DOE Office of Science
Office of Advanced Scientific Computing Research

Extreme Performance Scalable Operating Systems

Final Progress Report (July 1, 2008 – October 31, 2011)

Award: DE-FG02-08ER25846

Allen D. Malony¹ and Sameer Shende²

¹Department Computer and Information Science, University Oregon, Eugene, Oregon, 97403

²Performance Research Laboratory, University of Oregon, Eugene, Oregon, 97403

1 Introduction

This is the final progress report for the FastOS (Phase 2) (FastOS-2) project with Argonne National Laboratory and the University of Oregon (UO). The project started at UO on July 1, 2008 and ran until April 30, 2010, at which time a six-month no-cost extension began. The FastOS-2 work at UO delivered excellent results in all research work areas:

- scalable parallel monitoring
- kernel-level performance measurement
- parallel I/O system measurement
- large-scale and hybrid application performance measurement
- online scalable performance data reduction and analysis
- binary instrumentation

Progress reports in Years 1 and 2 provided detailed discussion of our accomplishments during these periods. These are summarized in Section §2 and Section §3, respectively. The majority of the final progress report reviews our achievements in Year 3 and the no-cost extension period. In Section §4, we begin reporting on improvements in the scalable online performance data reduction and analysis. Next, we discuss our continued work with hybrid performance measurement. New developments in parallel I/O were made during the period. Finally, we return to the kernel-level performance measurement work of Year 1 with a new development based on current support in Linux. As with the prior two reports, our meeting activities are reported; see Section §7.

2 Year 1 Accomplishments

The FastOS-2 project followed a successful FastOS effort where the University of Oregon delivered performance measurement and analysis support that was targeted to the ZeptoOS [6] environment.

Year 1 of FastOS-2 continue along this path targeting new support to enable a whole system performance perspective. The achievements made during this period are briefly summarized below. The Year 1 progress provides full details.

2.1 Performance Monitoring

We built a scalable parallel monitor (through the integration of TAU and MRNet) and evaluated its function and performance on several platforms. MRNets distributed programming capabilities were applied to efficiently perform useful reductions on the performance data on its path from application to monitor. Our research appeared in a STHEC workshop [1] and the CCPE journal [10].

In addition, we applied the TAUoverMRNet (ToM) system to benchmark online performance monitoring use and to implement profile snapshot functionality for the FLASH applications. This work was reported at the Cluster [4] and Euro-Par [5] conferences.

2.2 OS Performance Measurement and Analysis

Having developed the KTAU Linux kernel measurement infrastructure in the former FastOS project, our intent in Year 1 was to both the TAU and KTAU to the ZeptoOS V2.0 distribution for the IBM BG/P. This was accomplished and included extension of TAU/KTAU with specific BG/P support. Preliminary experimentation was out on the Surveyor system. Our work demonstrated that integrated user-system performance views could be obtained. This research was reported at a HPC tools workshop [3].

2.3 Parallel I/O Performance Measurement

Year 1 began our efforts to develop performance measurement support for Parallel I/O framework. The main problem studied was the different software layers in I/O stacks and how to track information flow between them. The main results were in development of initial instrumentation and measurement support within PVFS. A new TAU instrumentation capability for automatic wrapper generation was created during this activity.

3 Year 2 Accomplishments

The focus in the second year of FastOS-2 shifted slightly to replace the KTAU kernel-level performance work with an emphasis on large-scale and hybrid application performance measurement and binary instrumentation. Our parallel performance monitoring work continued with a focus on online scalable performance data reduction and analysis. Further improvements were also made in parallel I/O stack observability. The strategy in Year 2 was to tackle certain scaling challenges in the TAU instrumentation and measurement system to better position the FastOS project work for the potential hyper-scale systems appearing in Year 3. We briefly summarize our results below.

3.1 Parallel Performance Monitoring

Our work with MRNet during Year 2 mainly resulted in re-engineering TAUoverMRNet (ToM) to leverage v3.0 functionality. This improved certain aspects such as ToM startup and connection to the application. In parallel with ToM activities, we began work on a MPI-based monitoring implementation. We built a monitoring interface in TAU that was independent of the backend

implementation. Our dual efforts continue through the year and provide us basis for comparison on the Cray XT5 and BG/P platforms.

We produced significant results using the performance monitoring frameworks on the problem of event unification, data reductions, and scalable statistics computation. The tools were demonstrated on the PFLOTRAN application at large scale and results were presented at the Dagstuhl Seminar 10181.

3.2 Hybrid Performance Measurement

Year 2 saw the begin of our efforts to develop support in TAU for event-based sampling (EBS) and its integration with the probe-based measurement support that forms the foundation of TAU. Not only does this expand the scope of observable performance, but there are also opportunities for hybrid measurement. We were successful in building an initial version of the (so-called) TAUebs system during the year and published the work in the ICPP conference [11]. The implementation provided tracing support only, but this was enough to demonstrate important new capabilities to work with applications such as MADNESS, which proved to be problematic for TAU in the past.

3.3 Binary Instrumentation

During Year 2, the Dyninst project released support for binary rewriting. This provided a new mechanism for TAU to do instrumentation. It was an important development that had direct relevance to the FastOS-2 project. We were able to build a binary instrumentation tool and include it in the TAU distribution.

3.4 Parallel I/O Performance

TAU was extended to track I/O performance for applications using the MPI-IO constructs included in the MPI-2 standard. TAU's library wrapping tool was applied to build support for MPI-2 and MPI-IO, including the ability to examine the arguments that flowed through the MPI-IO calls. This allowed TAU's context events to be used to track the volume and bandwidth of I/O operations for different program callpaths.

4 Final Period Accomplishments

The final period of the FastOS-2 project focussed on improvements in the scalable online performance data reduction and analysis, continued work with hybrid performance measurement, new developments in parallel I/O, and a return to the kernel-level performance measurement work. The results from these activities are discussed in the following sections.

4.1 Parallel Performance Monitoring

Our work in Year 2 on scalable performance monitoring demonstrated working solutions using MR-Net and MPI. In the final period, we focussed our efforts on improving the MPI-based capabilities, primarily because these are portable across HPC platforms. In particular, we made the scalable data analysis operations more robust and added a means to apply multiple operations. It is now routine practice to run event unification, reduction, and basic statistics operations at the end of every TAU-instrumented MPI application.

We will continue to add analysis operations to the monitoring repertoire. There have been requests made to also provide a scripting capability so that derived metrics can be processed in a scalable manner online, instead of in TAU's ParaProf tool post-mortem. This is in development.

Figure 1: Probe-based and sampling-based profile approaches with sampling-based tracing.

4.2 Hybrid parallel profiling

The Year 2 report covered in detail our efforts to create support in TAU for hybrid performance measurement and analysis that integrates parallel *sampling-based* and *probe-based* techniques. Initially, this work targeted capture of a trace of samples, where each sample contained a depth-limited unwound call stack and a key to identify the TAU context. The trace was post-processed to produce merged hybrid profiles. Figure 1 shows a picture of traditional probe-based and sampling-based profiling together with the sampling-based tracing.

In the final period, we have made substantial progress in developing an online hybrid parallel profiling capability. This work will be released in the TAU distribution at the SC11 conference.

4.2.1 Approach

To enable hybrid profiling, it was necessary to first implement event-based sampling (EBS) support in TAU and capture EBS histograms (i.e., profiles) at runtime. This has been accomplished. However, the power of hybrid profiling comes from associating the TAU probe-based profile with the EBS profile. Figure 2 shows the general idea. At each EBS interrupt, the TAU event stack is queried and a key is generated for each unique TAU event path seen in the application. For each TAU event path, a EBS histogram is created. What this allows TAU to do is to contextualize EBS histograms with respect to TAU profiles.

Figure 2: TAUebs hybrid profiling.

4.3 Development

The development of the TAU hybrid profiling was based on existing timer-interrupt framework to trigger samples. It supports call stack unwinding and using BFD for PC address symbol resolution. The TAU event context can be controlled by an specifying the event path depth through an environment variable. Multi-threading is support for Posix threads, but not for OpenMP threads, currently. The TAU parallel profile format has been updated to merge hybrid profiles and TAU’s profile analysis tool, *ParaProf*, is now able to read and display this format.

4.3.1 Hybrid profiling examples

We are testing TAU hybrid profiling with several parallel applications. For demonstration purposes, Figure 3 shows a simple program profile with merge probe-based and sample-based profile data in a single view. Sample histograms are designated by [SAMPLE] and are positioned relative to the TAU event context in which they occurred.

Figure 4 shows the simple program profile when loop-level instrumentation is enabled in TAU. This highlights a unique feature of hybrid profiling. With EBS profiling alone, it would be necessary to post-process the EBS histogram to determine if a sample was seen within a loop. Similarly, probe-based profiling would have difficulty measuring loop execution details.

TAU hybrid profiling is targeted for parallel application using message communication and/or multi-threading. We experimented with the NAMD and FLASH applications on a Cray XE6 (the NERSC “Hopper” system). Figure 5 shows the use of hybrid profiling with NAMD. The mean profile is presented. NAMD is built with Charm++ and what is being displayed is the internal operation of Charm++’s runtime system *Idle* state. A significant fraction of the total execution time in this experiment is spent in this state, but Charm++’s runtime system is difficult to instrument in detail. We captured TAU events for the major RTS entry points (e.g., *Idle*) and then relied on EBS observation for the internal operation.

The mean hybrid profile in Figure 6 and Figure 6 demonstrates how hybrid profile analysis in *ParaProf* can be used to discover performance characteristics. Figure 6 exhibits a sequence of TAU events which form a path to lower levels in the event graph (call graph). The coloring used in the

Figure 3: TAUebs hybrid profiling for simple example.

ParaProf event graph view helps to identify performance “hot spots,” for instance, the **HYDRO** routine. Additional instrumentation in **HYDRO** allows the user to further search for where the performance is located. We see an unfolded event path to a loop that appears in the **EOS_HELMHOLTZ** routine. At this point, measurement switched to **EBS** and we see the histogram captured when the execution is within the loop.

Presentation of a hybrid profile in this manner could show samples from particular code regions associated with multiple event paths. If we were interested in seeing every event where a sample occurs, a reverse event/call path view needs to be produced. We implemented this feature in ParaProf. Figure 6 shows a reverse event/call path view, highlighting those events where a particular sample in **memcpy** occurs.

4.3.2 Continuing work

There are several areas where we will continue our hybrid profiling work. First, alternative mechanisms exist for unwinding call stacks. While we believe the **libunwind** library is generally robust, there may be some exceptions where other support might be better. Our approach is to create a call stack query bridge in TAU that could interface with different unwinding support. Second, hybrid profiles contain significantly more information than TAU’s probe-based profiles. We need to continue to improve the efficiency of maintaining context-based histograms during execution. This extends to scalable hybrid profile formats.

Lastly, a hybrid profiling approach allows the user to set the degree of probe-based versus sample-based measurement. This is done through event selection and the control of event path depth. In general, knowing where to set the optimal degree and depth is difficult for an application. It is possible that more guidance can be given with just a few bounding experiments.

Figure 5: NAMD hybrid profiling showing context.

the compilation process, at link time, and on the binary to track I/O calls.

Compilers for C and C++ will pre-process the source code and expand header files and macros before the code generation phase. This allows tools to intercept and replace I/O calls, such as POSIX I/O open, close, read, and write, with their instrumented counterparts. TAU utilizes this support to replace all references to the I/O call at the callsite in the source code with the corresponding instrumented call. However, this only works for C and C++. TAU does support instrumentation of Fortran I/O constructs by re-writing the source code.

The MPI message passing libraries provides a name-shifted interface that permits tools, including TAU [4], to intercept calls using the PMPI name-shifted interface. TAU additionally uses this support to create a wrapper library for MPI-IO calls (e.g., `MPI_File_read`) that internally calls the name-shifted interface (e.g., `PMPI_File_read`). This work was reported in Year 2 of the project. However, library interposition through name-shifted interfaces is only available as a technique if such interfaces are implemented in the library. This is not the case with POSIX I/O.

Many HPC operating systems such as Linux, Cray Compute Node Linux (CNL), IBM BlueGene Compute Node Kernel (CNK), Solaris permit pre-loading of a library in the address space of an executing application specifying a dynamic shared object (DSO) in an environment variable (`LD_PRELOAD`). TAU uses this technique to intercept all I/O operations by means of a wrapper-library where the POSIX I/O calls are redefined to call the global dynamically-linked routine. While it can resolve all POSIX-IO calls and operates on un-instrumented executables, it only supports dynamic executables. A different technique will be necessary to support static binaries on machines such as IBM BlueGene and Cray XE6 and XK6 systems.

A linker can redirect references to a wrapped routine when it is invoked with a special flag on the command line (`-Wl,-wrap,function_name`). In this case, the application does need to be re-linked to use the wrapped library, but this instrumentation technique overcomes the limitation of the previous approach provided by the runtime linker and may be used with both static and dynamic executables. TAU has applied this approach to instrument POSIX I/O calls by creating a wrapper library.

When the user needs to evaluate the time spent in un-instrumented I/O libraries (such as HDF5) and other system libraries, it is important to be able to generate custom user-directed wrapper libraries. These wrapper libraries may be pre-loaded at runtime or re-linked to create an

Figure 6: Hybrid profiling on FLASH showing event paths.

instrumented binary using linker-based instrumentation as described above. However, manually building these libraries may prove to be cumbersome. We created a TAU tool, `tau_gen_wrapper`, to automate generation of wrapper libraries. It takes an interface declaration of a library in the form of a header file and generates a wrapper library for TAU instrumentation. It uses the PDT static analysis tool to parse the header file and generates for each routine, a complete representation of its signature. A signature consists of the return type, the routine name, and a list of arguments. Each argument, in turn, consists of the argument type and an optional argument name. Internally, the tool invokes the parser TAU's wrapping tool, `tau_wrap` and builds the instrumented source code emitted by it. The wrapper generator tool allows the user to choose pre-processor, runtime preloading, or linker-based instrumentation techniques for creating a wrapper library.

4.4.2 GCRM

The Global Cloud Resolving Model (GCRM) being developed by Randall et al [??] will model climate on the entire globe at a horizontal grid spacing of at least 4km and vertical dimension on the order of 256 layers resulting in over 10 billion cells. A single cell-based variable written in single precision will require approximately 43 GB of disk storage. Corner data will require 85 GB and edge data 128 GB. A single snapshot of history data will require 1.8 TB of storage as currently configured. Climate scientists will want to write data as frequently as possible (down to the order of minutes) while maintaining an IO cost below 10% of the overall simulation. Obviously, the efficiency of the I/O is of critical importance.

Figure 7: Reverse event/call path view of FLASH hybrid profile.

4.5 Profiling GCRM

Understanding and optimizing the behavior of the I/O system for an application is difficult for several reasons. First there are several layers in the I/O stack, some of which are proprietary software. Second, there are many options for controlling these layers varying from optional arguments, to hints to alternative APIs. Third, there are often multiple implementations of some of the layers. Figure 8 shows the layers and alternatives considered for GCRM. It is still critical to be able to profile all the layers of the GCRM I/O in order to determine where the true bottlenecks reside. TAU provides the capabilities both to look deep into the various API layers and to organize and analyze the numerous configurations under evaluation.

We ran GCRM with TAU on the CRAY systems at NERSC using PNetCDF, the Cray MPI-IO library, and Lustre. The whole application was profiled with different phases of execution analyzed: initialization, I/O, and the numerical model itself. This enabled us to look at performance of each phase in its context and see the relative cost of each phase. Figure 9 shows the phase profile from our initial experiments.

Profile runs tend to be of very short duration, which causes an over-representation of the Init phase. The interesting take-away from this particular run is that I/O is taking a reasonable amount (8%) time relative to the numerical model itself. Right clicking on the I/O phase column and requesting the phase detail will result in the I/O phase detail shown in Figure 9 (right). In these early runs, we were clearly able to see that the collective open calls were taking significant time and that it was not due to the cost of POSIX open. Sharing these profiles with Cray engineers led to a detailed analysis of the cost of `MPI_file_open` and resulted in several changes to the Cray library that will be included with the next release.

A final important feature provided by the new TAU I/O profiling capability is a summary of the read and write sizes and read and write bandwidths per processor. An example is shown in

Figure 8: The GCRM I/O stack and some of the variations that make profiling a complex task.

Figure 9: Left: The initial TAU profile when using TAU’s phase capability. The first column represents the computational phase, the second column represents the initialization phase, and the third column represents the I/O phase. Right: I/O phase detail with columns MPI_Write_all, MPI_File_open, POSIX I/O write, and everything else. I/O aggregators are clearly identifiable.

Figure 10. Here we can see that we are successfully writing large chunks of data and that the mean is also large. Per-processor bandwidth and mean bandwidth can also be seen.

4.6 OS Performance Measurement and Analysis

In the first year of the FastOS-2 project, we further developed the KTAU (Kernel Tuning and Analysis Utilities) system-level performance measurement methodology and tool infrastructure to bridge the gap between the application and the OS/R components. KTAU couples kernel-level performance measurements with application-level events to provide an integrated parallel performance view from both kernel-wide and process-centric perspectives. Our specific Year 1 effort had the goal to port KTAU to the IBM BG/P at Argonne and demonstrate it integrated with TAU on a parallel MPI application. While our Year 1 goal was achieved, the KTAU source-level implementation was a non-starter for future use. In the final project period, we refocused our KTAU work on a course that leverages broader Linux development efforts. This work on *KTAU-ng* is presented below.

Name	Total	NumSampl...	MaxValue	MinValue	MeanValue	Std. Dev.
MPI_File_open()						
MPI_File_read_at()						
MPI_File_write_all()						
write()						
Bytes Written	10,123,013...	10,977	1,048,576	4	922,202.241	340,391.19
Bytes Written <file=../data80/exner_19010101_000000.nc>	415,237,696	400	1,048,576	8	1,038,094.24	104,220.35
Bytes Written <file=../data80/grapeL_mmr_19010101_000000.nc>	419,432,016	404	1,048,576	8	1,038,198....	103,707.722
Bytes Written <file=../data80/geopotential_19010101_000000.nc>	419,432,016	404	1,048,576	8	1,038,198....	103,707.722
Bytes Written <file=../data80/grid.nc>	102,760,712	313	1,048,576	4	328,308.984	461,889.127
Bytes Written <file=../data80/heat_flux_vdiff_19010101_000000.nc>	415,237,696	400	1,048,576	8	1,038,094.24	104,220.35
Bytes Written <file=../data80/cloud_ice_19010101_000000.nc>	419,432,016	404	1,048,576	8	1,038,198....	103,707.722
Bytes Written <file=../data80/cloud_water_19010101_000000.nc>	419,432,016	404	1,048,576	8	1,038,198....	103,707.722
Bytes Written <file=../data80/vorticity_19010101_000000.nc>	415,237,696	400	1,048,576	8	1,038,094.24	104,220.35
Bytes Written <file=../data80/swinc_19010101_000000.nc>	4,194,336	8	1,048,576	8	524,292	523,764.258
Bytes Written <file=../data80/temperature_19010101_000000.nc>	419,432,016	404	1,048,576	8	1,038,198....	103,707.722
Bytes Written <file=../data80/snow_mmr_19010101_000000.nc>	419,432,016	404	1,048,576	8	1,038,198....	103,707.722
Bytes Written <file=../data80/rel_vorticity_19010101_000000.nc>	415,237,696	400	1,048,576	8	1,038,094.24	104,220.35
Bytes Written <file=../data80/pressure_19010101_000000.nc>	419,432,016	404	1,048,576	8	1,038,198....	103,707.722
Bytes Written <file=../data80/rain_mmr_19010101_000000.nc>	419,432,016	404	1,048,576	8	1,038,198....	103,707.722
Bytes Written <file=../data80/qlr_19010101_000000.nc>	4,194,336	8	1,048,576	8	524,292	523,764.258
Bytes Written <file=../data80/mass_19010101_000000.nc>	415,237,696	400	1,048,576	8	1,038,094.24	104,220.35
Bytes Written <file=../data80/heating_lw_19010101_000000.nc>	419,432,016	404	1,048,576	8	1,038,198....	103,707.722
Bytes Written <file=../data80/heating_sw_19010101_000000.nc>	419,432,016	404	1,048,576	8	1,038,198....	103,707.722
Bytes Written <file=../dev/kdreg>	32,768	1,024	32	32	32	0
Bytes Written <file=../data80/wtr_flux_vdiff_19010101_000000.nc>	415,237,696	400	1,048,576	8	1,038,094.24	104,220.35
Bytes Written <file=../data80/wind_crn_ns_19010101_000000.nc>	830,472,200	794	1,048,576	8	1,045,934....	52,534.087
Bytes Written <file=../data80/wind_crn_ew_19010101_000000.nc>	830,472,200	794	1,048,576	8	1,045,934....	52,534.087
Bytes Written <file=../data80/wind_19010101_000000.nc>	1,245,708....	1,190	1,048,576	8	1,046,813....	42,929.966
Bytes Written <file=../data80/water_vapor_19010101_000000.nc>	419,432,016	404	1,048,576	8	1,038,198....	103,707.722
Write Bandwidth (MB/s) <file=../data80/heating_lw_19010101_000000.nc>		404	771.012	0.078	525.008	93.257
Write Bandwidth (MB/s) <file=../data80/heat_flux_vdiff_19010101_000000.nc>		400	762.047	0.084	521.865	101.895

Figure 10: Processor 0 Context Event Window showing write sizes and bandwidths.

4.6.1 Background

KTAU-ng is a Linux kernel module for creating performance data. KTAU-ng builds upon work of KTAU, LTT-ng, Oprofile, SystemTap, and D-Trace. The goals of our work differs from these tools in the following respects:

1. Flexibility to produce performance traces or profiles
2. Seamless integration with TAU
3. Ability to provide kernel-level performance information on production HPC-class system without compromising stability or security
4. Easy installation and operation

These particular goals have resulted in a feature set different from the tools above, and one that also paritally dictated an architecture for development.

4.6.2 KTAU-ng architecture design

The KTAU-ng architecture is shown in Figure 11. There are several things to point out. First, whereas the original KTAU used kernel patches to directly insert instrumentation points into the kernel source, KTAU-ng uses the tracepoint and Kprobes mechanisms added into the mainline Linux kernel. Tracepoints are statically-instrumented points in the kernel which have the advantage of low overhead when disabled. Conversely Kprobes allow dynamic instrumentation of nearly any location in the kernel, but but have higher overhead since they trigger a software interrupt and use traditional trampolining.

Second, KTAU-ng's functionality is restricted to a kernel module and a set of user-space utilities. It requires root access only to install, which is a necessary consequence of using tracepoints.

Figure 11: KTAU architecture showing the division of user-level and system-level development.

However, users can query performance application (about processes they start) without requiring super user access.

Third, one of the prominent features of KTAU-ng is its emphasis on per-process recording buffers – as opposed to the per-instrumentation point buffers user by other kernel performance measurement systems. Since KTAU-ng has been designed to be run solely as a Linux kernel module these recording buffers could not be added to the Linux kernel’s *task_struct*, which would require a kernel patch. Instead, we constructed a hash structure based on processor ID (pid) and process namespace. Using the task struct pointer stored at the bottom of the processes stack frame we can quickly locate the the necessary information to look-up the buffer for a given process, record the data, and continue executing the process. Since access to this hash could be performed simultaneously for multiprocessor systems we needed a locking structure that would allow for simultaneous access to the process structure in an atomic way. This was achieved by using the Read-Concurrent-Update mechanism.

Lastly, because KTAU-ng can potentially create large volumes of performance information it is necessary to insure that the recording of this performance information does not influence the results. To this end, KTAU-ng has been designed to transfer the data recorded by the kernel into user-space without having to copy the data while still insuring that the kernel does not export potentially sensitive information or read any data that might poison the integrity of the kernel.

4.6.3 Using KTAU-ng

KTAU maintains a per-process vantage point to allow instrumentation to be selected on a per process basis at run time. When instrumentation is no longer needed because the processes that it was specified for finishes, or is otherwise told to stop recording performance data, it is automatically disabled from this system.

While KTAU provides a growing set of instrumentation point to allow end pusers to easily evaluate system performance, specific hardware may require instrumentation beyond that which is provided by the KTAU system. Fortunately an API has been developed for the easy creation

and installation of instrumentation points. These points can be then registered into the KTAU infrastructure (at run time) and added into the performance data produced by KTAU.

Central to KTAU-ng's feature set is the ability to merge profiles and traces produced by TAU with those produced by KTAU, to this end the profiles and traces produced by KTAU-ng use the same format as TAU and will be easily mergable the profiles and traces produced by TAU.

5 Preliminary Results

Presently, KTAU-ng is still a work in progress. We have done preliminary testing to validate the functionality of profiling with a set of tracepoint instrumentation, including system calls, IRQs, and scheduling events. Once the profiling framework is robust and more instrumentation points have been enabled, we will develop the KTAU-ng tracing support.

One of the tests being used for KTAU-ng is a multi-process matrix multiplication (matmult) benchmark. Figure 12 shows a ParaProf display of one process from a matmul execution. Here we see merged application and system profiles. Notice the application-level events and the system-level events. If we look at the four-process matmult profile in Figure 13, we see the clearly the effects of the kernel swapping out the processes for another processes. This is happening because the matmult application is being run on a two-core machine

Figure 12: ParaProf display of one matmult process showing application and kernel events.

Figure 13: Full profile display of four-process matmult execution on two cores.

6 Future direction

As KTAU-ng continues to be developed, various system performance questions will be able to be addressed. In particular we believe KTAU-ng will hold great promise for analyzing performance on systems taking advantage of virtualization and those that utilize GPUs. The overhead of these two increasingly prevalent technologies is difficult to analyze with purely user-space utilities and we hope that KTAU-ng will help answer some of these questions.

7 Meetings

Throughout the FastOs-2 project, we participated in many external meetings we presented our research and development accomplishments. In addition to those presented in the Year 1 and Year 2 progress reports, the meetings in the final period included:

- **SC (Supercomputing) 2010**
 - TAU demonstrations in DOE NNSA ASC booth.
 - Tutorial, “Hands-On Practical Parallel Application Performance Engineering using PAPI, PerfSuite, Scalasca, Vampir and TAU”
- **DOE SciDAC CScADS workshops**
 - Performance Tools for Petascale Computing, August 2-5, 2010
 - Performance Tools for Extreme Scale Computing, August 1-4, 2011
- **Argonne Leadership Class Facility workshops**
 - INCITE Getting Started Workshop, January, 2010
 - Leap to Petascale, May 18-20, 2011
 - Leap to Petascale, June 7-9, 2011
- **DOE ACTS Collection Workshop**, High Performance Software Tools to Fast-Track The Development of Scalable and Sustainable
 - Tutorial, “TAU Performance System,” August 17-20, 2010
 - Tutorial, “TAU Performance System,” August 16-19, 2011

References

1. A. Nataraj, A. Malony, A. Morris, D. Arnold, and B. Miller. TAUoverMRNet (ToM) : A Framework for Scalable Parallel Performance Monitoring. In *International Workshop on Scalable Tools for High-End Computing (STHEC '08)*, 2008.
2. A. Nataraj, A. Malony, S. Shende, and A. Morris. Integrated Parallel Performance Views. *IEEE Cluster Computing Journal*, 11(1):57–73, Mar. 2008.
3. A. Malony, S. Shende, A. Morris, S. Biersdorff, W. Spear, K. Huck, and A. Nataraj. Evolution of a Parallel Performance System. *2nd International Workshop on Tools for High Performance Computing*, M. Resch, R. Keller, V. Himmler, B. Krammer, and A. Schulz, Eds., Stuttgart, Germany, Springer, pp. 169–190, July, 2008.
4. A. Nataraj, A. Malony, A. Morris, D. Arnold, and B. Miller. In Search of Sweet-Spots in Parallel Performance Monitoring. In *IEEE International Conference on Cluster Computing (Cluster 2008)*, Sept. 2008.
5. A. Morris, W. Spear, A. Malony, S. Shende. Observing Performance Dynamics using Parallel Profile Snapshots. In *14th International Euro-Par Conference on Parallel Processing (Euro-Par 2008)*, Aug. 2008.
6. ZeptoOS V2.0 Documentation – Using (K)TAU on ZeptoOS
[http://wiki.mcs.anl.gov/zeptoos/index.php/\(K\)TAU](http://wiki.mcs.anl.gov/zeptoos/index.php/(K)TAU)
7. H. Jagode, J. Dongarra, S. Alam, J. Vetter, W. Spear, A. Malony, A Holistic Approach for Performance Measurement and Analysis for Petascale Applications. *International Conference on Computational Science (ICCS)*. Eds. G. Allen et al., Springer-Verlag. LNCS 5545, Part II, 686–695, May, 2009. (Joint Workshop: Tools for Program Development and Analysis in Computational Science and Software Engineering for Large-Scale Computing).
8. M. Geimer, S. Shende, A. Malony, F. Wolf. A Generic and Configurable Source-Code Instrumentation Component. *International Conference on Computational Science (ICCS)*. Eds. G. Allen et al., Springer-Verlag. LNCS 5544, Part II, 696–705, May, 2009. (Joint Workshop: Tools for Program Development and Analysis in Computational Science and Software Engineering for Large-Scale Computing).
9. S. Biersdorff, C.W. Lee, A. Malony, and L. Kale. Integrated Performance Views in Charm++: Projections Meets TAU. *International Conference on Parallel Processing (ICPP)*, September, 2009.
10. A. Nataraj, A. Malony, A. Morris, D. Arnold, and B. Miller. A Framework for Scalable, Parallel Performance Monitoring. *Concurrency and Computation: Practice and Experience*. Vol. 22, No. 6, pp. 720–735, December, 2009. (special issue on Scalable Tools for High-End Computing).
11. A. Morris, A. Malony, and S. Shende. Design and Implementation of a Hybrid Parallel Performance Measurement System. *International Conference on Parallel Processing (ICPP)*, September 2010.
12. Global Cloud Resolving Model, 2011
<http://kiwi.atmos.colostate.edu/gcrm/>

13. S. Shende, A. Malony, W. Spear, and K. Schuchardt. Characterizing I/O Performance using the TAU Performance System. *International Conference on Parallel Computing (ParCo)*, September 2011.