Using PAPI for hardware performance monitoring on Linux systems

Jack Dongarra, Kevin London, Shirley Moore, Phil Mucci, and Dan Terpstra
Innovative Computing Laboratory, *University of Tennessee, Knoxville, TN*

Abstract

PAPI is a specification of a cross-platform interface to hardware performance counters on modern microprocessors. These counters exist as a small set of registers that count events, which are occurrences of specific signals related to a processor's function. Monitoring these events has a variety of uses in application performance analysis and tuning. The PAPI specification consists of both a standard set of events deemed most relevant for application performance tuning, as well as both high-level and low-level sets of routines for accessing the counters. The high level interface simply provides the ability to start, stop, and read sets of events, and is intended for the acquisition of simple but accurate measurement by application engineers. The fully programmable low-level interface provides sophisticated options for controlling the counters, such as setting thresholds for interrupt on overflow, as well as access to all native counting modes and events, and is intended for third-party tool writers or users with more sophisticated needs.

PAPI has been implemented on a number of platforms, including Linux/x86 and Linux/IA-64. The Linux/x86 implementation requires a kernel patch that provides a driver for the hardware counters. The driver memory maps the counter registers into user space and allows virtualizing the counters on a per-process or per-thread basis. The kernel patch is being proposed for inclusion in the main Linux tree. The PAPI library provides access on Linux platforms not only to the standard set of events mentioned above but also to all the Linux/x86 and Linux/IA-64 native events.

PAPI has been installed and is in use, either directly or through incorporation into third-party end-user performance analysis tools, on a number of Linux clusters, including the New Mexico LosLobos cluster and Linux clusters at NCSA and the University of Tennessee being used for the GrADS (Grid Application Development Software) project.

1 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 the performance. Performance tool developers can use these hardware counters to develop tools and interfaces that users can insert into their applications. The current

problem facing tool developers is that access to these counters is poorly documented, unstable or unavailable to the user level program. The focus of PAPI 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 may need to tune their software on different platforms. Our goals are to make it easy for users to gain access to the counters to aid in performance analysis, modeling, and tuning.

PAPI provides two interfaces to the underlying counter hardware: a simple, 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 low level interface manages hardware events in user-defined groups called *EventSets*. The high level interface simply provides the ability to start, stop and read the counters for a specified list of events. PAPI attempts to provide portability across operating systems and architectures wherever possible and reasonable to do so. PAPI includes a predefined set of events meant to represent a lowest common denominator of a 'good' counter implementation, the intent being that the same tool would count similar and possibly comparable events when run on different platforms. If the programmer chooses to use this set of standardized events, then the source code need not be changed and only a recompile is necessary. However, should the developer wish to access machine specific events, the low level API provides access to all available native events and counting modes.

In addition to raw counter access, PAPI 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.

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 like 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 through the use of a high resolution interval timer.

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 portions of code for accessing the counters.

More detailed information about the PAPI design and architecture may be found in [1, 2].

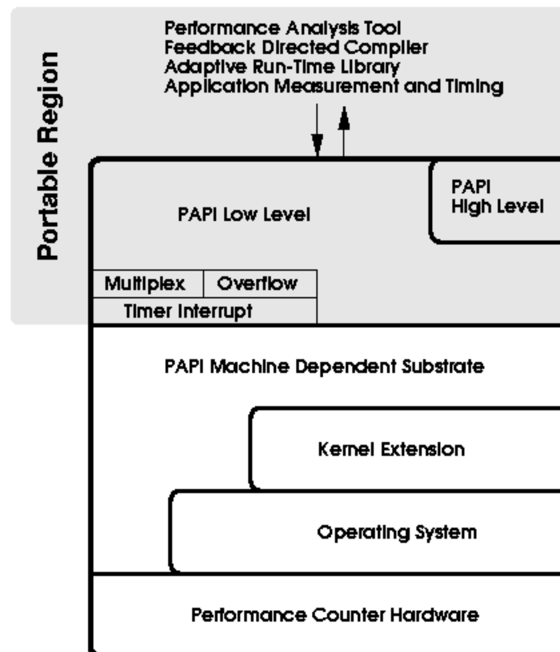

Figure 1. PAPI Architecture

2 Hardware counters

This section describes details of the hardware counters on some of the processors that run Linux.

2.1 IA-32 counters

The Pentium processor introduced model-specific performance monitoring counters to the Intel architecture[3]. These counters allow processor performance metrics to be monitored and measured. In the P6 family of processors, the performance counter mechanism has been enhanced to allow more events to be monitored and to allow greater control of event monitoring.

The Pentium and P6 processors both provide two 40-bit performance counters, allowing two events to be monitored simultaneously. These counters can either count events or measure duration. When counting events, a counter is incremented each time a specified event takes place or a specified number of events takes place. When measuring duration, a counter counts the number of processor clocks that occur while a specified condition is true. The counter can count events or measure durations that occur at any privilege level.

To allow user level access to the performance monitoring counters, the operating system needs to provide an event-monitoring device driver that includes procedures for initializing, starting, stopping, and reading the counters. The PAPI implementation for IA-32 Linux uses the perfctr Linux patch to enable access to the counters. See section 3 for details on the use of this patch.

The P6 family processors provide the option of generating a local APIC (Advanced Programmable Interrupt Controller) interrupt when a performance counter overflows. The primary use of this option is for statistical performance sampling. To use this option, the event monitor driver needs to provide an interrupt handler that saves context information, resets the counter, and returns from the interrupt. A profiling tool

can then read the information collected to analyze performance of the profiled application. See section 6 for details on how PAPI implements statistical profiling, either using the hardware support or by using software emulation where such hardware support is not provided.

The Pentium and P6 performance monitoring events are intended to be used as guides for performance tuning, and the counter values are not guaranteed to be absolutely accurate. See section 7 for a discussion of accuracy of hardware counter data.

Beginning with the Pentium processor, the Intel architecture defines a time stamp counter (TSC) that can be used to monitor the relative time of occurrence of processor events. The TSC as implemented in the Pentium and P6 family processors is a 64-bit counter that is set to zero following the hardware reset of the processor. Following reset, the counter is incremented every processor clock cycle. The RDTSC instruction reads the TSC and is guaranteed to return a monotonically increasing unique value whenever executed, except for 64-bit counter wraparound. The period for counter wrap is several thousands of years in the Pentium and P6 family processors. The RDTSC instruction is not serialized or ordered with respect to other instructions. The TSC is used as the basis of the PAPI real time timers, as described in more detail in section 5.

2.2 AMD Athlon counters

The AMD Athlon processor provides four 48-bit performance counters[]. These counters can either count events or measure duration. When counting events, a counter is incremented each time a specified event takes place or a specified number of events takes place. When measuring duration, a counter counts the number of processor clock cycles that occur while a specified condition is true. The counters can count events or measure durations that occur at any privilege level. The counters are not guaranteed to be fully accurate and should be used as a relative measure of performance to assist in application tuning.

The AMD Athlon processor provides the option of generating a debug interrupt when a performance monitoring counter overflows. The primary use of this option is for statistical performance sampling.

The AMD Athlon also provides a time stamp counter (TSC) and a RDTSC instruction that allows application code to read the TSC directly.

2.3 IA-64 counters

All Intel IA-64 processor implementations are to provide at least four performance counters and four performance counter overflow status registers[4]. Each counter can be configured to monitor events for any combination of privilege levels and one of several event metrics. All IA-64 processor models are required to provide at least two events: the number of retired instructions, and the number of processor clock cycles. Performance related events for the Itanium processor, the first IA-64 processor, are grouped into the following categories:

- Basic events: clock cycles, retired instructions
- Instruction execution: instruction decode, issue and execution, data and control speculation, and memory operations
- Cycle accounting events: stall and execution cycle breakdowns
- Branch events: branch prediction
- Memory Hierarchy: instruction and data caches
- System events: operating system monitors, instruction and data TLBs

These events are listed and described in Chapter 7: Performance Monitor Events of [4].

The Itanium processor has a three-level memory hierarchy. The instruction and the data stream work through separate L1 caches. The L1 data cache is a write-through cache. A unified L2 cache serves both the L1 instruction and data caches, and is backed by a large unified L3 cache. A large number of events are provided for measuring performance of this memory hierarchy.

On the Itanium processor, performance monitoring can be confined to a subset of all events. Events can be qualified for monitoring based on an instruction address range, a particular instruction opcode, a data address range, and/or the privilege level. The instruction address range check allows event monitoring to be constrained to a programmable instruction address range. This enables monitoring of dynamically linked libraries, functions, or loops of interest in a large IA-64 application. The data address range check allows event collection for memory operations to be constrained to a programmable data address range. This enables selective monitoring of data cache miss behavior of specific data structures.

To support profiling, performance monitor counts need to be associated with program locations. Event-based program counter sampling on IA-64 processors is supported by a dedicated performance monitor overflow interrupt mechanism. However, due to the super-scalar issue, deep pipelining, and out-of-order instructions completion of today's microprocessors, the sampled program counter value may not be related to the instruction address that caused an event. On the Itanium processor, the sampled program counter may be off by 48 dynamic instructions from the instruction that caused the event. If program counter sampling is used for event address identification on the Itanium processor, an event might be associated with an instruction almost five dynamic basic blocks away from where it actually occurred. Therefore, it is essential for hardware to precisely identify an event's address.

The Itanium processor provides a set of event address registers (EARs) that record the instruction and data addresses of data cache misses for loads, the instruction and data addresses of data TLB misses, and the instruction addresses of instruction TLB and cache misses. For example, the data cache EAR repeatedly captures the instruction and data addresses of actual data cache load misses. Whenever the counter overflows, event address collection is suspended until the EAR is read by the profiling software. Statistical sampling can achieve arbitrary event resolution by varying the number of events within an observation interval and by increasing the number of observation intervals.

IA-64 provides architected support for context switching of performance monitors by an IA64 operating system. If implemented by the operating system, this allows performance counter events to be broken down per thread or per process. To monitor processor events per thread/process, the operating system needs to save and restore performance monitor state at thread/process context switches. To monitor processor events system wide (across all processes and the operating system kernel), a monitor must be enabled continuously across all contexts. This can be achieved by configuring a privileged monitor.

3 Access to hardware counters under Linux

To gain access to the counters on Linux/x86 platforms, PAPI uses the Linux/x86 Performance Monitoring Counters Driver by Mikael Pettersson[5], otherwise known as perfctr. This package adds support to the Linux kernel for using the performance monitoring counters found in many modern x86-class processors. Supported processors are as follows:

- All Intel family 5 and 6 processors, i.e. Pentium, Pentium MMX, Pentium Pro, Pentium II, Celeron, and Pentium III.
- AMD K7 Athlon.
- Cyrix 6x86MX, MII, and III.
- WinChip C6, 2, 2A, 2B, and 3.

With perfctr, each Linux process has its own set of "virtual" counters. That is, to a process the counters appear to be private and unrelated to the activities of other processes in the system. The virtual counters have 64-bit precision, even though current processors only implement 40 or 48-bit counters in hardware. Each process also has a virtual Time-Stamp Counter (TSC). On most machines, the virtual counters can be sampled entirely in user-space without incurring the overhead of a system call. The driver also supports system-wide counters.

Starting with perfctr 1.8, preliminary interrupt-mode support using the local APIC interrupt is available for the P6. When an interrupt-mode virtual counter interrupts on overflow, the counters are suspended and a user-specified signal is sent to the process. The user's signal handler can read the trap pc from the memory

mapped virtual counter and should then issue an IRESUME ioctl to restart the counters. perfctr 2.0 will support buffering and automatic restart. PAPI does not yet make use of perfctr's counter overflow interrupt support but plans are to do so when PAPI is updated to use perfctr 2.0.

For access to the counters under IA-64 Linux, PAPI uses the latest IA-64 Linux patch available from http://www.kernel.org/.

4 Counter events available from PAPI

4.1 PAPI predefined events

Through interaction with the high performance computing community including vendors and users, the PAPI developers have chosen a set of hardware events deemed relevant and useful in tuning application performance. The complete list of PAPI predefined events can be found in the **papiStdEventDefs.h** file in the PAPI distribution. These events may differ in their actual semantics on different platforms, and all events may not be present on all platforms. However, 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 predefined events include accesses to the memory hierarchy, cache coherence protocol events, cycle and instruction counts, and functional unit and pipeline status. On each platform, PAPI maps as many of the predefined events as possible to native events on the platform. This mapping can either be one-to-one, or in the case of a derived event, one to many. For a derived event, the counts from one or more native events are combined using some operation, for example addition or subtraction. A utility program called **avail** is provided with the PAPI reference implementation that outputs which predefined events are available on a given platform and whether or not they are derived.

4.2 Access to native events

PAPI also provides access to native events on all supported platforms. Developers may need access to platform-specific events to tune specialized code. For example, the External Bus Logic (EBL) events available on the P6 might be useful for tuning a communication driver. The native event counter formats are explained in platform-specific README files in the papi/src directory of the PAPI distribution. The papi/src/tests subdirectory includes examples of the use of native events for the different platforms. The native event lists for the various platforms can be found in the processor architecture manuals[6, 3, 4].

5 Timers and system information

PAPI provides the routines **PAPI_get_real_cyc** and **PAPI_get_real_usec** for getting real time values. Both of these functions return the total real time passed since some arbitrary starting point. The time is returned in clock cycles or microseconds respectively. These calls are equivalent to wall clock time.

PAPI provides the routines **PAPI_get_virt_cyc** and **PAPI_get_virt_usec** for getting virtual time values. Both of these functions return the total number of virtual units from some arbitrary starting point. Virtual units accrue when the process is running in user-mode.

Both the real time and virtual time counters are guaranteed to exist on every platform PAPI supports. PAPI attempts to use the most accurate timer available on a given platform. For example, on the x86 platforms, PAPI uses the Time Stamp Counter (TSC) to obtain real time in clock cycles.

The **PAPI_get_executable_info** call returns the executable's address space information, such as the start and end addresses of the text, data, and bss segments. The **PAPI_get_hardware_info** call returns information about the hardware on which the program is running, such as the number of CPUs, CPU model information, and the cycle time of the CPU (which may be estimated with timings).

6 Statistical Profiling

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. This is accomplished by setting up a high resolution interval timer and installing a timer interrupt handler. For systems that do not support counter overflow at the operating system level, PAPI uses SIGPROF and ITIMER_PROF. 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 an associated with symbolic information contained in the executable. What results is a line-by-line account of where counter overflow occurred in the program. GNU `prof` in conjunction with the `-p` option of the GCC compiler performs exactly this analysis using process time as the overflow trigger. PAPI aims to generalize this functionality so that a histogram can be generated using any countable event as the basis for analysis.

PAPI provides support for execution profiling based on any counter event. The **PAPI_profil()** call creates an histogram of overflow counts for a specified region of the application code. In the exact manner of UNIX **profil(),** the identified region is logically broken up into equal size subdivisions. Each time the counter reaches the specified threshold, the current subdivision is identified and its corresponding hash bucket is incremented.

7 Accuracy of Hardware Counter Data

A number of questions have arisen concerning the accuracy of hardware performance data and how it should be used for application-level performance analysis. Accuracy problems caused when the granularity of the measured code is not sufficiently large to ensure that the overhead introduced by counter interfaces does not dominate the event counts are reported in [7]. Problems with the accurate attribution of hardware events to individual instructions in out-of-order processors is a well-known problem. We plan to collaborate with other researchers on developing microbenchmarks such as those described in [7] for determining the accuracy of hardware counter interfaces. Having the common PAPI interface available across platforms will make accuracy analysis much easier.

8 Tools

The PAPI project has developed an end-user tool that demonstrates graphical display of PAPI performance data in a manner useful to the application developer. This tool is meant to demonstrate the capabilities of PAPI rather than as a production quality tool. The tool front end is written in Java and can be run on the same machine or a separate machine from the program being monitored. All that is required for real-time monitoring and display of application performance is a socket connection between the machines or processes. The tool, called the *perfometer,* provides a runtime trace of a chosen PAPI metric, as shown in Figure 2 for floating operations per second (PAPI_FLOPS). 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 3. Any process can be selected for the larger display by clicking on its representation in the multiprocess display.

Visual Profiler, or vprof, is a tool developed at Sandia National Laboratory for collecting statistical program counter data and graphically viewing the results on Linux Intel machines[8]. vprof uses statistical

event sampling to provide line-by-line execution profiling of source code. vprof can sample clock ticks using the **profil** system call. The vprof developer has added support for PAPI so that vprof can also sample the wide range of system events supported by PAPI. A screenshot showing vprof examination of both **profil** and PAPI_TOT_CYC data is shown in Figure 4.

More end-user tools for interpretation of hardware data are needed that allow the data to be represented visually and that correlate counter data with application program components and correlate related events with each other. It is the intent of the PAPI project to make development of such tools much easier by providing a portable interface to processor-specific hardware counter data.

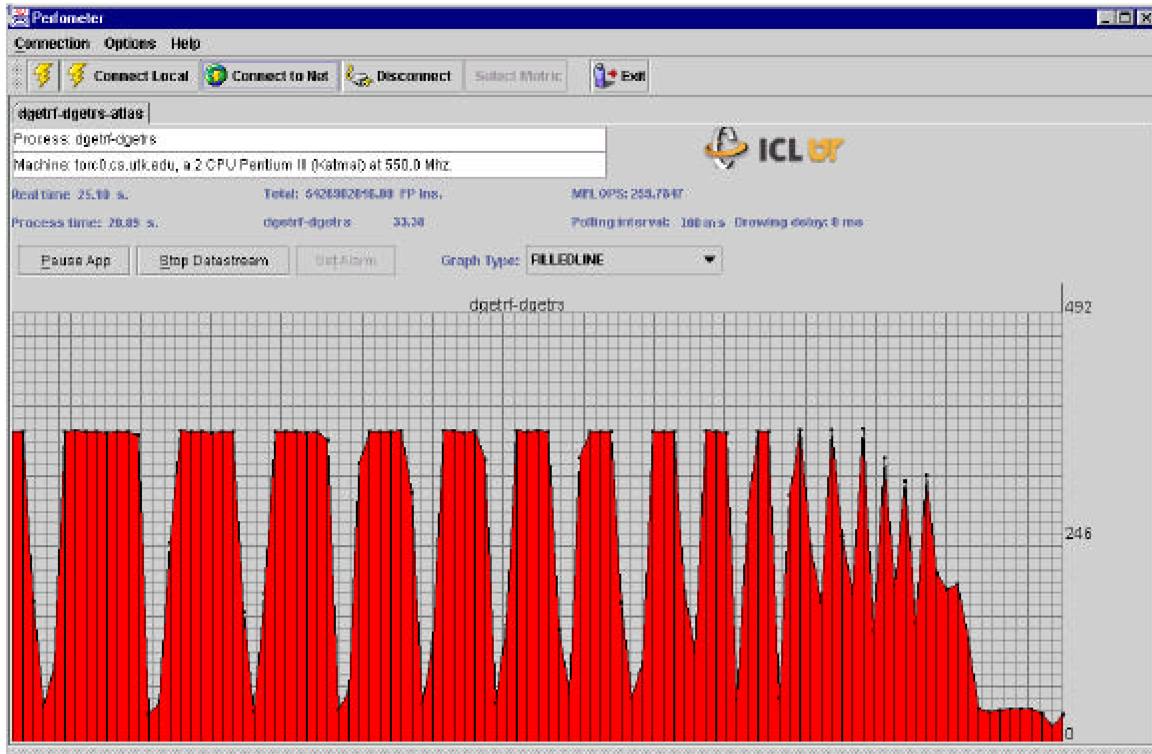

Figure 2. Perfometer monitoring a single-process application

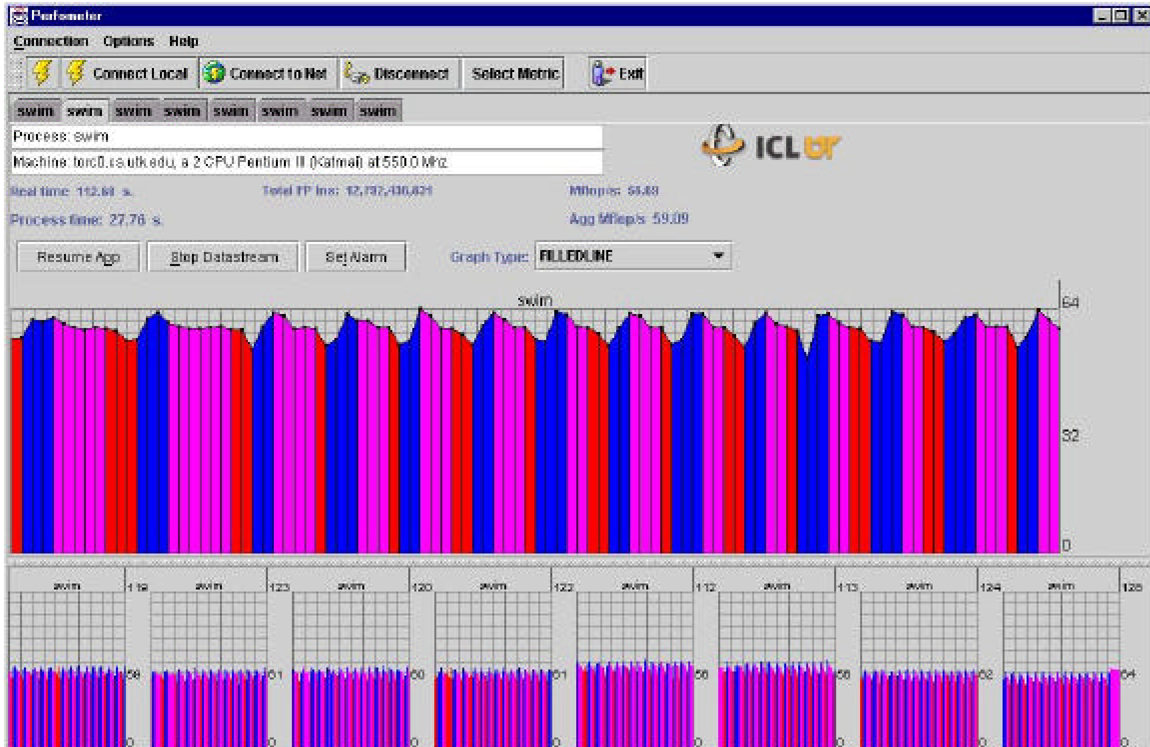

Figure 3. Perfometer monitoring a parallel application

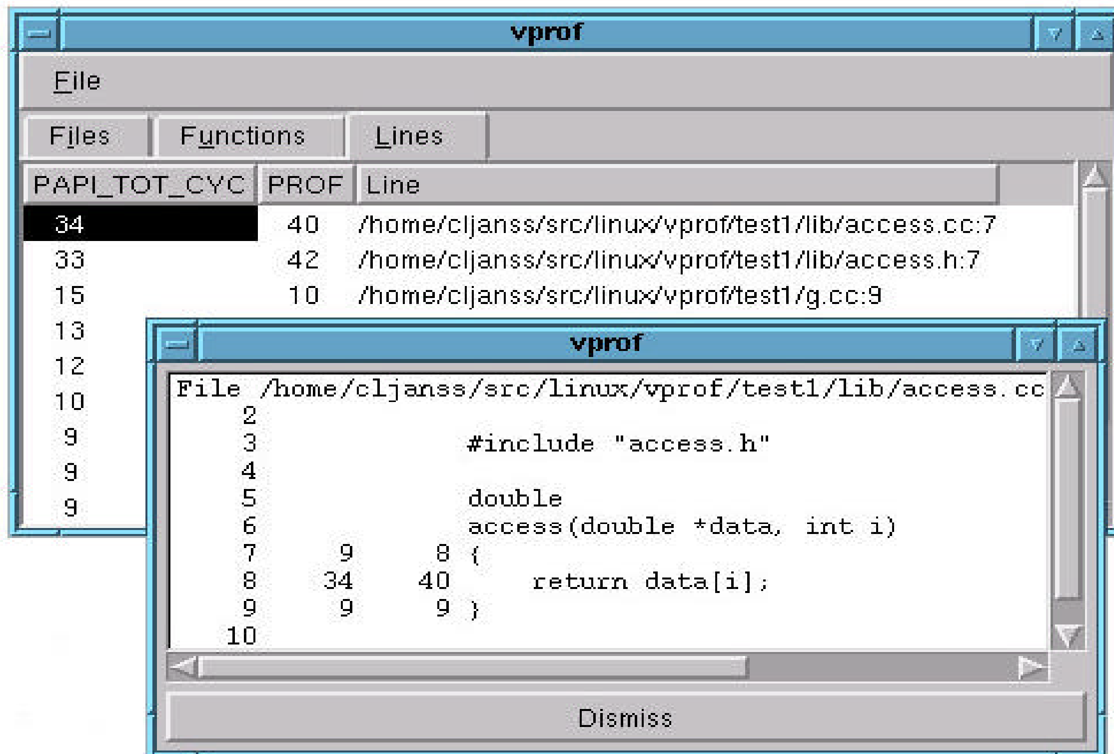

Figure 4. vprof displaying profil and PAPI_TOT_CYC data

9 Related Work

The Rabbit performance counters library provides routines for reading and manipulating Intel or AMD processor hardware event counters in C under Linux[9]. Arbitrary programs that do not use the hardware counters can be sample with Rabbit, which runs a child process alongside its own interval timer. Rabbit multiplexes through a list of events and manages data files in an output directory. Because Rabbit runs over unpatched Linux which does not save and restore the counters on process context switches, Rabbit measurements are attached to the processor, not to the process or thread. Rabbit uses the raw counters which are subject to overflow at high MHz due to their short (40- to 48-bit) length. Informative documentation concerning hardware performance monitoring on Linux platforms may be found at the Rabbit web site.

lperfex is a utility program for accessing the hardware counters on the Intel P6[10]. This utility does not require recompilation of the measure program. Its interface is similar to that of the SGI perfex utility. Some perfex options, such as counter multiplexing, are not supported by lperfex. lperfex uses Erik Hendrik's libperf Linux/x86 patch for access to the counters.

The Performance Counter Library (PCL) is a cross-platform interface for accessing performance counters built into modern microprocessors[11]. PCL supports query for functionality, start and stop of counters, and reading the current values of counters. Performance counter values are returned as 64-bit integers an all supported platforms. Performance counting can be done in user mode, system mode, or user-or-system mode. PCL supports nested calls to PCL functions to allow hierarchical performance measurements. However, nested calls must use exactly the same list of events. PCL functions are callable from C, C++, Fortran, and Java. Similar to PAPI, PCL defines a common set of events across platforms for accesses to the memory hierarchy, cycle and instruction counts, and the status of functional units, and translates these into native events on a given platform where possible. PAPI additionally defines events related to SMP cache coherence protocols and to cycles stalled waiting for various resources. Unlike PAPI, PCL does not support software multiplexing or user-defined overflow handling. The PCL API is very similar to the PAPI high-level API and consists of calls to start a list of counters, and to read or stop the counter most recently started. In the Linux implementation of PCL, counter values are not saved on context switches.

10 Conclusions

PAPI aims to provide the tool designer and application engineer with a consistent interface and methodology for use of the performance counter hardware found in most major microprocessor lines. The main motivation for this interface is the increasing divergence of application performance from near peak performance of most machines in the HPC marketplace. This performance gap is largely due to the disparity in memory and communication bandwidth at different levels of the memory hierarchy. With no viable hardware solution in sight, users requiring the optimal performance must expend significant effort on single processor and shared memory optimization techniques. To address this problem users need a compact set of robust and useful tools to quickly diagnose and analyze processor specific performance metrics. To that end, many design efforts have wastefully reinvented the software infrastructure necessary for a suite of program analysis tools. PAPI directly challenges this model by focusing on a reusable, portable and functionality-oriented infrastructure for performance tool design. It is hoped that through additional collaborative efforts, PAPI will become one of a number of modular components for advanced tool design and program analysis.

The PAPI specification and software, as well as documentation and additional supporting information, are available from the PAPI web site at http://icl.cs.utk.edu/projects/papi/.

References

1. Browne, S., 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), 2000, **p**p. 189-204.
2. Browne, S., J. Dongarra, N. Garner, K. London, and P. Mucci, *A Scalable Cross-Platform Infrastructure for Application Performance Optimization Using Hardware Counters*. in *Proc. SC'2000*, November 2000, Dallas, Texas.
3. *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, Intel document number 243192, http://developer.intel.com/.
4. *Intel IA-64 Architecture Software Developer's Manual, Volume 4: Itanium Processor Programmer's Guide,* Intel document number 245320-02, March 2001, http://developer.intel.com/.
5. Pettersson, M., *Linux x86 Performance-Monitoring Counters Driver*, http://www.csd.uu.se/~mikpe/linux/perfctr/.
6. *AMD Athlon Processor x86 Code Optimization Guide*, 2001, AMD publication number 22007, http://www.amd.com/.
7. Korn, W., P.J. Teller, and G. Castillo. *Just how accurate are performance counters?* in *20th IEEE International Performance, Computing, and Communications Conference (IPCCC 2001)*, April 2001, Phoenix, Arizona.
8. Janssen, C.L., *The Visual Profiler*, *Version 0.7*, April 2001, http://aros.ca.sandia.gov/~cljanss/perf/vprof/.
9. Heller, D., *Rabbit: A Performance Counters Library for Intel/AMD Processors and Linux*, http://www.scl.ameslab.gov/Projects/Rabbit/.
10. Baer, T., *lperfex: A Hardware Performance Monitor for Linux/IA32 Systems*, http://www.osc.edu/~troy/lperfex/.
11. Berrendorf, R. and H. Zeigler, *PCL -- the Performance Counter Library, version 2.0,* September 2000*,* http://www.gz-juelich.de/zam/PCL/.