TAUmon: Scalable Online Performance Data Analysis in TAU

Chee Wai Lee, Allen D. Malony, Alan Morris

Department Computer and Information Science,
University Oregon, Eugene, Oregon, 97403

Abstract. In this paper, we present an update on the scalable online support for performance data analysis and monitoring in TAU. Extending on our prior work with TAUoverSupermon and TAUoverMRNet, we show how online analysis operations can also be supported directly and scalably using the parallel infrastructure provided by an MPI application instrumented with TAU. We also report on efforts to streamline and update TAUoverMRNet. Together, these approaches form the basis for the investigation of online analysis capabilities in a TAU monitoring framework TAUmon. We discuss various analysis operations and capabilities enabled by online monitoring and how operations like event unification enable merged profiles to be produced with greatly reduced data volume just prior to the end of application execution. Scaling results with PFLOTRAN on the Cray XT5 and BG/P are presented along with a look at some initial performance information generated from FLASH and PFLOTRAN through our TAUmon prototype frameworks.

1 Introduction

As the level of parallelism increases in large-scale systems, performance measurement of parallel applications will be affected by the size of the performance data being maintained per process/thread, the effects of measurement overhead, and the cost of output, both during and at the end of execution. The traditional approach of post-mortem (offline) analysis of performance experiments will come under increasing pressure as the sheer volume and dimensionality of performance information drives up I/O and analysis complexity. Enhancing performance measurement systems with online monitoring support is a necessary step to address both challenges. In our prior research, we have explored extensions to the TAU performance system [9] that allow access to the parallel performance data measurement for an application at runtime. The TAUoverSupermon [13] (ToS) and TAUoverMRNet [12] (ToM) prototypes leveraged the online monitoring infrastructures, Supermon [15] and MRNet [1], respectively. While ToS and ToM demonstrate monitoring functionality, it is becoming increasing clear that we need to push forward on the scalable algorithms for performance analysis and evaluate their efficiency in real application scenarios.

In this paper, we reconsider the approaches and capabilities of online performance monitoring from a perspective of the online operations necessary to support scalable performance measurement and runtime analysis requirements. We describe changes to ToM to support operations on very large machines enabled by the latest MRNet versions that will soon be released. In addition, we investigate the approach of directly using the parallel infrastructure provided by MPI as an alternative monitoring framework, complementary to ToS / ToM. Together these different approaches for operations via different transports form the foundation of a framework for TAU performance monitoring we call *TAUmon*.

We will structure the rest of the paper as follows. In section 2, we discuss related literature for online monitoring. In section 3, we describe the two transport implementations of TAUmon and the statistical analysis operations that make use of them. Section 4 presents scaling results for analysis operations implemented using MPI and ToM for the PFLOTRAN [11] application and FLASH [4]. These experiments were conducted on Jaguar, Oak Ridge National Lab's Cray XT5 and Intrepid, Argonne National Lab's IBM BlueGene/P. We will then make our concluding remarks.

2 Related Work

The Online Monitoring Interface Specification (OMIS) [8] project provided a general interface between tools and a monitoring system. An event-action paradigm mapped events to requests and responses to actions as part of that interface. J-OMIS [2] was built on top of this interface to provide extensible monitoring facilities for the support of tools for Java applications. Specific target tools were performance characterization and debugging tools. Lee [7] explored the effectiveness of asynchronously collecting profile data transformed from performance traces generated within a running Charm++ application. The work demonstrated how an adaptive runtime system like Charm++ was able to serve as an effective transport medium for such purposes. Periscope [5] made use of hierarchical monitor agents working with applications and an external client in order to address scalability issues with transport. The Distributed Performance Consultant [10] in Paradyn made use of MRNet to support introspective online performance diagnosis. There are also a number of

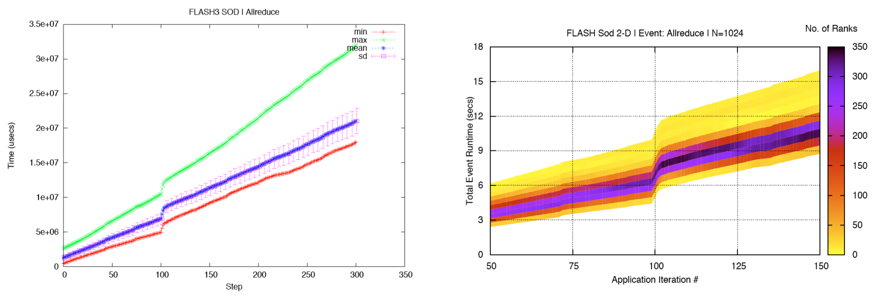

Fig. 1. Time series visualization of basic statistics (on the left) and histograms for cumulative exclusive FLASH events using ToM.

computation steering frameworks [6, 14, 16, 3] where performance information is collected, analyzed and fed through parameter-modifying interfaces in order to change an application's behavior. Compared with the literature above, what distinguishes TAUmon is the attempt to design as abstract as possible TAU's interface to the transport layer in order to provide maximum flexibility in that choice for delivering and processing performance data.

3 Online Performance Monitoring

Our efforts to build monitoring support have grown out of more general concerns of reducing the size, time, and number of files needed to offload parallel profile data at the end of the application execution. For several years, we have been extending the TAU measurement infrastructure with scalable capabilities to access performance data online via different transports, what we call *performance monitoring*. There are several benefits to performance monitoring including offloading of performance data, on-the-fly analysis, and performance feedback. We took advantage of the opportunity to explore these benefits.

Tree-based transports enable scalable aggregation and broadcast of performance information in analysis operations. We have currently provided a basic collective call *TAU_ONLINE_DUMP()* which can be invoked at the end of execution or around synchronous points during an application's lifetime.

With performance data offload, it becomes possible to capture a running time-series snapshots of event statistics, histograms and cluster-averages. Presentation of these snapshots as they appear permits on-the-fly analysis, possibly on a remote client while the application is still executing. Figure 1 shows two such visualization schemes supported by an older implementation of ToM. They both show how the application evolves as more iterations are executed. The leftmost figure shows the cumulative mean values of events' execution time as a function of iteration steps along the x-axis. The rightmost figure shows the corresponding histogram of a selected function, in this case MPI_Alltoall. One can observe an interesting and sudden change in not just the time range of the bins, but the distribution of processors across those bins as evidenced by the change in the color intensity. We now describe the monitoring operations currently supported as a part of the TAUmon framework.

Profile Merging By default, TAU creates a file for every thread of execution in the parallel program. In this case, the number of files required to save the full parallel profile grows as an application scales. We have added to the monitoring framework an operation that merges profile streams in order to reduce the number of files required. The operation is a concatenation operation which should only be used at the end of the run. In our implementation, the root processor requests for the profiles of the other processors piecemeal and in order. On receipt of these profiles, the data is immediately concatenated to a single file by the root processor. This permits the root processor to request more profiles without running out of memory. As we will discuss later in section 4, when profile merging is coupled with and preceded by event unification, there are significant gains in overall data volume when compared with the traditional approach of profile output.

Event Unification In TAU, because events are instantiated locally, each thread assigns a unique event identifier. This results in full event maps that need to be stored for each thread. Event unification begins by sorting each processor's events by name. The information is propagated up the transport's reduction tree, where at each stage, the output to the next level of the tree is the unified and sorted by the list of events from the set of inputs. At the same time, each intermediate node in the tree maintains the reverse event maps. When the fully-unified event information arrives at the root, it is broadcast back through the tree during which the full event map for each processor can be constructed using the local maps maintained at each intermediate node. It is important to note that event unification is the pre-requisite to all other monitoring operations that do not involve simple concatenation. This is because the result of any monitoring operation arriving at the root has to be globally consistent. Without per-processor event maps available, this is impossible.

Mean Profile Generation The mean profile is one that represents the averaged values for all events and metrics across all processors of the application. We take advantage of the simplicity of the operation to also piggy-back basic statistics of the execution like the minimum, maximum values of events and their standard deviation across processors. The latter values, while not part of this operation's output, are useful for supporting other operations like histogramming, which would otherwise have to calculate its own minimum and maximum values first. The mean profile is useful as a summary from which additional statistical information can be sought via other operations. As time-series data, it is capable of highlighting the relative significance of events and their relative rate of growth.

Histogramming We compute histograms by making use of the minimum and maximum values for each event previously calculated in the mean profile operation. Together with a specified bin size, each processor can determine which bin its event metric values fall into. Once determined, the bins for each event are simply propagated up the transport network tree and accumulated. Histograms are useful for highlighting work distribution across processors for events which cannot be captured by mean profiles.

3.1 Updates to TAUoverMRNet

Our original ToM instantiation scheme was designed to allow additional MRNet resources to be initialized flexibly and semi-transparently to both the application and job scheduling system. This was achieved to a reasonable degree through the splitting of MPI communicators as described in [12]. This instantiation scheme, however, did not foresee other flexibility problems in the way MRNet trees may have to be set up. For example, because of the way MRNet trees are currently implemented, intermediate tree nodes may only be allocated on the service nodes of BG/P.

As a result, we have updated ToM to make use of new versions of MRNet with the latter's support for much larger machines like the Cray XT5 and BG/P platforms and specialized batch environments. The user is now expected to allocate sufficient nodes for both the application and the desired MRNet transport tree. An independent code or script determines the topology of the MRNet tree given desired parameters. We currently use the *mrnet_topgen* utility provided by the developers of MRNet for generating the network's topology configuration. In particular, we have found it most convenient to allow *mrnet_topgen* to generate a tree given a fanout factor and the number of processor cores participating in the tree network as input. This makes it easy for users to decide how much computing resources to allocate to the network relative to the resources allocated to the application while retaining some control over fanout factors. Once a topology configuration is generated, a machine-dependent launch mechanism can be started to launch the MRNet front-end, the application which serves as the MRNet backends, and the intermediate nodes of the tree.

On the Cray XT5 and most linux systems, this is relatively simple. The MRNet front-end is started first and uses the resulting topology file to instantiate the tree and create the intermediate nodes. Once ready, the front-end will write out connection information to the leaf nodes the backends are supposed to connect to. Meanwhile, the TAU-instrumented application is simultaneously started. Rank 0 of the TAU-instrumented application back-ends now probe for a flag-file to be written to disk by the MRNet front-end. Once created, Rank 0 is then allowed to read the necessary connection information for each of the other application ranks. Once the front-end has received notice that every application back-end has connected to the MRNet tree, it broadcasts a signal to each application rank allowing them to proceed with the application code past MPI_Init().

On the BG/P, this process becomes more involved. Restrictions in the machine design force intermediate tree nodes to have to reside either on the head nodes or the service nodes while the leaf nodes of the tree must reside on the service nodes. Unlike the XT5 where the front-end knows where the leaf nodes reside, on the BG/P the application backends are the ones responsible for discovering their associated service nodes. The backends have to then inform the launching mechanism as to which service nodes the MRNet leaf nodes may be started. This information is also to be used by the front-end to connect itself to the rest of the tree. We currently do not have this capability ready for ToM.

Figure 2 illustrates the structure of ToM after the three components are launched, successfully connected and communicating.

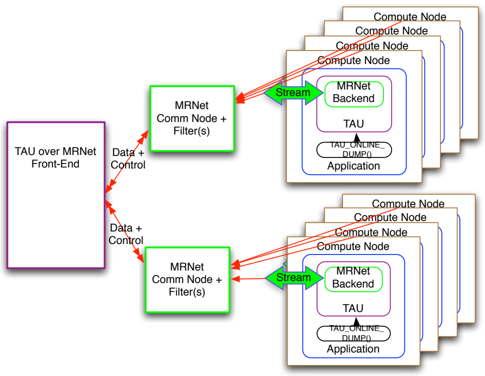

Fig. 2. ToM design.

3.2 MPI-based Monitoring

Our interest in using MPI as an online monitoring transport came from our work to reduce

the overheads associated with
end-of-execution profile output and analysis. As discussed above, it is necessary to deal with a large number of profile files and with efficient profile representation. We had separately implemented parallel event unification, profile merging, and profile data analysis (average, min, max and histogramming) using MPI for use at the end of the execution, and considered how these solutions could be applied to other monitoring operations. The monitoring operations were implemented in parallel using a binomial reduction tree based on the algorithms used in MPI reduction. Enabling them for online monitoring was a simple matter.

4 Experiments and Results

For our experiments with TAUmon, we targeted two applications: PFLOTRAN [11], a 3-D reservoir simulator that models subsurface reactive flows, and FLASH [4], a multi-physics multi-scale simulation code that has been used to simulate type Ia supernovae explosions. The input data set used for PFLOTRAN modeled a large 2 billion degree-of-freedom river simulation. This data set was used for both our preliminary experiments as well as our strong scaling experiments above 4,096 processor cores on the Cray XT5 and BG/P machines. For FLASH, we employed the Sod 2d input dataset with varying maximum refinement levels.

Initial PFLOTRAN Experiments Our preliminary TAUmon experiments were focused on the "end of execution" collection of parallel profiles and processing them to unify the event identifiers and merge them into a single output file. Initial experiments on PFLOTRAN used the MPI transport with 16,380 and 131,040 cores of the Cray XT5. Two levels of TAU instrumentation were enabled. Full instrumentation resulted in 1,131 events for measurement. Leaving fine-grained events uninstrumented, selective instrumentation reduced the measured events to 68. Each profiled event included six hardware counter values (FP OPS, TOT IN, L1 DCA/DCM, L2 TCA/TCM, RES STL) plus execution time (TOT CYC). With no event unification and profile merging, approximately 1.5 GB (16K cores) and 27 GB (131K cores) of parallel profile output were generated for full instrumentation; approximately 80MB was generated for partial instrumentation on 16K cores. When event unification was enabled along with profile merging, these sizes were reduced to 300 MB and 600 MB, respectively. Profile merging at the end of the run with 16K cores took 1.208 seconds and 12.96 seconds with 131K cores. I/O time was included in both results.

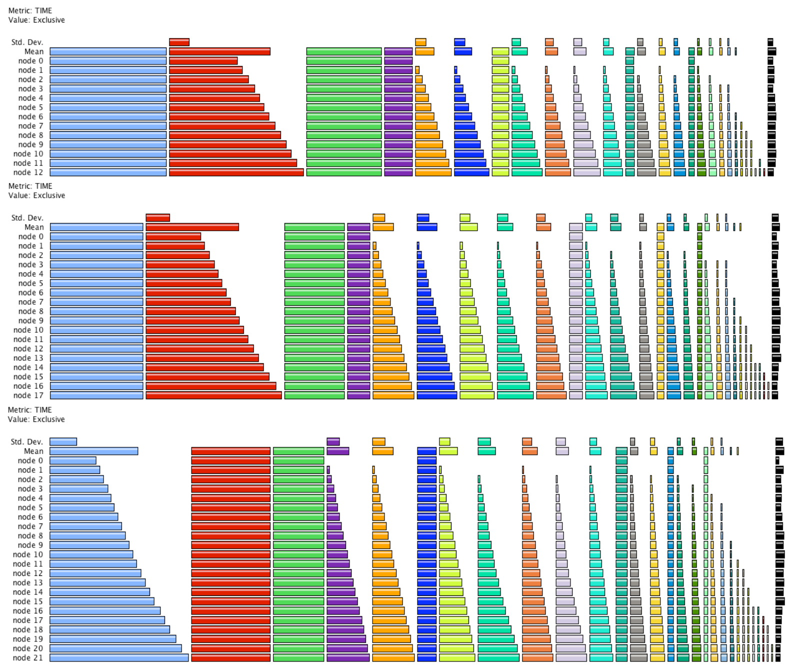

Fig. 3. Online profile snapshots, of 12 to 21 frames from top to bottom, of PFLOTRAN execution on 12K processes.

Based on these results, we experimented with online monitoring of a PFLOTRAN execution, concentrating on analysis operations to reduce the amount of parallel profile data. At each PFLOTRAN major iteration, event unification followed by per-event averaging and histogramming operations were performed. Figure 3 shows three average event profiles from a live online experiment between the NCIS Cray XT5 at the University of Tennessee and a laptop located in Dagstuhl, Germany[1]. The PFLOTRAN execution used 12K cores and the mean profile snapshots (frames) shown are those after 12, 17, and 21 iterations. The events are ordered and colored according to their normalized percent exclusive execution time. As a result, the event colors change from average profile snapshot to snapshot as evidenced by the transition from iteration 17 to iteration 21. We can see the exclusive execution time increasing for some of the events, but constant for others. Those with unchanging size incurred their execution time earlier in the computation.

[1] Our presentation was part of the Dagstuhl Seminar 10181 on "Program Development for Extreme-Scale Computing," http://www.dagstuhl.de/10181/.

Scaling Studies with PFLOTRAN Monitoring Having validated online monitoring functionality, we conducted scaling experiments for TAUmon with both MPI transport and MRNet (ToM). The goal was to measure the time taken by event unification and each statistical analysis operation (per-event averaging and histogramming (20 bins)). In the case of ToM, this is measured by the front-end process which marks the beginning and end of the operation's communication protocol. For MPI transport, the root process of our reduction tree takes responsibility for measuring when the collective operation was begun and when the performance data was finally gathered at the root. When executed without selective instrumentation, our input dataset for the strong scaling of PFLOTRAN generated 756 function events. With selective instrumentation, this number was reduced to 57 events.

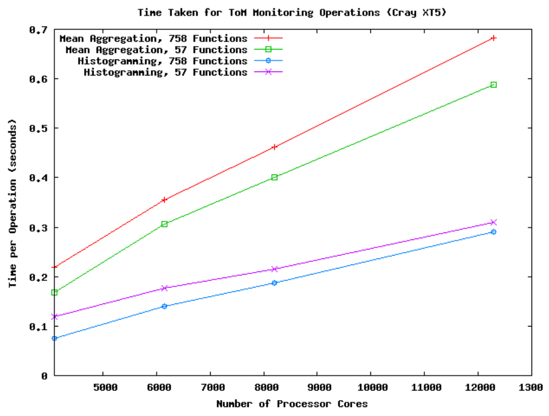

Fig. 4. Time taken for PFLOTRAN monitoring operations on the XT5 using ToM as the transport layer.

Figure 4 shows the scaling results for the Cray XT5 using ToM. Since event unification has not yet been implemented with MRNet, we used MPI event unification results for the other analyses. Two levels of instrumentation were tested (57 and 758 events) from 4K to 12K cores. All times are less than 0.7 seconds, with histogramming taking longer than averaging. Times increase with larger cores counts. Both of these results were expected. We are inves-

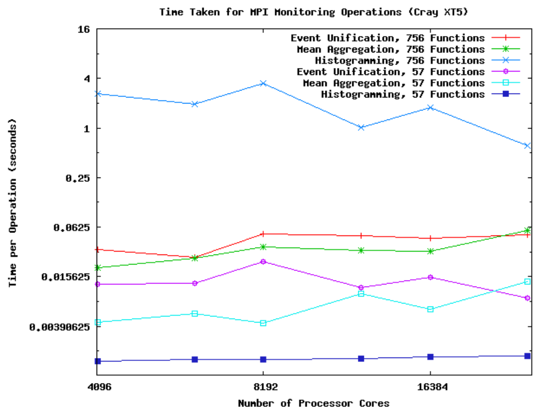

Fig. 5. Time taken for PFLOTRAN monitoring operations on the XT5 using MPI as the transport layer.

tigating performance in more detail to determine if optimizations are possible.

In contrast, Figure 5 shows the scaling results for the Cray XT5 using the MPI transport for 4K to 24K cores. Except for histogramming on 758 events, MPI analysis operations are all less than 0.06 seconds. Compared to ToM, this is significantly faster. Furthermore, there is little effect of scaling on these times. Clearly, the anomaly is the histogramming results for 758 events. More investigation is needed to uncover the poor performance here. Our suspicions are that there is an interaction between the histogramming algorithm and the core locality boundaries that disrupt performance. In addition to being high, the execution times have the weird behavior of declining at larger scale.

Moving to the IBM BG/P, Figure 6 shows the scaling results using MPI transport for 4K to 16K cores and 57 events[2]. Again, the execution times are all less than 0.06 seconds. The interesting effect is the larger event unification time relative to mean and histogram analysis. This is also represented in the XT5 results for 57 events, keeping in mind that Figure 5 is a log-log plot. As before, monitoring analyses with MPI transport appears to be minimally affected by scaling.

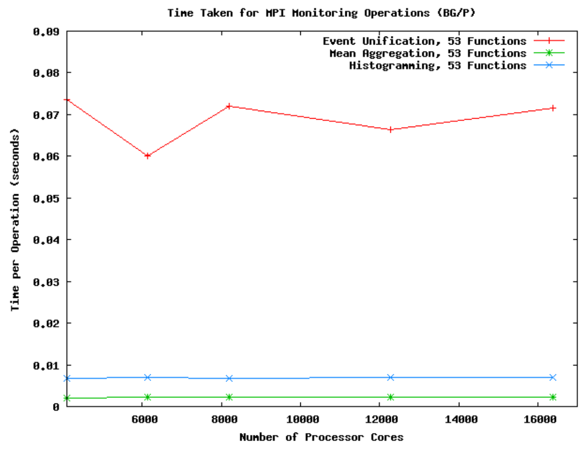

Fig. 6. Time taken for PFLOTRAN monitoring operations on the BG/P using MPI as the transport layer.

FLASH Experiments Our work with FLASH returned to online monitoring experiments to demonstrate how analysis of parallel profile snapshots taken during execution can highlight performance effects that would otherwise be missed in an aggregate profile. Figure 7 shows 34 frames of the mean profile from 1,536 processes running FLASH on the Cray XT5. The most significant five events are labeled. The particular frames were

[2] The 758 event experiments were not completed by the time of submission. Also, ToM is still being implemented on the BG/P.

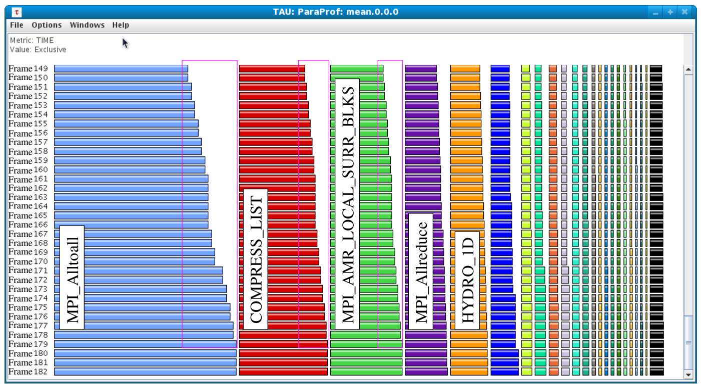
Metric: TIME
Value: Exclusive

Fig. 7. Online profile snapshots of FLASH execution on 1,536 Cray XT5 processes.

chosen because they show the step-like behavior in the events associated with AMR operations. These are highlighted in the figure.

5 Conclusions and Future Work

The TAU project is developing a scalable parallel monitoring framework called TAUmon, based on past research prototypes, but with an eye toward leveraging current scalable infrastructure like MRNet and high-performance MPI libraries. The results from initial experiments reported here give confidence that end-of-execution and online monitoring capabilities will provide opportunities for large-scale performance analysis. The parallel performance of the TAU event, merging, and reduction (mean, histogram) operations are good for both MRNet and MPI transport designs. We are currently developing other analysis operations, such as clustering and wavelet analysis, as well as tuning the monitoring analysis for higher efficiency. Long term, we hope to provide a monitoring interface for parallel applications to interrogate performance online from TAUmon, for purposes of adaptive performance optimization.

Acknowledgments The research was by a grant from the U.S. Department of Energy, Office of Science, under contract DE-SC0001777.

References

1. Dorian C. Arnold, Gary D. Pack, and Barton P. Miller. Tree-based Overlay Networks for Scalable Applications. In *11th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS 2006)*, April 2006.

2. Marian Bubak, Wlodzimierz Funika, Marcin Smetek, Zbigniew Kilianski, and Roland Wismuller. Architecture of monitoring system for distributed java applications. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, volume 2840 of *Lecture Notes in Computer Science*, pages 447–454. Springer Berlin / Heidelberg, 2003.
3. Greg Eisenhauer and Karsten Schwan. An object-based infrastructure for program monitoring and steering. In *SPDT '98: Proceedings of the SIGMETRICS symposium on Parallel and distributed tools*, pages 10–20, New York, NY, USA, 1998. ACM.
4. B. Fryxell, K. Olson, P. Ricker, F. X. Timmes, M. Zingale, D. Q. Lamb, P. MacNeice, R. Rosner, J. W. Truran, and H. Tufo. FLASH: An Adaptive Mesh Hydrodynamics Code for Modeling Astrophysical Thermonuclear Flashes. *The Astrophysical Journal Supplement Series*, 131(1):273–334.
5. M. Gerndt, K. Furlinger, and E. Kereku. Periscope: Advanced Techniques for Performance Analysis. *Parallel Computing: Current and Future Issues of High-End Computing*, pages 15–26, September 2005.
6. Weiming Gu, Greg Eisenhauer, Karsten Schwan, and Jeffrey Vetter. Falcon: Online monitoring for steering parallel programs. In *Ninth International Conference on Parallel and Distributed Computing and Systems*, pages 699–736, 1998.
7. Chee Wai Lee. *Techniques in Scalable and Effective Parallel Performance Analysis*. PhD thesis, Department of Computer Science, University of Illinois, Urbana-Champaign, December 2009.
8. T. Ludwig, R. Wismuller, V. Sunderam, and A. Bode. OMIS - on-line monitoring interface specification (version 2.0). *LRR-TUM Research Report Series*, 9, 1998.
9. Allen D. Malony, Sameer Shende, Robert Bell, Kai Li, Li Li, and Nick Trebon. Advances in the tau performance system. pages 129–144, 2004.
10. Barton P. Miller, Mark D. Callaghan, Jonathan M. Cargille, Jeffrey K. Hollingsworth, R. Bruce Irvin, Karen L. Karavanic, Krishna Kunchithapadam, and Tia Newhall. The paradyn parallel performance measurement tools. *Computer*, 28(11):37–46, 1995.
11. Richard Tran Mills, Chuan Lu, Peter C Lichtner, and Glenn E Hammond. Simulating Subsurface Flow and Transport on Ultrascale Computers using PFLOTRAN. *Journal of Physics: Conference Series*, 78 012051, 2007.
12. Aroon Nataraj, Allen D. Malony, Alan Morris, Dorian C. Arnold, and Barton P. Miller. A Framework for Scalable, Parallel Performance Monitoring using TAU and MRNet. *International Workshop on Scalable Tools for High-End Computing (STHEC 2008)*, June 2008.
13. Aroon Nataraj, Matthew Sottile, Alan Morris, Allen D. Malony, and Sameer Shende. TAUoverSupermon: Low-Overhead Online Parallel Performance Monitoring. *Lecture Notes in Computer Science*, 4641:85–96, August 2007.
14. Randy L. Ribler, Huseyin Simitci, and Daniel A. Reed. The autopilot performance-directed adaptive control system. *Future Gener. Comput. Syst.*, 18(1):175–187, 2001.
15. M. J. Sottile and R. G. Minnich. Supermon: a high-speed cluster monitoring system. pages 39–46, 2002.
16. C. Tapus, I-Hsin Chung, and J.K. Hollingsworth. Active harmony: Towards automated performance tuning. *Supercomputing, ACM/IEEE 2002 Conference*, pages 44–44, 16-22 Nov. 2002.