

PoLiMEr: An Energy Monitoring and Power Limiting Interface for HPC Applications

Ivana Marincic
University of Chicago
Chicago, IL, USA
imarincic@cs.uchicago.edu

Venkatram Vishwanath
Argonne National Laboratory
Lemont, IL, USA
venkat@anl.gov

Henry Hoffmann
University of Chicago
Chicago, IL, USA
hankhoffmann@cs.uchicago.edu

ABSTRACT

Power and energy consumption are now key design concerns in HPC. To develop software that meets power and energy constraints, scientific application developers must have a reliable way to measure these values and relate them to application-specific events. Scientists face two challenges when measuring and controlling power: (1) *diversity*—power and energy measurement interfaces differ between vendors—and (2) *distribution*—power measurements of MPI simulations should be unaffected by the mapping of MPI processes to physical hardware nodes. While some prior work defines standardized software interfaces for power management, these efforts do not support distributed environments. The result is that the current state-of-the-art requires scientists interested in power optimization to write tedious, error-prone application- and system-specific code. To make power measurement and management easier for scientists, we propose PoLiMEr, a user-space library that supports fine-grained application-level power monitoring and capping. We evaluate PoLiMEr by deploying it on Argonne National Laboratory's Theta system and using it to measure and cap power, scaling the performance and power of several applications on up to 1024 nodes. We find that PoLiMEr requires only a few additional lines of code, but easily allows users to detect energy anomalies, apply power caps, and evaluate Theta's unique architectural features.

CCS CONCEPTS

• **Hardware** → **Power and energy**; • **Computer systems organization** → *Distributed architectures*;

KEYWORDS

power monitoring, power capping, RAPL, Cray XC40, HPC

1 INTRODUCTION

Power consumption and energy efficiency are key design considerations for architecting current and future high performance computing (HPC) systems. There are two main challenges preventing complete and accurate power monitoring and control in HPC applications: *diversity* and *distribution*.

The *diversity* challenge concerns the growing abundance of power monitoring and control options on current HPC systems. Wallace et al. provide an overview and comparison of vendor supplied environmental power monitoring capabilities, highlighting the differences between vendors in accessing and processing environmental data [19]. This diversity makes taking and interpreting simple power measurements a daunting task to scientists interested in studying or improving their application's energy efficiency. Interpretation of measurements is further obfuscated by poor documentation of the low-level, vendor-specific power monitoring/control capabilities (e.g. Desrochers et al. note the difficulty understanding Intel's Running Average Power Limiting (RAPL) interface documentation [3]). Lack of documentation is also true for HPC environments. For instance, while IBM systems have a well documented API for measuring the power of specific hardware components (CPU, memory, optics, PCIe, network transceivers, etc), Cray systems provide a much smaller set of less descriptive power monitoring counters; e.g., their "Energy" counter is not indicative of what it measures at first sight. Measuring application power consumption is thus currently reserved only for expert users well versed in the power characteristics and tools for their specific system.

Even advanced users, however, experience difficulty measuring HPC application power, due to the *distribution* challenge. This issue concerns the difficulty of obtaining power measurements for applications executed in distributed environments using MPI. Widely used tools with power measuring capabilities, like PAPI [14] or Linux perf tools [2], do not handle power measurement of parallel applications arbitrarily scheduled on physical nodes. Traditional power profiling tools in an MPI setting add the significant burden of correctly mapping processes to nodes and limit researchers to experimental setups designed for easier power measurement (e.g., always scheduling one process per node). If multiple MPI ranks are scheduled on the same node and read power, users obtain an overestimate of the power consumption. Similarly, if too few processes read power, users may miss some nodes' power consumption. With the more sophisticated scheduling policies expected from future systems, this problem becomes even worse. Power capping studies face the same issue, and are currently limited to fixed a priori settings given the available tools, providing little flexibility in studying the effects of dynamic power limiting.

To address the diversity and distribution challenges, we propose PoLiMEr—a user-space library for power monitoring and capping of MPI applications. Using known system power

ACM acknowledges that this contribution was authored or co-authored by an employee, contractor, or affiliate of the United States government. As such, the United States government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for government purposes only.
E2SC'17, November 12–17, 2017, Denver, CO, USA
© 2017 Association for Computing Machinery.
ACM ISBN 978-1-4503-5132-4/17/11...$15.00
https://doi.org/10.1145/3149412.3149419

interfaces and topology-aware process management together with its easy-to-use API, PoLiMEr enables all HPC users to navigate the power monitoring options on their system, profile large-scale HPC applications and explore power capping effects. PoLiMEr exhibits extremely low overhead and successfully scales to 1024 nodes on Argonne's system *Theta*, one of the cutting-edge supercomputing platforms of today.

PoLiMEr fully harnesses a system's power monitoring and control capabilities by utilizing all available power monitoring and control interfaces and providing an overview of all measureable components without requiring users to know what these power domains are. Simultaneously, through automated process and thread detection, PoLiMEr can profile applications that use any combination of processes and threads. PoLiMEr's power monitoring and control functionalities are thorough and versatile: it monitors power and energy consumption over time, profiles specific portions of code, sets power caps at any time during application execution and on any combination of nodes. We demonstrate PoLiMEr's component-wise power consumption breakdown on Theta, as well as power consumption characteristics and impacts of selective power capping during application runtime on a variety of workloads. Our results demonstrate that PoLiMEr not only provides a wealth of information about an application's power consumption, but also enables power savings without performance loss by applying power capping during specific application phases.

2 RELATED WORK

PoLiMEr was inspired by an MPI power profiling application for Blue Gene/Q systems developed at the Argonne Leadership Computing Facility, called MonEQ [20]. PoLiMEr takes power monitoring of MPI applications a step further by adding power capping capabilities, extra features, and customizable power monitoring and capping, while supporting the latest Cray supercomputers.

Several HPC profiling tools solve the diversity challenge, but none simultaneously address the distribution challenge posed by MPI applications. An early evaluation of Argonne's XC40 system Theta [15], uses PAPI [14] to measure power on a single node. PAPI users, however, are required to know low-level details corresponding to their specific system. Energymon [7] is a portable energy measurement tool that works on a variety of hardware systems. As is the case with PAPI, Energymon has no support for MPI applications. Work such as the Linux "perf" tools [2] can be invoked from a script and wrap an MPI executable as part of job submission. In this case, users must manage MPI process scheduling themselves to obtain an average power consumption per node. Other tools enable collection of out-of-band power information, such as PDUs, the IPMI, PowerInsight [5, 10] or Cray's Power Management Database (PMDB) [13]. The limitation of script-based approaches and out-of-band tools is the added burden of correctly post-processing the data, mapping readings to corresponding nodes and processes. Furthermore, these data only provide total power measurements, without any further insight into finer-grained application power characteristics.

Power capping's impact on applications is increasingly important in the research community. Pedretti et al. show how power capping affects an MPI application [16]. They perform node-level power capping using CapMC on a Haswell-based Cray XC40 system and evaluate its effects across various static P-states. The authors demonstrate poor performance scaling under power caps, even with manual selection of P-states. PoLiMEr offers the ability to selectively apply power caps for specific workloads, which can yield to greater insights into the relationship between performance and power. Lawson et al. demonstrate a runtime model for dynamic power capping on the Xeon Phi [11]. Their experiments show an improvement in energy savings on single-node, multi-threaded benchmarks, suggesting that workload-aware, runtime power-capping can save energy without affecting performance. We believe that PoLiMEr—with its ability to set power caps during application runtime—will further research in this area beyond single-node applications.

Two notable projects take a global perspective on power monitoring and control: Power API [6] and GEOPM [4]. Both aim to standardize system-wide power monitoring and control protocols. PoLiMEr shares similar objectives, however Power API and GEOPM operate within a grand vision to unify applications, schedulers, workload managers, and system software to obtain a holistic power monitoring and control system. PoLiMEr 's goals are comparatively modest, focusing on helping scientists facing the diversity and distribution challenges, by providing an application-level tool that runs at scale with minimal overhead. PoLiMEr is tested at scales up to 1024 nodes—a third of Argonne's 10 PF Theta system—with overheads as low as 0.4%. In comparison, Power API is demonstrated on a single node [11], while GEOPM is evaluated on 12 nodes. In the future, we hope to compare the overheads, capabilities and implications of PoLiMEr, Power API and GEOPM on HPC applications, should these projects become available on Argonne's systems.

3 POLIMER: AN ENERGY MONITORING AND POWER CONTROL API

This section describes PoLiMEr's[1] general properties. To better understand PoLiMEr's features, we explain power monitoring and capping capabilities of Argonne's XC40 system as a case study, followed by an extensive user guide. Finally, we describe PoLiMEr's outputs upon finalization and provide an example plot.

PoLiMEr is a user-level library that addresses the power monitoring diversity and the distribution challenge specific to MPI applications. PoLiMEr leverages a system's power monitoring and capping capabilities without the user needing to explicitly specify what those capabilities are. PoLiMEr provides the necessary implementation abstractions for power monitoring, power capping and MPI process management, which can be used to add support for various systems. Currently, users simply need to compile PoLiMEr for Intel supercomputers with or without the Cray stack. Future releases of

[1] PoLiMEr - **Po**wer **Li**miting & **M**onitoring of **E**nergy

PoLiMEr will account for more diverse architectures, including IBM systems, and systems with accelerators. Section 3.1 illustrates how specific power interfaces on Cray's XC40 are used by PoLiMEr.

Through a topology-aware process organization mechanism, PoLiMEr enables accurate power and energy measurements, reflecting actual utilization of physical nodes and processors, regardless of how the processes or threads are scheduled. It is not affected by user's process management at the application level, thus making PoLiMEr completely application agnostic. PoLiMEr detects which MPI processes run on which physical nodes using any available processor management interface (e.g. PMI [1]). In the absence of such interface, PoLiMEr uses node hostnames as a fall-back option. The processes are accordingly grouped using communicators specific to each node. One rank per node is then designated as a monitor which reads power and energy, manages PoLiMEr's internal mechanisms, sets power caps and handles output of collected data. In applications also using OpenMP, PoLiMEr will designate one rank and one thread per node as monitor. PoLiMEr is suitable for large- and small-scale, distributed, parallel or serial applications written in C, Fortran and C++. PoLiMEr itself is implemented in C and is available as open source[2].

Using PoLiMEr, a single line of code automates the process of detecting and initializing power monitors. This is a tedious, system- and application-specific process that would otherwise require hundreds of lines of code. Such code would need to be written by an expert user and would not be portable to new systems or even new mappings of processes to nodes on the same system.

3.1 Power Monitoring and Limiting on the XC40

PoLiMEr is designed for any Intel-based distributed system with power monitoring and/or power capping capabilities. Here we give a case study of Argonne's XC40 system Theta to demonstrate the key challenges of diversity of such capabilities. Table 1 gives a brief overview of the power monitoring and power capping options on Theta.

On the XC40 Cray provides several out-of-band and in-band power monitoring capabilities [13]. The software-accessible in-band power monitoring capabilities are exposed as unrestricted read-only files called pm_counters, which PoLiMEr uses to obtain power and energy measurements of the node, CPU and memory. These power domains are architected with energy and power counters updated at a frequency of 10 Hz.

Node-level power capping is possible through Cray's CapMC, an administrative tool requiring elevated privileges. We designed PoLiMEr to enable power capping studies on applications without needing special privileges via Intel's RAPL.

RAPL provides energy monitoring and power capping options on the socket level, corresponding to a single Xeon Phi (KNL or Knights Landing) CPU on Theta. RAPL counters are accessible through Model Specific Register (MSR) files with root privileges, however, when available, PoLiMEr utlizes the msr-safe [18] module, which is installed on Theta. Msr-safe allows users without special privileges to access a set of whitelisted MSRs, sufficient for all power monitoring and capping needs.

[2]https://xgitlab.cels.anl.gov/open-source/PoLiMEr/tags/v1.0

Table 1: Power Monitoring and Capping Capabilities on Theta

	Cray	Intel
Power interface	pm_counters	RAPL interface
Power domains	Node, CPU, Memory	Package, Core, DRAM
Counter types	energy, power	energy
Update interval	100 ms	1 ms
Power Capping	Node-level using CapMC	Package, Core, DRAM

RAPL records energy readings in designated MSRs, one for package, core or Power Plane 0 (PP0) and DRAM power domain. On Theta, DRAM power capping is not supported, nor is energy monitoring of PP0. Energy can be read for the package and DRAM, while power caps can be set to package and PP0. PoLiMEr determines which MSRs are listed for a particular Intel CPU.

Power limiting on Intel CPUs is performed by maintaining average power consumption over a specified moving time window. RAPL allows for brief violations of the power cap by setting a secondary, shorter time window with a different power cap. Time windows and power caps are configurable by writing to the corresponding MSRs. Reading and writing to the MSRs is handled by any tools accessing RAPL, but there are no application-level power limiting tools. PoLiMEr is a self-contained tool with its own ability to correctly parse, read and set MSR values in MPI applications.

3.2 PoLiMEr Usage

To use PoLiMEr, it suffices to only initialize and finalize it, as shown in Example 1. Once initialized, PoLiMEr collects total energy, power and timing information from start to finish. PoLiMEr also polls the power usage at a certain time interval, resulting in power and energy trace information. This interval is set to 0.5 seconds by default.

Example 1: Basic usage of PoLiMEr

```
/* application initialization */
MPI_Init(&argc, &argv);
poli_init();
... //main application work
poli_finalize();
/* application finalization */
MPI_Finalize();
```

To profile specific application tasks, users can tag the corresponding code blocks, as shown in Example 2. Tags can be arbitrarily named and used in many different combinations.

Example 2: Code tagging in PoLiMEr

```
/* basic tagging */
poli_start_tag("my_tag_name");
/* do work */
poli_end_tag("my_tag_name");

/* tagging with loops */
poli_start_tag("outside_loop");
for ( i = 0; i < 10; i++) {
    poli_start_tag("inside_loop");
```

```
    /* do work */
    poli_end_tag("inside_loop");
}
poli_end_tag("outside_loop");
```

Tags are reported and time-stamped in the order they are invoked, providing per-call statistics in addition to aggregate information. In other words, users can observe energy, power and time of each call to `inside_loop` tag in Example 2, or obtain an average value over all iterations.

Power caps can be set in any arbitrary location in the code with several options. PoLiMEr supplies a default power capping function which is applied to the RAPL package domain, but it also lets users specify further RAPL parameters to set long- and short-term power caps on specific power domains (zones). Likewise, users have the option to retrieve and print power cap values, as well as reset power cap values to default values (PoLiMEr resets power caps on finalization automatically). See Example 3.

Example 3: Setting power caps in PoLiMEr

```
/* Sets general 150 W long-term and short-term power
    cap to RAPL package domain */
poli_set_power_cap(150);
/* do work*/

/* Returns current long-term power cap set on RAPL
    package domain */
double result_watts;
poli_get_power_cap(&result_watts);

/* Setting specific power cap */
poli_set_power_cap_with_params(char *zone_name,
    double watts_long, double watts_short, double
    seconds_long, double seconds_short);
//zone names in KNL: PACKAGE, CORE, DRAM

/* Getting a specific power cap parameter*/
poli_get_power_cap_for_param ("PACKAGE", "watts_long"
    , &result_watts);

/* Resetting power caps*/
poli_reset_system();

/* Printing power cap information*/
poli_print_power_cap_info();

/* Getting maximum and minimum supported power cap*/
double min, max;
poli_get_power_cap_limits("PACKAGE", &min, &max);
```

PoLiMEr can return the MPI subcommunicator, the node name/ID of the monitoring rank as well as monitor status of any rank. This allows advanced users to apply power caps on a specific set of nodes in a controlled way. Example 4 demonstrates how to set power caps on some subset of nodes, where the nodes are labeled numerically on Theta. PoLiMEr will accept any format of node labels.

Example 4: Setting power caps on some select nodes

```
int node_num;
poli_get_monitor_node(&node_num);
if (node_num > 3000)
        poli_set_power_cap(150);
```

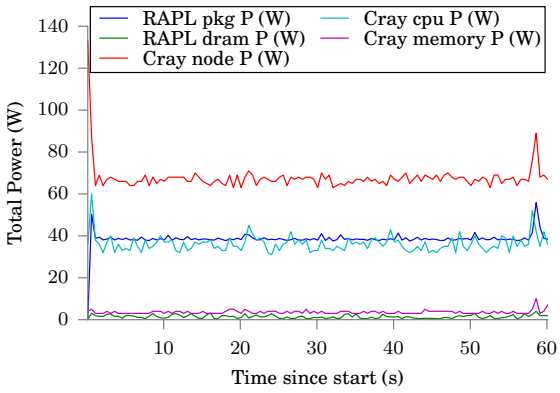


Figure 1: Idle power consumption on a single node on Theta over time across power domains measured by PoLiMEr.

3.3 PoLiMEr Output

After PoLiMEr is finalized, a set of three output files will be produced for each node the application used:

- PoLiMEr_*node-id*_*job-id*.txt - contains the polled power measurements over time (power trace), marking the start and end of all user tags and power caps
- PoLiMEr_energy-tags_*node-id*_*job-id*.txt - contains a summary for all user tags, displaying total energy, power and time of the tagged code blocks. This file also includes a special, generated tag called application_summary which provides total energy, time and power from when PoLiMEr was initialized to when it was finalized
- PoLiMEr_powercap-tags_*node-id*_*job-id*.txt - contains records of when and what power caps were set

PoLiMEr comes with a supplementary Python script that can process these files as tab-separated CSV files (they can be read by any CSV-file processing tools). Furthermore, setting of the PoLi_PREFIX environment variable allows users to specify additional naming or location of the output files.

All files contain time, power and energy specific columns. The time columns track time stamps, elapsed times since initialization or between two points in time. The power and energy columns record power and energy measurements for the RAPL zones and additional system-specific counters (e.g. Cray counters on the XC40). The trace file also records the polled power caps for each time interval, while the power cap tag file lists all user (energy) tags that were active during a particular power cap.

Figure 1 shows the output of the trace file of an idle node running an MPI program that sleeps for a minute. On Theta we can measure power consumption of the package and DRAM RAPL domain, as well as access the Cray power counters for node, CPU and memory. A user would see this breakdown of components without explicitly specifying them.

The RAPL package domain and Cray CPU counter correspond in measurements, as is the case with RAPL DRAM and Cray memory. The Cray node power counter measures power draw of other peripheral components such as the on-board SSD and the NIC. The slight increases in power consumption

Table 2: Hardware specifications of Theta (XC40)

# Nodes	3624
CPU	7230 KNL
# Cores	64
# nodes/rack	192
L1 cache	32 KB instr. + 32 KB data
L2 cache	1 MB
High-bandwidth memory	16 GB
Main memory	192 GB
NVRAM/node	128 GB SSD
Interconnect	Cray Aries Dragonfly

at the endpoints can be attributed to PoLiMEr polling power draw while partially in the initialization and finishing stage, though because each correspond to a single peak over one second, this should not be of concern.

4 EVALUATION

In this section we demonstrate the plethora of insights PoLiMEr can reveal. We show how PoLiMEr can identify application phases and optimization targets for achieving energy or performance goals on real simulation code (LAMMPS) at scale up to 1024 nodes. We demonstrate PoLiMEr's ability to set power caps at select phases during application runtime, and show how Theta's architectural features, specifically memory configuration, can benefit from power capping. Finally, we present overhead measurements of PoLiMEr. The experiments were run on Theta. Table 2 contains an overview of Theta's specifications.

4.1 Characterizing Application Power Consumption Behavior using PoLiMEr

Here we give an overview of the types of insights into application power consumption characteristics PoLiMEr can provide using LAMMPS [17], a molecular dynamics simulation, as a case study. The experiments presented here use the default scaled Rhodopsin benchmark that is supplied with the source code [9], which allows users to specify the system dimensions (also referred to as repetitions in the benchmark setup file) and number of steps or iterations. On Theta, LAMMPS is compiled with the USER-INTEL and USER-OMP packages. We used the flat memory mode and the quad cluster mode on KNL.

Figure 2 shows the power trace obtained from running LAMMPS on 128 nodes. Approximately the first half of the runtime was spent in the initialization phase. The simulation was performed in 100 iterations. The main simulation phases in LAMMPS can be identified in the verlet.cpp source file, which builds Verlet neighbor lists and runs the key components of the simulation. We used PoLiMEr's tagging feature on this file to obtain energy values for some of LAMMPS's key phases, including setup and building of the neighbor list (neighbor_setup, neighbor_build), pair computation (force_pair), Kspace computation (force_kspace), atom property computation (force_atom), Verlet cleanup, and writing of output. PoLiMEr's energy tags file lists tags as they were called, which allows for easy counting of how many

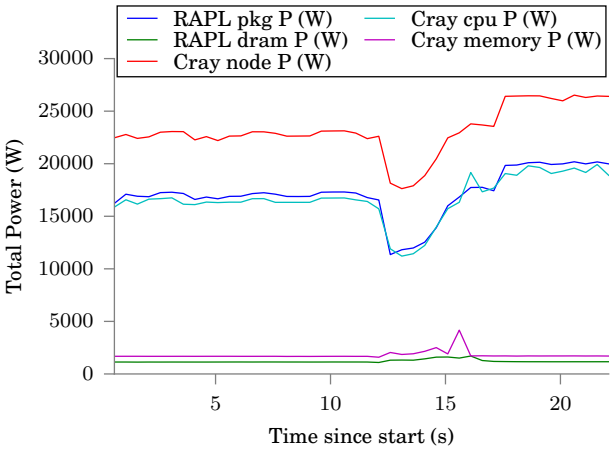


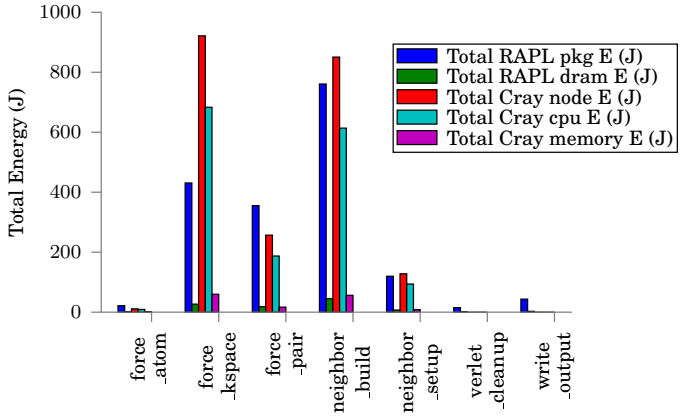
Figure 2: LAMMPS Power Trace. 128 nodes, 64 ranks/node, 2 threads/rank. Rhodopshin benchmark with $6 \times 6 \times 6$ repetitions (6.9 million atoms total) and 100 iterations.

times a particular code block was executed. In the 100 iterations, neighbor setup, atom, Kspace and pair computations are executed at each iteration. Neighbor build is performed 11 times, writing of output twice, and the cleanup phase is called once. Figure 3 shows the average energy consumption and time per tag per iteration. The Kspace computation and neighbor build are the most energy consuming phases, yet their energy consumption is not proportional to the duration of these phases, as shown in Figure 3b. Thus, PoLiMEr's tagging feature enables us to identify potential optimization strategies with minimal changes to the source code (a few tags in a single source file): to optimize for time, the neighbor build phase needs to be addressed, and the Kspace computation to optimize for energy.

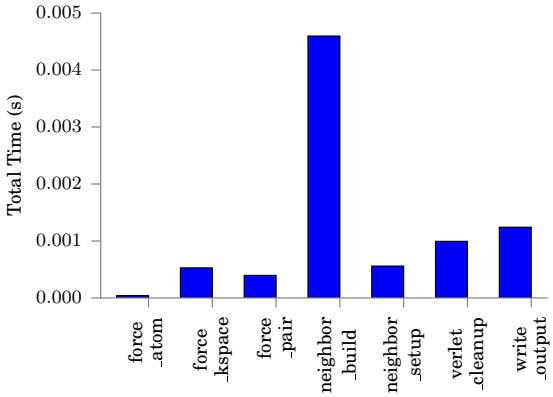
In addition to user specified tags, PoLiMEr will generate an application_summary tag, used to obtain total application energy, power and time measurements per node. Figure 4 shows the total time and energy per node for two separate runs of LAMMPS on 128 nodes and on 1024 nodes. Each of the data points represents a single node. From Figure 4 it is evident that some nodes are more time efficient, others more energy efficient. In the 128-node run there is a 4× performance range, while in the large-scale run there is a difference in energy consumption of close to 20%. Since PoLiMEr's output files are labeled per node, it is easy to identify the exact node label for each of these data points. Therefore, PoLiMEr can enable future research on node variability in energy consumption.

4.2 Power Capping with PoLiMEr

We demonstrate PoLiMEr's capability to set power caps during application runtime using Intel's MKL DGEMM on 64 nodes (1 rank/node, 64 threads/rank, 1 thread/core). Power caps were set on the RAPL package domain using PoLiMEr's default power capping function.

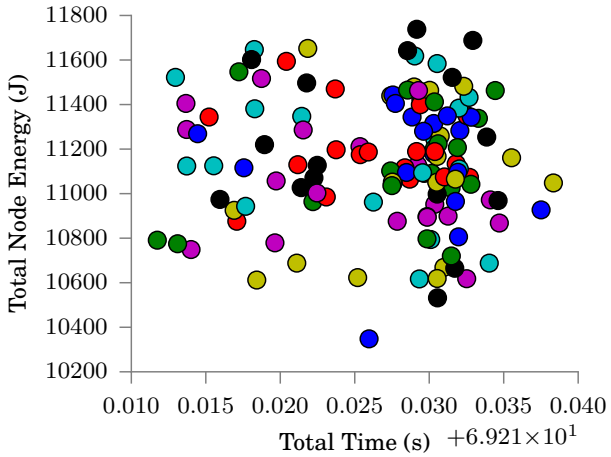


(a) Average energy consumption per tag per iteration.

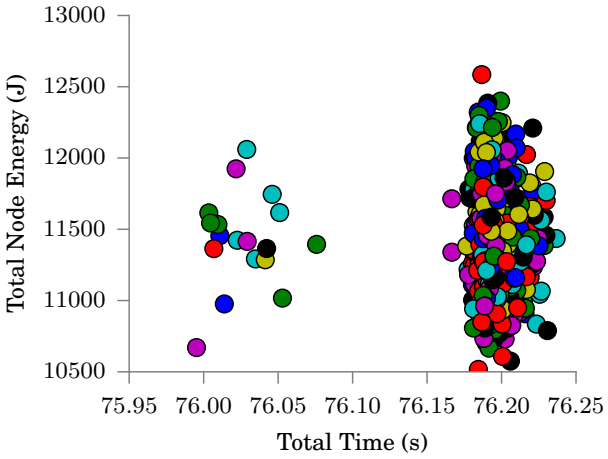


(b) Average time per tag per iteration.

Figure 3: Energy and time values per tag per iteration for LAMMPS on 128 nodes.



(a) 128 Nodes



(b) 1024 Nodes

Figure 4: LAMMPS total energy consumption and time per node for two runs using 128 and 1024 nodes respectively.

We used a matrix size of 1024×1024, and DGEMM was run for 1000 iterations. An increasing power cap was set every 100 iterations, and the power trace is presented in Figure 5. As a compute bound benchmark, DGEMM is very susceptible to changes in performance with changing power caps, each time consuming as much power as it was allowed to. The system's responsiveness to changes in power cap is fairly quick, taking roughly a second for changes in power consumption to be observable.

4.3 Power Capping with PoLiMEr: An Architectural Perspective

Power and performance impacts under power cap are not only application-dependent, but also system dependent. Here we use PoLiMEr to explore differences in power and performance between Theta's in-package (MDRAM) and external (DDR) DRAMs under power cap. We ran STREAM on 64 nodes (1 rank/node, 64 threads/rank, 1 thread/core), using the flat memory and quadrant cluster mode. Each instance used 15 GB of memory, and the four STREAM benchmarks, copy, add, scale and triad were executed in a loop of 100 iterations.

Figure 6a shows the power consumption over time for STREAM using the slower (90 GB/s) but larger DDR memory. Table 3 shows the power and time difference between DDR and the faster (480 GB/s) MCDRAM. Each node consumed on average 120 W - 140 W during the main benchmark phase when using DDR memory, and around 200 W in the MCDRAM case.

Figure 6b shows the total power consumption over time for STREAM on DDR with a 110 W power cap, with a decrease in node and package power consumption. The memory power consumption was unaffected, and all four benchmarks finished equally fast in both the capped and the uncapped version. In contrast, STREAM's verification phase past the 70 second

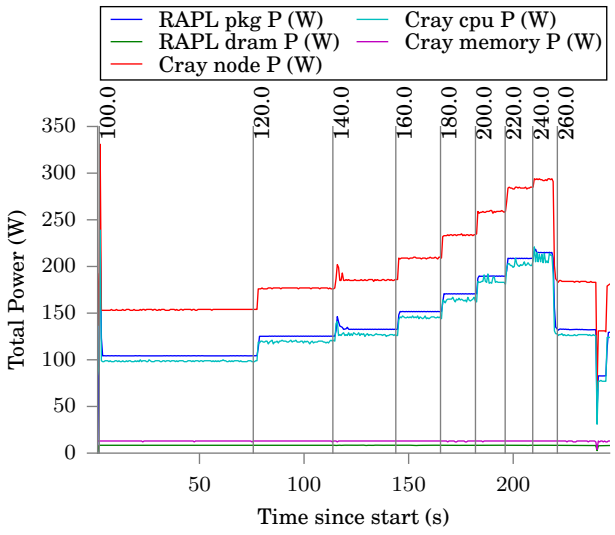


Figure 5: Intel MKL DGEMM executed on a single node, 1000 iterations (1024×1024 matrix size). RAPL package power cap was increased every 100 iterations by 20 W, starting with 100 W. Vertical lines mark increase in power cap.

Table 3: Power and time differences of STREAM using DDR and MCDRAM memory, with and without power cap.

	DDR	MCDRAM
Time for main loop	70 s	18 s
Max Node Power	14 kW	18 kW
Max Package Power	9 kW	14 kW
Max Memory Power	2 kW	1 kW
Time for main loop w/ 110 W cap	70 s	39 s
Max Node Power w/ 110 W cap	12 kW	11 kW
Max Package Power w/ 110 W cap	7 kW	7 kW
Max Memory Power w/ 100 W cap	2 kW	1 kW

mark is significantly impacted by the power cap. The reported bandwidth in the uncapped version for copy and scale was 82 GB/s, and 87 GB/s for add and triad. In the capped version, copy and scale bandwidth was 81 GB/s and 86 GB/s for add and triad. These results suggest that the package power cap had little to no impact on STREAM's performance when using the DDR memory. On the other hand, there is a performance penalty in the IPM case, as can be seen in Table 3.

4.4 Overhead Measurements

We present overhead measurements of PoLiMEr below. We ran DGEMM on 16, 128 and 512 nodes with PoLiMEr, and timed PoLiMEr's essential features. The results were reported for 1 rank and are displayed in Table 4.

Initialization and finalization time increase with increasing number of nodes. Finalization also depends on the number of user tags, as PoLiMEr has to process all tags, check they have been completed correctly, and write them out to file.

Table 4: Overhead Measurements using DGEMM

# Nodes	16	128	512
application time (s)	192.103	191.56	201.410
initialize (s)	0.061	0.098	0.201
finalize - 2 tags (s)	0.092	0.209	0.84
start 1 tag (s)	0.004	0.003	0.003
end 1 tag (s)	0.0003	0.0003	0.0003
start + end tag 100× (s)	3.361	3.539	2.015
finalize w/ 100 tags (s)	18.531	23.467	38.639
set power cap 100× (s)	0.002	0.002	0.001
get power cap 100× (s)	0.0004	0.0005	0.0005
polling overhead 1000× (s)	0.00048	0.00042	0.00041

A tag may be used multiple times if it is inside a loop, for instance. We emulated this behavior by measuring the overhead of starting and ending a tag 100 times, and are reporting the overhead of finalizing PoLiMEr in that case. Tagging incurs relatively low overhead, while finalization can take quite some time. Users can mitigate this cost by surrounding loops with tags, or lowering the number of tags. The overhead of setting and getting power caps in such a scenario is near negligible. Users are free to set and poll power caps as many times as they wish. The overhead to poll the system 1000 times in a row shows there is close to no cost of polling the system regardless of the polling interval.

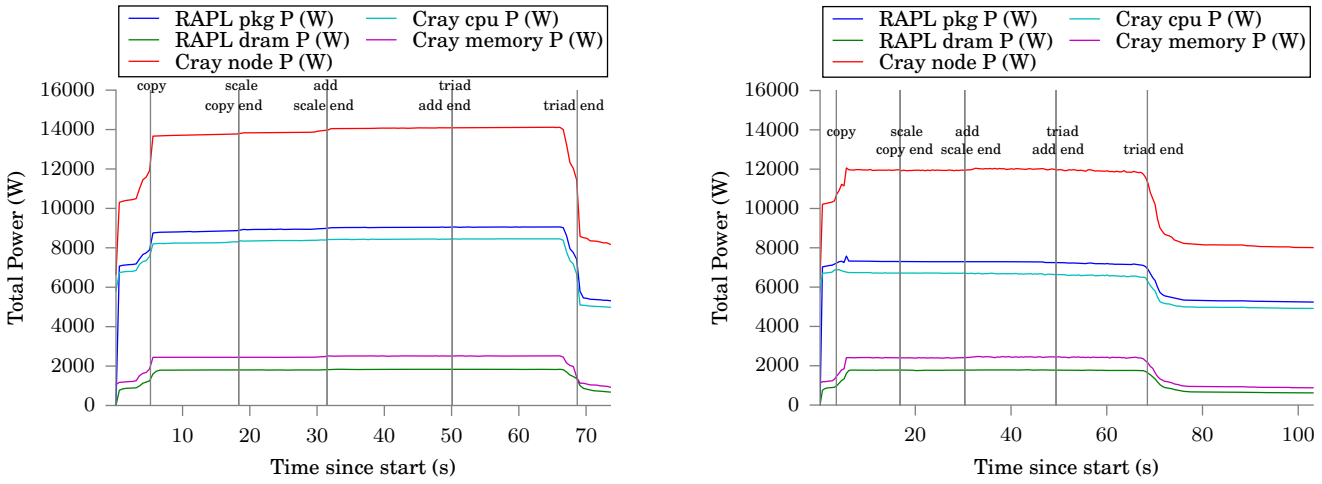
Any actions performed by PoLiMEr during the main execution of the application incur near negligible cost. The heavier initialization and finalization phases are outside of the main application tasks, thus not affecting the outcomes of the application.

5 CONCLUSION

PoLiMEr serves as an important foundation for future research on power-related issues facing future exascale systems. PoLiMEr can be used to gain valuable, fine-grained insight into power consumption characteristics of even complex simulation codes, node variabilities at scale, and application and system response to power caps, all of which point to potential resource management strategies. Key architectural features of the XC40 such as its memory hierarchy, shows new power saving opportunities even in the face of power constraints. We have shown how we can use PoLiMEr to identify DDR memory-intensive application tasks and apply a stringent power cap without any performance losses. Large-scale HPC applications with significant in-memory data requirements, such as data-driven science, are excellent candidates for future power capping studies. With its user-friendly API and abstraction of system-dependent power interfaces, PoLiMEr now enables HPC users of all backgrounds to engage in these new research endeavors.

ACKNOWLEDGMENTS

This research has been funded in part and used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract



(a) No power cap

(b) 110 W RAPL package power cap

Figure 6: Power consumption trace of STREAM using DDR memory on 64 nodes (1 MPI rank/node, 64 threads/rank, 1 thread/core), without and with power cap. Flat memory mode, quad cluster mode on KNL. 15 GB of memory, 100 iterations.

no. DE-AC02-06CH11357. The effort on this project is also funded by the U.S. Government under the DARPA BRASS program, by the NSF under CCF 1439156, and by a DOE Early Career Award.

The authors would like to thank Christopher Knight, Francois Tessier, Derek Jensen, Ryan Milner, Sudheer Chunduri and Benjamin Allen for their help and valuable insights.

REFERENCES

[1] Pavan Balaji, Darius Buntinas, David Goodell, William Gropp, Jayesh Krishna, Ewing Lusk, and Rajeev Thakur. 2010. PMI: A scalable parallel process-management interface for extreme-scale systems. *Recent Advances in the Message Passing Interface* (2010), 31–41.

[2] Arnaldo Carvalho de Melo. 2010. The new linux perf tools. In *Slides from Linux Kongress*, Vol. 18.

[3] Spencer Desrochers, Chad Paradis, and Vincent M Weaver. 2016. A Validation of DRAM RAPL Power Measurements. In *Proceedings of the Second International Symposium on Memory Systems*. ACM, 455–470.

[4] Jonathan Eastep, Steve Sylvester, Christopher Cantalupo, Federico Ardanaz, Brad Geltz, Asma Al-Rawi, Fuat Keceli, and Kelly Livingston. 2016. Global extensible open power manager: a vehicle for HPC community collaboration toward co-designed energy management solutions. *Supercomputing PMBS* (2016).

[5] Vladimir Getov, Darren J. Kerbyson, Matt Macduff, and Adolfy Hoisie. 2015. Towards an Application-specific Thermal Energy Model of Current Processors. In *Proceedings of the 3rd International Workshop on Energy Efficient Supercomputing (E2SC '15)*. ACM, New York, NY, USA, Article 5, 10 pages. https://doi.org/10.1145/2834800.2834805

[6] R. E. Grant, M. Levenhagen, S. L. Olivier, D. DeBonis, K. T. Pedretti, and J. H. Laros III. 2016. Standardizing Power Monitoring and Control at Exascale. *Computer* 49, 10 (Oct 2016), 38–46. https://doi.org/10.1109/MC.2016.308

[7] Connor Imes, Lars Bergstrom, and Henry Hoffmann. 2016. A portable interface for runtime energy monitoring. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 968–974.

[8] Intel. 2015. 64 and IA-32 Architectures Software Developer's Manual. *Volume 3A: System Programming Guide, Part* (2015).

[9] LAMMPS. 2004. LAMMPS WWW Site. http://lammps.sandia.gov. (2004). Accessed: August, 2017.

[10] James H Laros, Phil Pokorny, and David DeBonis. 2013. Powerinsight-a commodity power measurement capability. In *Green Computing Conference (IGCC), 2013 International*. IEEE, 1–6.

[11] Gary Lawson, Vaibhav Sundriyal, Masha Sosonkina, and Yuzhong Shen. 2016. Runtime power limiting of parallel applications on Intel Xeon Phi processors. In *Proceedings of the 4th International Workshop on Energy Efficient Supercomputing*. IEEE Press, 39–45.

[12] SJ Martin, D Rush, and M Kappel. 2015. Cray advanced platform monitoring and control (CAPMC). In *Proc. Cray Users' Group Technical Conference (CUG)*.

[13] S Martin, D Rush, M Kappel, M Sandstedt, and J Williams. 2016. Cray XC40 Power Monitoring and Control for Knights Landing. *Proceedings of the Cray User Group (CUG)* (2016).

[14] Philip J Mucci, Shirley Browne, Christine Deane, and George Ho. 1999. PAPI: A portable interface to hardware performance counters. In *Proceedings of the department of defense HPCMP users group conference*, Vol. 710.

[15] Scott Parker, Vitali Morozov, Sudheer Chunduri, Kevin Harms, Chris Knight, and Kalyan Kumaran. 2017. Early Evaluation of the Cray XC40 Xeon Phi System Theta at Argonne. *Cray User Group 2017 proceedings* (2017).

[16] Kevin Pedretti, Stephen L. Olivier, Kurt B. Ferreira, Galen Shipman, and Wei Shu. 2015. Early Experiences with Node-level Power Capping on the Cray XC40 Platform. In *Proceedings of the 3rd International Workshop on Energy Efficient Supercomputing (E2SC '15)*. ACM, New York, NY, USA, Article 1, 10 pages. https://doi.org/10.1145/2834800.2834801

[17] Steve Plimpton. 1995. Fast parallel algorithms for short-range molecular dynamics. *Journal of computational physics* 117, 1 (1995), 1–19.

[18] Kathleen Shoga, Barry Rountree, Martin Schulz, and Jeff Shafer. 2014. Whitelisting MSRs with msr-safe. In *3rd Workshop on Exascale Systems Programming Tools, in conjunction with SC14*.

[19] Sean Wallace, Venkatram Vishwanath, Susan Coghlan, Zhiling Lan, and Michael E Papka. 2015. Comparison of vendor supplied environmental data collection mechanisms. In *Cluster Computing (CLUSTER), 2015 IEEE International Conference on*. IEEE, 690–697.

[20] Sean Wallace, Zhou Zhou, Venkatram Vishwanath, Susan Coghlan, John Tramm, Zhiling Lan, and Michael E Papka. 2016. Application power profiling on IBM Blue Gene/Q. *Parallel Comput.* 57 (2016), 73–86.

[21] Vincent M Weaver, Matt Johnson, Kiran Kasichayanula, James Ralph, Piotr Luszczek, Dan Terpstra, and Shirley Moore. 2012. Measuring energy and power with PAPI. In *Parallel Processing Workshops (ICPPW), 2012 41st International Conference on*. IEEE, 262–268.