Measuring Energy and Power with PAPI

Vincent M. Weaver, Matt Johnson, Kiran Kasichayanula, James Ralph,
Piotr Luszczek, Dan Terpstra, and Shirley Moore
Innovative Computing Laboratory
University of Tennessee
{vweaver1,mrj,kirankk,ralph,luszczek,terpstra,shirley}@eecs.utk.edu

***Abstract*—Energy and power consumption are becoming critical metrics in the design and usage of high performance systems. We have extended the Performance API (PAPI) analysis library to measure and report energy and power values. These values are reported using the existing PAPI API, allowing code previously instrumented for performance counters to also measure power and energy. Higher level tools that build on PAPI will automatically gain support for power and energy readings when used with the newest version of PAPI.**

We describe in detail the types of energy and power readings available through PAPI. We support external power meters, as well as values provided internally by recent CPUs and GPUs. Measurements are provided directly to the instrumented process, allowing immediate code analysis in real time. We provide examples showing results that can be obtained with our infrastructure.

***Index Terms*—energy measurement; power measurement; performance analysis**

I. Introduction

The Performance API (PAPI) [1] framework has traditionally provided low-level cross-platform access to the hardware performance counters available on most modern CPUs. With the advent of component PAPI (PAPI-C) [2], PAPI has been extended to provide a wider variety of performance data from various sources. Recently a number of new components have been added that provide the ability to measure a system's energy and power usage.

Energy and power have become increasingly important components of overall system behavior in high-performance computing (HPC). Power and energy concerns were once primarily of interest to embedded developers. Now that HPC machines have hundreds of thousands of cores [3], the ability to reduce consumption by just a few Watts per CPU quickly adds up to major power, cooling, and monetary savings. There has been a lot of HPC interest in this area recently, including the Green 500 [4] list of energy-efficient supercomputers.

PAPI's ability to be extended by components allows adding support for energy and power measurements without any changes needed to the core infrastructure. Existing code that is already instrumented for measuring performance counters can be re-used; the new power and energy events will show up in event listings just like other performance events, and can be measured with the same existing PAPI API. This will allow current users of PAPI on HPC systems to analyze power and energy with little additional effort.

There are many existing tools that provide access to power and energy measurements (often these come with the power measuring hardware). PAPI's advantage is that it allows measuring a diverse set of hardware with one common interface. Users only instrument their code once, and then can use it with minimal changes as their code is moved between different machines with different hardware. Without PAPI the instrumented code would have to be re-written depending on what power measurement hardware it is running on.

Another benefit of PAPI is that in addition to measuring energy and power, it also provides access to other values, such as CPU performance counters, GPU counters, network, and I/O. All of these can be measured at the same time, providing for a richer analysis environment. Many of the other advanced PAPI features, such as sampling and profiling, can potentially be used in conjunction with these new power and energy events. Higher-level tools that build on top of PAPI (such as TAU [5], HPCToolkit [6], or Vampir [7]) automatically get support for these new measurements as soon as they are paired with an updated PAPI version.

We will describe in detail the various types of power and energy measurements that will be available in the PAPI 5.0 release, as well as showing examples of the data that can be gathered.

II. Related Work

There are various existing tools that provide access to power and energy values. In general these tools do not have a cross-platform API like PAPI, nor are they deployed as widely. PAPI has the benefit of allowing energy measurements at the same time as CPU and other performance counter measurements, allowing analysis of low-level energy behavior at the source code level. PAPI can also act as an abstraction library, so most of the tools listed below could be given PAPI component interfaces.

The tool that provides the most similar functionality to PAPI is the Intel Energy Checker SDK [8]. It provides an API for instrumenting code and gathering energy information from a variety of external power meters and system counters. It provides support for various operating systems, but is limited to Intel architectures.

PowerPack [9] provides an interface for measuring power from a variety of external power sources. The API provides routines for starting and stopping the gathering of data on the remote machine. Unlike PAPI, the measurements are gathered out-of-band (on a separate machine) and thus cannot be directly provided to the running process in real time.

IBM Power Executive [10] allows monitoring power and energy on IBM blade servers. As with PowerPack, the data is gathered and analyzed by a tool (in this case IBM Director) running on a separate machine.

Shin et al. [11] construct a power board for an ARM system that estimates power and communicates with a front-end tool via PCI. Various tools are described that use the gathered information, but there is not a generic API for accessing it.

The Linux Energy Attribution and Accounting Platform (LEA^{2}P) [12] acquires data on a system with hardware custom-modified to provide power readings via a data acquisition board. These values are passed into the Linux kernel and made available via the `/proc` filesystem and can be read in-band.

PowerScope [13] uses a digital multimeter to perform offline analysis using statistical sampling. It provides a kernel-level interface (via system calls) to start and stop measurements; this requires modifying the operating system. The benefit of this system is that power information is kept in the process table, allowing one to map energy usage in a detailed per-process way.

The Energy Endoscope [14] is an embedded wireless sensor network that provides detailed real-time energy measurements via a custom-designed helper chip. The Linux kernel is modified to report energy in `/proc/stat` along with other processor stats.

Isci and Martonosi [15] combine external power meter measurements with performance counter results to generate power readings with a modeled CPU. The readings are gathered on an external machine.

Bellosa [16] proposes *Joule Watcher*, an infrastructure that uses hardware performance counters to estimate power and provide this information to the kernel for scheduling decisions. He proposes a generic API to provide this information to users.

III. Background

PAPI users have recently become more concerned with energy and power measurements. Part of this is due to the addition of embedded system support (including ARM and MIPS processors) and part is from the current interest in energy-efficiency in PAPI's traditional HPC environment.

With PAPI-C (component PAPI) it is straightforward to add extra PAPI "components" that report values outside of the usual hardware performance counters that were long the mainstay of PAPI. The PAPI API returns unsigned 64-bit integers; as long as a power or energy value can fit that constraint no changes at all need to be made to existing PAPI code.

A. New PAPI Interfaces

The existing PAPI interface is sufficient for providing power and energy values, but the recent PAPI 5.0 release adds many features that improve the collection of this information.

The most important new feature is enhanced event information support. The user can query an event and obtain far richer details than were available previously. The new interface allows specifying units for a returned value, allowing a user to know if the values they are getting are in "Watts", "Joules" or perhaps even "nano-Joules" without having to look in the system documentation. Another new feature is the ability to return values other than unsigned integers, including floating point. This allow returning power values in human-friendly amounts such as 96.45 Watts rather than 96450 milliwatts.

Additional event information is provided that will help external tools analyze the results, especially when trying to correlate power results with other measurements. PAPI now provides the frequency with which the value is updated and whether the value returned is instantaneous (like an average power reading) or cumulative (total Energy).

B. Limitations

There are some limitations when measuring power and energy using PAPI. Typically these readings are system-wide: it is not possible to exactly map the results exactly to the user's code, especially on multi-core systems. Often a user is interested in knowing where the power usage comes from: power supply inefficiencies, the CPU, network card, memory, etc. With external power meters it is not possible to break down the full-system power measurements into per-component values. Since power optimization for various hardware components require different strategies, having only total system power might not provide enough information to allow optimization.

Ideally one could correlate power and energy with CPU and other PAPI measurements. This can be done; values can be measured at the same time (although in separate event sets). However due to the nature of the measurements it is hard to get an exact correlation.

Another issue is that of measurement overhead. Since PAPI has to run on the system gathering the results, it contributes to the overall power budget of the system. Tools that measure power externally do not have this problem.

IV. PAPI Energy and Power Components

The new PAPI 5.0 release adds support for various power and energy components.

PAPI components measure power and energy in-band: a program is instrumented with PAPI calls and can read measurement data into the running process. The data *can* be stored to disk for later offline analysis, but by default it is available for immediate action. This contrasts with other tools that only support out-of-band measurements: they can only analyze code at a later time, and the program being profiled is not aware of its current power or energy status.

We use linear algebra routines that perform one-sided factorization of dense matrices to compare various methods of measuring energy. In particular, we test Cholesky factorization from PLASMA [17] on the processor side and LU factorization on the GPU using MAGMA [18]. Both of these are computationally bound and thus show variable power draw by the computing device: either CPU or GPU. Our tests also show memory effects by including memory bound operations such as filling the matrices with initial values.

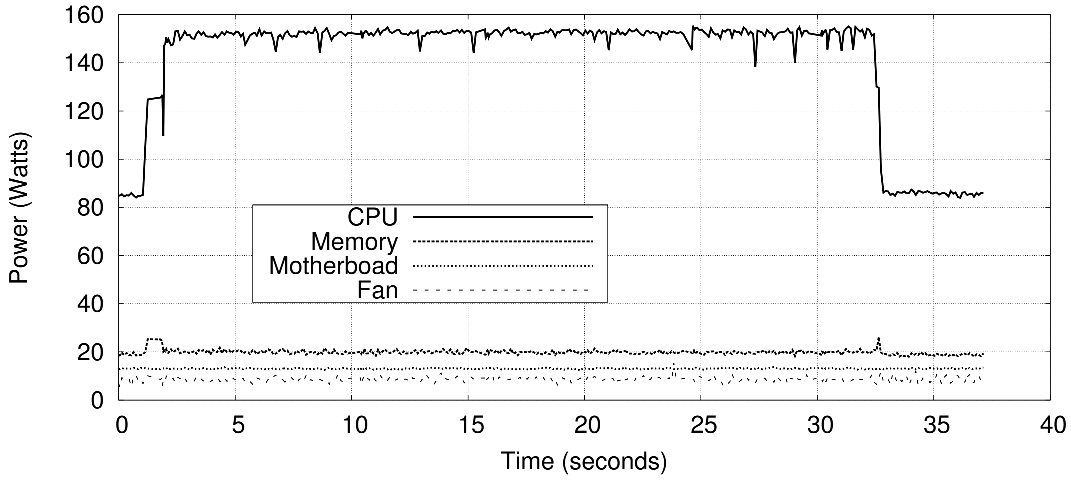

Fig. 1. PLASMA Cholesky power usage gathered by PowerPack (not PAPI). Results were gathered out-of-band; PAPI can gather similar data in-band.

For comparison purposes, Figure 1 shows PLASMA Cholesky results gathered with PowerPack [9] (not PAPI) on a machine custom-wired for power measurement. Results are gathered on an unrelated machine (which has the advantage of not including the overhead of the measurement in the power readings). We show that PAPI can generate similar results from a variety of power measurement devices.

A. External Measurement

The most common type of power measurement infrastructure is one where an external power meter is used. For PAPI to access the data, the values have to be passed back to the machine being measured. This is usually done via a serial or USB connection.

The easiest type of equipment to use in this case is one where a power pass-through is used; this device looks like a power strip, and allows measuring the power consumption of anything plugged into the device.

More intrusive full-system instrumentation can be done, where wires are hooked into power supplies, disks, processor sockets, and DIMM sockets. This enables fine-grained power measurement but usually requires extensive installation costs.

1) Watt's Up Pro Power Meter: The *Watt's Up Pro* powermeter is an external measurement device that a system plugs into instead of a wall outlet; it provides various measurements via a USB serial connection. The metrics collected include average power, voltage, current, and various others. Energy can be derived based on the average power and time. The results are system-wide and low resolution, with updates only once a second.

Writing a PAPI driver for this device is nontrivial, as the results become available every second whether requested or not. Any data can potentially be lost if the on-board logging memory is full and a read does not happen in the one-second time window. Since PAPI users cannot be expected to have their code interrupt itself once a second to measure data, the PAPI component forks a helper thread that reads the data on a regular basis, and then returns overall values when an instrumented program requests it.

Some data gathered from a Watt's Up Pro device are shown in Figure 2. The results are coarse due to the one-second sampling frequency of the device. This can be good enough for doing validation and global investigations, but probably not detailed enough when tuning code for energy efficiency. However, the general trends in power consumption for the code in question (Cholesky factorization from PLASMA [17]) are similar to the much finer-grain graph in Figure 1.

In Figure 2 the initial spike in power consumption to about 50 W (two seconds into the run) represents data generation (creation of a random matrix) and corresponds to a flat ledge at about 130 W in Figure 1. Four seconds into the run, both figures indicate a fluctuation around the maximum power level for the whole run. The fluctuations are much more accurately portrayed in Figure 1, indicating the need for granularity substantially lower than 1 second available for the Watt's Up Pro device.

2) PowerMon 2: The powermon2 [19] card sits between a system's power supply and its various components. It measures voltage and current on 8 different lines, monitoring most of the power going into the computer. Measurements happen at a frequency of up to 3kHz; this is multiplexed across a user-selected subset of the 8 channels.

We are working on a PAPI component for this device, but support is currently not available. We foresee using this device to provide energy results at a detail not available with other external power meters.

B. Internal Measurement

Recent computer hardware includes support for measuring energy and power consumption internally. This allows fine-grained power analysis without having to custom-instrument the hardware.

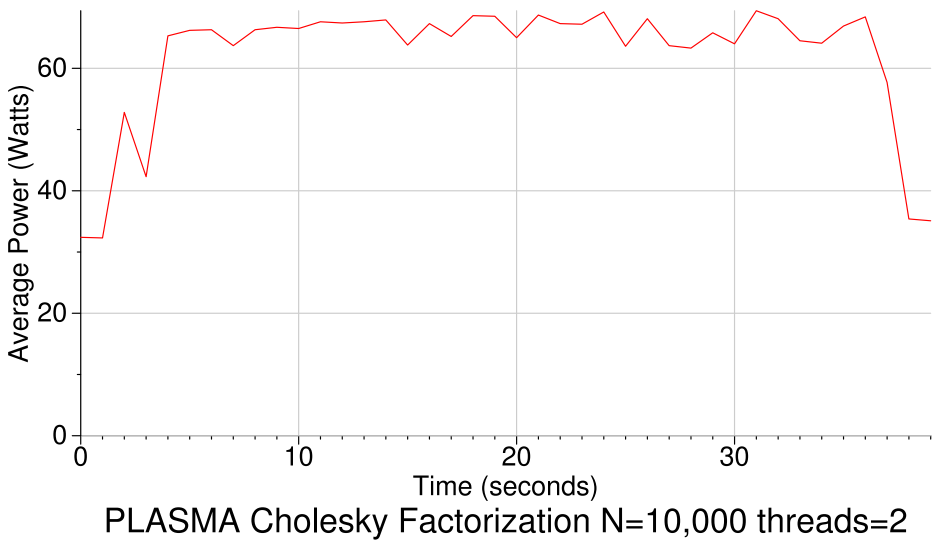

Fig. 2. PLASMA Cholesky power gathered with a Watt's Up Pro device on an Intel Core2 laptop. Coarse results due to one-second sampling frequency.

Access to the measurements usually requires direct low-level hardware reads, although sometimes the operating system or a library will do this for you.

1) Intel RAPL: Recent Intel SandyBridge chips include the "Running Average Power Limit" (RAPL) interface, which is described in the Intel Software Developer's Manual [20]. RAPL's overall design goal is to provide an infrastructure for keeping processors inside of a given user-specified power envelope. The internal circuitry can estimate current energy usage based on a model driven by hardware counters, temperature, and leakage models. The results of this model are available to the user via a model specific register (MSR), with an update frequency on the order of milliseconds. The power model has been validated by Intel [21] to closely follow actual energy being used. PAPI provides access to the values returned by the power model.

Accessing MSRs requires ring-0 access to the hardware; typically only the operating system kernel can do this. This means accessing the RAPL values requires a kernel driver. Currently Linux does not provide such a driver; one has been proposed [22] but it is unlikely it will be merged into the main kernel tree any time soon. To get around this problem, we use the Linux "MSR driver" that exports MSR access to userspace via a special device driver. If the MSR driver is enabled and given proper read-only permissions then PAPI can access these registers directly without needing kernel support.

There are some limitations to accessing RAPL this way. The results are system-wide values and cannot easily be attributed to individual threads. This is not worse than measurements of any shared resource; on modern Intel chips last level caches and the uncore events share this limitation.

RAPL reports various energy readings. This includes the energy usage for the total processor package and the total combined energy used by all the cores (referred to as Power-Plane 0 (PP0)). PP0 also includes all of the processor caches. Some versions of SandyBridge chips also report power usage by the on-board GPU (Power-Plane 1 (PP1)). Sandybridge EP chips do not support the GPU measurement, but instead report energy readings for the DRAM interface.

While the RAPL values can be measured in-band and consumed by the program, since RAPL is system-wide a separate process may be used to measure energy and power. In this way the running code does not need to be instrumented and some of the PAPI overhead can be avoided. We use this method to gather the results presented.

We take measurements on a Sandybridge EP machine. It has 2 CPU packages, each with 8 cores, and each core with 2 threads. Figure 3 shows some average power measurements gathered while doing Cholesky factorization using the PLASMA library. Notice that the energy usage by each package varies, despite all of the cores doing similar work. Part of this is likely due to variations in the cores at the silicon level, as noticed by Rountree et al. [23]. Figure 4 shows the same measurements using the Intel MKL library [24].

Figure 5 shows some energy measurements comparing the same Cholesky factorization using both PLASMA and Intel MKL on the same hardware. The PAPI results show that for this case, PLASMA uses energy more quickly, but finishes faster and uses less total energy for the calculation.

2) AMD Application Power Management: Recent AMD Family 15h processors can report "Current Power In Watts". [25] via the "Processor Power in TDP" MSR. We are investigating PAPI support for this and hope to deploy a component similar in nature and scope to the Intel RAPL component.

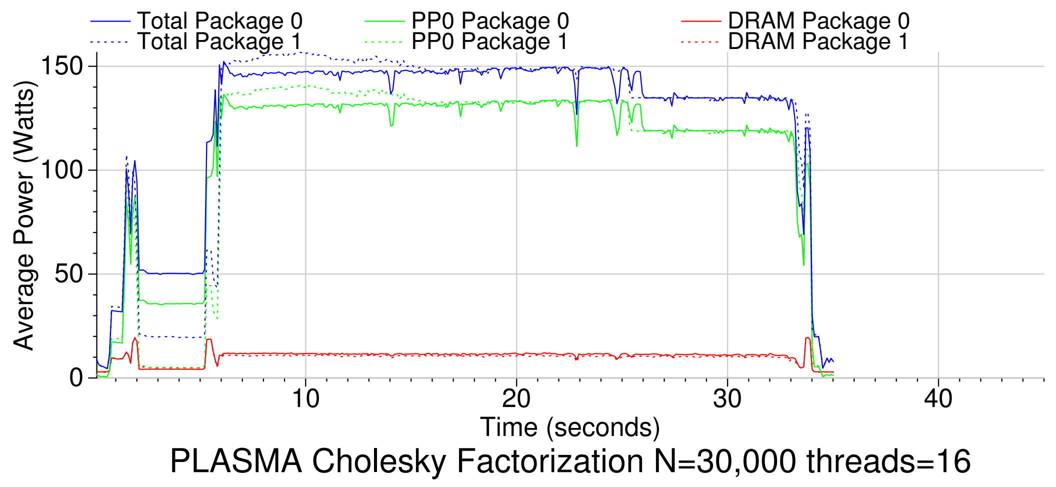

Fig. 3. PLASMA Cholesky power usage measured with RAPL on Sandybridge EP. Power Plane 0 (PP0) is total usage for all 8 cores in a package.

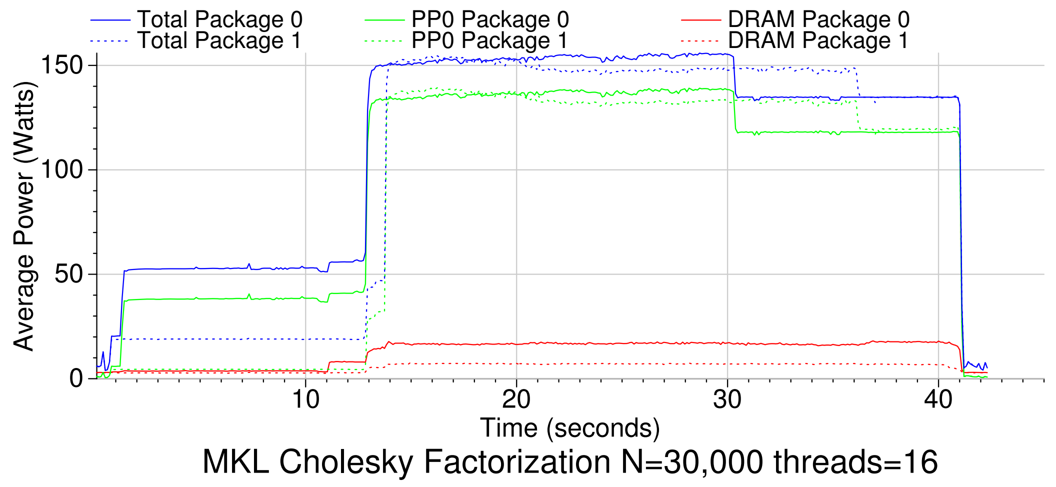

Fig. 4. Intel MKL Cholesky power usage measured with RAPL on Sandybridge. Power Plane 0 (PP0) is total usage for all 8 cores in a package.

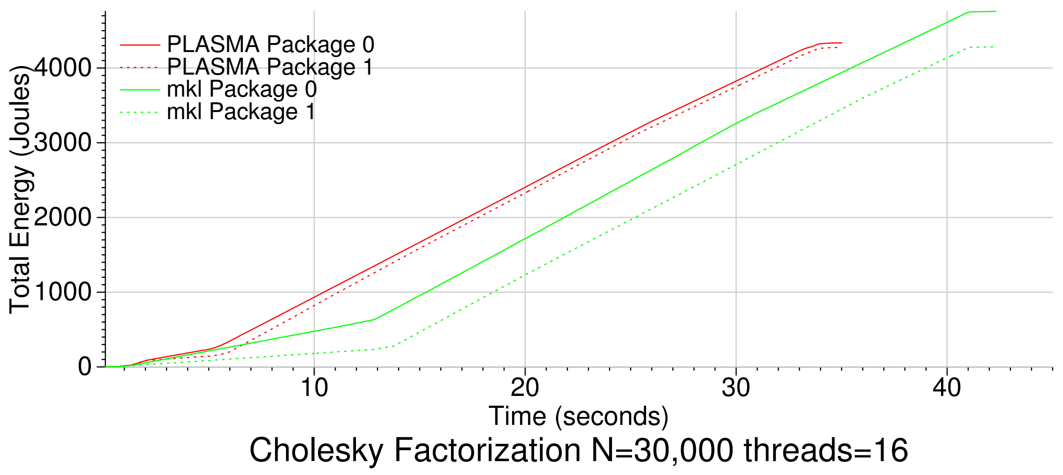

Fig. 5. Energy usage of two different implementations (PLASMA and MKL) of Cholesky on Sandybridge EP measured with RAPL.

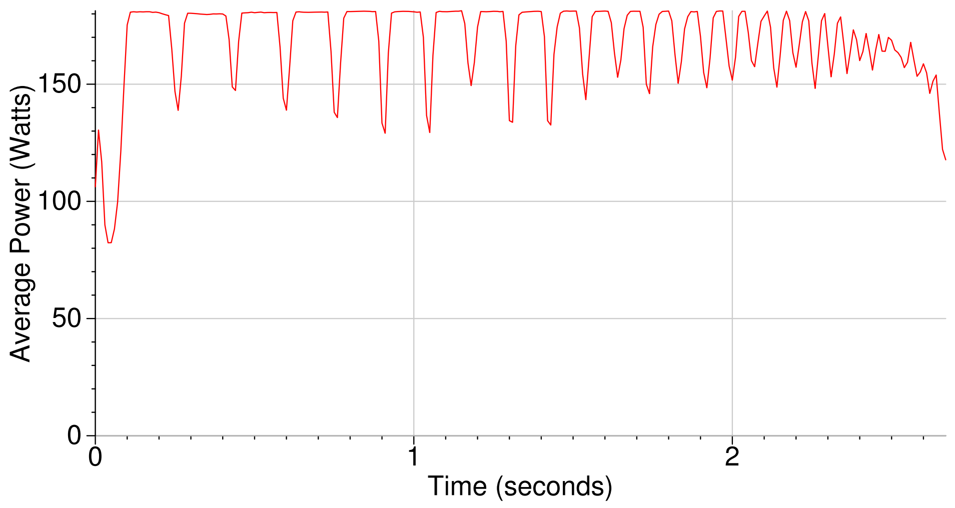

Fig. 6. MAGMA LU with size 10,000 power measurement on an Nvidia Fermi C2075, gathered with NVML.

3) NVIDIA Management Library: Recent NVIDIA GPUs can report power usage via the NVIDIA Management Library (NVML) [26]. The `nvmlDeviceGetPowerUsage()` routine exports the current power; on Fermi C2075 GPUs it has milliwatt resolution within ±5W and is updated at roughly 60Hz. The power reported is that for the entire board, including GPU and memory.

Gathering detailed performance information from a GPU is difficult: once you dispatch code to a GPU the running CPU has no control over it until the GPU returns upon completion. This means that it is not generally possible to attribute what GPU code corresponds to what power readings. Nvidia provides a high-level utility called `nvidia-smi` which can be used to measure power, but its sample rate is too long to obtain useful measurements.

In order to provide better power measurements we have constructed an NVML component [27] for PAPI and have validated the results using a "Kill-A-Watt" power meter.

Figure 6 shows data gathered on an Nvidia Fermi C2075 card running a MAGMA [28] kernel using the LU algorithm [29] with a matrix size of 10k.

The MAGMA LU factorization is a compute bound algorithm (expressed in terms of GEMMs); it uses a hybridization methodology to split the computation between the CPU host and GPU. The split aims to match LU's algorithmic requirements to the architectural strengths of the GPU and the CPU. In the case of LU, this translates into having all matrix-matrix (GEMM) multiplication done on the Gmy_PU, and the panel factorizations on CPU. The design of the algorithm allows for big enough matrices to totally overlap the CPU work with the large matrix-matrix multiplications on the GPU. As a result, the performance of the MAGMA LU algorithm runs at the speed of performing GEMMs on the GPU.

Our experiments have shown that the use of MAGMA GEMM operations on GPU completely utilize it, maximizing the power consumption. This explains why the hybrid LU factorization also maximizes the GPU power consumption, which reduces time taken so the overall energy consumption is minimized.

C. Estimated Power

Various researches have proposed using hardware performance counters to model energy and power consumption [15], [30], [31], [32], [33], [16], [34], [35], [36]. Goel et al. [36] have shown that power can be modeled to within 10% using just four hardware performance counters.

Using the PAPI user-defined events infrastructure [37] an event can be created that derives an estimated power value from the hardware counters. This can be used to measure power on systems that do not have hardware power measurement available.

V. Conclusion

The PAPI library can now provide transparent access to power and energy measurements via existing interfaces. Existing programs that already have instrumentation for PAPI for CPU performance measurements can quickly be adapted to measure power, and existing tools will gain access to the new power events with a simple PAPI upgrade.

With larger and larger clusters being built, energy consumption has become one of the defining constraints. PAPI has been continually extended to provide support for the most up-to-date performance measurements on modern systems. The addition of power and energy measurements allow PAPI users to stay on top of this increasingly important area in the always rapidly changing HPC environment.

Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant No. 0910899 and the U.S. Department of Energy Office of Science under contract DE-FC02-06ER25761.

References

[1] S. Browne, 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*, vol. 14, no. 3, pp. 189–204, 2000.

[2] D. Terpstra, H. Jagode, H. You, and J. Dongarra, "Collecting performance data with PAPI-C," in *3rd Parallel Tools Workshop*, 2009, pp. 157–173.

[3] "Top 500 supercomputing sites," http://www.top500.org/.

[4] "Top green 500 list:: Environmentally responsible supercomputing," http://www.green500.org/.

[5] S. Shende and A. Malony, "The Tau parallel performance system," *International Journal of High Performance Computing Applications*, vol. 20, no. 2, pp. 287–311, 2006.

[6] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, and N. Tallent, "HPCToolkit: Tools for performance analysis of optimized parallel programs," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 6, pp. 685–701, 2010.

[7] W. Nagel, A. Arnold, M. Weber, H.-C. Hoppe, and K. Solchenbach, "VAMPIR: Visualization and analysis of MPI resources," *Supercomputer*, vol. 12, no. 1, pp. 69–80, 1996.

[8] Intel, *Intel Energy Checker: Software Developer Kit User Guide*, 2010.

[9] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K. Cameron, "PowerPack: Energy profiling and analysis of high-performance systems and applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 6, May 2010.

[10] P. Popa, "Managing server energy consumption using IBM PowerExecutive," IBM Systems and Technology Group, Tech. Rep., 2006.

[11] D. Shin, H. Shim, Y. Joo, H.-S. Yun, J. Kim, and N. Chang, "Energy-monitoring tool for low-power embedded programs," *IEEE Design & Test of Computers*, vol. 19, no. 4, pp. 7–17, July/August 2002.

[12] S. Ryffel, "LEA^{2}P: The linux energy attribution and accounting platform," Master's thesis, Swiss Federal Institute of Technology, Jan. 2009.

[13] J. Flinn and M. Satyanarayanan, "PowerScope: a tool for profiling the energy usage of mobile applications," in *Proc. of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, Feb. 1999, pp. 2–10.

[14] T. Stathopoulos, D. McIntire, and W. Kaiser, "The energy endoscope: Real-time detailed energy accounting for wireless sensor nodes," in *Proc. of the International Conference on Information Processing in Sensor Networks*, Apr. 2008, pp. 383–394.

[15] C. Isci and M. Martonosi, "Runtime power monitoring in high-end processors: Methodology and empirical data," in *Proc. IEEE/ACM 36th Annual International Symposium on Microarchitecture*, Dec. 2003.

[16] F. Bellosa, "The benefits of event: driven energy accounting in power-sensitive systems," in *Proceedings of the 9th workshop on ACM SIGOPS European workshop*, 2000.

[17] *PLASMA Users' Guide, Parallel Linear Algebra Software for Multicore Architectures, Version 2.3*, University of Tennessee Knoxville, Nov. 2010.

[18] S. Tomov, R. Nath, H. Ltaief, and J. Dongarra, "Dense linear algebra solvers for multicore with GPU accelerators," in *Proc. 24th IEEE/ACM International Parallel and Distributed Processing Symposium*, Apr. 2010.

[19] D. Bedard, R. Fowler, M. Linn, and A. Porterfield, "PowerMon 2: Fine-grained, integrated power measurement," Renaissance Computing Institute, Tech. Rep. TR-09-04, 2009.

[20] Intel, *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*, 2009.

[21] E. Rotem, A. Naveh, D. Rajwan, A. Anathakrishnan, and E. Weissmann, "Power-management architecture of the Intel microarchitecture code-named Sandy Bridge," *IEEE Micro*, vol. 32, no. 2, pp. 20–27, 2012.

[22] Z. Rui. (2011, May) [patch 2/3] introduce intel_rapl driver. linux-kernel mailing list. [Online]. Available: http://thread.gmane.org/gmane.linux.kernel/1145973

[23] B. Rountree, D. Ahn, B. de Supinski, D. Lowenthal, and M. Schulz, "Beyond DVFS: A first look at performance under a hardware-enforced power bound," in *Proc. of 8th Workshop on High-Performance, Power-Aware Computing*, May 2012.

[24] Intel, *Intel, Math Kernel Library (MKL)*, http://www.intel.com/software/products/mkl/.

[25] AMD, *AMD Family 15h Processor BIOS and Kernel Developer Guide*, 2011.

[26] *NVML Reference Manual*, NVIDIA, 2012.

[27] K. Kasichayanula, "Power aware computing on GPUs," Master's thesis, University of Tennessee, Knoxville, May 2012.

[28] E. Agullo, C. Augonnet, J. Dongarra, H. Ltaief, R. Namyst, S. Thibault, and S. Tomov, "Faster, cheaper, better - a hybridization methodology to develop linear algebra software for GPUs," *LAPACK Working Note 230*.

[29] S. Yamazaki, S. Tomov, and J. Dongarra, "One-sided dense matrix factorizations on a multicore with multiple GPU accelerators," in *Proc. of the 2012 International Conference on Computational Science*, Jun. 2012.

[30] K. Singh, M. Bhadauria, and S. McKee, "Real time power estimation of multi-cores via performance counters," *Proc. Workshop on Design, Architecture and Simulation of Chip Multi-Processors*, Nov. 2008.

[31] I. Kadayif, T. Chinoda, M. Kandemir, N. Vijaykirsnan, M. Irwin, and A. Sivasubramaniam, "vEC: virtual energy counters," in *Proc. of the 2001 ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*, Jun. 2001.

[32] V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software: a first step towards software power minimization," *IEEE Transactions on VLSI*, vol. 3, no. 4, pp. 437–445, 1994.

[33] J. Russell and M. Jacome, "Software power estimation and optimization for high performance, 32-bit embedded processors," in *Proc. IEEE International Conference on Computer Design*, Oct. 1998, pp. 328–333.

[34] R. Joseph and M. Martonosi, "Run-time power estimation in high-performance microprocessors," in *Proc. IEEE/ACM International Symposium on Low Power Electronics and Design*, Aug. 2001, pp. 135–140.

[35] J. Haid, G. Kaefer, C. Steger, and R. Weiss, "Run-time energy estimation in system-on-a-chip designs," in *Proc. of the Asia and South Pacific Design Automation Conference*, Jan. 2003, pp. 595–599.

[36] B. Goel, S. McKee, R. Gioiosa, K. Singh, M. Bhadauria, and M. Cesati, "Portable, scalable, per-core power estimation for intelligent resource management." in *First International Green Computing Conference*, Aug. 2010.

[37] S. Moore and J. Ralph, "User-defined events for hardware performance monitoring," in *Proc. 11th Workshop on Tools for Program Development and Analysis in Computational Science*, Jun. 2011.