The PAPI Cross-Platform Interface to Hardware Performance Counters

Kevin London, Shirley Moore, Philip Mucci, and Keith Seymour
University of Tennessee-Knoxville
{london, shirley, mucci, seymour}@cs.utk.edu

Richard Luczak
University of Tennessee-Knoxville/ASC MSRC
luczakr@asc.hpc.mil

Abstract

The purpose of the PAPI project is to specify a standard API for accessing hardware performance counters available on most modern microprocessors. These counters exist as a small set of registers that count "events", which are occurrences of specific signals and states related to the processor's function. Monitoring these events facilitates correlation between the structure of source/object code and the efficiency of the mapping of that code to the underlying architecture. This correlation has a variety of uses in performance analysis and tuning. The PAPI project has developed a standard set of hardware events and a standard cross-platform library interface to the underlying counter hardware. The PAPI library has been implemented for a number of Shared Resource Center platforms. The PAPI project is developing end-user tools for dynamically selecting and displaying hardware counter performance data. PAPI support is also being incorporated into a number of third-party tools.

Introduction

For years collecting performance data on applications programs has been an imprecise art. The user has had to rely on timers with poor resolution or granularity, imprecise empirical information on the number of operations performed in the program in question, vague information on the effects of the memory hierarchy, etc. Today hardware counters exist on every major processor platform. These counters can provide application developers valuable information about the performance of critical parts of the application and point to ways for improving performance. Performance tool developers can use these hardware counters as a basis for tools and interfaces that provide users with insight into diagnosing and fixing performance problems. The problem facing users and tool developers is that access to these counters has previously been poorly documented, unstable or unavailable to the user level program. The focus of the PAPI project is to provide an easy to use, common set of interfaces that will gain access to these performance counters on all major processor platforms, thereby providing application developers the information they need to tune their software on different platforms [1, 2]. The goal is to make it easy for users to gain access to the counters to aid in performance analysis, modeling, and tuning.

The approach of the PAPI project has been to work with the high performance computing community, including users and vendors, to choose a common set of hardware events and to define a cross-platform library interface to the underlying counter hardware. The common set of events are those considered to be most relevant and useful in tuning application performance. As large a subset as possible has been mapped to the corresponding machine-specific events on the major HPC platforms and it is hoped that most of these events will be made available in the future on all major HPC platforms to improve the capability for tuning applications across multiple platforms. The intent is that the same metric would count similar, and possibly comparable, events on different platforms. Direct comparison between systems is *not* the intention of the PAPI predefined events. Rather, the intention is to standardize the names for the metrics, not the exact semantics, which necessarily depend on the processor under study. A detailed description of the PAPI metrics and how they can be used for application performance tuning may be found in [1].

The PAPI library provides two interfaces to the underlying counter hardware: a high level interface for the acquisition of simple measurements, and a fully-programmable, thread-safe low level interface directed towards users with more sophisticated needs. The high level interface simply provides the capability to start, stop, and read the counters for a specified list of events. The target audience for the high level interface is application engineers and benchmarking teams looking to quickly and simply acquire some rudimentary performance measurements. The tool developer will likely find the high level interface too restrictive. The low level interface provides the more sophisticated functionality of user callbacks on counter overflow and hardware based SVR4 compatible profiling, regardless of whether or not the operating system supports it. These features provide the necessary basis for any source level performance analysis software. Thus, for any architecture with even the most rudimentary access to hardware performance counters, PAPI provides the foundation for truly portable, source level performance analysis tools based on real processor statistics.

PAPI Architecture and Design

The PAPI architecture uses a layered approach, as shown in Figure 1. Internally, the PAPI implementation is split into portable and machine-dependent layers. The topmost portable layer consists of the high and low level PAPI interfaces. This layer is completely machine independent and requires little porting effort. It contains all of the API functions as well as numerous utility functions that perform state handling, memory management, data structure manipulation, and thread safety. In addition, this layer provides advanced functionality not always provided by the operating system, namely event profiling and overflow handling. The portable layer calls the substrate, the internal PAPI layer that handles the machine-dependent specifics of accessing the counters. The substrate uses whatever methods appropriate to facilitate counter access, whether that be register level operations (T3E), customized system calls (Linux/x86), or calls to another library (AIX 4.3). The substrate interface and functionality are well defined, leaving most of the code free from conditional compilation directives. For each architecture/operating system pair, only a new substrate layer needs to be written.

Experience indicates that no more than a few weeks are required to generate a fully functional substrate for a new platform, if the operating system provides the necessary support for accessing the hardware counters. Substrates have been implemented for Pentium Pro/II/III and P6 on Linux, SGI/MIPS R10000/R12000 on IRIX 6.x, IBM Power 604/604e/630 on AIX 4.3, Cray T3E/EV5 on Unicos/mk, and Sun Ultra on Solaris 8. Work is underway on the Compaq Alpha Tru64 Unix, Linux IA-64, and Microsoft Windows substrates. For each of the substrates, as many of the PAPI predefined events as possible have been mapped to the hardware counter events available on the given processor, sometimes deriving a PAPI event from a combination of available hardware events. A test program is available that outputs which PAPI events are available and how they are mapped on a given platform.

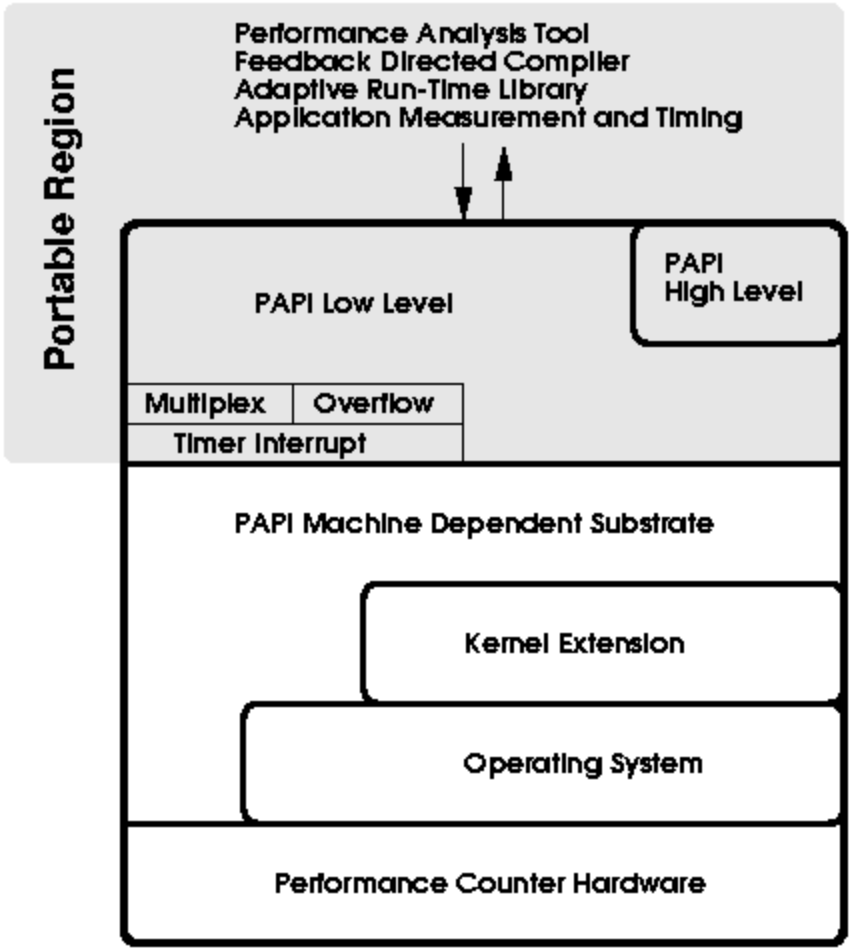

Figure 1. PAPI Architecture

PAPI provides an abstraction from particular hardware events called *EventSets.* An EventSet consists of events that the user wishes to count as a group. There are two

reasons for this abstraction. The first reason is efficiency in accessing the counters through the operating system. Most operating systems allow the programmer to move the counter values in bulk without having to make a separate system call for each counter. By exposing this grouping to the user, the PAPI library can greatly reduce its overhead when accessing the counters. This efficiency is especially important when PAPI is used to measure small regions of code inside loops with large iteration counts. The second reason for *EventSets* is that users can evolve their own specialized counter groupings specific to their application areas. More often than not, relevant performance information is obtained by relating different metrics to one another. For example, the ratio of loads to level 1 cache misses is often the dominant performance indicator in dense numerical kernels. EventSets are managed by the user via integer handles, which simplifies inter-language calling interfaces. Multiple EventSets may be used simultaneously and may share counters.

Most modern microprocessors have a very limited number of events than can be counted simultaneously. This limitation severely restricts the amount of performance information that the user can gather during a single run. As a result, large applications with many hours of run time may require days or weeks of profiling in order to gather enough information on which to base a performance analysis. This limitation can be overcome by multiplexing the counter hardware. By subdividing the usage of the counter hardware over time, multiplexing presents the user with the view that many more hardware events are countable simultaneously. This unavoidably incurs a small amount of overhead and can adversely affect the accuracy of reported counter values. Nevertheless, multiplexing has proven useful in commercial kernel level performance counter interfaces such as SGI's IRIX 6.x. Hence, on platforms where the operating system or kernel level counter interface does not support multiplexing, PAPI provides the capability to multiplex in software through the use of a high resolution interval timer. However, multiplexing must be explicitly enabled by the user.

One of the most significant features of PAPI for the tool writer is its ability to call user-defined handlers when a particular hardware event exceeds a specified threshold. For systems that do not support counter overflow at the operating system level, this is accomplished by setting up a high resolution interval timer and installing a timer interrupt handler. PAPI handles the signal by comparing the current counter value against the threshold. If the current value exceeds the threshold, then the user's handler is called from within the signal context with some additional arguments. These arguments allow the user to determine which event overflowed, how much it overflowed, and at what location in the source code. Statistical profiling is built upon the above method of installing and emulating arbitrary callbacks on overflow. Profiling works as follows: when an event exceeds a threshold, a signal is delivered with a number of arguments. Among those arguments is the interrupted thread's stack pointer and register set. The register set contains the program counter, the address at which the process was interrupted when the signal was delivered. Performance tools such as UNIX `prof` extract this address and hash the value into a histogram. At program completion, the histogram is analyzed and associated with symbolic information contained in the executable. What results is a line-by-line account of where counter overflow occurred in

the program. Most existing profilers use process time as the overflow trigger. PAPI generalizes this functionality so that a histogram can be generated using any countable event as the basis for analysis.

End-user Tools

While PAPI provided access to the counters, it still requires users to instrument their code with calls to PAPI. Also, changing the metric(s) being monitored requires changing the code, recompiling and running the application again. For these reasons, a number of end-users tools have been developed for PAPI that provide automatic or semi-automatic instrumentation. In addition, these tools provide a graphical display of the collected performance data that assists the user in analyzing and interpreting the data[4].

The *perfometer* tool developed by the PAPI project provides a real-time graphical view of hardware performance counter data, allowing users to quickly see where performance bottlenecks are in their applications. Only one function call has to be added to the source code to take advantage of perfometer. This makes it quick and simple to add and remove instrumentation from a program. Also, perfometer allows users to change the event they are monitoring during execution. Add the ability to monitor parallel applications, set alarms and a Java front-end that can run anywhere, and this gives the user a powerful tool for quickly discovering where and why a bottleneck exists. Also, perfometer can monitor one or more processes in a heterogeneous parallel programming environment making it a valuable tool in clustered and grid computing environments

Perfometer has three different portions the GUI, the server and the library. The server runs on any platform that supports pthreads and the GUI runs on any platform that supports Java 1.2 or higher, while the library runs on every platform that PAPI supports. The library is layered on top of PAPI and supports both Fortran and C programs. All three portions communicate using TCP sockets so they need not be on the same machine. If monitoring just one application the server is not needed and the GUI can connect directly to the application as seen in figure 2. However, if monitoring more than one application the GUI has to connect to the server, which is responsible for filtering any information and passing the performance data back to the GUI as seen in figure 3.

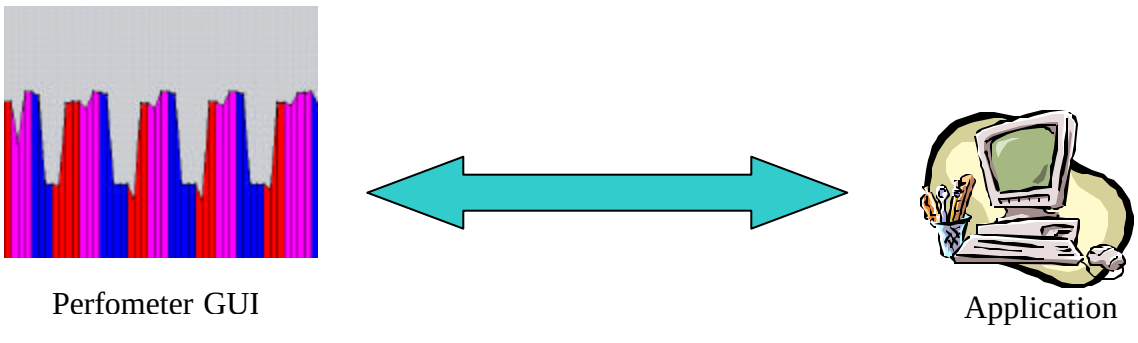

Figure 2: Perfometer being used to connect to a single process application

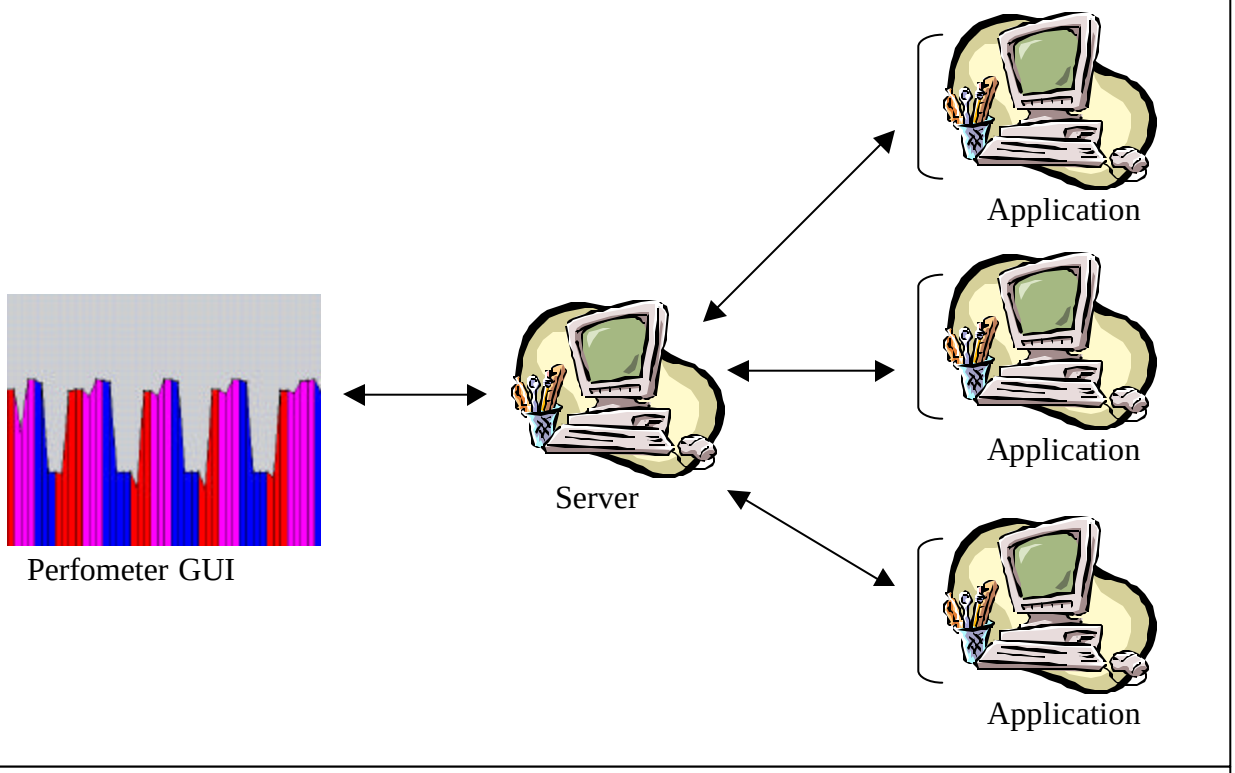

Figure 3: Perfometer being used to monitor multiple applications.

The graphical user interface as seen in figure 4 is where all the performance information is displayed in a graph. Current work involves extending the graphical interface to display performance data for multithreaded and multiprocess programs in a scalable manner. Currently data for multiple processes are displayed as shown in Figure 5. Any process can be selected for the larger display by clicking on its representation in the multiprocess display.

PAPI has been incorporated into a number of third-party research and commercial tools, including SvPablo from University of Illinois and IBM [3], DEEP/MPI from Pacific Sierra Research, and TAU from the University of Oregon [5]. In addition, PAPI support is being incorporated into the VAMPIR and KAPPro tools already available on some Shared Resource Center machines.

VAMPIR is a performance analysis tool for MPI parallel programs developed by Pallas in Germany. The next version of VAMPIR will support OpenMP in addition to MPI and will use PAPI to access hardware performance counters. With PAPI, VAMPIR will be able to use hardware counter data to guide detailed event based analysis which will allow

users to quickly locate performance bottlenecks in their applications. VAMPIR users previously were able only to scan the visualization of a trace file and attempt to correlate events in that view with cumulative statistics displays. Displaying various PAPI metrics alongside the VAMPIR timeline view will allow users to quickly pinpoint the location of performance bottlenecks.

The **KAPPro** toolkit, developed by Intel/KAI for OpenMP programming, consists of the Assure debugger, the Guide OpenMP compiler, and the GuideView OpenMP performance analysis tool. Pallas and Intel/KAI are developing a new performance analysis tool set for combined MPI and OpenMP programming which uses PAPI to access the hardware performance counters. PAPI's standard performance metrics, which include metrics for shared memory processors (SMPs), will provide accurate and relevant performance data for the clustered SMP environments targeted by the new tool set.

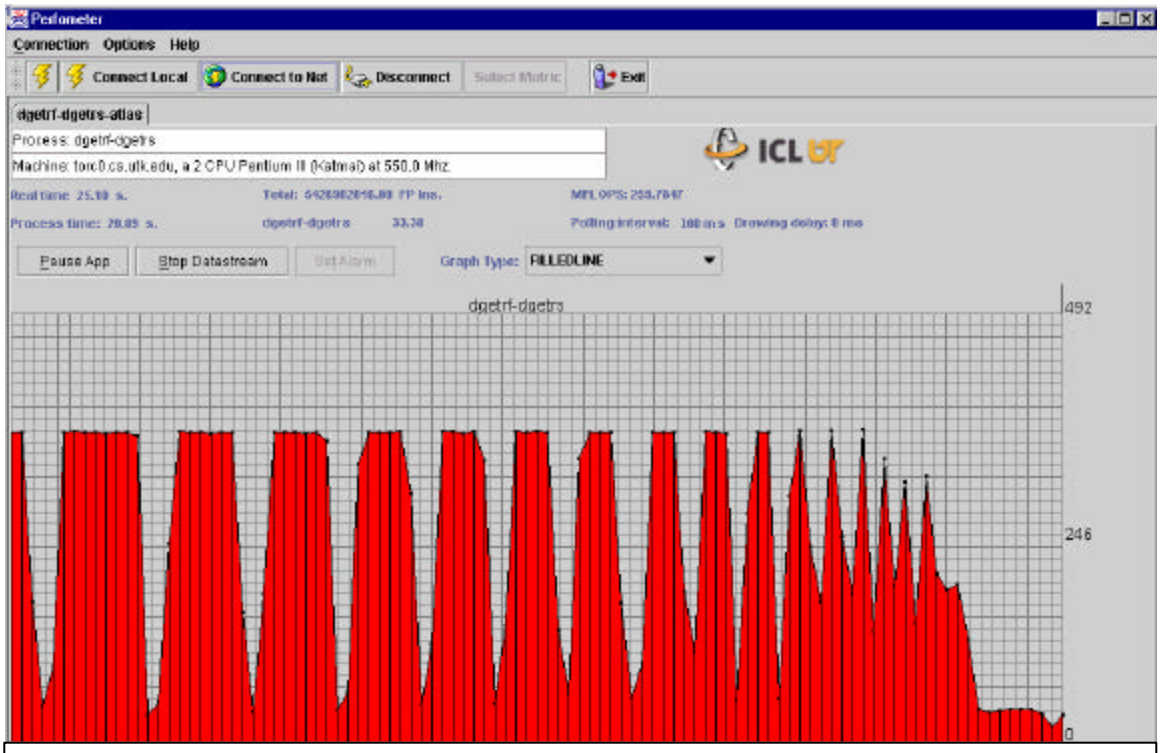

Figure 4: Example of perfometer monitoring a single application

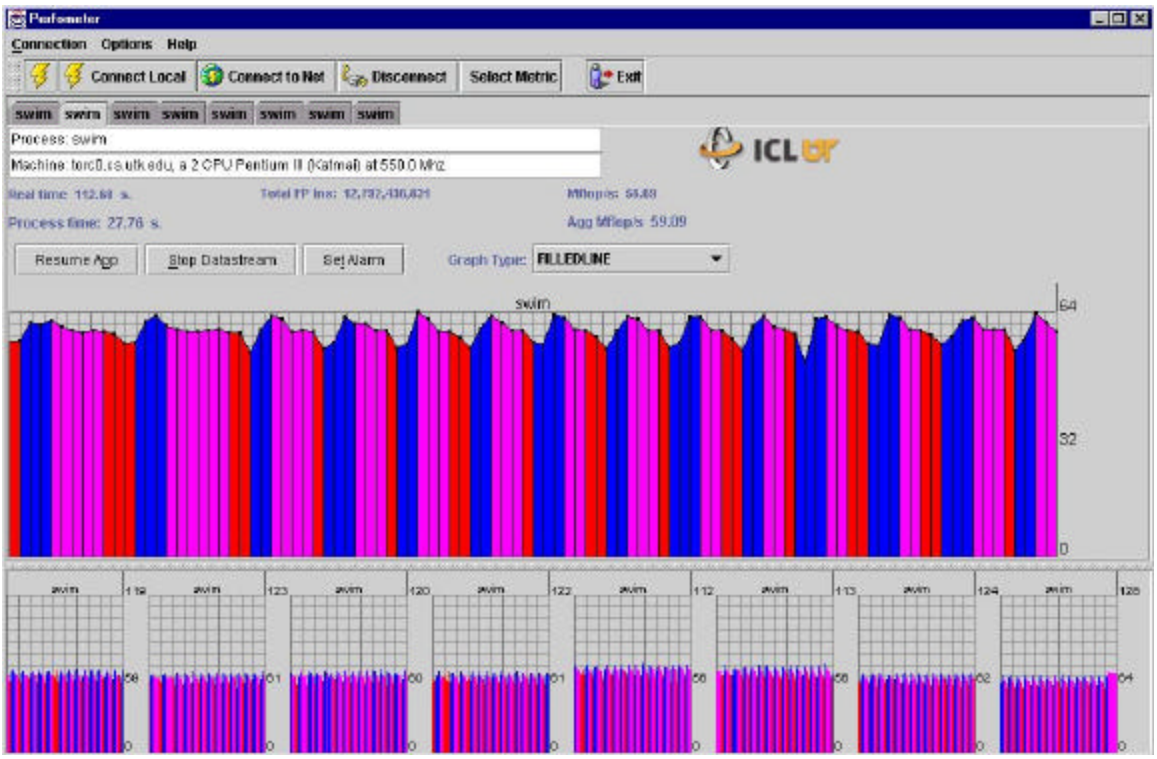

Figure 5. Example of perfometer monitoring a parallel application

Conclusions and Future Work

PAPI has been approved by the ASC MSRC Configuration Control Board for official installation on HPC systems at ASC MSRC, and installation is proceeding on the SGI Origin 2000 and the IBM Power3 SMP. The port to the Compaq Alpha is awaiting a bug fix to a device driver from Compaq. PAPI is being used for performance evaluation of SIP CHSSI codes at ASC.

PAPI has been installed on the ERDC MSRC Cray T3E and Origin 2000 and is being used for performance evaluation of EQM codes. Installation is planned on the IBM Power3 SMP.

In addition to the instruction counts, cache and memory access events, and SMP cache coherence events currently monitored by PAPI, the next version of PAPI will support cross-platform collection of memory utilization information. This information will be used to analyze the code for the DoD Challenge project "Hybrid Particle Simulation of High Altitude Nuclear Explosions in 3-D" led by Dr. Stephen Brecht. The memory utilization extension is expected to be generally useful to other application developers.

References

[1] S. Browne, J. J. Dongarra, N. Garner, G. Ho and P. Mucci, *A Portable Programming Interface for Performance Evaluation on Modern Processors*, International Journal of High Performance Computing Applications, 14:3 (Fall 2000), pp. 189-204.

[2] S. Browne, J. J. Dongarra, N. Garner, K. London and P. Mucci, *A Scalable Cross-Platform Infrastructure for Application Performance Optimization Using Hardware Counters*, *Proc*. *SC'2000*, Dallas, Texas, November 2000.

[3] L. DeRose and D. A. Reed, *SvPablo: A Multi-language Performance Analysis System*, *Proc. 1999 International Conference on Parallel Processing*, September 1999, pp. 311-318.

[4] K. London, J. J. Dongarra, S. V. Moore, P. Mucci, K. Seymour and T. Spencer, *End-user Tools for Application Performance Analysis Using Hardware Counters*, *Proc*. *International Conference on Parallel and Distributed Computing Systems*, Dallas, Texas, August 2001 (to appear).

[5] A. D. Malony, *TAU: Tuning and Analysis Utilities*, 2001, http://www.cs.uoregon.edu/research/paracomp/tau/.