Observing Performance Dynamics Using Parallel Profile Snapshots

Alan Morris, Wyatt Spear, Allen D. Malony, and Sameer Shende

Performance Research Laboratory
Department of Computer and Information Science
University of Oregon, Eugene, OR, USA
{amorris,wspear,malony,sameer}@cs.uoregon.edu

Abstract. Performance analysis tools are only as useful as the data they collect. Not just accuracy of performance data, but accessibility, is necessary for performance analysis tools to be used to their full effect. The diversity of performance analysis and tuning problems calls for more flexible means of storing and representing performance data. The development and maintenance cycles of high performance programs, in particular, stand to benefit from exploration of and expansion of the means used to record and describe program execution behavior. We describe a means of representing program performance data via a time or event delineated series of performance profiles, or profile snapshots, implemented in the TAU performance analysis system. This includes an explanation of the profile snapshot format and means of snapshot analysis.

Keywords: Parallel computing, performance measurement and analysis.

1 Introduction

In the evolution of parallel performance tools, the two general methods of measurement and analysis – *profiling* and *tracing* – have each found their strong advocates and critics, yet both continue to be dominant in performance engineering practice. In fundamental terms, the differences between the two are clear. Profiling methods compute performance statistics at runtime based on measurements of events made either through sampling or direct code instrumentation. Tracing, on the other hand, merely records the measurement information for each event in a trace for future analysis. Whereas the statistics calculated in profiling can be computed by trace analysis, certain performance results can only be generated from traces. These results are best characterized as *time-dependent*. Indeed, tracing is fundamentally distinguished from profiling in that the trace data retains complete event timing information.

If the potential for temporal analysis of performance is tracing's forte, its weakness is the generation of large, sometime prohibitively large, traces. In general, trace sizes are proportional to both the number of processes in a parallel application and the total execution time. Applications of a thousand processes

E. Luque, T. Margalef, and D. Benítez (Eds.): Euro-Par 2008, LNCS 5168, pp. 162–171, 2008.

running several hours can easily produces traces in the hundreds of gigabytes. While tracing systems have made excellent advances in dealing with trace size in measurement and analysis (e.g., the Vampir [1] and Kojak/Scalasca projects [2,3]), the use of profiling methods does not suffer such drastic size concerns.

Unfortunately, profiling also loses track of time, to put it simply. The statistics it produces are summary in nature and do not allow performance to be observed in a time relative manner. At least, this is a reasonable synopsis of what we might call "classical profiling." An interesting question is whether profiling methods could be enhanced to introduce a time reference, in some way, in the performance data to allow time-oriented analysis.

In this paper, we introduce the concept of *parallel profile snapshots* and describe how profile snapshots are implemented in the *TAU performance system*^{TM} [4]. While the approach we describe in Section §2 is simple, it is powerful in that it opens many opportunities for its use and generates a few problems to resolve along the way. Section §3 discusses the TAU implementation of parallel profile snapshots, both the measurement mechanisms and the profile snapshot output. Performance profiles snapshots can also be generated from trace analysis. Section §4 discusses how this is done in the TAU trace-to-profile tool. However, the goal of a profile snapshot capability is to add temporal analysis support to performance profiling, while avoiding resorting to tracing methods. In Section §5 we show results from the application of profile snapshots. We provide background on related work in Section §6 and give final remarks in Section §7.

2 Design

A *parallel profile snapshot* is a recording of the current values of parallel profile measurements during program execution. It is intended that multiple profile snapshots are made, with each snapshot marked by a time value indicating when it took place. In this manner, by analyzing the profile snapshot series, temporal performance dynamics are revealed. Figure 1 shows a high-level view of the performance profile snapshot workflow. For any snapshot, the profile data for each thread of execution is recorded. Depending on the type of parallel system, thread-level (process-level) profiles may be stored in different physical locations during execution. However, analysis of temporal performance dynamics requires all parallel profile snapshots to be merged beforehand.

A *snapshot trigger* determines when a profile snapshot is taken. Triggers are defined with respect to actions that occur during execution, either externally, within the performance measurement system, or at the program level. Timer-based triggers initiate profile snapshots at regular fixed time internals. These intervals can be changed during the execution. Triggers can be conditional, determined by performance or application state. The key issue is where trigger control is located. User-level trigger control allows a profile snapshot to be taken at any point in the program. The performance measurement system can invoke triggers at points where it has gained execution control.

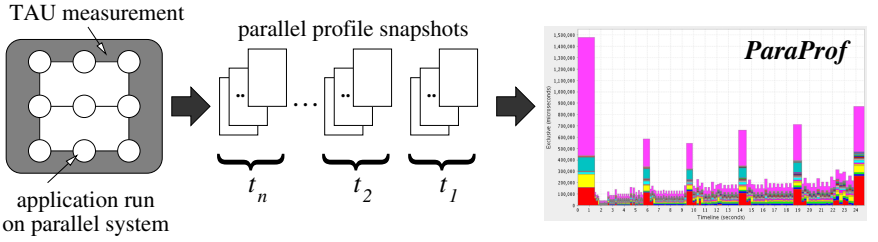


Fig. 1. Profile snapshot architecture

Because the profile snapshots being taken are from parallel executions, the triggers are also executed in parallel. There may be different triggers for different threads of execution and they may be based on different conditions. Thus, it is possible for profile snapshots to be made at different times for different threads for different reasons. Profile snapshots can also record any portion of the parallel profile state by selecting which performance events are of interest and what performance data (e.g., time and counters) should be stored.

A series of parallel profile snapshots is a time-sequence representation of the changing performance measurements of a program's execution. Flexibility in trigger control and profile snapshot selection is important to allow the desired views of temporal performance dynamics to be obtained. For instance, timer-based triggers allow performance frequency and rates to be calculated. However, interpreting the relationship between profile snapshots and between different threads of execution for the same 'logical' snapshot can be tricky, especially when the per thread snapshots are recorded at different time points. One of the challenges of parallel profile snapshots is capturing "synchronized" performance state across the whole parallel execution. As the application scales, this becomes more difficult. In general, by associating profile snapshots (triggers and profile selection) with application semantics (e.g., between computation phases), analysis results can be more meaningfully interpreted.

3 Profile Snapshots in TAU

We have implemented profile snapshot data collection in TAU. The measurement system, API, and analysis tools have all been extended to support this new technique.

We have implemented a new file format in TAU to support scalable profile snapshot data collection. We created an open XML based format with concise data blocks for raw performance data. Identifier information is normalized and separated in definition blocks. Metadata blocks store machine and OS information collected by the measurement system. The format supports streamability for use in online performance monitoring. This is achieved through incremental definition blocks interleaved with performance data blocks such that there is no final index that must be read before data can be interpreted. Additionally,

full definition blocks can recur in the stream to support readers that may start reading partway through the stream. Record blocks can be split across multiple files. The measurement system writes the snapshots for each thread of execution in a separate file. This way there is no need for locking between threads and no need for synchronization across process boundaries. The files can be merged by simply appending one to another, or they can be interleaved, which could be done during online monitoring using a tree network such as MRNet [5] or Supermon [6]. Indeed, we have already shown how a performance measurement system can be integrated with a transport layer such as Supermon [7] to transport distributed profiles at runtime from many sources to a single sink. We have also integrated TAU output with MRNet [8], which allows us to utilize data reductions and filters as profiles are collected through the system.

Profile snapshots are created by applications instrumented with TAU using an API call similar to existing calls for exporting profiles to disk. Some identifying information such as a character string and/or integer value can be associated with each snapshot to correlate them with application level events. For example, in many scientific simulations, there is a main iteration or timestep loop. We would instrument this by placing a snapshot call before the loop begins (called the initialization snapshot), then another at the end of the loop, identified with the iteration number. The initialization snapshot is necessary so that the first iteration of the loop is distinguished from the execution of the application before the main loop begins. At the end of execution, a final snapshot is written. The raw data from the final snapshot is the same as what would normally be written as regular profile.

The option to perform snapshot profiling is orthogonal to all of the other profiling options in TAU. It can be combined with callpath profiling, phase profiling, and any other specialized profiling options. We capture both interval events with begin/end semantics (such as timers) as well as atomic user-defined events that are triggered with application specific data values.

We have tested the scalability of this format with hundreds of processors and hundreds of events on long running applications. Depending on the application, creating snapshots at coarse intervals still provides a wealth of useful information. Our profile snapshot format typically requires on the order of 20 bytes per event recorded. In the execution of a 6 minute simulation of FLASH across 128 processors, we generated 320MB of uncompressed data (58MB compressed) with 6.3% overhead above regular profiling (which had a 4.6% overhead) . By comparison, with tracing, the same run generated 397GB of trace data and imposed a 130.3% overhead (13m31s).

We have extended the PerfDMF [9] system to support reading the new snapshot format. For those components that cannot interpret the intermediate snapshot data, such as PerfExplorer [10], the data is interpreted as a single profile representing the entire execution, just as if regular profiling had been done. Other tools, such as ParaProf [11], can take advantage of the full range of snapshot profiles.

ParaProf has been significantly enhanced to support profile snapshots. Each chart and table can display both individual and cumulative data from snapshots, all linked through a separate window with a slider to control snapshot position. Because the snapshots are timestamped, we present it as a timeline and allow automatic playback of the execution.

4 Trace File Conversion

In addition to generating profile snapshots via TAU instrumentation, we have implemented a tool for extracting them from existing trace data. A full program trace implicitly contains all profile data that might have been extracted from the traced application. By converting traces to profile snapshots we open the wealth of data contained in traces to profile and profile snapshot analysis methods.

We developed the trace-to-snapshot functionality by extending an existing trace-to-profile utility in TAU. It operates by reading trace files and maintaining a time-stamped callstack of trace events. From this, it is able to produce profile files through simulation of the original program execution, essentially using the trace data as a script.

Each thread of execution in the trace is assigned an individual callstack which records the inclusive and exclusive time spent in routines as they are entered and exited. The callstack data structure also aggregates any user defined event data contained in the trace. Profile snapshots are produced by periodically finalizing the current callstacks, as if all active routines exited at the current timestamp. The collected profile data is then written to disk. The converter then resumes the simulation with a non-finalized version of the callstack.

The trace-to-profile converter generates snapshots on the basis of specified time intervals by default. However, all collected metrics remain visible to the snapshot logic throughout the conversion process. Wall clock time intervals are merely one special case. Other triggers for snapshot generation, such as inclusive or exclusive time spent in a given routine, the number of calls to a given routine or the value of a given hardware counter can be just as easily designated as triggers for snapshot generation. Because they are driven by program behavior rather than static time increments, snapshots generated by these triggers may reveal more about the program's behavior with respect to certain phases of execution.

The process of converting multi-threaded traces to profile snapshots is embarrassingly parallel. So long as the trace files themselves are not merged, the trace-to-profile converter can be independently invoked on the trace output of distinct processes. No programmatic means of taking advantage of this parallelism, which also exists in merged traces, has been implemented, however.

Because event driven trace files typically share analogous content it is straightforward to extend the converter's format recognition. We leveraged this innate flexibility by separating the trace event processing routines from the trace reader API in use. This makes it fairly simple to extend the converted to support other trace formats.

5 Application of Profile Snapshots

We have applied our profile snapshot technique to the FLASH [12] application from the ASC Flash Center at the University of Chicago.

FLASH is a parallel adaptive-mesh multi-physics simulation code designed to solve nuclear astrophysical problems related to exploding stars. The FLASH code solves the Euler equations for compressible flow and the Poisson equation for self-gravity.

We instrumented FLASH with TAU using PDT [13] for routine level timing information. The instrumentation has been optimized so that not all routines are instrumented (to reduce overhead, small routines which are called with very high frequency are often excluded [14]). We instrumented the main iteration loop for snapshots using the method described in section §3. Each iteration of the loop takes a different amount of time due to a variety of factors. The profile snapshot technique allows us to verify what is happening in the simulation.

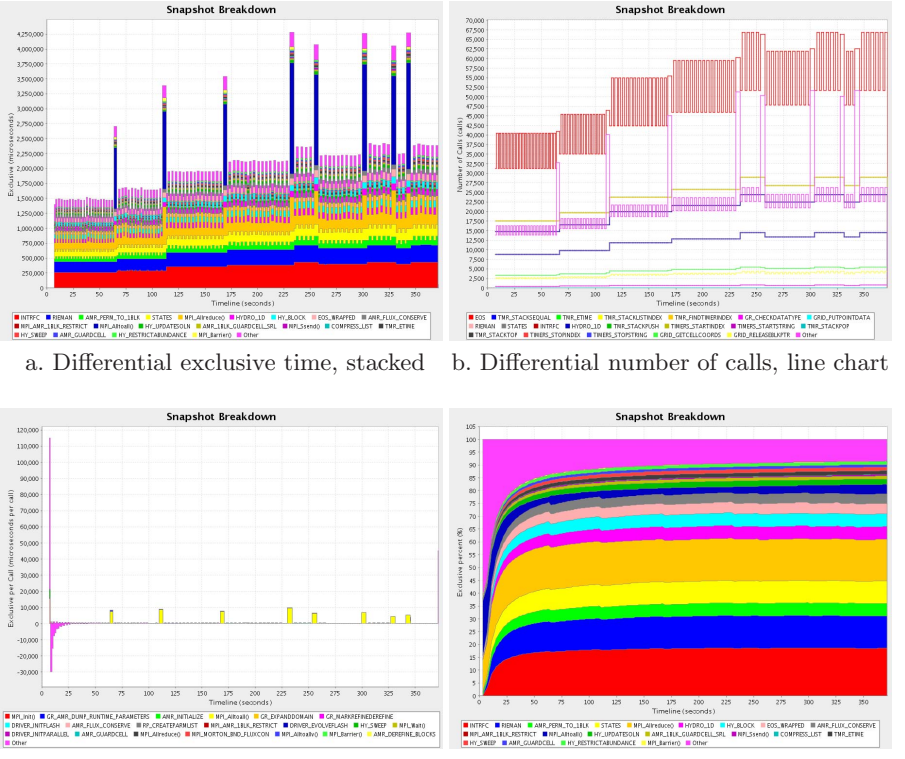
Some iterations perform different tasks such as checkpointing and AMR regridding. Regular profiling will tell us the aggregate time that these operations take for the entire execution, but the per-iteration snapshots will show how these operations perform within the context of the iteration. Additionally, we find that the amount of work per iteration increases due to the mesh refinement. There are more total leaf blocks in the grid and thus more work to do. We can verify this behavior and its effects with snapshot profiles.

Figure 2 shows a variety of analysis displays from ParaProf showing performance data from each of the profile snapshots. The data shown here is for MPI rank 0 from a 6 minute 128 processor run on LLNL's Atlas machine. Figure 2(a) shows the time taken in each of the top 20 events for each snapshot, across a timeline, as a stacked bar chart. The data here is *differential*, meaning that the snapshots are viewed as the difference between the timing information at the start of the snapshot vs. the end of the snapshot. Alternatively, we can view each chart in cumulative mode. Figure 2(b) shows a line chart of the number of calls for the top 20 routines. The AMR grid used in FLASH is refined as the simulation proceeds, so we expect to see more calls and more time spent in later timesteps. This is verified by the data we see in the profile snapshots.

Figure 2(c) shows the differential per call exclusive time. Because the per call value can decrease between snapshots, the values here can be negative. Here we see that the *MPI_Alltoall()* call has spikes of large per-call values during certain iterations. While the regular snapshot view would show us that more time is spent in this routine, it could be due to more *MPI_Alltoall()* calls being made, however, with this view, we are able to determine that the duration of each call has actually increased.

Figure 2(d) shows the cumulative exclusive percent of time spent in each of the top 20 routines. As expected, the cumulative percentage of time spent in each routine stabilizes as the simulation proceeds.

Each of ParaProf's regular displays can show cumulative or differential snapshot data. A slider is used to select which snapshot is currently viewed across all windows. Additionally, the execution history can be replayed automatically



a. Differential exclusive time, stacked b. Differential number of calls, line chart

c. Differential exclusive time per call d. Cumulative exclusive percent

Fig. 2. ParaProf charts of profile snapshots from FLASH

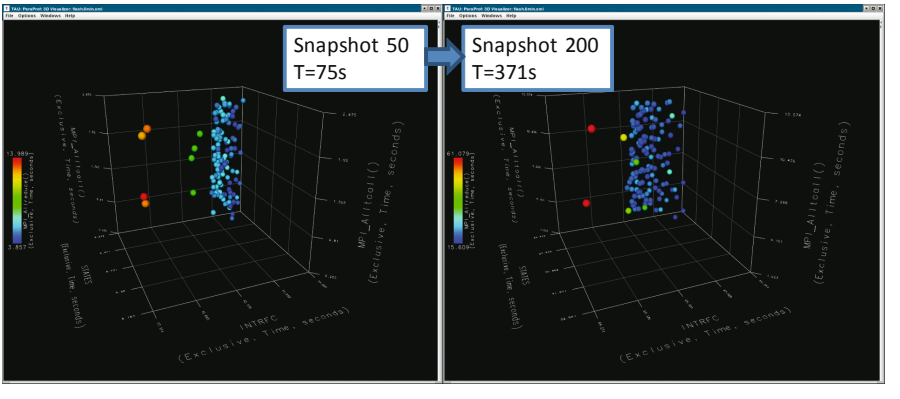


Fig. 3. Three-dimensional scatterplots from cumulative snapshots showing the change in clustering over time

and the user can watch as each window animates through the history of the execution, similar to a saved history of an online monitoring framework.

Figure 3 shows one of the three dimensional charts from ParaProf for two different snapshots. The 3D scatterplot can be used to determine performance data clustering. With a snapshot profile, we can see how the clustering changes over the course of the execution of the application.

Using the new capabilities provided by profile snapshots in TAU, we are able to verify our understanding of the execution of FLASH and look for inconsistent program behavior over time.

6 Related Work

The concept of parallel profile snapshots complements the work by Fürlinger on "incremental profiling" [15,16] of OpenMP parallel programs. Incremental profiling shares the same objectives as TAU's profile snapshot to capture temporal performance dynamics. Fürlinger developed the *ompP* tool and has shown it to be effective at tracking variations in performance properties of OpenMP loops over time, in particular overheads associated with loop imbalance and thread synchronization. Our work on profile snapshots generalizes incremental profiling to operate with any parallel program and at any scale. TAU implements a broader spectrum of snapshot triggers than ompP's regular, fixed-length time triggers. Also, a wider choice of analysis and display is available. We should note that the incremental profile data captured by ompP can be easily converted to the TAU profile snapshot format, making it possible to use ParaProf's performance displays with ompP.

Both TAU and ompP use direct measurement for performance profiling. Other tools use statistical sampling to build profile data. The *Intel Thread Profiler*, part of the *VTune Performance Envionment* [17], and the *Sun Studio Performance Analyzer* [18] both use statistical profiling. In addition, they provide views of performance dynamics. The Intel Thread Profiler has a timeline view that highlights thread state and performance activity. Similarly, the Sun Studio Performance Analyzer captures a profile sample trace for every thread of execution that includes data about call stacks, thread microstate, synchronization delay, memory allocation, and operating system interactions, and then displays this information as a function of time. However, because they are based on statistical sampling, both tools are limited in their support for general parallel programs and profile snapshot points (i.e., triggers) are constrained to sampling events (e.g., timer interrupts). In contrast, TAU performance snapshots are more scalable and flexible in both measurement and analysis.

Statistical profiling systems include those that can externally measure application performance and provide online access to the data. The Digital Continuous Profiling Infrastructure (DPCI) [19], *OProfile* [20], and Sun's *DTrace* [21] all implement infrastructure for gathering a wide variety of application and system statistics, and connecting with daemons to process and store the results. While providing the capability to observe temporal performance dynamics, these

technologies are application agnostic and not targeted to parallel applications per se. They lack the generality of the profile snapshot approach in TAU to capture and study performance behavior with respect to application semantics and performance optimization needs.

7 Conclusions and Future Work

Profile snapshots extend the capability of profiling systems with the addition of temporal data analysis without the need for full application tracing. This new capability opens up an area of time-based differential profile analysis that allows us to see how an application's performance changes through the course of its execution.

Our future plans include enhancing the snapshot triggering mechanism of both the runtime measurement system and the trace conversion utility. We would like to be able to specify and create a more rich set of triggers for snapshot creation based on runtime measurements, application level events, and conditionals. Additional enhancements to the trace-to-profile tool will include the option to specify lock step or thread-specific snapshot generation.

Profile snapshots are an ideal vehicle for integration with an online analysis infrastructure. We plan to create a robust, extendable online analysis system. Several key additions to the snapshot infrastructure will be necessary. We will need the capability to perform selective snapshots (e.g. subsets of events, counters), and allow for a reverse control channel for runtime adjustment of this selection. Along these lines, we can also build in mechanisms to buffer snapshot data for those cases where scalable I/O is not available or not necessary, in the case of online data collection.

Acknowledgments

This research is supported by the U.S. Department of Energy, Office of Science, under contracts DE-FG02-05ER25680 (Application-Specific Performance Technology for Productive Parallel Computing) and DE-FG02-07ER25826 (Knowledge-Based Parallel Performance Technology for Scientific Application Competitiveness).

References

1. Brunst, H., Kranzlmüller, D., Nagel, W.E.: Tools for Scalable Parallel Program Analysis - Vampir NG and DeWiz. Distributed and Parallel Systems, Cluster and Grid Computing 777 (2004)
2. Wolf, F., Mohr, B.: Automatic Performance Analysis of Hybrid MPI/OpenMP Applications. Journal of Systems Architecture 49(10-11), 421–439 (2003); Special Issue Evolutions in parallel distributed and network-based processing
3. Geimer, M., Wolf, F., Wylie, B., Mohr, B.: Scalable Parallel Trace-Based Performance Analysis. In: Mohr, B., Träff, J.L., Worringen, J., Dongarra, J. (eds.) PVM/MPI 2006. LNCS, vol. 4192, pp. 303–312. Springer, Heidelberg (2006)

4. Shende, S., Malony, A.D.: The TAU Parallel Performance System. The International Journal of High Performance Computing Applications 20, 287–331 (2006)
5. Roth, P., Arnold, D., Miller, B.: MRNet: A Software-Based Multicast/Reduction Network for Scalable Tools. In: SC 2003: ACM/IEEE conference on Supercomputing (2003)
6. Sottile, M., Minnich, R.: Supermon: A High-Speed Cluster Monitoring System. In: CLUSTER 2002: International Conference on Cluster Computing (2002)
7. Nataraj, A., Sottile, M., Morris, A., Malony, A.D., Shende, S.: TAUoverSupermon: Low-Overhead Online Parallel Performance Monitoring. In: Europar 2007: European Conference on Parallel Processing (2007)
8. Nataraj, A., Morris, A., Malony, A.D., Arnold, D., Miller, B.: Scalable Online Monitoring of Parallel Applications (under submission)
9. 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 (ICPP 2005), pp. 473–482 (2005)
10. 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 2005), Washington, DC, USA. IEEE Computer Society, Los Alamitos (2005)
11. Bell, R., et al.: A Portable, Extensible, and Scalable Tool for Parallel Performance Profile Analysis. LNCS, vol. 2790, pp. 17–26. Springer, Heidelberg (2003)
12. Rosner, R., et al.: Flash Code: Studying Astrophysical Thermonuclear Flashes. Computing in Science and Engineering 2, 33–41 (2000)
13. Lindlan, K.A., Cuny, J., Malony, A.D., Shende, S., Mohr, B., Rivenburgh, R., Rasmussen, C.: A Tool Framework for Static and Dynamic Analysis of Object-Oriented Software with Templates. In: Proceedings of SC 2000: High Performance Networking and Computing Conference (2000)
14. Shende, S., Malony, A.D., Morris, A.: Optimization of Instrumentation in Parallel Performance Evaluation Tools. In: Kågström, B., Elmroth, E., Dongarra, J., Waśniewski, J. (eds.) PARA 2006. LNCS, vol. 4699, pp. 440–449. Springer, Heidelberg (2007)
15. Fürlinger, K., Dongarra, J.: On Using Incremental Profiling for the Performance Analysis of Shared Memory Parallel Applications. In: Proceedings of the 13th International Euro-Par Conference on Parallel Processing (Euro-Par 2007) (August 2007) (accepted for publication)
16. Fürlinger, K., Moore, S.: Continuous Runtime Profiling of OpenMP Applications. In: Proceedings of the 2007 Conference on Parallel Computing (PARCO 2007), pp. 677–686 (September 2007)
17. Intel Vtune Performance Analyzer, `http://www.intel.com/cd/software/products/asmo-na/eng/vtune/239144.htm`
18. Itzkowitz, M.: Sun Studio Performance Analyzer (2007)
19. Anderson, J., et al.: Continuous Profiling: Where Have All the Cycles Gone (July 1997)
20. Oprofile, `http://oprofile.sourceforge.net/`
21. Cantrill, B., Shapiro, M., Leventhal, A.: Dynamic Instrumentation of Production Systems. In: USENIX Annual Technical Conference (ATEC 2004), p. 2 (2004)