Extending a Standard Hardware Counter Interface to Simultaneous Multiple Measurement Domains[⋆]

Jack Dongarra, Kevin London, Shirley Moore, Daniel Terpstra, and Haihang You

Innovative Computing Laboratory, University of Tennessee
Knoxville, TN 37996-3450 USA
{dongarra,london,shirley,terpstra,you}@cs.utk.edu

Abstract. Modern high performance computer systems continue to increase in size, speed, and architectural complexity. Tools to measure application performance in these increasingly complex environments must also increase the richness of their measurements to provide insights into the increasingly intricate ways in which software and hardware interact. This paper reports on initial efforts to extend the widely used PAPI portable hardware counter interface to support the monitoring of multiple measurement domains simultaneously. The paper gives a description of the architectural changes needed to generalize the PAPI interface, followed by examples of simultaneous measurements of processor hardware counters, temperature sensors, and network counters for some benchmark codes from the HPC Challenge suite.

1 Introduction

The Performance Application Programming Interface (PAPI) provides a portable interface to the performance counters available on most modern microprocessors [1]. Developers can use these counters to understand exactly how their application is running on the processor and use this information to determine the best way to optimize the application. The PAPI interface has been extended to support monitoring of other domains simultaneously with processor hardware counters. Such domains include, but are not limited to, thermal sensors and network counters, examples of which will be presented below. An important consideration in extending a widely accepted interface such as PAPI is to make extensions in such a way as to preserve the original interface as much as possible for the sake of backward compatibility.

As processor speeds and densities climb, the thermal properties of high performance systems are becoming increasingly important. Such systems contain large numbers of densely packed processors which require a great deal of electricity. Power and thermal management issues are becoming critical to successful resource utilization [2, 3]. Standardized interfaces for accessing the thermal sensors are available, but may be difficult to use for runtime power-performance adaptation [4]. Extending the PAPI interface to simultaneously monitor processor metrics and thermal sensors can provide clues for

[⋆] This work was supported by the U.S. Department of Energy Los Alamos Computer Science Institute under subcontract R7A827-79200 through Rice University.

correlating algorithmic activity with thermal system responses thus help in developing appropriate workload distribution strategies.

Many network switches and network interface cards (NICs) contain counters that can monitor various events related to performance and reliability. Possible events include checksum errors, dropped packets, and packets sent and received. Although the set of network events is necessarily somewhat dependent on the underlying hardware, extending PAPI to the network monitoring domain provides a portable way to access native network events and allows correlation of network events with other domains. Because communication in OS-bypass networks such as Myrinet is handled asynchronously to the application, hardware monitoring, in addition to being low overhead, may be the only way to obtain some important data about communication performance.

Initial results reported in this paper illustrate how monitoring multiple measurements domains simultaneously can provide insight into the mapping of an algorithm onto the underlying hardware. Results of using the extended version of PAPI to simultaneously monitor processor counters, ACPI thermal sensors, and Myrinet network counters while running the FFTE and HPL HPC Challenge benchmarks [5] on a AMD Opteron Linux cluster are presented.

2 Extending PAPI to Multiple Substrates

The PAPI library was originally developed to address the problem of accessing the processor hardware counters found on a diverse collection of modern microprocessors in a portable manner [1]. Other system components besides the processor, such as the memory interface chips, network interface cards, and network switches, also have counters that count various events related to system reliability and performance. Further, other system health measurements, such as board level temperature sensors, are also available and useful to monitor in a portable manner. Unlike on-processor counters, the off-processor counters and sensors usually measure events in a system-wide rather than a process or thread-specific context. However, when an application has exclusive use of a machine partition, or as in the case of single threaded operating systems on the compute nodes of machines such as the IBM BlueGene or the Cray XT3, it may be possible to interpret such events in the context of the application. The current situation with off-processor counters is similar to the situation that existed with on-processor counters before PAPI. A number of different platform-specific interfaces exist, some of which are poorly documented or not documented at all.

A number of software design issues became apparent in extending the PAPI interface for multiple measurement domains. The PAPI library consists of two internal layers: a large layer optimized for platform independence and portability; and a smaller layer, called the substrate, containing the platform dependent code. By compiling and linking the independent layer with a specific substrate , an instance of the PAPI library could be produced for a specific platform and hardware architecture. At compile time the substrate provided common definitions and data structure sizes to the independent layer, and at link time it satisfied unresolved function references across the layers. Since there was a one-to-one relationship between the independent layer and the substrate, initialization and shutdown logic was straightforward, and control and query routines could be directly implemented. In migrating to a multi-substrate model, this one-to-one relationship was replaced with a one-many coupling between independent layer and substrates, requiring that previous code dependencies and assumptions be carefully identified and modified as necessary.

When linking multiple substrates into a common object library, each substrate exposes a similar set of functionality to the independent layer. To avoid name-space collisions in the linker, each substrate was modified to hide the function names by declaring them as 'static' inside the substrate code. The substrate then publishes a list of these function pointers through a newly created and uniquely named initialization entry point. The independent layer builds a table of function pointers at run-time by calling this entry point. All later accesses to the substrate occur through this function table. In this way, the independent layer can transparently manage initialization of and access to multiple substrates. Our experiments have shown that the extra level of indirection introduced by calls through a function pointer adds a negligible additional overhead to the call time, even in time-critical routines such as reading counter values.

Countable events in PAPI are either preset events, defined uniformly across all architectures, or native events, unique to a specific substrate. To date preset events have only been defined for processor hardware counters, making all events on off-processor substrates native events. By convention, an event to be counted is added to a collect of events in an eventset, and eventsets are started, stopped, and read to produce event count values. Each eventset in multi-substrate PAPI belongs to a specific substrate and can only contain events associated with that substrate. Mechanisms have been added to PAPI to search for events across the substrate space and identify the substrate to which an event belongs. Multiple eventsets can be active simultaneously, as long as only one eventset per substrate is invoked. Example substrates have been implemented in the development version of PAPI for the ACPI temperature sensors and the Myrinet network counters. An implementation of an Infiniband network substrate is underway.

3 Measurements of the HPC Challenge Benchmarks

The HPC Challenge suite is a set of scalable, computationally intensive benchmarks with different memory access patterns that examine the performance of HPC architectures [5]. For our experiments, we chose two global kernel benchmarks, High Performance Linpack (HPL) and FFT. We instrumented both benchmarks to gather total floating-point operations, temperature and packets sent and received through the Myrinet network. With the extension made to PAPI for measuring multiple substrates simultaneously, we were able to easily instrument the program by simply providing the desired event names in PAPI calls. We ran our experiments on a 65-node AMD Opteron cluster. Both benchmarks ran on eight nodes. We instrumented functions fft235 in FFT and pdgesvK2 in HPL, and gathered data for each iteration that called these functions.

The measurements for the FFT benchmark on two of the nodes are shown in Figure 1. We can see the periodic nature of the computation and communication. Note the difference in temperature between the two nodes.

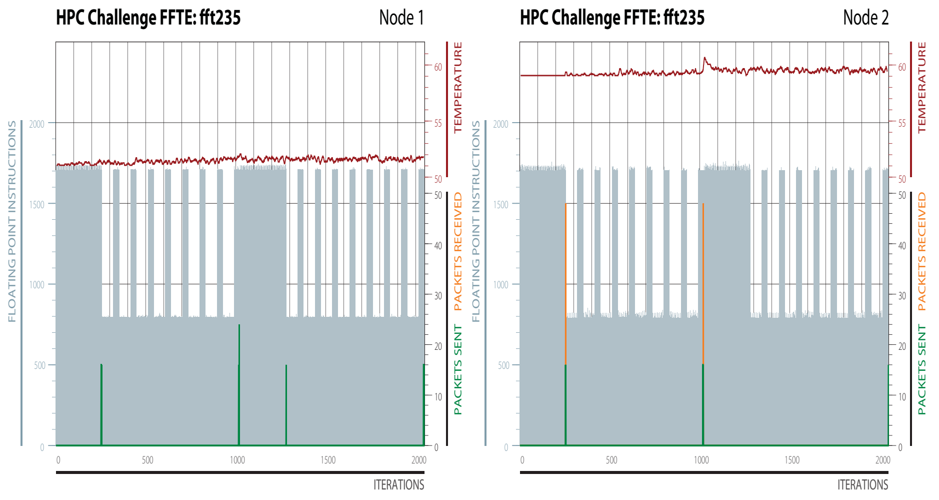

Fig. 1. FFT Results

4 Conclusions and Future Work

The widely accepted PAPI interface has been extended to support the simultaneous monitoring of multiple measurement domains. This extension has been done in such a way as to cause minimal disruption to the current user base while providing flexible opportunities to gain new insights into application and system performance. Preliminary measurements on HPC Challenge benchmark codes offer suggestions of the kinds of measurements that might be made in the future with such a tool. Efforts are underway to harden existing substrates in the network and temperature monitoring domains and to implement new substrates. Further work is needed in automating the configuration and build systems to more fully and flexibly accommodate a wide range of hardware and substrate architectures.

References

1. Shirley Browne, Jack Dongarra, Nathan Garner, George Ho, and Philip Mucci. A portable programming interface for performance evaluation on modern processors. *International Journal of High-Performance Computing Applications*, 14(3):189–204, 2000.
2. Kirk W. Cameron, Rong Ge, and Xizhou Feng. High-performance, power-aware distributed computing for scientific applications. *Computer*, 38(11):40–47, 2005.
3. Wu chun Feng. The importance of being low power in high performance computing. *CTWatch Quarterly*, 1(3), August 2005.
4. Fincent W. Freeh, David K. Lowenthal, Feng Pan, and Nandani Kappiah. Using multiple energy gears in MPI programs on a power-scalable cluster. In *Principles and Practices of Parallel Programming (PPOPP)*, June 2005.
5. P. Luszczek, J. Dongarra, D. Koester, R. Rabenseifner, B. Lucas, J. Kepner, J. McCalpin, D. Bailey, and D. Takahashi. Introduction to the hpc challenge benchmark suite. Technical report, March 2005.