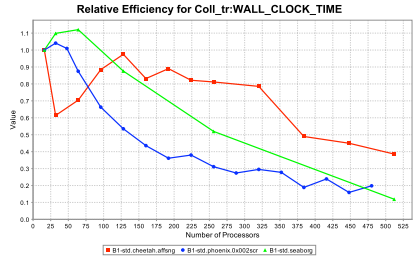
EXPERT[9], from the KOJAK[10] project, is an automatic event-trace analysis tool for MPI and OpenMP applications. It searches the traces for execution patterns indicating low performance and quantifies them according to their severity. The patterns target both problems resulting from inefficient communication and synchronization as well as from low CPU and memory performance.



(a) first caption



(b) second caption



(c) third caption

method	$n = 16$	$n = 32$	Diff	%
RUNTIME	1777	1350	-427	31.6
Coll	106.8	54.8	-52.1	95.1
Coll_tr	86.9	85.9	-1.0	1.1
extras	97.2	65.1	-32.1	49.2
field	499.6	377.5	-122.1	32.4
lin_RHS	111.5	60.6	-50.9	83.9
NL	496.7	475.4	-21.3	4.5
NL_tr	371.3	224.3	-147.0	65.5

(d) fourth caption

Fig. 3. Total Caption

Unlike our proposed approach, EXPERT searches for known problems, rather than focusing on characterization and new problem discovery. Also, the performance data analyzed is trace data. CUBE[11] is a graphical browser suitable for displaying a wide variety of performance measurements for parallel programs including MPI and OpenMP applications, and is the primary analysis viewer for EXPERT. CUBE implements Performance Algebra[12], a technique for performing difference, merge and aggregation operations on parallel performance profile data. While CUBE provides a powerful interface for visualization and exploratory analysis of the differences between two performance data sets, there is no mechanism for linking the performance behavior to the performance context, and providing the user with a meaningful explanation of why the performance differs between the two profiles.

Paradyn[13] utilizes the Performance Consultant[12] and Distributed Performance Consultant[14] for run-time and offline discovery of known performance problems. The latest version of the Performance Consultant uses historical performance data to help guide bottleneck detection. The performance data is mapped into the Program Space, which is split into two pieces: the Space Map and the Event Map. The Space Map consists of the metadata describing the specific execution, in the form of a resource hierarchy. Filtering of experiments comes by selecting a focus from each of the trees in the resource hierarchy. The event map consists of the structural information about the application gathered during compile time and execution time. The performance data is a mapping of the unique program event identifier, metric, focus, time interval and the actual performance result. Structural merge and difference operators are used to perform comparisons between two experiments. Mappings are used to compare dissimilar resource hierarchies. With this interface, it is possible to automatically describe differences between two runs of a program, both the structural differences (differences in program source code and the resources used at runtime), and the performance variation (how were the resources used and how did this change from one run to the next). While the Performance Consultant does include contextual information about the runtime environment to help explain performance differences, there doesn't appear to be a mechanism for including additional expert knowledge about the application, such as data or event relationships. And like the aforementioned tools, the Performance Consultant's strength is in diagnosing known performance problems, rather than general performance characterization.

KappaPi[15, 16] (Knowledge-based Automatic Parallel Program Analyzer for Performance Improvement) and KappaPI2[17] are tools which use trace files from PVM and MPI applications, detect known performance bottlenecks, and determine causes by applying inference rules. The causes are then related back to the source code and suggest recommendations to the user.

Performance Assertions[18] have been developed to confirm that the empirical performance data of an application or code region meets or exceeds that of the expected performance. By using the Assertions, the programmer can relate expected performance results to variables in the application, the execution

configuration (i.e. number of processors), and pre-evaluated variables (i.e. peak FLOPS for this machine). This technique allows users to encode their performance expectations for regions of code, confirm these expectations with empirical data, and even make runtime decisions about component selection based on this data. The use of performance assertions requires extensive annotation of source code, and requires the application developer's experience and intuition in knowing where to insert the assertions, and what kind of performance result to expect.

JavaPSL[19] is a Java Performance Specification Language, designed to be used to specify techniques for searching for known performace problems such as poor scaling, load imbalance, and communication overhead. The specification language could be useful in the application of search heuristics in a particular diagnosis process, and represents a good example of the type of low-level analysis whose results could be used in conjunction with expert knowledge and context metadata to suggest the causes of performance phenomena.

5 Conclusion

References

1. Huck, K.A., Malony, A.D.: PerfExplorer: A performance data mining framework for large-scale parallel computing. In: Conference on High Performance Networking and Computing (SC'05), Washington, DC, USA, IEEE Computer Society (2005)
2. Huck, K., Malony, A., Bell, R., Morris, A.: Design and implementation of a parallel performance data managment framework. In: Proceedings of the International Conference on Parallel Computing, 2005 (ICPP2005). (2005) 473–482
3. The R Foundation for Statistical Computing: R project for statistical computing. `http://www.r-project.org` (2007)
4. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques. 2nd edn. Morgan Kaufmann, San Francisco (2005) `http://www.cs.waikato.ac.nz/~ml/weka/`.
5. Shende, S., Malony, A.D.: The tau parallel performance system. The International Journal of High Performance Computing Applications **20**(2) (2006) 287–331
6. Li, L., Malony, A.D., Huck, K.: Model-based relative performance diagnosis of wavefront parallel computations. In: HPCC, Munich, Germany (2006)
7. Li, L., Malony, A.D.: Knowledge engineering for automatic parallel performance diagnosis. Concurrrency and Computation: Practice and Experience *to appear* (2006)
8. Li, L., Malony, A.D.: Model-based performance diagnosis of master-worker parallel computations. In: Europar 2006. (2006)
9. Song, F., Wolf, F., Bhatia, N., Dongarra, J., Moore, S.: An algebra for cross-experiment performance analysis. In: Proceedings of 2004 International Conference on Parallel Processing (ICPP'04), Montreal, Quebec, Canada (2004) 63–72
10. KOJAK: Kojak. `http://www.fz-jeulick.de/zam/kojak/` (2006)
11. Wolf, F., Mohr, B.: Automatic performance analysis of SMP cluster applications. Technical Report 05, Research Centre Julich (2001)

12. Karavanic, K., Miller, B.: A Framework for Multi-Execution Performance Timing. In: On-Line Monitoring Systems and Computer Tool Interoperability. Nova Science Publishers, New York, USA (2003)
13. Miller, B., Callaghan, M., Cargille, J., Hollingsworth, J., Irvin, R., Karavanic, K., Kunchithapadam, K., Newhall, T.: The Paradyn parallel performance measurement tool. Computer **28**(11) (1995) 37–46
14. Roth, P.: Towards automatic performance diagnosis on thousands of nodes. 5th International APART Workshop, SC2003 Conference (2003)
15. Espinosa, A., Margalef, T., Luque, E.: Automatic performance evaluation of parallel programs. In: IEEE Proceedings of the 6th Euromicro Workshop on Parallel and Distributed Processing. (1998)
16. Espinosa, A., Margalef, T., Luque, E.: Automatic performance analysis of PVM applications. In: EuroPVM/MPI. Volume LNCS 1908. (2000) 47–55
17. Jorba, J., Margalef, T., Luque, E.: Performance analysis of parallel applications with kappapi2. Parallel Computing: Current & Future Issues of High-End Computing, Proceedings of the International Conference ParCo 2005 **33** (2006) 155–162
18. Vetter, J.S., Worley, P.H.: Asserting performance expectations. In: Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing, Los Alamitos, CA, USA, IEEE Computer Society Press (2002) 1–13
19. Fahringer, T., Júnior, C.S.: Modeling and detecting performance problems for distributed and parallel programs with JavaPSL. In: Supercomputing '01: Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM), New York, NY, USA, ACM Press (2001) 35–35