New Developments for PAPI 5.6+

Anthony Danalis
University of Tennessee
Knoxville, TN, USA
adanalis@icl.utk.edu

Heike Jagode
University of Tennessee
Knoxville, TN, USA
jagode@icl.utk.edu

Vince Weaver
University of Maine
Orono, ME, USA
vincent.weaver@maine.edu

Yan Liu
University of Maine
Orono, ME, USA
yan.liu@maine.edu

Jack Dongarra
University of Tennessee
Knoxville, TN, USA
dongarra@icl.utk.edu

ABSTRACT

The high-performance computing (HPC) community has relied on the Performance API (PAPI) monitoring library to track low-level hardware operations for more than 15 years. In that time, the needs of software developers have changed immensely, and the PAPI team aims to meet these demands through a better understanding of deep and heterogeneous memory hierarchies and finer-grain power management support. With this in mind, and to better serve the HPC software development community, the PAPI team has focused on improving and transforming PAPI to meet the latest requirements of emerging architectures and advanced technologies.

The accompanying poster demonstrates how PAPI enables power tuning to reduce overall energy consumption without, in many cases, a loss in performance. Furthermore, we discuss efforts to develop microbenchmarks intended to assist application developers who are interested in performance analysis by automatically categorizing and disambiguating performance counters. This work also aims to assist developers by highlighting non-obvious behaviors of modern hardware that can lead to unexpected counter values.

Finally, the poster illustrates efforts to update PAPI's internal sanity checks—designed to inspect that PAPI's predefined events are in fact measuring the values they claim to measure—and modernize the implementation of critical API functions, e.g., `PAPI_read()`, and the sampling interface so that more information can be captured and reported with lower overhead.

ACM Reference format:
Anthony Danalis, Heike Jagode, Vince Weaver, Yan Liu, and Jack Dongarra. 2017. New Developments for PAPI 5.6+. In *Proceedings of ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, Denver, Colorado, USA, Nov 2017 (SC'17),* 3 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

SC'17, Denver, Colorado, USA

DOI: 10.1145/nnnnnnn.nnnnnnn

1 PAPI FOR POWER-AWARE COMPUTING

Designing numerical libraries and scientific applications with energy consumption as a primary constraint presents significant challenges for developers. Our research poster presents results of a study where PAPI's new power control functionality was used to limit the power consumption in a set of representative kernels on a modern computing platform. The development of the PAPI `powercap` component, which uses the Linux powercap interface (available in the Linux kernel 3.13 release) to expose the RAPL settings to user-space, has an active interface to allow *writing* values in addition to *reading* them.

The figures of the first part of the poster demonstrate a detailed analysis of the performance of four kernels and the power consumption of the different components of the hardware when the power limit is set to different values using the PAPI `powercap` component. In particular, the four pairs of figures demonstrate the behavior of:

- compute-intensive matrix-matrix multiplication (`dgemm`),
- memory-bound matrix-vector multiplication (`dgemv`),
- the Jacobi method for solving the Helmholtz equation, and
- a Lattice-Boltzmann simulation taken from SPEC 2006.

These four kernels were chosen, not only because they represent a significant fraction of real-world scientific workloads, but also because they have different ratios of computation to memory traffic. All experiments were performed on an Intel Knights Landing CPU configured in FLAT mode (i.e., the MCDRAM is used as addressable memory). In each of the four pairs of graphs, the left graph demonstrates the case where the data was allocated in the standard 4th-generation double data rate (DDR4) synchronous DRAM memory, and the right graph demonstrates the case where the data was allocated in the high-bandwidth multi-channel DRAM (MCDRAM).

As explained in detail in the poster, depending on the computational density of the kernel and the speed of the memory used to store the data, the power consumption of some of these kernels could be limited using PAPI in ways that lead to an overall reduction of energy consumption (e.g up to 25% for the Jacobi iterative method) without reducing the performance of the kernel!

This clearly demonstrates that the usefulness of PAPI's `powercap` component is not only theoretical but can be used in the libraries and kernels that will drive the software stack of tomorrow's supercomputers. This feature allows applications to control the power limit of the hardware through the de facto standard PAPI interface in a portable way, regardless of whether PAPI is used directly or through third-party performance toolkits.

2 COUNTER INSPECTION TOOLKIT

Improving application performance requires some understanding of where performance bottlenecks exist. This understanding can be achieved by examining the values of hardware performance counters during program execution. However, because of the increasing complexity of modern CPUs, the number of events offered by the hardware has been fast-growing, and many of these events have complex descriptions and contain multiple flags that modify what exactly is being monitored.

The cache hierarchy is a perfect case in point. The level at which a code is reusing the caches of a CPU usually has a substantial effect on performance. To help assess how well cache reuse is achieved by an application, hardware vendors offer several events that count different behaviors of the cache hierarchy. Unfortunately, the complexity of modern cache subsystems has led to multiple event types, some of which have non-obvious names and functionality.

To assist developers in choosing which event to use and understand what each event measures, we used microbenchmarks that stress the cache subsystem. The underlying idea of the code is to control the way memory is accessed—and the amount of memory accessed—and observe how the measured events change. For our measurements, we use streaming along with a technique known as "pointer chaining" (or "pointer chasing"), which is common in the benchmark literature [2, 4, 5].

In the second section of the poster, we include figures that show the results of running our memory access benchmark with a variable array size while measuring the value of different events. As shown in the figures, the values of each event show sharp transitions between 0% and 100% at the boundaries of the different caches (L1=32KB, L2=256KB, and L3=32MB in this example). These sharp transitions can function as signatures that enable us to categorize events based on whether they measure a *hit* or a *miss* and which cache level they relate to, regardless of the name of each event.

Prefetching is another interesting behavior that we can probe with our microbenchmarks. In particular, as we illustrate in the poster, altering the minimum distance between memory accesses (64 B, 128 B, or 256 B) or the number of operating system pages that are accessed by our benchmark (4–35) can have dramatic effects on the efficacy of the prefetching unit.

Finally, using an example that involves branch instructions and speculative execution, the poster demonstrates how seemingly trivial modifications to a simple code can lead to non-obvious changes in event counter values.

In summary, the Counter Inspection Toolkit effort aims to deliver a set of benchmarks and analyses that will assist developers who care about performance and optimizing their applications but are not hardware wizards with a perfect understanding of chip design or event counter semantics.

3 PAPI INFRASTRUCTURE IMPROVEMENTS

3.1 Test Infrastructure Modernization

The hardware, operating systems, and compilers supported by PAPI are always changing, and keeping pace with this constant evolution requires a robust testing methodology. To this end, the PAPI distribution contains multiple tests used to ensure counter accuracy. These tests, written in both C and Fortran and designed to exercise as much of the PAPI interface as possible, are performed and verified before each software release. For the latest version of PAPI, the entire code base was audited, and failing tests were fixed and rewritten as necessary.

In addition to fixing the existing code, new event validation tests were written. These tests are simple sanity checks designed to check that the PAPI predefined events are in fact measuring the values they claim to measure. The vendor-provided "native" events are not always well documented, so these new tests can help ensure that the correct events are chosen from the processor data sheets. These tests also act as an additional sanity check when making other changes to the PAPI code base.

3.2 Low-Overhead `PAPI_read()` Support

PAPI is often used for self monitoring, which involves adding targeted instrumentation code around areas of interest. The monitoring code introduces `PAPI_read()` calls deep in the critical sections of the code being measured; as a result, this routine must have low overhead to avoid interfering with the values being monitored and recorded. Traditionally, `PAPI_read()` uses the standard Linux `read()` system call, which can be intrusive (essentially it is a software-triggered interrupt) and slow (around 1,000 cycles on modern machines). Modern x86 hardware supports a special `rdpmc` instruction that bypasses the kernel and only uses 200 cycles (a 5× speedup). Current Linux distributions support this `rdpmc` interface, and we added support for the interface to PAPI. We also found four major Linux bugs related to the interface, but these have since been fixed upstream, allowing us to fully support this interface in the latest version of PAPI.

The figures shown in the third part of the poster illustrate how the new `rdpmc` support compares against the classic `perfctr` and `perfmon2` interfaces that predated `perf_event`'s inclusion Linux. As shown, the overhead of `PAPI_read()` remained the same across PAPI versions until the introduction of `rdpmc` support.

3.3 Enhanced Sampling Interface

Sampling is a common use case for PAPI. For example, a counter is set up to periodically interrupt the processor and store the instruction pointer. These values can be saved and later analyzed to reconstruct program behavior. Recent processors support more advanced sampling interfaces, e.g., Intel's Precise Event-based Sampling (PEBS) [1] and AMD's Instruction Based Sampling (IBS) [3]. These more advanced interfaces allow more accurate and more detailed sampling. In addition to the instruction pointer, data can also be gathered about full register state and other architectural information, including a list of taken branches, the cause of a cache miss, location of cache miss within the hierarchy, and the instructions' cycle latency. The latest version of PAPI has been extended to include this type of sampling information.

ACKNOWLEDGMENTS

This material is based upon work supported in part by the National Science Foundation NSF under awards No. 1450429. A portion of this research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

REFERENCES

[1] Intel Corporation. 2017. *Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3B: System Programming Guide, part 2.*

[2] Anthony Danalis, Piotr Luszczek, Gabriel Marin, Jeffrey S. Vetter, and Jack Dongarra. 2014. BlackjackBench: Portable Hardware Characterization with Automated Results' Analysis. *Comput. J.* 57, 7 (2014), 1002. DOI: https://doi.org/10.1093/comjnl/bxt057

[3] P.J. Drongowski. 2007. Instruction-Based Sampling: A new performance analysis technique for AMD Family 10h Processors. http://developer.amd.com/wordpress/media/2012/10/AMD_IBS_paper_EN.pdf. (2007).

[4] Larry McVoy and Carl Staelin. 1996. lmbench: portable tools for performance analysis. In *ATEC'96: Proceedings of the Annual Technical Conference on USENIX 1996 Annual Technical Conference*. USENIX Association, Berkeley, CA, USA, USENIX Association., 23–23.

[5] Phillip J. Mucci and Kevin London. 1998. *The CacheBench Report.* Technical Report. Computer Science Department, University of Tennessee, Knoxville, TN.