Global Extensible Open Power Manager: A Vehicle for HPC Community Collaboration on Co-Designed Energy Management Solutions

Jonathan Eastep(✉), Steve Sylvester, Christopher Cantalupo, Brad Geltz, Federico Ardanaz, Asma Al-Rawi, Kelly Livingston, Fuat Keceli, Matthias Maiterth, and Siddhartha Jana

Power Pathfinding to Product (P3) Team, Data Center Group, Intel Corporation, Hillsboro, OR, USA
jonathan.m.eastep@intel.com

Abstract. The power scaling challenge associated with Exascale systems is a well-known issue. In this work, we introduce the Global Extensible Open Power Manager (GEOPM): a tree-hierarchical, open source runtime framework we are contributing to the HPC community to foster increased collaboration and accelerated progress toward software-hardware co-designed energy management solutions that address Exascale power challenges and improve performance and energy efficiency in current systems. Through its plugin extensible architecture, GEOPM enables rapid prototyping of new energy management strategies. Different plugins can be tailored to the specific performance or energy efficiency priorities of each HPC center. To demonstrate the potential of the framework, this work develops an example plugin for GEOPM. This power rebalancing plugin targets power-capped systems and improves efficiency by minimizing job time-to-solution within a power budget. Our results demonstrate up to 30% improvements in the time-to-solution of CORAL system procurement benchmarks on a Xeon Phi cluster.

1 Introduction

Performance of future large-scale HPC systems will be constrained by power costs. Some HPC centers are already incentivized through government taxes to operate their systems at more energy-efficient points below peak performance and power [3]. Others may prefer peak performance today, but they face cost-pressure of a different kind to deploy more efficient systems in the future: system power draw is increasing by a substantial factor generation-over-generation, and without a breakthrough in system energy efficiency, industry trends forewarn that large-scale systems will exceed the limits of the existent power delivery infrastructure at typical centers by a 2–3× margin by 2022. This forces costly upgrades or limited system scale. Overcoming the 2–3× gap will require co-designed hardware and software system energy management solutions as well as increased collaboration between hardware vendors and the HPC software community.

J.M. Kunkel et al. (Eds.): ISC High Performance 2017, LNCS 10266, pp. 394–412, 2017.
DOI: 10.1007/978-3-319-58667-0_21

In this paper, we introduce the Global Extensible Open Power Manager (GEOPM). GEOPM is an open source, plugin extensible runtime for power management. The primary goal of the project is to provide an open platform for community collaboration and research on co-designed energy management solutions to close the energy efficiency gap. We demonstrate a power rebalancing plugin for GEOPM targeting power-constrained systems which leverages feedback from the application to identify which nodes are on the critical path then adjusts processor power cap settings to accelerate the critical path and improve the application's time-to-solution. Subject to the power cap it is given, each processor attempts to maximize its performance while our software provides coordination of power budgets (and thus performance) across nodes. Through this marriage of software and hardware management of power and performance, we obtain up to 30% improvements in time-to-solution for CORAL procurement benchmarks on a power-constrained Knights Landing system.

In contributing this paper and the first plugin for GEOPM, we have taken a significant step toward closing the 2–3× energy efficiency gap. Much community collaboration will be required to close the remainder. For example, hardware vendors will need to provide improved or new software-tunable knobs in the future; GEOPM is influencing research along these lines at Intel. Additionally, the HPC software community will need to expose tunable knobs from various software layers to GEOPM (e.g. the application, runtime, system software, or operating system layers). Fully leveraging these knobs will require algorithmic advances in GEOPM and extensions enabling it to target different knobs than are supported today. These extensions will be developed in collaboration with the HPC community and will be added to GEOPM via plugins over time.

The GEOPM runtime framework is being developed for broad deployment on Xeon, Xeon Phi, and other HPC system architectures. The first deployment is expected on the Theta system, a Knights Landing Xeon Phi system at Argonne. The GEOPM software package is available under the BSD three clause open source software license in the GEOPM source code repository on GitHub (project page: http://geopm.github.io/geopm). The GEOPM runtime framework, test infrastructure, and power rebalancing plugin are all open source.

The remainder of the paper is organized as follows. Section 2 highlights GEOPM's primary contributions over prior works. Section 3 overviews GEOPM's design. Section 4 analyzes time-to-solution improvements obtained with the power rebalancing plugin for CORAL procurement and other benchmarks. Section 5 concludes and discusses future work.

2 Related Work

To our knowledge, GEOPM is the first open extensible runtime framework to be contributed to the community by a hardware vendor with the intent of collaborative research toward software-hardware co-designed energy management solutions in future HPC systems. This vision and early work was first publicized broadly to the community in a short workshop paper in PMBS'16 [1] and the

Emerging Technologies Showcase at SC'16. In this ISC paper, we have further developed the early work presented at PMBS for a full conference publication.

There are parallel software efforts to GEOPM contributed by a hardware vendor: OpenHPC [2] facilitates community collaboration on the HPC software stack by providing a framework for integrating, configuring, and testing open source components. OpenHPC has not been focused on fostering co-designed energy management solutions, but we note that we intend to submit the GEOPM package to the OpenHPC Technical Steering Committee for inclusion in the OpenHPC distribution when the production version 1.0 of GEOPM is released. There are parallel software-hardware co-design efforts to GEOPM such as OpenPOWER [19]. While OpenPOWER enables the community to customize systems based on the IBM POWER architecture, we are not aware of activity within the OpenPOWER project to research software-hardware co-designed energy management solutions exploiting runtime feedback from applications. While GEOPM only currently provides plugins supporting x86 systems, users can add platform implementation plugins supporting POWER or other system architectures.

To our knowledge, GEOPM is the first open source job-level power management runtime for HPC systems to support extensible energy management control strategies through a plugin architecture, making it suitable for the differing energy management needs of a wide range of HPC installations around the world. The Power API Specification from Sandia [18] is a synergistic effort, but it is an orthogonal effort because it emphasizes power interfaces rather than runtime techniques for optimizing energy. The Power API project is defining community-standard interfaces for power monitoring and control at various granularities throughout the HPC stack. Runtimes like GEOPM and other components can collectively target these interfaces to achieve interoperability. We are collaborating with Sandia to explore changes targeted at future releases of the specification to increase support for GEOPM and its interfaces.

In this work, we develop a plugin for GEOPM for power rebalancing within a job. Prior works such as Conductor [4], Adagio [5], and Jitter [6] have demonstrated effective algorithms for reallocating power between nodes to compensate for application load imbalance – whether for the purpose of increasing application performance under a job power cap by accelerating the critical path or improving application energy efficiency by reducing performance in nodes off of the critical path. While these algorithms are effective at smaller scales (i.e. less than a few thousand nodes), their centralized designs are not intended for today's large-scale deployments or future Exascale deployments. The key difference is that the GEOPM power balancing plugin has a flexible tree-hierarchical design suitable for deployments ranging in scale from rack-scale to extreme-scale deployments. We note, however, that we have a collaboration underway with the authors of these prior works to compare approaches and meld together the best aspects of each approach in a future GEOPM plugin and paper.

There is a parallel work to GEOPM called the Argo project [26] which is developing a task-based programming model and runtime for Exascale HPC systems. Its design includes a hierarchical power manager. Unlike GEOPM, the

Argo power manager is not intended as a vehicle for the community and hardware vendors to collaborate on researching new energy management solutions. Furthermore, while the Argo project envisions this power manager performing automatic hierarchical power budgeting, that functionality is not complete to our knowledge. What has been demonstrated is hierarchical enforcement of power budgets that were adjusted manually at runtime. That said, the authors are interested in exploring if Argo's algorithms could be implemented as GEOPM plugins and brought to fruition in production deployments through GEOPM.

We note that there have also been orthogonal efforts [27] to develop hierarchical power management frameworks for enterprise data centers. They employ significantly different energy management strategies suitable for enterprise workloads and virtualized environments. There have been other related works that focused on saving power given a time bound. Some have used linear programming to optimize energy savings with nearly no runtime increase [21]. Others have achieved bigger power savings in exchange for small performance degradations [22–25].

Aside from prior works on saving energy while maintaining performance levels, hierarchical power capping, and rebalancing power across nodes to increase job performance under a power cap, there have also been prior works on power-aware scheduling algorithms for energy management at the system level [7–9]. These algorithms comprehend system-level power caps and assign a different power cap to each job based on its runtime and power characteristics with the goal of reducing job wait times or optimizing overall system throughput. GEOPM is synergistic with these works: the intent is for GEOPM to integrate with a power-aware scheduler in an extended energy management hierarchy. In particular, the scheduler can view GEOPM as a mechanism for optimizing the job's performance or energy efficiency within the scheduler-specified job power budget, and the scheduler can optimize system performance and efficiency by deciding the best allocation of the system budget among concurrent jobs. For maximum benefits, GEOPM supports dynamic adjustments to the job cap.

3 GEOPM Design Overview

This section provides an overview of the GEOPM design, beginning with discussion of how GEOPM integrates into the HPC system stack. We cover GEOPM's interfaces and responsibilities as well as its scalable, extensible design.

3.1 GEOPM Interfaces and Integration Architecture

Figure 1 illustrates how the GEOPM runtime fits into the HPC system stack. GEOPM is a job-level power manager. The GEOPM runtime interacts with the scheduling functions of the workload manager through the workload manager interface. This interface lets future power-aware schedulers assign an objective for the job and configure which energy management plugin GEOPM should use to manage the job. Supported objectives include but are not limited to managing

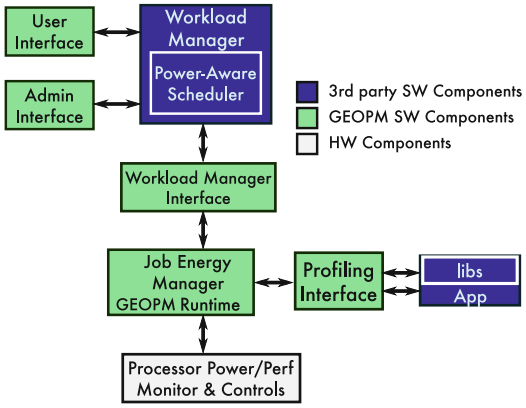

Fig. 1. GEOPM interfaces and HPC system stack integration

the job to stay within a power budget while optimizing job time-to-solution; in this case, the scheduler would use the interface to assign a job power budget as well. The workload manager interface allows GEOPM to report back how much power the job consumed and statistics about the job that GEOPM has collected. There is an option for the interface to be used at job start and finish (statically) or periodically while the job is running (dynamically). The GEOPM runtime runs in user space. Therefore, GEOPM does not control resources that are shared between users like network switches or the distributed file system; its scope is control of power and performance knobs in compute node resources.

There is also an interface to the application software or libraries shown at the middle right of the figure. The interface functions are listed in Table 1. This software profiling interface allows the programmer to mark up their code to hint to GEOPM about loops with global synchronization events in the application that could result in performance loss if some MPI ranks fall behind in the computation and reach synchronization points late (i.e. epochs). The interface also enables programmers to hint to GEOPM about phases (i.e. regions) in the application or library code between synchronization events as well as provide an application-level performance signal (i.e. progress) that GEOPM can use to adapt its decisions as the application transitions between phases. For

Table 1. Function list for GEOPM profiling interface

`geopm_prof_epoch()`:	Synchronization loop iteration beacon
`geopm_prof_region()`:	Get region ID from name
`geopm_prof_enter()`:	Mark region entry
`geopm_prof_exit()`:	Mark region exit
`geopm_prof_progress()`:	Report region progress

example, GEOPM may use region information to monitor for memory-intensive or communication-intensive phases where processor frequency can be decreased to save power with little or no impact on runtime.

The GEOPM profiling interface is designed to be lightweight and minimally invasive, but future work will explore methods of automatically inferring phase and performance information to enable use cases where GEOPM can make per-phase decisions effectively without requiring programmers to mark up application or library code. See the GEOPM man pages in [10] for full details on the signatures and use of these functions. We also provide further documentation, tutorials, and example MPI applications in the GEOPM source code repository illustrating how to use them. See [20] for tutorial video walk-throughs on YouTube.

As depicted in Fig. 1, GEOPM provides interfaces to the user or administrator enabling them to configure GEOPM and request specific energy management plugins for a job. The interface is a JSON configuration file. The GEOPM software package provides a tool to generate configuration files from the command line called `geopmpolicy` and a C interface as well through the `geopm_policy_*()` APIs. The GEOPM configuration file is selected at runtime through the `GEOPM_POLICY` environment variable. On a system deploying the SLURM workload manager, SLURM's plugin infrastructure can be used to generate the file and set the environment variable. It can also be used to launch the GEOPM runtime and configure CPU affinity. Some other workload managers offer similar infrastructure. For those that do not, wrappers can be placed around MPI launch commands to configure and enable GEOPM.

3.2 GEOPM Scalable Tree-Hierarchical Design

The GEOPM runtime is designed for use on a wide range of system scales. This is accomplished through a flexible tree-hierarchical design. As illustrated in Fig. 2, the GEOPM runtime is implemented as a hierarchical feedback guided control system using a balanced tree. The energy management strategy employed is extensible through a plugin architecture. The depth and fan-out of the tree are automatically adjusted by the GEOPM runtime to accommodate different job sizes.

Controllers in the tree (and therefore energy management plugins) take a recursive approach to coordinating energy and performance policy decisions globally, across all nodes in the job. The root controller sets policy for its children, each of its children set policy for their children, and so on. Policies are defined hierarchically such that the parent constrains the space of policies that its children can select from and, in so doing, effects their decisions. Decisions at each level of the tree are based on feedback from each child. This feedback consists of a history of energy, performance, and other statistics collected over the last few control intervals. For scalability, the feedback is aggregated as it is communicated back from the leaves toward the root. Thus, decisions at the root are informed by feedback from the leaves, and decisions flowing down the tree effect decisions made at each leaf.

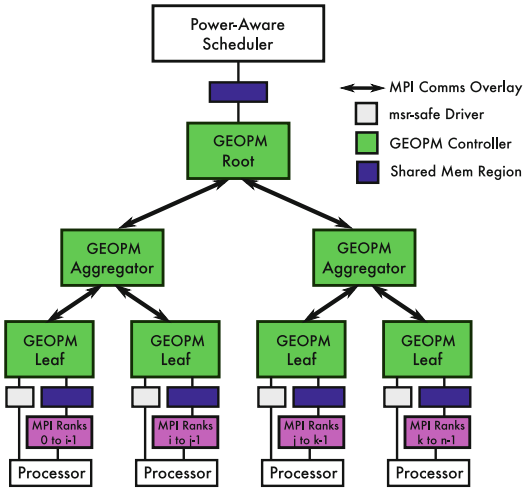

Fig. 2. GEOPM hierarchical design and communication mechanisms

To run the control tree hierarchy, the GEOPM runtime launches one user thread on each compute node. This thread runs for the duration of the job. On each compute node, this thread executes the responsibilities of the leaf controller. On some compute nodes, this thread also executes responsibilities of the aggregator controllers at higher levels in the tree. On one compute node in the job, the thread also executes responsibilities of the root controller of the tree.

The GEOPM thread can be launched in several ways, and the affinity of this thread should be controlled for best performance. On manycore systems with low single-thread performance and high-cost context switching, it may be a general performance benefit to leave a core unused by the application so that the operating system can execute threads without interrupting the application. In such systems, the GEOPM thread can be affinitized to this core and use it as well. In cases where GEOPM is run with computationally-intense plugins, results may be best if the GEOPM thread runs on the core that the application stays off of; developers and users should keep this in mind. For other application and system hardware combinations, it may benefit performance for the GEOPM thread to share a core with the application through context switching. See [10] for further information on GEOPM's launch and affinitization options.

Dynamic communication between levels of the GEOPM control tree hierarchy is currently achieved using MPI over the application's in-band network fabric. We use MPI's Cartesian topology functionality to map the leaf, aggregator, and root controllers to the GEOPM runtime threads on the compute nodes. When GEOPM is built against an MPI implementation with optimized Cartesian topology functionality, this minimizes communication distances over the network fabric for the controllers. We also use MPI's Cartesian topology to efficiently implement a balanced tree hierarchy supporting a wide range of job node

counts. All communication uses one-sided operations through the `MPI_Put()` interface. In Sect. 4.4, we provide measurements of GEOPM's communication bandwidth requirements on the OmniPath network fabric in our test system. We demonstrate that bandwidth use is orders of magnitude less than 1% of the total available bandwidth. GEOPM can be extended to support out-of-band communication in the future.

Inter-process shared memory is used both for dynamic communication between the GEOPM root controller and power-aware scheduler and for communication between leaf controllers and application processes on the compute nodes. Communication between leaf controllers and processors is achieved via GEOPM `PlatformImp` plugins (discussed in Sect. 3.3). In the case of Intel systems, processors expose Model Specific Registers (MSRs) [17] for communication with software. GEOPM `PlatformImp` plugins for Intel systems perform MSR access from userspace via the msr-safe Linux driver developed by LLNL [11].

3.3 GEOPM Extensible Plugin Architecture

There are three types of plugins supported by GEOPM which enable user extension of the runtime. From lowest to highest level of abstraction these are: the `PlatformImp`, the `Platform`, and the `Decider`. The `PlatformImp` plugin is used to expose low-level hardware features to `Platform` plugins. The GEOPM package provides `PlatformImp` plugins for a range of Intel platforms exposing hardware features implemented with Model Specific Registers (MSRs). Support for other hardware platforms would be implemented with this type of plugin. The `Platform` plugin is used to express higher-level abstractions of the hardware features exposed by the `PlatformImp`, and it provides the bridge interface called by the controller to enforce a policy provided by a leaf `Decider`.

There are two types of `Decider`s: tree `Decider`s and leaf `Decider`s. The leaf `Decider` is responsible for managing power or other controls within a single compute node. The tree `Decider` is used for all levels of the control tree hierarchy above the leaf level. The lowest-level tree `Decider` is the parent of a leaf `Decider`. Collectively, the tree `Decider`s are responsible for managing power across nodes. The leaf and tree `Decider`s are each selected by name in the GEOPM configuration file provided by the end user, administrator, or workload manager.

All `Decider`s have six main responsibilities: collecting feedback information from their children in the GEOPM control tree hierarchy (or the software profiling interface and `Platform` in the case of leaf `Decider`s), aggregating this data into a reduced form, passing this reduced version up to their parent in the tree, receiving policy information from their parent above them in the tree, deciding how to set policy for their children (or how to set node power controls or other controls in the case of leaf `Decider`s) based on the policy given by their parent, and passing policy decisions down to their children (or to the `Platform` in the case of leaf `Decider`s).

The GEOPM policy defines a power budget, so each `Decider` is taking in a power budget from its parent and deciding how to divide that budget among its

children (or how to divide that budget among the node hardware components that support control in the case of the leaf `Decider`). The leaf `Decider` may also manage controls beyond power limit controls to enable various additional optimizations. Each `Decider` plugin defines for itself the objective function it will try to maximize when making policy decisions for its children and implements an algorithm to maximize this objective function. Thus, by selecting a particular `Decider`: GEOPM users, administrators, or the workload manager are selecting a particular objective function and a particular energy management algorithm to try to maximize that objective function.

To help the `Decider` tree hierarchy achieve control stability, the GEOPM `Decider` interface includes functions that express convergence. Before introducing a new policy, the parent `Decider` waits for its children to signal that the existing policy has been enacted stably. Child `Decider`s wait until the aggregated feedback they would send to their parent would reflect the current policy before signaling convergence to their parent.

GEOPM is designed to support per-phase adaptation of node hardware controls and other node-level controls via the leaf `Decider`. To support this, the GEOPM framework and leaf `Decider`s must collect feedback and adjust controls at the cadence of application phase transitions. The current implementation of the GEOPM framework and governing leaf `Decider` provided with the GEOPM software package can sustain a 5 ms cadence with standard deviation of less than .5 ms when running on an Intel Knights Landing Xeon Phi platform. The control loop includes computations to feed input to the governing `Decider` and enact its decisions in the `Platform`. Those computations include sampling processor performance counters, reading from a log of application interface calls stored in shared memory, extrapolating application progress forward to when the processor counters were read, estimating power consumption over the last interval by applying linear least squares parameter fit over a moving window of energy counter readings, and writing RAPL MSRs to enact the governing `Decider`'s allocation. The control loop computation also includes the algorithm in the governing `Decider` that adaptively allocates power among the processor and external DRAM from the node power budget.

3.4 Example Power Balancing Plugin

We have developed an example plugin for GEOPM to demonstrate both its extensible architecture and a scalable hierarchical power management strategy to address the performance variation challenges expected in Exascale systems due to their need for power-capping. Power-capping exposes differences in the energy efficiency of like hardware components. These differences derive from manufacturing variation. Under power caps, even like hardware components from the same Stock Keeping Unit (SKU) will exhibit different performance which results in the nodes of the system taking different amounts of time to complete equal amounts of work [12,13].

The GEOPM power balancing plugin mitigates this performance variation, minimizing its impact on application time-to-solution. The plugin targets

iterative bulk-synchronous MPI applications running on power-capped systems and leverages application-awareness to first identify the nodes that are on the critical path due to their lower performance at a given power cap then accelerate them by diverting power away from nodes that are off of that path. This provides an overall improvement in application time-to-solution. The source code for this power balancing tree `Decider` plugin is hosted in the GEOPM repository [10].

The tree `Decider`s identify and accelerate the critical path hierarchically. The GEOPM controller framework provides the lowest-level tree `Decider` with samples of its children's runtime (not including time spent waiting for MPI synchronization) between application calls to the `geopm_prof_epoch()` function averaged over a moving window. The `geopm_prof_epoch()` call acts as a beacon, signaling each time the application reaches a new iteration of an outer loop containing an inter-node synchronization operation. The tree `Decider` takes in a power budget from its parent, compares the runtime reported by its children, then computes how to divide its power budget among its children such that they will reach the synchronization point at roughly the same time, avoiding wait time and associated performance loss. Each tree `Decider` reports the max of its children's runtime as aggregated runtime feedback to its parent.

4 Results

This section presents our analysis of the power balancing plugin for GEOPM. We describe our experimental setup, we demonstrate the improvements to application time-to-solution that the plugin provides, we analyze how the plugin obtains these improvements, and we report measurements of GEOPM's computational, communication, and memory requirements.

4.1 Experimental Setup

Our experiments use standard benchmark output as the final reference for time-to-solution and the other statistics we report. However, our analysis additionally leverages GEOPM's report and trace features. GEOPM may be configured with the environment variables `GEOPM_REPORT` and `GEOPM_TRACE` to generate two types of profiles after each application run: a summarizing report or a time series trace. The report file aggregates performance and energy metrics for the application both overall and for each individual region that the programmer has annotated in the application using the profiling interface described in Sect. 3.1. The trace file is a table of time series data containing samples of processor performance counters, information collected via application calls to the profiling interface, and control knob setting outputs recorded by GEOPM during the application run. This table contains exactly the same data provided to the leaf `Decider` plugin in the GEOPM control tree hierarchy.

In this paper, we performed our experiments on a cluster of 12 compute nodes. Each compute node has one Intel Xeon Phi Knights Landing processor (KNL-F B0 Beta SKU) and 256 GB of external DRAM. This processor SKU

has 64 Turbo-enabled, 4-way hyperthreaded cores each with a 1.3 GHz sticker frequency. It has 16 GB of MCDRAM on-package memory, an integrated Omni-Path HFI used for communication over the network fabric, and a 230 W Thermal Design Power (TDP). The operating system is CentOS 7 Linux with the 'performance' frequency governor enabled. The C/C++ and Fortran software was compiled with the Intel tool-chain while using the MVAPICH2 MPI implementation. We used version 0.2.2 of the GEOPM software package [10].

In our experiments, we targeted the following workloads: Qbox, HACC, Nekbone, AMG, miniFE, CoMD, and FFT. Qbox is a quantum molecular dynamics code, HACC is a cosmology code for simulating the evolution of the universe, Nekbone is a thermal hydraulics code, AMG is an algebraic multi-grid solver for unstructured meshes, miniFE is a finite element code, CoMD is a proxy molecular dynamics simulation code, and FFT is a discrete 3-d Fast Fourier Transform kernel. Qbox, HACC, and Nekbone are Tier 1 scalable science workloads from the CORAL procurement benchmarks; AMG is a Tier 1 throughput benchmark; and miniFE is a Tier 2 throughput benchmark from the CORAL benchmarks [14]. CoMD is an ExMatEx benchmark for software-hardware co-design [15]. FFT is a key kernel from the NAS Parallel Benchmarks suite [16].

When configuring workloads, we applied standard conventions. We sized the problem to use the majority of the MCDRAM (on-package memory) in each node. With the system not power-capped – i.e. with the processors running at TDP – we swept over the different numbers of MPI ranks and OpenMP threads per rank using up all or almost all of the available hyperthreads in the processor; we then determined which configuration resulted in the best runtime for each workload and used it in all evaluations of our power balancing plugin. We found that all workloads performed best if they were affinitized to leave Linux CPU 0 unused by the application to avoid interference by operating system threads. We found that miniFE and CoMD performed best if using two hyperthreads per core, while all other workloads performed best if using one. Using the GEOPM profiling interface, we added mark up to these workloads to enable tuning them with the power balancing plugin. The modifications are available in [10].

To study how much application speedup our power-balancing plugin provides in power-constrained systems, we swept over a range of job power caps and compared the workload runtime achieved while using our power-balancing plugin versus a baseline. Our power-balancing plugin dynamically reallocates the job power budget among nodes to mitigate load imbalance while the baseline applies a static uniform division of the job power budget among nodes. In the baseline, all tree `Decider`s are inactive. However, both cases employ active leaf `Decider`s to enforce the node-level power budgets.

The leaves enforce the budget as follows: they dynamically measure the power consumed in the external DRAM via the processor RAPL feature, they subtract this power from the node budget (obtained from their parents in the GEOPM control tree hierarchy), and then they set the RAPL socket power limit equal to the remaining power so that the sum of socket and external DRAM power matches the node power budget. Node power budgets are defined in terms of

the dynamic power controllable via the processor RAPL feature. The remainder of node power is not included but it is approximately static. Power consumed by the job in shared resources like the network fabric interface is not currently accounted for but may be in future work.

The workloads under study have well-balanced assignments of work across ranks yet they still exhibit load imbalance deriving from the effects of hardware manufacturing variation which have been discussed in Sect. 3.4. When interpreting the results in this paper, it is important to note that, while the analysis focuses on manufacturing variation, the GEOPM power balancing plugin can address load imbalance due to imbalanced work assignments across ranks as well. However, evaluating benefits of the plugin in that scenario is beyond the scope of this paper. We also note that we made no attempt to cherry-pick processors from extreme ends of the power efficiency distribution in the processor SKU. Therefore, we do not know if the processors in our cluster reflect the full potential for load imbalance. We will explore this in future work.

In our power cap sweep experiments, we set the max job power cap equal to the power at which each workload's time-to-solution reached its minimum (i.e. unconstrained performance), and we set the min job power cap to the value at which performance scaling hit an inflection point where the processor spent in excess of 8% of its time throttling inefficiently to reach the required power. Results at power caps below this inflection point may be meaningful in some research or production scenarios but they are omitted from this paper for brevity.

4.2 Runtime Improvements with Power Balancing Plugin

Figure 3 shows the mean runtime improvements obtained by our power balancing plugin over a range of job power caps. These experiments were repeated 5 times. The lighter colored bars are the results with our power balancing plugin, and lower values are better. Runtimes are normalized based on the rightmost darker colored bar (representing the baseline data) for each plot such that this bar always has a value of 1.0. The red whiskers that are above and below the top of the bars represent the max and min (respectively) of the observed runtimes. As the figure indicates, our power balancing plugin is able to provide substantial runtime improvements of up to 30% for Nekbone, miniFE, and CoMD. For the other workloads, the runtime improvements are up to 9–23%.

The amount of improvement varies depending on the power cap and the workload, but it tends to increase as the job power is increasingly constrained since the critical path can be operated at a higher and higher frequency relative to the other nodes. At the right side of each graph, job power is not very constrained and the nodes have enough power to run at closer to full frequency, so the critical path cannot be accelerated.

In all experiments, we note that we confirmed that the power balancing plugin obtains its runtime improvements without going over the job power budget. We also note that, in other clusters, the improvements may vary if the processors exhibit differing amounts of manufacturing variation than seen in our cluster.

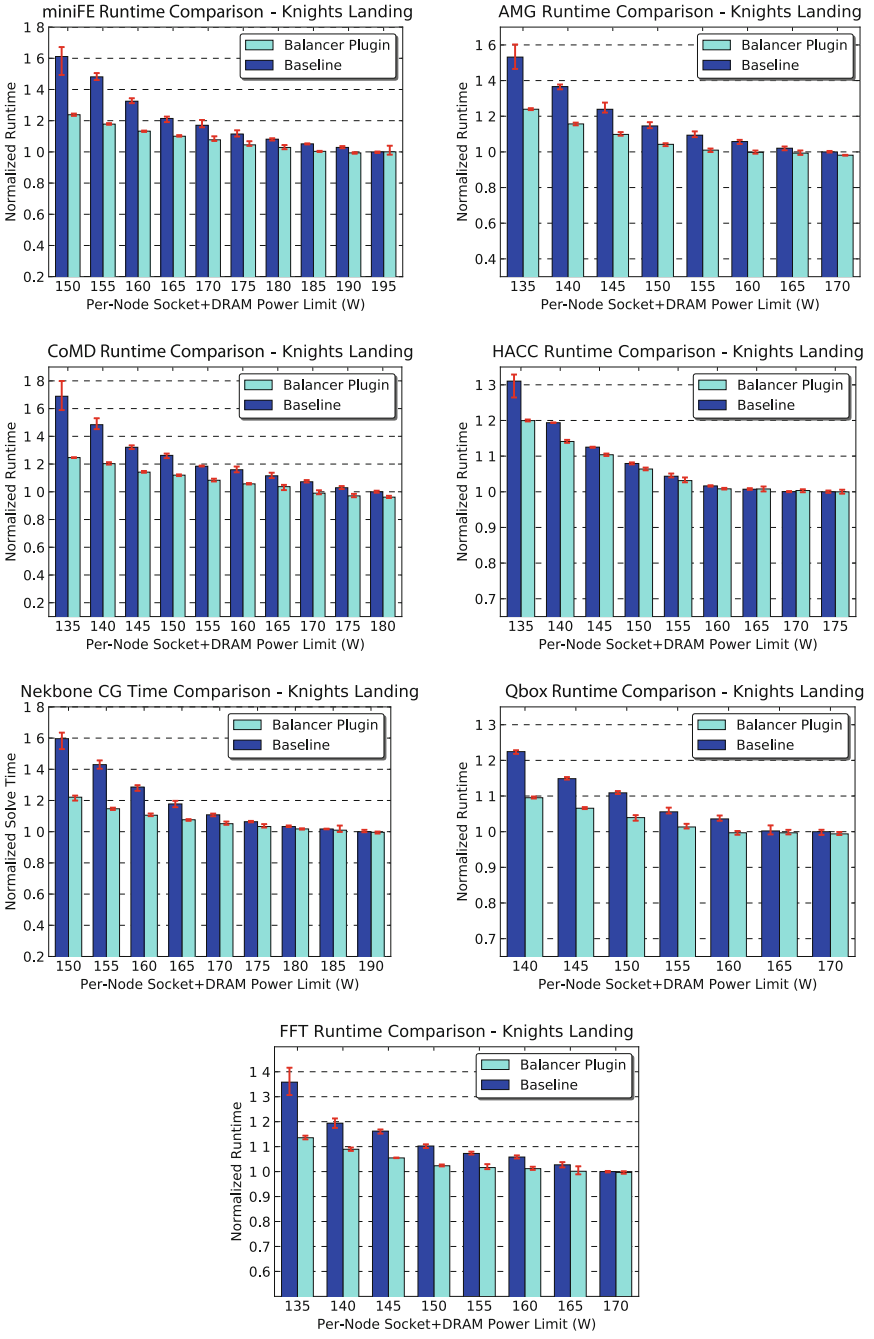

Fig. 3. Runtime improvements obtained with GEOPM power balancing plugin on a 12-node Knights Landing cluster. 5 runs averaged for each bar. (Color figure online)

4.3 Analysis of Runtime Improvements via Traces

Next, we trace the action of the power balancing plugin over the course of a run to show how the runtime improvements were obtained. In the left column of plots in Fig. 4, we show traces from a run of the HACC workload. In the right column, we show traces from a run of the Nekbone workload. For each, we highlight a run from one of the power caps studied in our sweeps. For brevity, we omit results collected for the other power caps and other benchmarks, but we note that we observed consistent trends in that data.

The top plot in the HACC traces shows the normalized runtime of each iteration of the HACC outer loop in the critical path node (i.e. the node with lowest power efficiency due to manufacturing variation) and compares the time taken when using the power balancing plugin versus the baseline. In the middle plot, we plot the power allocated to each node for each iteration of the outer loop when using the power balancing plugin. In the bottom plot, we plot the mean frequency that each node's processor runs at in each iteration of the outer loop when using the power balancing plugin. These traces were collected through GEOPM's tracing features.

As demonstrated in the top plot, the power balancing plugin is able to reduce the runtime of each iteration of the HACC outer loop which reduces the overall time-to-solution. The middle plot demonstrates how the power balancing plugin achieves this: it identifies the critical path nodes and allocates them a larger portion of the job power budget. In particular, Node 8 is allocated more power.

The power allocation is tuned using an objective function that penalizes variance in the time it takes the nodes to complete each iteration. From one iteration to the next, the amount of computation needs not be constant. In fact, the top plot demonstrates that the computation is not constant in HACC. Nonetheless, the power balancing plugin readily handles it. The bottom plot confirms that the variance-minimizing power allocation was the allocation that equalized frequency across processors in all nodes. This is expected when manufacturing variation is the cause of variation in iteration runtime across nodes.

The right column of Fig. 4 shows the corresponding traces for Nekbone, a more complicated example. The iteration loop time data in the top plot exhibits two phases. In the first phase, the runtime of the outer loop is slightly better than the baseline runtime when using the power balancing plugin, but in the second phase the power balancing plugin significantly improves the runtime. The two phases can be explained by observing that the Nekbone benchmark executes two conjugate gradient computations of different problem sizes. The second one is more sensitive to manufacturing variation because it is more compute-intensive. Thus, it offers more opportunity for acceleration.

In the middle and bottom plots, the traces confirm that the power balancing plugin is responding to differences in the outer loop runtime across nodes. In particular, Node 8 is allocated more power. This is expected based on additional experiments we performed to confirm that Node 8 has the processor with the lowest power efficiency (due to manufacturing variation) in our cluster: over a sweep of different power caps, we compared the average frequency each node's processor

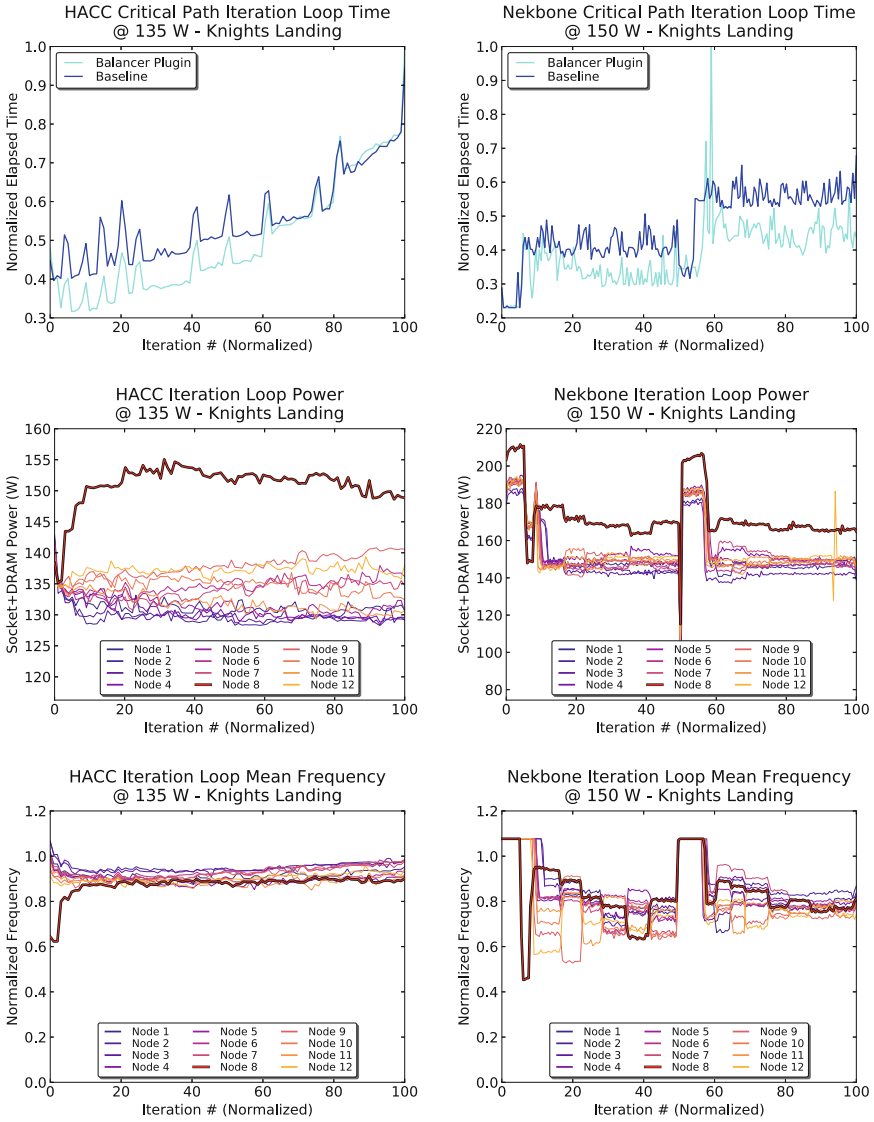

Fig. 4. Traces for an example run and power cap of HACC (Left) and Nekbone (Right). Top to bottom: time taken in the critical path node to complete each iteration, power allocated to each node in each iteration, and mean frequency in each node in each iteration

achieved when running a single-node compute-intensive synthetic workload and confirmed that the average frequency was consistently lowest on Node 8. We also note that the data demonstrates that our plugin adapts readily when Nekbone moves from the first conjugate gradient computation to the second. When the second begins, the plugin realizes that the previous power allocation is no longer ideal and it learns a new power allocation.

4.4 Overhead Measurements

Next we measure and report the memory usage and communication bandwidth costs associated with running the GEOPM framework and power balancing plugin as well as the overhead to the application's runtime associated with calling into GEOPM's profiling interface functions. GEOPM and its interfaces have been designed to minimize these costs.

To measure the memory working set, we queried the peak resident set size statistic (VmHWM) provided by Linux in /proc/<pid>/status/ for the GEOPM process on each node, at GEOPM shutdown time. To track communication bandwidth usage, we implemented accounting logic in the GEOPM code for tree communications to accumulate how many payload bytes are sent over the network. To track application overhead, we wrapped each GEOPM interface function with timers and implemented logic to accumulate the total time spent in all interface function calls. For each type of overhead, we obtain measurements on each node and report the maximum overhead across nodes.

We note that taking the maximum actually overestimates both the average network bandwidth usage per node and the overall application overhead. The node in which the GEOPM root controller lives uses more communication bandwidth than any other node, but it is the value we are reporting. Overhead on the critical path node will have greatest impact to overall application runtime, but we are reporting the maximum across any node; in our experiments, the overhead was typically lowest on the critical path node.

Table 2. Per-node memory usage, communication bandwidth, and application overhead

Workload	Memory working set	Communication BW (upper bound)	Application overhead (upper bound)
Qbox	40.8 MB	7.8 B/sec	2.32%
HACC	48.7 MB	36.2 B/sec	0.54%
Nekbone	37.1 MB	1121.3 B/sec	1.45%
AMG	34.9 MB	24.4 B/sec	0.97%
miniFE	34.8 MB	414.9 B/sec	2.38%
CoMD	34.7 MB	735.8 B/sec	2.88%
FFT	38.4 MB	338.6 B/sec	4.52%

Nonetheless, the costs are minimal as demonstrated in Table 2. They are easily outweighed by the large improvements in application time-to-solution presented earlier in this paper. We note, however, that we have not yet made a thorough effort to optimize the GEOPM code, so the overheads may be further reduced in the future.

5 Conclusion and Future Work

This paper introduced an open source, extensible, scalable runtime framework called GEOPM. GEOPM is being contributed to the community to accelerate collaboration and research toward software-hardware co-designed HPC energy management solutions. To demonstrate GEOPM's potential as a framework, this paper developed a power balancing plugin for GEOPM, and it presented results from our experiments with that plugin which demonstrated substantial improvements in time-to-solution for key CORAL system procurement and other benchmarks in power-capped systems.

In future work, we will expand upon our studies of the power balancing plugin to (a) determine bounds on how much benefit the plugin will provide in systems with processors spanning the full range of manufacturing variation possible in a given SKU, (b) evaluate benefits on additional benchmarks, and (c) demonstrate that the plugin's tree-hierarchical algorithm scales as well as expected in larger systems. In fact, the first scaling studies have already begun through a collaboration with Argonne National Laboratory. They are planned for the Theta system, a production system based on Intel Knights Landing hardware and Cray Aries Interconnect.

Lastly, the promising results presented in this paper motivate future work to spin up additional collaborations with the community to research new energy optimization strategies through GEOPM's plugin framework. It would be especially interesting to prototype plugins for GEOPM that optimize energy-to-solution or other objective functions beyond those demonstrated in this paper. It would also be interesting to explore optimizations that run in conjunction with power balancing optimizations to achieve speedups and energy efficiency improvements on top of the benefits of power balancing.

The authors are also seeking collaborations to (a) explore further integration of GEOPM with emerging power-aware scheduling functions in SLURM (or other workload managers) and (b) explore tuning power-performance knobs in software libraries/runtimes like MPI or OpenMP as well as knobs in the library or application layer of the HPC stack.

Acknowledgments. The authors would like to thank the following individuals for their input on this work: Vitali Morozov and Kalyan Kumaran of Argonne; Barry Rountree, Martin Schulz, and their teams from LLNL; James Laros, Ryan Grant, and their team from Sandia; and Richard Greco, Tryggve Fossum, David Lombard, Michael Patterson, and Alan Gara of Intel. Development of the GEOPM software package has been partially funded through contract B609815 with Argonne National Laboratory.

References

1. Eastep, J., Sylvester, S., Cantalupo, C., et al.: Global extensible open power manager: a vehicle for HPC community collaboration toward co-designed energy management solutions. In: Supercomputing PMBS (2016)
2. Schulz, K., Baird, C.R., Brayford, D., et al.: Cluster computing with OpenHPC. In: Supercomputing HPC Systems Professionals (2016)
3. Auweter, A., et al.: A case study of energy aware scheduling on SuperMUC. In: Kunkel, J.M., Ludwig, T., Meuer, H.W. (eds.) ISC 2014. LNCS, vol. 8488, pp. 394–409. Springer, Cham (2014). doi:10.1007/978-3-319-07518-1_25
4. Marathe, A., Bailey, P.E., Lowenthal, D.K., Rountree, B., Schulz, M., de Supinski, B.R.: A run-time system for power-constrained HPC applications. In: Kunkel, J.M., Ludwig, T. (eds.) ISC High Performance 2015. LNCS, vol. 9137, pp. 394–408. Springer, Cham (2015). doi:10.1007/978-3-319-20119-1_28
5. Rountree, B., Lowenthal, D.K., de Supinski, B., Schulz, M., Freeh, V.W.: Adagio: making DVS practical for complex HPC applications. In: ICS (2009)
6. Kappiah, N., Freeh, V.W., Lowenthal, D.K.: Just in time dynamic voltage scaling: exploiting inter-node slack to save energy in MPI programs. In: Supercomputing (2005)
7. Etinski, M., Corbalan, J., Labarta, J., Valero, M.: Optimizing job performance under a given power constraint in HPC centers. In: IGCC (2010)
8. Etinski, M., Corbalan, J., Labarta, J., Valero, M.: Linear programming based parallel job scheduling for power constrained systems. In: HPCS (2011)
9. Sarood, O., Langer, A., Gupta, A., Kale, L.: Maximizing throughput of overprovisioned HPC data centers under a strict power budget. In: Supercomputing (2014)
10. Global Extensible Open Power Manager Project. Intel Corporation (2016). http://geopm.github.io/geopm/
11. Shoga, K., Rountree, B., Schulz, M., Shafer, J.: Whitelisting MSRs with MSR-safe. In: Supercomputing Exascale Systems Programming Tools (2014)
12. Rountree, B., Ahn, D.H., de Supinski, B.R., et al.: Beyond DVFS: a first look at performance under a hardware-enforced power bound. In: HPPAC (2012)
13. Inadomi, Y., Patki, T., Inoue, K., et al.: Analyzing and mitigating the impact of manufacturing variability in power-constrained supercomputing. In: Supercomputing (2015)
14. CORAL Procurement Benchmarks. Livermore National Lab (2016). https://asc.llnl.gov/CORAL-benchmarks/CORALBenchmarksProcedure-v26.pdf
15. Mohd-Yusof, J.: Codesign molecular dynamics (CoMD) proxy app. In: ExMatEx All-Hands Meeting (2012)
16. Bailey, D., Barszcz, E., Barton, J., Browning, D., Carter, R., Dagum, L., Fatoohi, R., Frederickson, P., Lasinski, T., Schreiber, R., et al.: The NAS parallel benchmarks summary and preliminary results. In: Supercomputing (1991)
17. Intel: Intel-64 and IA-32 Architectures Software Developer's Manual, vols. 3A and 3B. System Programming Guide, Intel Corporation (2011)
18. Laros, J., DeBonis, D., Grant, R., et al.: High performance computing - power application programming interface specification, version 1.0. Sandia National Laboratories, Technical report SAND2014-17061 (2014)
19. Gschwind, M.: OpenPOWER: reengineering a server ecosystem for large-scale data centers. In: Hot Chips Symposium (HCS) (2014)
20. GEOPM Video Tutorials: Intel Corporation (2016). https://www.youtube.com/playlist?list=PLwm-z8c2AbIBU-T7HnMi_Pux7iO3gQQnz

21. Rountree, B., Lowenthal, D.K., Funk, S., et al.: Bounding energy consumption in large-scale MPI programs. In: Supercomputing (2007)
22. Cameron, K.W., Feng, X., Ge, R.: Performance-constrained distributed DVS scheduling for scientific applications on power-aware clusters. In: Supercomputing (2005)
23. Ge, R., Feng, X., Feng, W., Cameron, K.W.: CPU MISER: a performance-directed, run-time system for power-aware clusters. In: ICPP (2007)
24. Hsu, C.-H., Feng, W.-C.: A power-aware run-time system for high-performance computing. In: Supercomputing (2005)
25. Li, D., de Supinski, B., Schulz, M., Cameron, K., Nikolopoulos, D.: Hybrid MPI/OpenMP power-aware computing. In: IPDPS (2010)
26. Ellsworth, D., Patki, T., Perarnau, S., et al.: Systemwide power management with Argo. In: Parallel and Distributed Processing Symposium Workshops (2016)
27. Raghavendra, R., Ranganathan, P., Talwar, V., Wang, Z., Zhu, X.: No "power" struggles: coordinated multi-level power management for the data center. In: ASPLOS (2008)