The Institute for Sustained Performance, Energy, and Resilience (SUPER)

DOE Final Technical Report

(reporting period: 09/01/2015 – 08/31/2016)

DE-FC02-11ER26059/DE-SC0006733

University of Tennessee, Knoxville (UTK)

PI: Heike Jagode

Co-PIs: George Bosilca, Anthony Danalis, Jack Dongarra

Subcontractor:

University of Texas El Paso (UTEP)

PI: Shirley Moore

November 30, 2016

1 Executive Summary

The University of Tennessee (UTK) and University of Texas at El Paso (UTEP) partnership supported the three main thrusts of the SUPER project—performance, energy, and resilience. The UTK-UTEP effort thus helped advance the main goal of SUPER, which was to ensure that DOE's computational scientists can successfully exploit the emerging generation of high performance computing (HPC) systems. This goal is being met by providing application scientists with strategies and tools to productively maximize performance, conserve energy, and attain resilience.

The primary vehicle through which UTK provided performance measurement support to SUPER and the larger HPC community is the Performance Application Programming Interface (PAPI). PAPI is an ongoing project that provides a consistent interface and methodology for collecting hardware performance information from various hardware and software components, including most major CPUs, GPUs and accelerators, interconnects, I/O systems, and power interfaces, as well as virtual cloud environments. The PAPI software is widely used for performance modeling of scientific and engineering applications—for example, the HOMME (High Order Methods Modeling Environment) climate code, and the GAMESS and NWChem computational chemistry codes—on DOE supercomputers. PAPI is widely deployed as middleware for use by higher-level profiling, tracing, and sampling tools (e.g., CrayPat, HPCToolkit, Scalasca, Score-P, TAU, Vampir, PerfExpert), making it the de facto standard for hardware counter analysis. PAPI has established itself as fundamental software infrastructure in every application domain (spanning academia, government, and industry), where improving performance can be mission critical. Ultimately, as more application scientists migrate their applications to HPC platforms, they will benefit from the extended capabilities this grant brought to PAPI to analyze and optimize performance in these environments, whether they use PAPI directly, or via third-party performance tools. Capabilities added to PAPI through this grant include support for new architectures such as the lastest GPU and Xeon Phi accelerators, and advanced power measurement and management features.

Another important topic for the UTK team was providing support for a rich ecosystem of different fault management strategies in the context of parallel computing. Our long term efforts have been oriented toward proposing flexible strategies and providing building boxes that application developers can use to build the most efficient fault management technique for their application. These efforts span across the entire software spectrum, from theoretical models of existing strategies to easily assess their performance, to algorithmic modifications to take advantage of specific mathematical properties for data redundancy and to extensions to widely used programming paradigms to empower the application developers to deal with all types of faults. We have also continued our tight collaborations with users to help them adopt these technologies to ensure their application always deliver meaningful scientific data.

Large supercomputer systems are becoming more and more power and energy constrained,

and future systems and applications running on them will need to be optimized to run under power caps and/or minimize energy consumption. The UTEP team contributed to the SUPER energy thrust by developing power modeling methodologies and investigating power management strategies. Scalability modeling results showed that some applications can scale better with respect to an increasing power budget than with respect to only the number of processors. Power management, in particular shifting power to processors on the critical path of an application execution, can reduce perturbation due to system noise and other sources of runtime variability, which are growing problems on large-scale power-constrained computer systems.

2 University of Tennessee, Knoxville

During the last year of the grant, UTK focuses primarily on activities related to two central areas of SUPER: (1) architecture aware areas focusing on performance and energy measurements and engineering, and (2) resilience and fault tolerance. Each of these areas will be discussed in further detail below.

2.1 Architecture Awareness

The primary workhorse through which UTK provides performance measurement support to SUPER and the larger HPC community is the Performance Application Programming Interface (PAPI). PAPI is an ongoing project that provides a consistent interface and methodology for performance counter information of varied hardware and software components, including most major CPUs, GPUs, accelerators, interconnects, I/O systems, and power interfaces as well as virtual cloud environments. The PAPI software is available for download from the PAPI website at `http://icl.cs.utk.edu/papi/`.

The UTK PAPI development team also works closely with vendors and tool developers to incorporate PAPI into their products and research software. For instance, the release of the most recent PAPI version 5.5 incorporates power measurements and power capping control for recent Intel CPUs using the Linux powercap interface. Energy efficiency has been identified as a major concern for extreme-scale platforms, and PAPI can provide details about the power efficiency of an implementation. Additionally, we have worked closely with Intel—long before the official release of Intel's Knights Landing (KNL)—to add core and uncore support for KNL as well as power monitoring via the RAPL and powercap components. This close collaboration with vendors enabled us to ship PAPI with KNL support shortly after the processor was officially released.

We released two PAPI versions during the reporting period:

1. PAPI 5.4.3 was released in January 2016.
2. The most recent version is PAPI 5.5, released in August 2016.

The progress on each of these accomplishments is reported and explained, in further detail, in the subsections below.

2.1.1 PAPI support for multiple GPUs

The PAPI CUDA component is a hardware performance counter measurement technology for the NVIDIA CUDA platform that provides access to the hardware counters inside the GPU. The PAPI CUDA component is based on CUPTI support in the NVIDIA driver library. In any environment where the CUPTI-enabled driver is installed, the PAPI CUDA component is able to provide detailed performance counter information regarding events on the GPU kernels.

As a major extension, and as part of our effort during the last year of the SUPER project, the PAPI CUDA component has undergone a major redesign to provide CUDA support for multiple GPU devices and multiple CUDA contexts. Part of the PAPI CUDA component enhancement is the newly added test case, so that users can easily take advantage of `LD_PRELOAD` on a Linux system to intercept function calls and "PAPI-enable" an uninstrumented CUDA binary. These CUDA enhancements were tested and hardened, and officially released with PAPI in January 2016.

2.1.2 Power Management via PAPI

Activities during the last year of this grant focused primarily on the exploration of incorporating power measurements and power control into PAPI. Current directions for PAPI development include providing applications the ability to achieve a trade-off between power and performance. The 2016 PAPI releases have incorporated power measurements and power capping control for recent Intel CPUs using the RAPL interface. Energy efficiency has been identified as a major concern for exascale platforms, and PAPI can provide details about the power efficiency of an implementation. This power-efficiency information can determine the choice between alternative algorithmic implementations of an operation (e.g., linear algebra block-layout versus tile-layout algorithms), leading to the design of energy-efficient algorithms. We have developed two new power-management components, which are discussed below:

A. Power Management using the Intel RAPL Interface (via msr-safe and libmsr)
We've been experimenting with a PAPI component that includes control aspects (i.e., this component has an *active* interface that writes values to the RAPL/MSR interface). This is a significant change from all prior PAPI components that have had an entirely passive measurement interface to read information and events. In our SUPER project report from last year (2015), we included a detailed discussion about our early version of the RAPL/MSR component. This component was further optimized and tested, and released in January 2016

(PAPI version 5.4.3). We are awaiting feedback from our interdisciplinary partners who integrate our software with their tools and applications.

B. Power Management using the Intel RAPL Interface (via the Linux powercap interface) The August 2016 PAPI release has a new component that provides read and write access to the power/energy information and controls exposed via the Linux powercap interface. This interface exposes various power-related values to the user via a sysfs interface. The information is displayed in the form of sets of directories with subdirectories. These directories are organized by "zones" or "domains". Essentially, the top level is the power planes. Some systems will have one power plane, and others will have two. What is nested inside a power plane also varies; these can be referred to as "subdomains" or "subzones." For some processors, memory, uncore, and core "subdomains" may be nested within a given power plane. Each of these "parent" and "children" power domains contains individual power-constraint information that is divided up between short- and long-term time windows. This is presented to the user in the form of system files whose name indicates the type of value they contain. For example, `constraint_0_power_limit` would contain the value associated with the power limit relative to the time window that is associated with ID number 0.

This PAPI powercap component uses the Linux interface that was designed to expose power constraint values for various Intel processors. Each processor has a different set of power constraints available via this Linux interface. The PAPI powercap component will show only the available constraints for the specific system that it is being run on. It is important to note that some inconsistencies exist in the Linux interface. Some writable constraints for one processor may not be writable for another processor. This has been identified as a known issue by Intel.

Figure 1 shows a Hessenberg reduction kernel from the MAGMA library (`http://icl.cs.utk.edu/magma/`) computed on the Intel Xeon Phi Knights Landing processor utilizing all 68 cores utilizing all 4 hardware threads per core. For the power readings, we are using the PAPI powercap component and the measurement utility with a sampling rate of 100 milliseconds. The Hessenberg reduction is computed nine times, each with a different matrix size. The numbers on top of each curve show the chosen matrix size. The measured power data clearly mimics the computational intensity of the Hessenberg computations. The factorization begins on the entire matrix—in this case consuming most of the power—and as the factorization progresses, it operates on smaller and smaller matrices, resulting in less and less power usage.

Next steps/goals: As one of our next steps, we are working on consolidating all the different PAPI power components into one component rather than having multiple approaches of measuring and controlling the power.

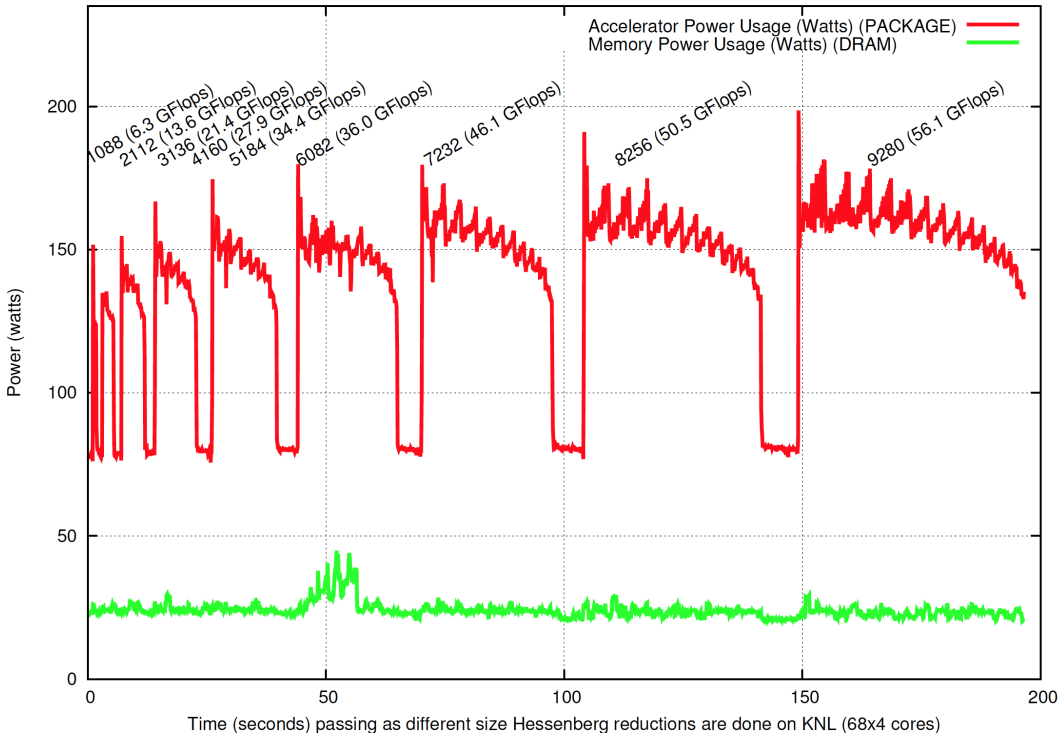

Figure 1: Example of power controlling with PAPI powercap component.

2.1.3 New Architecture Support in PAPI

Another objective that is always crucial to PAPI's success is extending and verifying support for new processor architectures.

A. Intel Xeon Phi Knights Landing We have worked closely with Intel—long before the official release of Intel's Knights Landing (KNL)—to add core and uncore support for KNL as well as power monitoring via the RAPL and powercap components. This close collaboration with the vendor enabled us to ship PAPI with KNL support shortly after the processor was officially released.

A.1 Native Events: In Figure 2, we compare the PAPI support for the first Xeon Phi co-processor (Knights Corner) and the latest Xeon Phi processor (Knights Landing). PAPI offers two components for CPU counters:

1. A CORE component for on-core-specific events that are not shared (e.g., instruction count, cache-related events, etc.).
2. An UNCORE component for shared counters.

There was no UNCORE support for Knights Corner. But for Knights Landing, we offer

PAPI Components	Knights Corner	Knights Landing
perf_event: Linux perf_event CPU **core** events	**PMU's supported**: perf, perf_raw, knc **# of Native Events**: 140 **# of Preset Events**: 14 **# of Counters**: 2	**PMU's supported**: perf, perf_raw, knl, ix86arch **# of Native Events**: 182 **# of Preset Events**: 26 **# of Counters**: 5
perf_event_uncore: Linux perf_event CPU **uncore** and northbridge events	---	**PMU's supported**: e.g. Memory, on-die interconnect, IO, Memory-to-PCIe event support **# of Native Events**: 894

Figure 2: PAPI CPU components: Knights Corner versus Knights Landing.

almost 900 UNCORE events (such as memory-related events like any kind of "retired memory operations", event support for the 2D-mesh interconnect that's on KNL, IO support, event support for memory-to-PCI operations, etc.). On Knights Landing, we have five counters that are available (per hardware thread) for simultaneous event monitoring. However, three of these five are fixed-purpose counters. Each can measure only one specific event:

1. Number of all Instruction at retirement.
2. Number of Core cycles when CPU is not halted.
3. Number of Reference Cycles when CPU is not halted.

That leaves two general-purpose counters (equal to what was available on Knights Corner) that can be used to measure any event.

A.2 Predefined Events: A PAPI predefined event (preset event) is typically derived using a combination of native events to provide guidance to the user regarding meaningful performance counter monitoring, and also to check for correctness. Figure 3 shows the list of the 26 predefined events that PAPI provides on KNL.

Preset Events	Description
PAPI_L1_DCM	Level 1 data cache misses
PAPI_L1_ICM	Level 1 instruction cache misses
PAPI_L1_TCM	Level 1 cache misses
PAPI_L2_TCM	Level 2 cache misses
PAPI_TLB_DM	Data translation lookaside buffer misses
PAPI_L1_LDM	Level 1 load misses
PAPI_L2_LDM	Level 2 load misses
PAPI_STL_ICY	Cycles with no instruction issue
PAPI_BR_UCN	Unconditional branch instructions
PAPI_BR_CN	Conditional branch instructions
PAPI_BR_TKN	Conditional branch instructions taken
PAPI_BR_NTK	Conditional branch instructions not taken
PAPI_BR_MSP	Conditional branch instructions mispredicted
PAPI_TOT_INS	Instructions completed
PAPI_LD_INS	Load instructions
PAPI_ST_INS	Store instructions
PAPI_BR_INS	Branch instructions
PAPI_RES_STL	Cycles stalled on any resource
PAPI_TOT_CYC	Total cycles
PAPI_LST_INS	Load/store instructions completed
PAPI_L1_DCA	Level 1 data cache accesses
PAPI_L1_ICH	Level 1 instruction cache hits
PAPI_L1_ICA	Level 1 instruction cache accesses
PAPI_L2_TCH	Level 2 total cache hits
PAPI_L2_TCA	Level 2 total cache accesses
PAPI_REF_CYC	Reference clock cycles

Figure 3: PAPI Preset event on Knights Landing.

B. Floating-point operation counter issues Another activity we focused on is the exploration of the issues with counting accurate floating-point operations.

On previous Intel architectures, measuring floating-point operations (FLOPs) was either a mess or not possible at all. For instance, on Intel's Sandy Bridge and Ivy Bridge products, the counters were there but they didn't accurately count FLOPs. They were mainly useful for determining what kinds of floating-point instructions were being executed (scalar, 128-bit vector, 256-bit vector) and in what precision (single, double).

On the next generation, the Intel Haswell cpu architecture, Intel decided to disable FLOP counters altogether. The good news is that, on *some* of Intel's latest products, specifically the Broadwell and Skylake CPU architectures, we have a number of FLOP events that are supposedly counted at retirement. For instance, for Intel's Skylake CPU, the PAPI double-precision FLOP counter is a combination of the following four native events:
DOUBLE-precision FLOPs =

```
1 * FP_ARITH_INST_RETIRED.SCALAR_DOUBLE +
2 * FP_ARITH_INST_RETIRED.128B_PACKED_DOUBLE +
4 * FP_ARITH_INST_RETIRED.256B_PACKED_DOUBLE +
8 * FP_ARITH_INST_RETIRED.512B_PACKED_DOUBLE
```

Similarly, PAPI's predefined counter for counting single-precision FLOP operations is a com-

bination of another four native events:
SINGLE-precision FLOPs =

```
1 * FP_ARITH_INST_RETIRED.PACKED_SINGLE +
4 * FP_ARITH_INST_RETIRED.128B_PACKED_SINGLE +
8 * FP_ARITH_INST_RETIRED.256B_PACKED_SINGLE +
16 * FP_ARITH_INST_RETIRED.512B_PACKED_SINGLE
```

Still, this does not allow us to count the total number of FLOPs in a single run (without multiplexing) as this would require eight general-purpose registers while there are only four general-purpose registers available on Broadwell and Skylake. Nonetheless, being able to count double- and single-precision FLOPs separately is an improvement compared with earlier generations of Intel processors.

The bad news is that, on Intel's latest Knights Landing (KNL) processor, events for counting accurate floating-point instructions/operations are not publicly available. The current KNL events (`UOPS_RETIRED.SCALAR_SIMD` and `UOPS_RETIRED.PACKED_SIMD`) cannot be used to accurately measure FLOPs, as they count the number of retired micro-ops, not instructions.

2.1.4 C. Other Intel Processor Support

In addition to the Xeon Phi work, PAPI support for other new processors was enhanced. Support was added for the new Intel Skylake processors, and support for the Intel Broadwell processors was improved.

2.1.5 Impact of PAPI on Domain Disciplines

The PAPI software is widely used for performance modeling of applications in other disciplines of science and engineering. For example, PAPI has been used to model the performance of the HOMME (High Order Methods Modeling Environment) climate code, and the GAMESS and NWChem codes—both computational chemistry suites—on DOE supercomputers. PAPI is widely deployed as a middleware by third-party profiling, tracing, and sampling tools (e.g., CrayPat, HPCToolkit, Scalasca, Score-P, TAU, Vampir, PerfExpert). As more application scientists migrate their applications to HPC platforms, PAPI will enable these scientists to use familiar tools to analyze and optimize performance in these environments as well.

Continued effective collaboration with systems developers at Cray and Intel ensures that PAPI is a viable component of the software stack that they deliver with their high-performance computing systems. We also collaborate with operating system vendors, such as RedHat, which ships PAPI to their downstream users.

2.2 Resilience

UTK developed a multi-criteria research approach for resilience. From a software perspective we investigated different application behavior patterns and programming models to comprehend the most recurrent constructs in HPC. Simultaneously, we developed different practical approaches to fault mitigation, and built theoretical models allowing simulation of current and future execution platforms.

From the perspective of programability, our approach to resilience involves the combination of multiple existing techniques, ranging from checkpoint/restart fault-tolerance, to automatically tolerating failures, to algorithm-specific approaches that involve resilient middleware and resilient algorithms that exploit internal redundancy to detect and correct failures. In order to assess the overhead and the cost (in terms of time-to-completion and energy as an example) of these techniques, we have developed theoretical models of some of the most commonly used fault management techniques (checkpoint/restart, incremental checkpointing, and ABFT). These models were then validated using in-house simulators as well as experiments on some of the largest platforms available today.

In this light, we continued our effort on the Resilience area in complementary directions. While overall we strictly followed the milestones defined in the original project, the last year the stronger push was toward theoretically defining bounds for the different programming constructs added for handling faults, and delivering efficient software components critical to the users communities.

2.2.1 Resilience Models

Despite the increasing importance of fault tolerance in achieving sustained, predictable performance, the lack of models and predictive tools has restricted the analysis of fault tolerant protocols to experimental comparisons only, which are painfully difficult to realize in a consistent and repeated manner. We developed a set of comprehensive models for a wide range of checkpoint/restart protocols, allowing their quantitative assessment.

Reliability Fault-tolerant protocols have different overheads in fault-free and recovery situations. These overheads depend on many factors (type of protocols, application characteristics, system features, etc.) that introduce complexity and limit the scope of experimental comparisons conducted in the past. Using the models developed in the context of this project, we can estimate the waste (overhead) of particular types of applications on specific execution environments. As an example, Figure 4, instantiates the C/R model with features of the

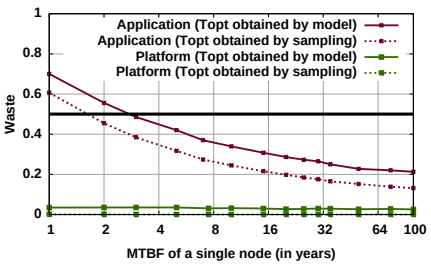

Figure 4: The waste on a modeled K-Computer

K-Computer; the checkpoint growth rate is set according
to a matrix-matrix multiply operation. The results present the waste from the platform perspective (green lines) and from the application perspective (red lines). The optimal checkpoint period is computed by minimizing the model-computed waste.

Energy While reliability is a major concern for Exascale, another key challenge is to minimize energy consumption, both for economic and environmental reasons. One of the most power-consuming components of todays systems is the processor: even when idle, it dissipates a significant fraction of the total power. However, For future Exascale systems, the power dissipated to execute I/O transfers, to create a reliable storage for checkpoints, is likely to play a crucial role, because the relative cost of communication is expected to dramatically increase, both in terms of latency and consumed energy [2].

2.2.2 Message Passing building box for application-level fault mitigation algorithms

User Level Failure Mitigation is a set of MPI interface extensions enabling Message Passing programs to restore MPI communication capabilities affected by process failures. It supports rebuilding communicators, RMA windows, and I/O Files. No particular recovery model is imposed or favored; instead a set of versatile APIs is included that provides support for different recovery styles. The application directs the recovery using the constructs, as necessary, in order to pay for the cost of repairing only the necessary MPI objects. The ULFM specification is a crucial infrastructure for enabling the deployment of advanced, production quality fault tolerant techniques, and is a versatile solution for improving the efficiency of novel and established fault tolerant techniques.

Working together with application developers of two chemistry oriented frameworks, Fenix [9] and LFLR [17], we have improved the performance of one of the most critical fault management routines. Based on communication concepts used by most parallel programming paradigms, we have presented an algorithm that implements an Early Returning Agreement (ERA) in pseudo-synchronous systems. This algorithm optimistically allows a process to resume its activity while guaranteeing strong progress on an agreement operation. We proved the correctness of our ERA algorithm and exposed its logarithmic behavior, an extremely desirable property for any algorithm targeting future exascale platforms. We have detailed a practical implementation of this consensus algorithm in the context of an MPI library, and evaluated both its efficiency and scalability through a set of benchmarks and two fault tolerant scientific applications. The results of this study have been presented at SC'15 [11].

2.3 2015-2016 Peer-reviewed Publications

The peer-reviewed publications during the final year (Sep/2015 to Aug/2016) of the grant are listed below:

- Jagode, H., YarKhan, A., Danalis, A., Dongarra, J. **"Power Management and Event Verification in PAPI"**, Tools for High Performance Computing 2015, 9th International Workshop on Parallel Tools for High Performance Computing, September 2015, Dresden, Germany, Springer International Publishing, Andreas Knuepfer et al. (Eds.), pp. 41-51, 2016.
- Jagode, H., Danalis, A., Bosilca G., Dongarra, J. **"Accelerating NWChem Coupled Cluster through dataflow-based Execution"**, Parallel Processing and Applied Mathematics: 11th International Conference, PPAM 2015, Krakow, Poland, September 6-9, 2015, Springer International Publishing, R. Wyrzykowski et al. (Eds.): PPAM 2015, Part I, LNCS 9573, pp. 366-376, 2016.
- Danalis, A., Jagode, H., Bosilca G., Dongarra, J. **"PaRSEC in Practice: Optimizing a legacy Chemistry application through distributed task-based execution"**, 2015 IEEE International Conference on Cluster Computing, Chicago, Illinois, USA, IEEE, pp. 304-313, September 8-11, 2015.
- Herault, T., Bouteiller, A., Bosilca, G., Gamell, M., Teranishi, Keita, Parashar, M., Dongarra, J. **"Practical Scalable Consensus for Pseudo-synchronous Distributed Systems"**, Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, Austin, Texas, ACM, pp. 31:1–31:12, Nov, 2015.
- Bosilca, G., Bouteiller, A., Guermouche, A., Herault, T., Robert, Y., Sens, P., Dongarra, J. **"Failure Detection and Propagation in HPC systems"**, The International Conference for High Performance Computing, Networking, Storage and Analysis, November 24-26, 2016.

2.4 Summary of Project Activities for entire Grant

The PAPI software is widely used in the field of computer science for application level as well as system-level performance monitoring and modeling. This DOE research funding has allowed us to make substantial progress in providing PAPI performance and power measurements for new architectures. Over the entire period of the grant, the following aspects of the goals of hardware performance counter monitoring have been achieved:

PAPI provides a coherent, operating-system-independent interface to performance counter information for varied hardware and software components, including CPUs [18], GPUs [12], memory, networks [13, 6], I/O systems [18] and power interfaces [14, 8, 5] as well as virtual

cloud environments [19, 20]. Below is a list of architectures that are currently supported by PAPI:

- AMD
- ARM Cortex A8, A9, A15, ARM64
- CRAY: Gemini and Aries interconnects, power/energy
- IBM Blue Gene Series, Q: 5D-Torus, I/O system, EMON power/energy
- IBM Power Series
- Intel Westmere, Sandy Bridge, Ivy Bridge, Haswell, Broadwell, Skylake, Knights Corner, Knights Landing
- Intel power/energy: Knights Corner, Knights Landing
- Intel RAPL (power/energy); power capping
- InfiniBand
- Lustre FS
- NVIDIA Tesla, Kepler: CUDA support for multiple GPUs; PC Sampling
- NVIDIA NVML (temperature, etc.)
- Virtual Environments: VMware, KVM

PAPI can be used independently as a performance-monitoring library and tool for application analysis. However, PAPI finds its greatest utility as middleware component for a number of third-party profiling, tracing, and sampling toolkits (e.g., CrayPat [7], HPCToolkit [1], Scalasca [10], Score-P [15], TAU [16], Vampir [3], PerfExpert [4]), making it the de facto standard for hardware counter analysis. As the middleware component, PAPI handles the details for each hardware component in order to expose a clean API to the higher-level toolkits. The sustained use of PAPI on large DOE hardware procurement over the years confirms the validity of having one high-level interface (PAPI) that provides a consistent API platform as well as operating-system-independent access to hardware-performance counters within CPUs, GPUs, and across the entire compute system.

UTK References

[1] L. Adhianto et al. "HPCTOOLKIT: tools for performance analysis of optimized parallel programs". In: *Concurrency and Computation: Practice and Experience* 22.6 (2010), pp. 685–701. ISSN: 1532-0634. DOI: 10.1002/cpe.1553.

[2] Guillaume Aupy et al. "Optimal Checkpointing Period: Time vs. Energy". In: *CoRR* abs/1310.8456 (2013).

[3] Holger Brunst and Andreas Knöpfer. "Vampir". In: *Encyclopedia of Parallel Computing*. Ed. by David Padua. Springer US, 2011, pp. 2125–2129. ISBN: 978-0-387-09765-7. DOI: 10.1007/978-0-387-09766-4_60.

[4] Martin Burtscher et al. "PerfExpert: An Easy-to-Use Performance Diagnosis Tool for HPC Applications". In: *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–11. DOI: 10.1109/SC.2010.41.

[5] *December 2013 Programming Environments Release Announcement for Cray XE and Cray XK Systems*. http://docs.cray.com/books/S-9407-1312//S-9407-1312.pdf.

[6] *Using the PAPI Cray NPU Component*. http://docs.cray.com/books/S-0046-10//S-0046-10.pdf.

[7] *The CrayPat Performance Analysis Tool*. http://docs.cray.com/books/S-2315-50/html-S-2315-50/z1055157958smg.html.

[8] J. Dongarra et al. "Energy Footprint of Advanced Dense Numerical Linear Algebra Using Tile Algorithms on Multicore Architectures". In: *Cloud and Green Computing (CGC), 2012 Second International Conference on*. Nov. 2012, pp. 274–281. DOI: 10.1109/CGC.2012.113.

[9] Marc Gamell et al. "Exploring Automatic, Online Failure Recovery for Scientific Applications at Extreme Scales". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '14. New Orleans, Louisana, 2014, pp. 895–906. ISBN: 978-1-4799-5500-8.

[10] Markus Geimer et al. "The Scalasca performance toolset architecture". In: *Concurrency and Computation: Practice and Experience* 22.6 (Apr. 2010), pp. 702–719. DOI: 10.1002/cpe.1556.

[11] Thomas Herault et al. "Practical Scalable Consensus for Pseudo-synchronous Distributed Systems". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '15. Austin, Texas: ACM, 2015, 31:1–31:12. ISBN: 978-1-4503-3723-6. DOI: 10.1145/2807591.2807665. URL: http://doi.acm.org/10.1145/2807591.2807665.

[12] Allen D. Malony et al. "Parallel Performance Measurement of Heterogeneous Parallel Systems with GPUs". In: *Proceedings of the 2011 International Conference on Parallel Processing*. ICPP '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 176–185. DOI: 10.1109/ICPP.2011.71.

[13] H. McCraw et al. "Beyond the CPU: Hardware Performance Counter Monitoring on Blue Gene/Q". In: *Proceedings of the International Supercomputing Conference 2013*. ISC'13. Leipzig, Germany: Springer, Heidelberg, June 2013, pp. 213–225.

[14] H. McCraw et al. "Power Monitoring with PAPI for Extreme Scale Architectures and Dataflow-based Programming Models". In: (June 2014).

[15] Marc Schlütter et al. "Profiling Hybrid HMPP Applications with Score-P on Heterogeneous Hardware". In: *Parallel Computing: Accelerating Computational Science and Engineering (CSE)*. Vol. 25. Advances in Parallel Computing. IOS Press, 2014, pp. 773–782. DOI: 10.3233/978-1-61499-381-0-773.

[16] Sameer S. Shende and Allen D. Malony. "The Tau Parallel Performance System". In: *Int. J. High Perform. Comput. Appl.* 20.2 (May 2006), pp. 287–311. DOI: 10.1177/1094342006064482.

[17] Keita Teranishi and Michael A. Heroux. "Toward Local Failure Local Recovery Resilience Model Using MPI-ULFM". In: *Proceedings of the 21st European MPI Users' Group Meeting*. EuroMPI/ASIA '14. Kyoto, Japan: ACM, 2014, 51:51–51:56. ISBN: 978-1-4503-2875-3. DOI: 10.1145/2642769.2642774. URL: http://doi.acm.org/10.1145/2642769.2642774.

[18] D. Terpstra et al. "Collecting Performance Data with PAPI-C". In: *Tools for High Performance Computing 2009* (2009),

[19] V.M. Weaver et al. "Measuring Energy and Power with PAPI". In: *ICPPW '12 Proceedings of the 2012 41st International Conference on Parallel Processing Workshops*. Sept. 2012, pp. 262–268. DOI: 10.1109/ICPPW.2012.39.

[20] V.M. Weaver et al. "PAPI 5: Measuring Power, Energy, and the Cloud". In: *Performance Analysis of Systems and Software (ISPASS), 2013 IEEE International Symposium on*. Apr. 2013, pp. 124–125. DOI: 10.1109/ISPASS.2013.6557155.

3 University of Texas at El Paso

Funding at the University of Texas at El Paso (UTEP) began September 1, 2012. Four graduate students have been partially supported under the project. Work focused on the performance and energy areas as described below. UTEP also continued to maintain the SUPER website and publish the SUPER newsletter.

3.1 Performance

we contributed to performance analysis of the XGC1 fusion code by analyzing performance of the CPU version using HPCToolkit from Rice University and PerfExpert from Texas Advanced Computing Center (TACC). Our analysis, which was presented in the SUPER performance poster at the July 2013 SciDAC PI meeting, determined the causes for low performance in terms of instructions per cycle (IPC) in two key subroutines called by the pushe kernel. In one of the routines, low IPC was due to a high number of expensive floating point operations, such as square root. In the particle search routine, low IPC was due to a high number of data accesses.

We worked with the XGC1 developers on a software engineering approach to map computationally intensive portions of the code to GPUs using OpenACC directives. The particle push routine has already been ported to GPUs using CUDA Fortran, but this approach required maintaining two separate versions of the code for the CPU-only and CPU+GPU implementations. Using OpenACC directives has the advantage of maintaining only a single code base. We worked with the new development version of the XGC1 code that contains the new collision operator. Our original plan was to collect data about data access patterns using the MACPO [16] tool from TACC and to automate the generation and optimization of the OpenACC code using an autotuning framework being developed by TACC. Note that this is not a completely separate approach from the SUPER autotuning framework since the TACC framework uses some of the same components, such as PAPI, PerfExpert, and ROSE, and the two frameworks could be made to interoperate and even be more tightly integrated. The TACC framework is based on ROSE, and unfortunately, the Fortran support in ROSE is not robust enough to parse the XGC1 code and construct the required abstract syntax tree on which the TACC framework operates. Thus, we changed our approach to use the Cray performance tools, including Reveal, which gives guidelines on placement of the OpenACC directives and on optimizing data movement. To use the Cray tools, we had to build XGC1 and the libraries it depends on, including a special thread-safe version of PETSc, using the Cray compiler. We were able to build the code, but it crashed when we tried to run it. Although we were able to obtain some performance data using CrayPat with the code built with the PGI compiler, we were not able to complete the software engineering workflow with the Cray tools.

This work was part of master's project that was completed in December 2014 and presented

at the SE4HPC15 workshop in May 2015 [4]. Although our attempts at using software engineering methodology and tools to produce an OpenACC port of the XGC1 collision operator code did not result in a successful port, such a methodology and tool chain could be highly effective in increasing developer productivity if further work were done on the tools. Currently, porting a code to a new architecture is done by trial and error and takes on the order of man years of effort. Furthermore, the codes that result are hard to maintain and port to new platforms. Scientific application developers desperately need methodologies and tools to help shorten code development time and increase code quality. Our study showed that two proposed methodologies and their corresponding tool chains are not yet effective for codes such as XGC1, but we pointed out weaknesses that need to be addressed to achieve the desired goals. We found that inadequate Fortran 90 support with open source tools such as MACPO and ROSE severely limits their usefulness with scientific application codes written in Fortran. Vendor tools, while more robust, require using the vendor compiler and programming environment. Switching compilers requires recompiling the library dependencies. Specifically, we made the following recommendations:

1. Use a version control system for application code development. We obtained multiple versions of the code from different developers and it was hard to keep straight which version was which. We kept detailed notes but it would have been much easier to have version numbers for the different code versions.

2. Document the application codes with proper comments in the code. We spent much time in email and phone calls with the code developers to try to understand what various parts of the code did. It would save our time and theirs to have the code better documented.

3. Remove compiler dependencies with libraries. This is not something that application developers can accomplish on their own. If libraries had clean interfaces rather than relying on shared data structures, the requirement to use a specific library version compiled with a specific compiler could be reduced, thus enabling more agile code development and porting.

4. Incorporate tool use into code development and increase collaboration between tool and application developers. Such collaboration would expose tool inadequacies and motivate tool developers to address them and provide better interoperability and language support.

To help make performance tools that analyze non-uniform memory access (NUMA) performance problems more accessible to users, we worked on the design and preliminary implementation of a NUMA sampling interface for PAPI [10]. Recent processors include hardware support for detailed instruction sampling. For example, Intel Precise Event Based Sampling with Load Latency (PEBS-LL) and AMD Instruction Based Sampling (IBS) enable gathering detailed information for a sample, including register state, pipeline status, and stall accounting. For memory access events, information such as access latency, operand address, data source for loads (e.g., cache level or memory and whether the access is local or remote)

can be returned. Such information is extremely useful for performance tuning on modern multicore systems. For example, being able to sample memory accesses that exceed a specified latency threshold and map these accesses to the data addresses responsible can help code developers analyze and improve data layout on a NUMA system. Performance tools such as HPCToolkit-NUMA from Rice University [] and MemAxes from LLNL [] use such facilities to collect NUMA data for analysis and visualization of NUMA performance problems. TAU developers have expressed interest in extending TAU to include NUMA data in TAU profiles. Other NUMA analysis tools that have been developed by the research community include Memphis [] and MemProf [].

NUMA analysis tools have used either the low-level Linux perf_event interface [] or a custom kernel module to collect NUMA data. It is a burden for tool developers to have to keep updating their data collection tools to work with the lastest Linux kernel releases. It is also a burden for users to have to install the data collection tools or their dependencies and they may not have permissions to do so. Having a tool interface for NUMA sampling as part of PAPI would enable a tool built on top of this interface to work anywhere PAPI is installed and supported, thus improving accessibility and usability of the tool.

In our experimental PAPI-NUMA prototype implementation [Lopez 2015], the client code calls PAPI_sample_init() to set up sampling on a desired hardware event. The client code is then required to set up an mmap buffer for and associate it with each file descriptor that is returned by the PAPI_sample_init() call. The client code also must provide an interrupt handler that gets called whenever there is an overflow for the sampled event. Although not part of the PAPI library, a utility routine is provided that the developer of the client code can use as is or modify as needed to parse the mmap buffer and read the sample data. PAPI-NUMA has routines for configuring, starting, and stopping sampling.

Performance portability and programming productivity are becoming increasingly important in the current petasacle and the approaching exascale eras. Project managers are interested in applying software engineering methodologies to address these issues. We have carried out preliminary investigations in scientific programming productivity [13] and performance portability [14]. Our work on an evaluation framework for scientific programming productivity pointed out that long-term productivity goals, such as portability and maintainability of scientific codes, need to be evaluated with metrics, in addition to short-term development time metrics. In fact, "quick and dirty" development practices can shorten development time but adversely affect maintability and portability. We used our evaluation framework to compare programming productivity with linear algebra libraries with two different groups of domain scientists using 1) library and platform documentation and 2) the Lighthouse tool [5]. We found that while Lighthouse assisted in automatically generating working PETSc codes, the generated code needed modifications to work with the target problem and more guidance for the user was needed to make correct and efficient use of the code. In our work on performance portability, we compared code complexity and performance for three different GPU implementations (CUDA, OpenCL, and OpenACC) for three benchmark codes (Game of Life, CloverLeaf, LULESH). We measured the source lines of code (SLOC) and

various types of complexity (cyclomatic, design, and essential complexity), as well as the runtimes for the different implementations of the benchmarks on a GPU node on the Titan supercomputer. The CUDA implementations ran significantly faster than the OpenACC implementations, and the CUDA and OpenCL implemenations had higher SLOC counts. Code complexity measures are used by the software engineering community to estimate coding error rates and amount of programmer effort to develop or maintain codes. Our complexity results were pretty much the same across different GPU implementations, with the exception that the CUDA implementation tended to have lower cyclomatic complexity due to not having branches in the routines. This result is counter to the intuition in the scientific programming community that CUDA and OpenCL codes are harder to develop, port, and maintain than OpenACC implementations. Thus, our conclusion is that different complexity measures are needed for GPU programming.

3.2 Energy

We formulated a new metric called iso-power-efficiency that describes scaling under a power budget. The idea is to scale up the problem size with increasing power budget, rather than just with increasing numbers of processors. For many applications, speedup saturates and parallel efficiency decreases if the problem size is held fixed while increasing the number of processors (the form of scaling known as strong scaling). For some problems, it is possible to maintain a fixed parallel efficiency by increasing both the problem size and the number of processing elements. The rate at which the problem size must increase to maintain constant efficiency for a given rate of increase of the number of processors is given by the iso-efficiency function [3]. We have developed a new scalability function called *iso-power-efficiency* that determines the rate at which the problem size must increase to maintain constant efficiency for a given rate of increase of the application's power budget. For a given power budget, an application can choose to use a larger number of processors running at lower power. As shown in [15], speedup can often be obtained within a given power budget by such overprovisioning. Deriving the iso-power-efficiency function for a given problem involves 1) determining optimal configurations for problem instance/power budget pairs, and 2) expressing the parallel overhead as a function of problem size and power budget. We have shown that the rate of growth required for problem size can be lower with iso-power-efficiency than with iso-efficiency, thus yielding better scalability, and we have developed a regression modeling methodology, similar to the focused regression modeling described in [1], for fitting observed execution data to an iso-power-efficiency function. This work is expected to lead to a PhD disseration, and the UTEP PhD student had an internship for summer 2014 to work on this topic in collaboration with LLNL researchers. A publication described the iso-power-efficiency metric and including initial experimental results [9].

We worked with researchers at Oak Ridge National Laboratory on a preliminary implementation of power modeling for the ASPEN performance modeling tool [12]. ASPEN is part of the COMPASS modeling framework [6] and is a semi-analytical performance modeling tool.

ASPEN machine and application models are generated in the ASPEN modeling language, either by hand or with tools such as OpenARC for source code analysis. The machine and application models are used to predict the runtime performance of the application on the machine architecture. Since ASPEN is semi-analytical and not black-box, machine and application characteristics can be varied to predict performance on architectural variants (e.g, more cores with decreased memory bandwidth per core) or of possible application optimizations (e.g., if a certain degree of vectorization could be achieved).

We used the Berkeley Roofline Toolkit benchmark [7] to vary W (flops) and Q (bytes moved from memory to cache) and did several runs for which we measured runtime and energy (using RAPL). We fit the data to the regression equation $E = W\epsilon_flop + Q\epsilon_byte + \pi_1 T$, where E is energy in Joules, $epsilon_flop$ is energy per flop, $epsilon_byte$ is energy per byte moved, π_1 is static power, and T is runtime. We used the power model from [2] to implement a powerline tool that constructs and graphically displays a powerline for a given machine model. The powerline for one node of Catalyst, which is an IvyBridge system at LLNL with 12-core nodes, is shown in Figure 5. We can also use the measured machine parameters to predict runtime and energy consumption for an application code, provided we can determine the application parameters. Examples are shown in Tables **??** and **??**. In these tables, p is the number of processing cores and n is the problem size. We determined application parameters with both source code analysis and hardware counter measurements. The predicted runtimes are generally 15-20% under the measured runtimes. This is probably because we are assuming complete overlap of computation and data movement and we are not considering costs of moving data up and down the cache hierarchy. The predicted energy consumption is very close to the measured energy consumption. This is because we are using a black-box model parameterized by benchmarks that have the same characteristics as our application (for example, vectorized or non-vectorized). We are currently working on: increasing the accuracy of the runtime prediction by considering the costs of accessing various levels of cache, 2) making our energy prediction more analytical by better understanding contributions from different components of the system.

In collaboration with researchers at Lawrence Livermore National Laboratory, we have investigated aspects of power shifting in a power bounded system for 1) mitigating the effects of noise and 2) ensuring that power shifting is carried out in a safe manner. As the number of cores in HPC systems grows, so does the effect of system noise on applications running on these systems. With the knowledge that future large-scale parallel computer systems, including exascale systems, will operate under an overall power bound, we propose solutions that can counter the effects of noise. We have developed two methods that estimate the effects of noise on an application and then redistribute power among nodes, such that the effects of noise are hidden [8]. The workload investigated is a load balanced stencil code. In the first method, we create groups of nodes. The size of these groups must be small enough so that the communication cost amongst nodes in these groups does not outweigh our benefit. Every T iterations, each node compares its runtime to that of its immediate neighbors, and then sends the largest runtime to its group leader. Each node also sends its current power

budget. The group leader then takes these runtimes and power budgets and redistributes power among the nodes in its group. Power is redistributed in such a way that slower nodes get more power and faster nodes get less. This is done in an attempt to compensate for any noise bubbles that may have occurred. This process is continued every T iterations, but each iteration the nodes in a given group shift by one. By doing this, all nodes will eventually have their power redistributed appropriately. Our second method builds on the first. The only difference is that we do not shift nodes in a group every iteration and our leader nodes are part of an additional group. Lets call the groups from the first method the local group and the group of leader nodes the global group. So every T steps when power is to be redistributed, the leader nodes gather the same information from the nodes in their local group. They then broadcast the worst runtime and total power from their local group to all other leader nodes in the global group. Each leader node takes this information, does the exact same calculations, and redistributes total power among local groups. Once figuring out their own local groups power, they redistribute power within their local group as previously shown in the first method. Our first method has the advantage of very little communication cost among nodes, but the disadvantage of not globally redistributing power. Through both an analytical model and simulation using actual noise data, we showed that the first method is more effective in hiding system noise. We are working on extending the method to handle arbitrary DAG (directed acyclic graph) representations of parallel computations.

Future large-scale parallel computer systems, including exascale systems, will operate under an overall power bound. This bound will be enforced by having nodes adhere to local power caps, the sum of which cannot exceed the global power bound. To make efficient use of the available power, protocols have been developed to shift power between nodes, for example to compensate for load imbalance or system noise. The distributed power shifting protocol needs to provide a guarantee of safety that the global power bound will never be exceeded and it needs to have acceptably low overhead under normal operating conditions. We present a variant of the Paxos protocol in [11] that guarantees safety. We give an upper bound for the communication overhead of this protocol, along with suggestions for how the overhead can be lowered.

UTEP References

[1] Bradley J. Barnes et al. "Using focused regression for accurate time-constrained scaling of scientific applications". In: *International Parallel and Distributed Processing Symposium (IPDPS)*. 2010, pp. 1–12.

[2] Jee Choi et al. "Algorithmic time, energy, and power on candidate HPC building blocks". In: *IEEE International Parallel and Distributed Processing Slymposium (IPDPS)*. Phoenix, AZ, USA, May 2014.

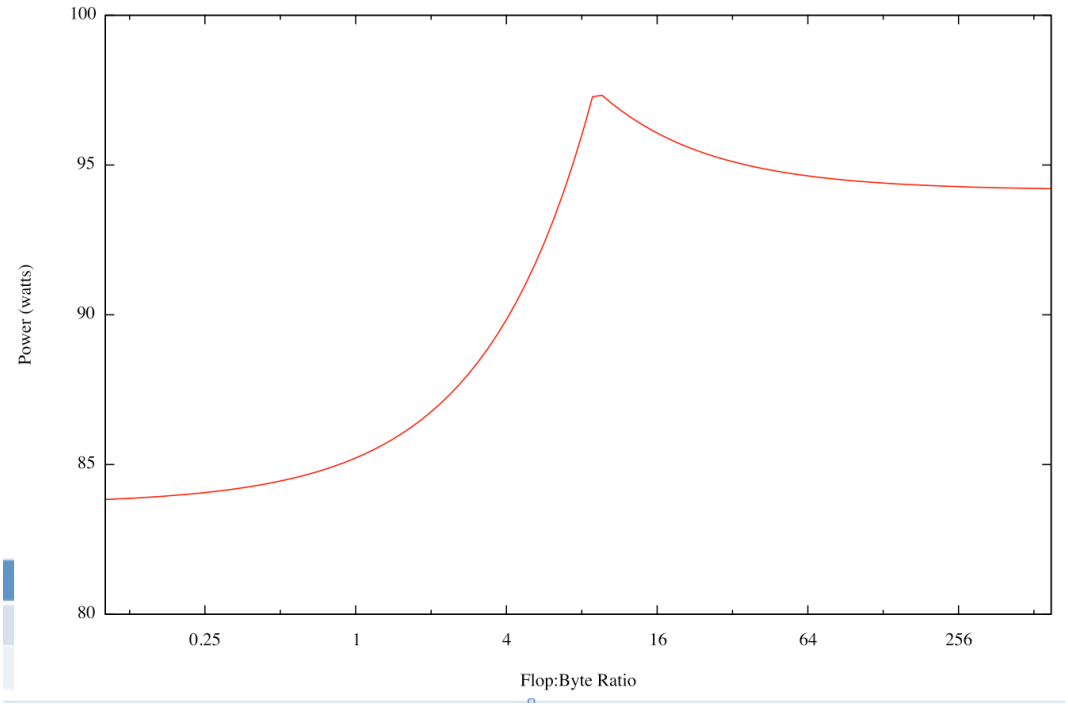

Figure 5: Power draw for one node of Catalyst at LLNL modeled by ASPEN powerline tool

[3] Ananth Y. Grama, Anshul Gupta, and Vipin Kumar. "Isoefficiency: measuring the scalability of parallel algorithms and architectures". In: *IEEE Parallel and Distributed Technology* 1 (3 Aug. 1993), pp. 12–21.

[4] Madhu Hari and Shirley Moore. "Using software engineering methodologies to port a scientific code to GPUs". In: *International Workshop on Software Engineering for High Performance Computing in Science*. Firenze, Italy, May 2015.

[5] Elizabeth Jessup et al. *Lighthouse: A User-Centered Web Interface to Matrix Algebra Software*. `http://lighthouse-taxonomy.googlecode.com/files/lighthouse.pdf`. Argonne, IL, Apr. 2012.

[6] Seyong Lee, Jeremy S. Meredith, and Jeffrey S. Vetter. "COMPASS: A framework for automated performance modeling and prediction". In: *International Conference on Supercomputing (ICS 2015)*. Newport Beach, CA, June 2015.

[7] Yu Jung Lo et al. "Roofline model toolkit: a practical tool for architectural and program analysis". In: *Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. New Orleans, LA, USA, Nov. 2014.

[8] Rogelio Long and Shirley Moore. "Countering the noise-induced critical path problem". In: *Twelfth Workshop on High-Performance Power-Aware Computing (HPPAC)*. Chicago, IL, USA, May 2016.

[9] Rogelio Long, Shirley Moore, and Barry Rountree. "Iso-power-efficiency: an approach to scaling application codes with a power budget". In: *Eleventh Workshop on High-Performance Power-Aware Computing (HPPAC)*. Hyderabad, India, May 2015.

[10] Ivonne Lopez, Shirley Moore, and Vincent Weaver. "A prototype sampling interface for PAPI". In: *XSEDE15*. St. Louis, MO, July 2015.

[11] Shirley Moore. "Achieving safety for power shifting in overprovisioned high performance computing systems". In: *Twelfth Workshop on High-Performance Power-Aware Computing (HPPAC)*. Chicago, IL, USA, May 2016.

[12] Shirley Moore et al. "Performance and power modeling using ASPEN (poster abstract)". In: *Smoky Mountains Computational Sciences and Engineering Conference*. Gatlinburg, TN, USA, Sept. 2015.

[13] Umayanganie Munipala and Shirley Moore. "An evaluation framework for scientific programming productivity". In: *International Workshop on Software Engineering for Science*. Austin, TX, USA, May 2016.

[14] Umayanganie Munipala and Shirley Moore. "Code complexity versus performance for GPU-accelerated scientific applications". In: *Fourth International Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering*. Salt Lake City, Utah, USA, Nov. 2016.

[15] Tapasya Patki et al. "Exploring hardware overprovisioning in power-constrained, high performance computing". In: *International Conference on Supercomputing (ICS'13)*. Eugene, Oregon, June 2013.

[16] Ashay Rane and James Browne. "Enhancing performance optimization of multicore chips and multichip nodes with data structure metrics". In: *21st International Conference on Parallel Architectures and Compilation Techniques (PACT 2012)*. Minneapolis, MN, Sept. 2012.