Chapter

Characterizing Power and Energy Efficiency of Legion Data-Centric Runtime and Applications on Heterogeneous High-Performance Computing Systems

Song Huang, Song Fu, Scott Pakin and Michael Lang

Abstract

The traditional parallel programming models require programmers to explicitly specify parallelism and data movement of the underlying parallel mechanisms. Different from the traditional computation-centric programming, Legion provides a data-centric programming model for extracting parallelism and data movement. In this chapter, we aim to characterize the power and energy consumption of running HPC applications on Legion. We run benchmark applications on compute nodes equipped with both CPU and GPU, and measure the execution time, power consumption and CPU/GPU utilization. Additionally, we test the message passing interface (MPI) version of these applications and compare the performance and power consumption of high-performance computing (HPC) applications using the computation-centric and data-centric programming models. Experimental results indicate Legion applications outperforms MPI applications on both performance and energy efficiency, i.e., Legion applications can be 9.17 times as fast as MPI applications and use only 9.2% energy. Legion effectively explores the heterogeneous architecture and runs applications tasks on GPU. As far as we know, this is the first study to understand the power and energy consumption of Legion programming and runtime infrastructure. Our findings will enable HPC system designers and operators to develop and tune the performance of data-centric HPC applications with constraints on power and energy consumption.

Keywords: power consumption, Legion programming model, legion runtime, high performance computing, energy efficiency

1. Introduction

The U.S. Department of Energy (DOE) announced to invest $258 million to the exascale computing project in 2017. With funding from the six selected companies, the total investment reaches over $430 million to achieve the goal of delivering at least one exascale-capable supercomputer by 2021 [1]. Building an exascale high performance computing (HPC) system has to overcome four major

challenges: parallelism, memory and storage, reliability, and energy consumption. An exascale system, if built using the existing technologies, will consume half of a gigawatt of power, which highly exceeds the expected power limit specified by DOE. Therefore, innovative technologies are needed to enhance the power and energy efficiency and improve the system performance with a low power consumption.

Compute nodes are a major power and energy consumer inside an HPC system. Deng et al. [2] found that about 60% of system power is consumed by CPU, around 30% of power is allocated to memory, and other components account for 10%. This situation becomes more obvious in HPC environments where compute intensive and data intensive computation keeps a system always busy. Hence, reducing power and energy consumption of computing units and memory is the major challenge for efficiency of the whole system.

The message passing interface (MPI) is the de facto standard for writing HPC applications. It is a computation-centric programming model, where MPI processes are independent execution units that contain instructions and state information, use their address spaces, and interact with each other via inter-process communication mechanisms defined by MPI. Application programmers focus on writing computation processes and dealing with their communication, while data-related components, including data layout, data placement, and data movement, are implicitly determined by computation. As the volume, variety, and velocity of data dramatically increase, computation-centric programming becomes inefficient. Data-centric programming is increasingly addressing these problems, because focusing on the data makes the big-data problems much simpler to express. It enables programmers to define data properties including organization, partitioning, privileges, and coherence, also allows runtime systems to control data movement, communication, task scheduling, and execution.

Legion, which is jointly developed by Stanford University, Los Alamos National laboratory, and Nvidia, is a data-centric parallel programming system for writing portable high performance programs targeted at heterogeneous architectures [3, 4]. Legion provides abstractions which allow programmers to describe properties of program data, such as independence and locality [3]. By making the Legion programming system aware of the structure of program data, it can automate many of the tedious tasks programmers currently face, including correctly extracting task- and data-level parallelism and moving data around complex memory hierarchies.

Existing works mainly focus on improving the performance of Legion applications. Little is known about the energy efficiency of the Legion system and many questions have not been answered, such as:

- Unlike the traditional HPC programming systems, what are the distinct characteristics of power and energy consumption of Legion runtime and applications?
- Can Legion applications achieve better power and energy efficiency, at the same time as accelerate the execution and increase the throughput?
- How well do Legion runtime and applications utilize computing and memory resources on both homogeneous and heterogeneous systems?

In this chapter, we study these critical questions and analyze the energy efficiency of Legion applications and runtime system. We test a number of benchmark

applications with varying configurations on a CPU-GPU heterogeneous platform. We run both the MPI version and the Legion version applications. The heterogeneous system offers pure CPU and CPU-GPU execution environments. We use a variety of power profiling tools such as PAPI [5], RAPL [6], PowerAPI [7], and NVML [8] to measure runtime power consumption and characterize power consumption, energy consumption, and resource utilization of applications run on Legion. Important contributions include: (1) Legion Helper affects the performance and power consumption of applications; (2) Legion-based GPU applications perform better with regards to energy efficiency and execution time for larger problem size.

As far as we know, this is the first investigation of the performance and energy properties of Legion applications and data-centric Legion runtime system. The findings and results produced from this work will improve our understanding of Legion and develop resource scheduling to maximize system performance while operating under static/dynamic power caps.

The remainder of this chapter is structured as follows. Section 2 briefly presents the data-centric programming model and Legion runtime. The test environment (hardware, benchmarks, and profiling tools) is described in Section 3. Section 4 presents the results on performance and energy efficiency on servers with only CPU. The results on heterogeneous servers with both CPU and GPU are presented in Section 5. Key findings are highlighted in Section 6. Section 7 describes and related research and Section 8 provides the conclusion.

2. Legion programming and runtime system

Legion [3, 4] is a data-centric programming model and it provides runtime system to reduce expensive data movement in the complex memory hierarchy and to write highly portable and data intensive programs for heterogeneous system. Legion Runtime extracts independent tasks and allocates them to available computer resources to speed up parallel execution.

Compared to current computation-centric programming models, such as MPI and OpenMP, which require that programmers to explicitly specify the communication between compute nodes and data transfer for underlying parallel mechanisms, Legion focus more on defining data properties and the relationship between different data units [3]. Application developers can explicitly declare the properties of program data, including data organization, independence, partition, and locality. Therefore, Legion hides the operations of extracting parallelism and data movement and provides auto mapping to avoid suffering data moving overhead. Also, Legion allows programmers to customize optimal mapping for specific applications or infrastructure.

A dynamic scheduling approach called SOOP ("out-of-order" processor) is provided by the Legion runtime to map the dependences of tasks, distribute the tasks onto processor, map to physical instance for execution [4]. SOOP determines the task dependency at the logical region level by comparing the privileges and coherence modes to detect dependency between a newly registered task and a previous registered task. After the task dependency is satisfied, the task will be mapped and placed into the mapping queue, and scheduled to processors. Then task execution is performed and resources are recovered after execution. This whole process is automatic and hidden from Legion users. In our next discussion, we use the Legion Helper to refer to the set of processes that detect the dependency, map and dispatch of Legion tasks.

3. Evaluation environment

Before showing the experiment and discussing the results, we detail the platforms in our experimental environment in this section, provide the specifications of the homogeneous servers and heterogeneous servers, and describe the benchmark applications and profiling tools.

3.1 Hardware configurations

In the experiments, we use a homogeneous HPC server that consists of Enterprise version of Haswell processor, and a heterogeneous HPC server that has both CPU processor and a GPU accelerator. They will be referred to as the CPU server or GPU server in the following discussion.

3.1.1 CPU server

The CPU server is a Dell PowerEdge T630 computer that has two sockets with Intel Xeon E5-2683 v3 processors, 128 GB RAM and 28 TB SSD. **Table 1** contains the specification.

3.1.2 GPU server

To understand the power and energy characteristics of Legion on a heterogeneous environment, we run applications on a HP server having both Intel Xeon processor and NVIDIA Tesla K40c GPU accelerator, and another HP server with the same CPU processor and NVIDIA Tesla P100 GPU accelerator. **Table 2** shows their specifications.

3.2 Benchmark applications

To demonstrate the characteristics of the power and energy consumption of Legion runtime and application, we select two benchmark applications, which are compute-intensive, to run on both servers using Legion and MPI programming models.

3.2.1 MiniAero

MiniAero is a fluid dynamics mini-application [9, 10] designed to evaluate the programming model and hardware. It is an explicit unstructured finite volume code, which use Runge-Kutta four-order method to solve the compressible

Compute server	Dell PowerEdge T630
CPU Processor	2xIntel Xeon E5-2683 v3 (Haswell-EP)
Number of cores per socket	14
Number of threads per socket 28	28
Base frequency	2 GHz
Turbo frequency	3 GHz
Thermal design power per Socket	120 W

Table 1.
Configuration of the CPU server.

Compute server	HP ProLiant heterogeneous server
GPU processor	NVIDIA Tesla K40c
Number of CUDA cores	2880
DRAM	12 GB
Thermal design power per Socket	235w
GPU processor	NVIDIA Tesla P100
Number of CUDA cores	3584
DRAM	12 GB
Thermal design power per Socket	250w
CPU processor	Intel(R) Xeon(R) CPU X3460
Number of cores per socket	4
Number of threads per socket	8
Base frequency	2.8 GHz
Turbo frequency	3.46 GHz
Thermal design power per Socket	95 W

Table 2.
Configuration of the GPU server.

Navier-Stokes equations. It has the usual calculation and communication patterns on 3D unstructured mesh [11]. These meshes are generated on the CPU and then move to the devices (e.g. the CPU itself, GPU accelerator, or Xeon Phi). The original version of MiniAero uses multi-dimensional Kokkos arrays to store connectivity and flow data. Because MiniAero has a small dependency on tasks, the Legion version of MiniAero extracts concurrency from program data and maps it to physical regions to speed up the execution.

3.2.2 Circuit

Circuit [3] is a sample application that simulates on any graph of integrated circuit components and wires [10]. An explicit iterative solver step through time and calculates the updated voltages and currents on each node and wire. It computes the current by examining the voltage differential across every wire, updates the charge for each node with new current, and then re-calculate the voltage for every node according to the charge. The Legion runtime controls the resource allocation, performs task scheduling, and moves program data. These operations decompose independent data and allocate it to different computational units for scalability.

3.3 Profiling tools and performance metrics

3.3.1 PAPI

The Performance API (PAPI) [5] provides a set of standard APIs to access the hardware performance counter to capture real-time statistics from multiple hardware devices. The counter exist as a small set of registers, which record the occurrence of signals and events, for instance, Machine Specific Register (MSR). PAPI provides portability across different platforms via the ability to accept platform specific counter numbers. This enables the users to access a variety of devices for these counters and enable performance monitoring and tuning of these components.

3.3.2 RAPL

The Running Average Power Limit (RAPL) [6], introduced by Intel Xeon processors, which use a software power model to estimate the power and energy consumption of hardware. It can be used for monitoring of heat and energy and coverage of multiple domains such as PKG (Package Power), PP0 (Core), PP1 (uncore) and DRAM. The Haswell EP processor used in our experiments does not support PP0 and PP1 domains. Meanwhile, the RAPL counters can help to tune the performance of processors and balance the computing workloads on the nodes. In our experiments, we use the RAPL module in PAPI to profile the power consumption of the processor in the packet and DRAM domains.

3.3.3 PowerAPI

PowerAPI [7] provides a library for measuring power consumption at the process level. PowerAPI is a pure software approach to estimate power consumption of various hardware devices based on energy analytical models. Additional, the library is actor-based framework that the users can choose modules to fit for their requirements, which enables lowering computational cost and high accuracy. Moreover, PowerAPI can provide performance statistics of a particular process.

3.3.4 NVML

The NVIDIA Management Library (NVML) [8] monitors and manages NVIDIA GPU devices. It provides interfaces for querying and controlling device states, handling events, and reporting errors. Real-time query-able statistics such as ECC error counting, active processes and utilization, temperature and energy consumption can be captured via these interfaces. Also, some modifiable state can be accessed (e.g. ECC mode, compute mode, Persistence mode). In our K40c GPU and P100 GPU, we record the real-time board power draw by querying the performance counters.

4. Legion power and energy consumption on CPU server

To better understand the power and energy consumption patterns of Legion applications, we compare the performance of processors and power consumption of both MPI versions and Legion versions of *MiniAero* and the *circuit* with different problem sizes on different CPU cores.

To reduce noise and measurement errors, we perform each experiment 10 times and calculate the average of the measurements, and each run has the same initial conditions. The two applications are computationally intensive. Package and DRAM are the most important consumers of energy. To better characterize Legion applications and runtime, we separately measure and analyze the power and power consumption of Legion helper and computational processes.

4.1 Experimental results of MiniAero

For the MPI version of MiniAero, their processes have to be explicitly defined and they only share a part of the problem. The Legion version of MiniAero has some calculation processes and Legion helpers. We test the 3D-Sod with three problem

sizes, that is 128 × 128 × 4, 256 × 256 × 4 and 512 × 512 × 4, on one, two and four CPU cores.

4.1.1 CPU utilization of MiniAero application

The CPU utilization, which is used to estimate the system performance, measures the percentage of CPU cycles used on a core. On a multi-core processor, a load of more than 100% indicates that two or more cores are being used by applications. **Figure 1** shows CPU usage of MiniAero with different problem sizes running on different number of CPU cores. The figures show that the CPU usage of the MPI version is relatively stable and reaches about 100%. However, for the Legion version, the CPU cycles are not fully utilized by the Legion helper when the number of compute cores is less, but the usage keep increasing as the number of cores increases. On the other hand, those CPU cycles used by computational processes are reduced in our experiments. When the core number increases from 1 to 4, shown in **Figure 1a–f** and **g–i**, the Legion helper CPU usage increases from about 48% to more than 92%. In contrast, the average CPU utilization of the calculation processes drops from 75–25%. This suggests that identifying dependencies, mapping, and scheduling tasks on Legion can cause significant overhead that interferes with the useful calculation. The problem size, however, does not affect the CPU usage very much. In **Figure 1a, d** and **g**, the execution time and CPU utilization of the Legion version are almost identical, while the execution time of the MPI version increases exponentially. Other experimental results with same number of compute cores show a similar trend. The Legion runtime system offers better scalability.

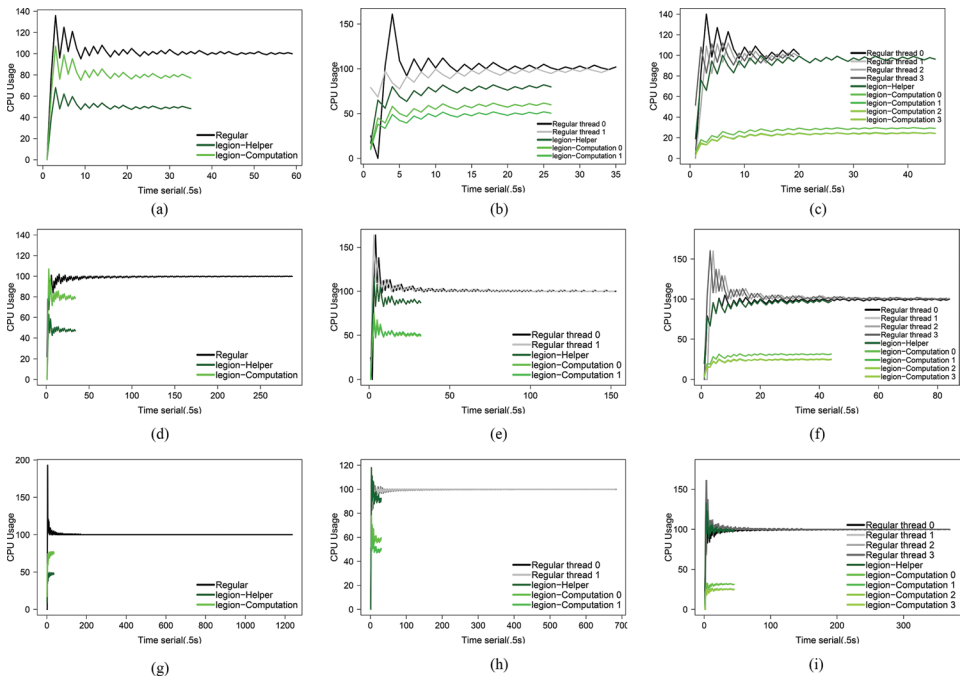

Figure 1.
*CPU utilization of MPI and Legion versions of the MiniAero application (a) Workload:128*128*4, 1 core, (b) Workload: 128*128*4, 2 cores, (c) Workload: 128*128*4, 4 cores, (d) Workload: 256*256*4, 1 core, (e) Workload: 256*256*4, 2 cores, (f) Workload: 256*256*4, 4 cores, (g) Workload: 512*512*4, 1 core, (h) Workload: 512*512*4, 2 cores, and (i) Workload: 512*512*4, 4 cores.*

4.1.2 Power usage of MiniAero application

The power consumption of the package and DRAM of both processors is similar. The biggest difference which is 12 W between packages is observed when MiniAero runs on a core as shown in **Figure 2a, d** and **g**. With the Legion runtime, the threads for arithmetic computational tasks are evenly pinned to cores of the two processors, while the Legion helper threads hovers between the cores and migrate across the cores some time. When the Legion helper floats to a processor running computational processes, the power consumption of that processor increases. For example, **Figure 2b** shows that the Legion helper is running on processor 0 and two computation processes are running on processors 0 and 1. Therefore, Package 0 draws 5.1 W more power when the processor is running at peak power. In **Figure 2c**, however, the Legion Helper runs on processor 1, which leads to more power consumption through this package. Despite this uncertainty, the total power consumption of both packages does not vary much when using the same number of cores. For example, in **Figure 2c, f** and **i**, the total power consumption of the package is 83.7–85.2 W. Memory consumes a small amount of power, that is 3.1–4.5 W and the variation is small as well. Combined with the CPU utilization results discussed in the previous subsection, we can observe that when the number of cores increases, the Legion Helper uses more CPU cycles and power consumption are also increased.

The total power consumption when running the Legion version, including both the computational tasks and the Legion helper, is 71.3–80.7% of that for the MPI version. The two limits are reached when the workload is 128 × 128 × 4. The

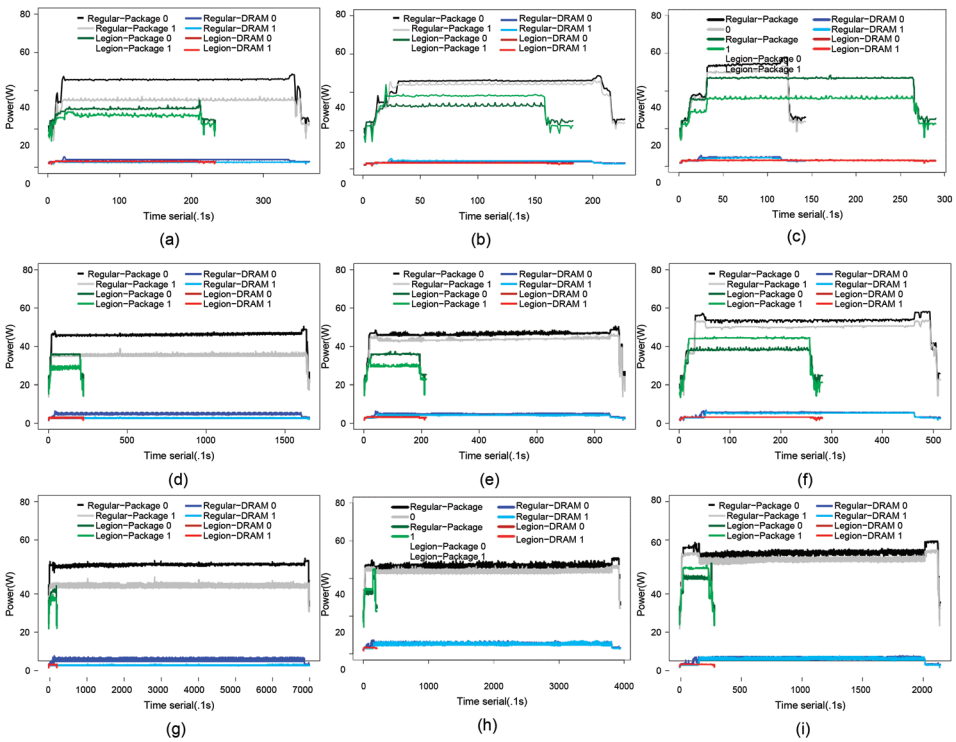

Figure 2.
*Package and DRAM power consumption of MPI and Legion versions of the MiniAero application (a) Workload: 128*128*4 1 core, (b) Workload: 128*128*4, 2 cores, (c) Workload: 128*128*4, 4 cores, (d) Workload: 256*256*4 1core, (e) Workload: 256*256*4, 2 cores, (f) Workload: 256*256*4 4 cores, (g) Workload: 512*512*4, 1 core, (h) Workload: 512*512*4, 2 cores, and (i) Workload: 512*512*4, 4 cores.*

power consumption of the Legion version on 1, 2 and 4 cores is 57.8, 72.6 and 81.7 W respectively, while the MPI counterpart consumes 81.1, 90 and 101.2 W.

In addition, we use PowerAPI to measure the power consumption of the MPI version and Legion version of MiniAero at the process level. The power consumption is depicted in **Figure 3**. **Figure 3** compares the power consumption of processors measured by *PowerAPI* on varying problem sizes and settings (128 × 128 × 4, 256 × 256 × 4, 512 × 512 × 4 on 1,2,4 cores respectively). The power consumption measured by PowerAPI is close to the results provided by RAPL. Overall, the difference between the two tools is within a range of [2.4w, 2.9w]. As both versions of MiniAero consume a little amount of power from DRAM, we do not include it in the figure.

4.1.3 Execution time and energy consumption of MiniAero

The execution time as shown in **Figure 4** and energy consumption as shown in **Figure 5** of Legion-version of MiniAero are relatively stable except the rise, when the Legion helper sends tasks to more cores for parallelism. In contrast, the MPI version follows the normal trend, where more cores accelerate execution and save energy. The results indicate that while Legion provides more partitions for the application, it distributes the workload equally among the cores and slows down tasks, which can be caused by the Legion helper. As a result, the power consumption of each processor does not change much while the execution time is prolonged, resulting in increased power consumption. The MPI version, on the other hand, fully exploits the extra cores, reducing execution time and power consumption.

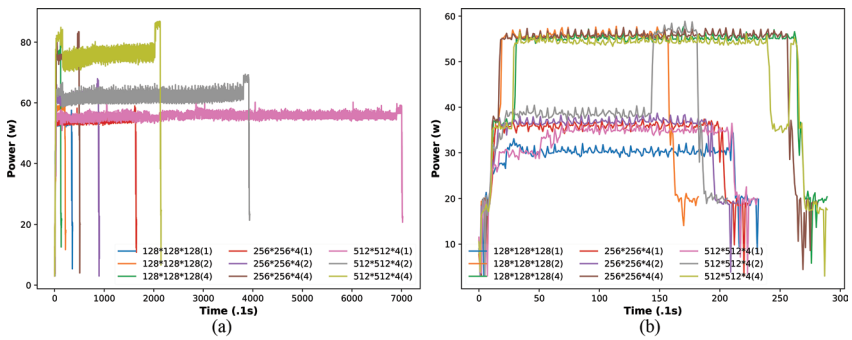

Figure 3.
Power consumption of MPI version of MiniAero measured by Power API (a) Power consumption of MPI version of MiniAero mea- sured by Power API, and (b) Power consumption of Legion version of MiniAero measured by Power API.

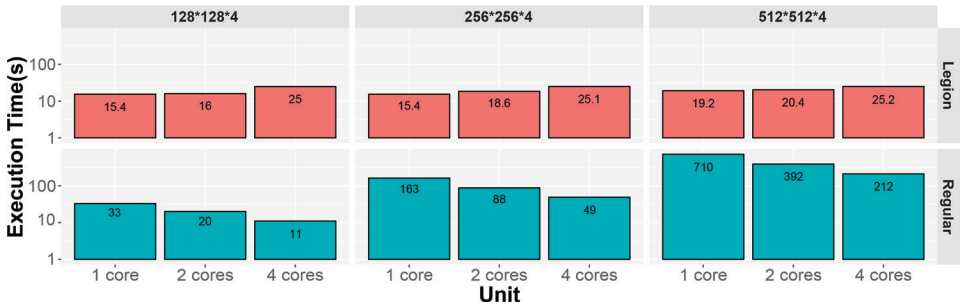

Figure 4.
Execution time of the MiniAero application.

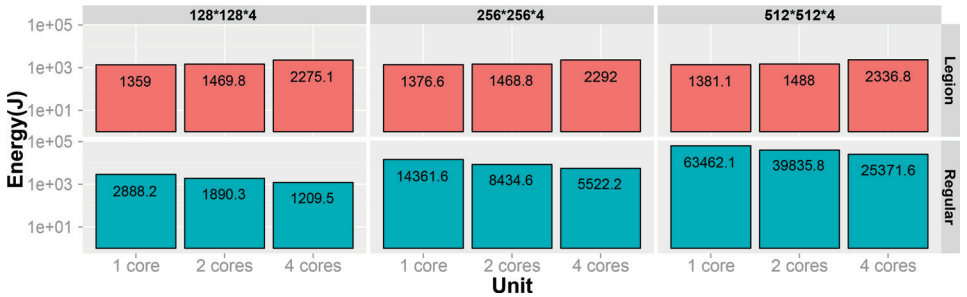

Figure 5.
Energy consumption of the MiniAero application.

The highest reduction in Legion execution time and energy is achieved when using a single core for a 512 × 512 × 4 problem size. The MPI version requires 36 times more execution time, and 45 times more energy than the Legion version respectively. Although the Legion helper causes more overhead, it reduces 89.1% of execution time and saves 90.8% energy compared to its MPI counterpart.

4.2 Experimental results of circuit

The Legion version of the circuit application is much more scalable than Legion version of MiniAero. The CPU utilization of Legion Helper jump to 17% at the beginning of the execution, and then the amount of utilization drops to 5% for the rest of the execution, as shown in **Figure 6a**. On the other hand, the CPU cores that perform computational tasks are fully used and the utilization is over 100% sometime. All the execution of *Circuit* on different numbers of cores has similar pattern.

In **Figure 6b** display that more power is consumed by the packet domain when more cores are used for computational tasks. From one core to two cores, power consumption increases by 6.6 W and an additional 6.8 W is consumed by two cores into four cores. In the meanwhile, the power draw of DRAM remains low and constant.

It is also shown in **Figure 7** that with more cores for computational tasks, execution time and power consumption are reduced. For example, if you run on two cores and four cores, 49.7 and 73.3% of execution time and 42.3 and 64.1% of energy, respectively, is reduced as if only one core is running.

The power per watt of the Legion version of the circuit is 6.7 MFLOPS/W on a core. It reaches 11.6 MFLOPS/W on two cores and 18.0 MFLOPS/W on 4 cores,

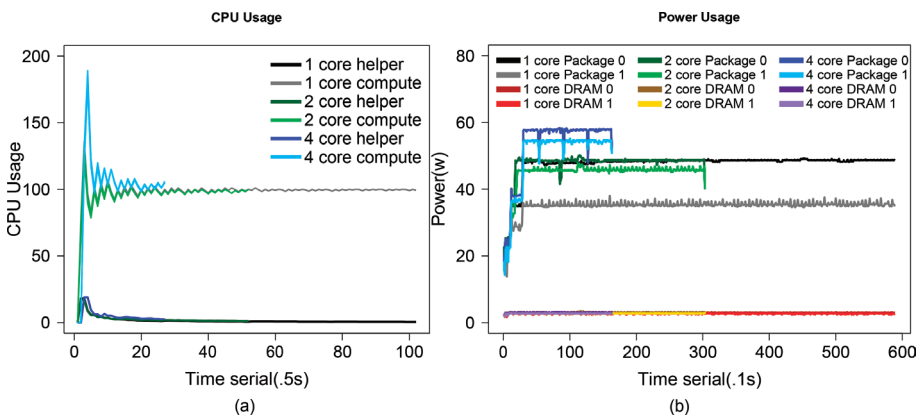

Figure 6.
CPU utilization and power consumption of the circuit application (a) CPU utilization, and (b) Power consumption.

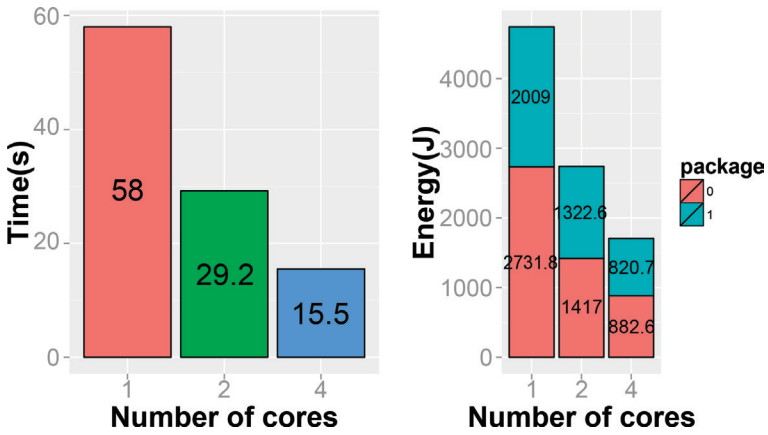

Figure 7.
Execution time and energy consumption of the circuit application.

which is 1.73 times and 1.55 times higher than one core and two cores. This observation indicates good scalability and energy efficiency of Legion runtime and application.

5. Power and energy consumption with legion on CPU-GPU server

To discover the power and energy consumption of Legion runtime and application on heterogeneous platform, we perform *circuit* tasks on GPU cores and compare them to the results of the CPU server. The Legion helper of the circuit identifies dependencies, maps logical areas, schedules tasks on CPU cores, and performs tasks on GPU CUDA cores.

5.1 CPU utilization of circuit

Figure 8 shows the resource usage when connecting to the CPU server and the heterogeneous server. During the initialization phase, the CPU utilization on both platforms has a steep jump and reach beyond 100%, while the GPU utilization remain 0. After that, the GPU starts with parallel circuit tasks. For the two problem sizes (2 loops and 4 pieces, 4 loops and 8 pieces) shown in **Figure 8**, the circuit tasks run at a high CPU utilization of nearly 100%. The *Circuit* takes advantage of 2880 CUDA cores in the Tesla K40c GPU, and the massive parallelism leads to a distinct reduced execution time. In **Figure 8a**, the execution time of the circuit on the GPU server is only about 1/3 of that on the CPU server, although the initialization phase requires another 7.6 s. The larger problem size, as shown in **Figure 8b**, causes the execution time on the CPU server to be increased 3.2-fold, while the increase on the GPU is only 0.8-fold. This indicates that Legion scales are scaled very well in the heterogeneous CPU-GPU environment.

5.2 Power consumption of circuit on heterogeneous server

Figure 9 shows the power consumption of the circuit on the CPU server and heterogeneous server with Tesla K40c. The power consumption of the CPU at the process level, measured with PowerAPI [7], is 3.05 W and remains stable in both cases. The power consumption of GPU, as measured by NVML [8], varies as the problem size changes; which is 50.5 W for 2 loops and 4 pieces of components and 55.1 W for 4 loops and 8 pieces of components respectively. This is because GPU

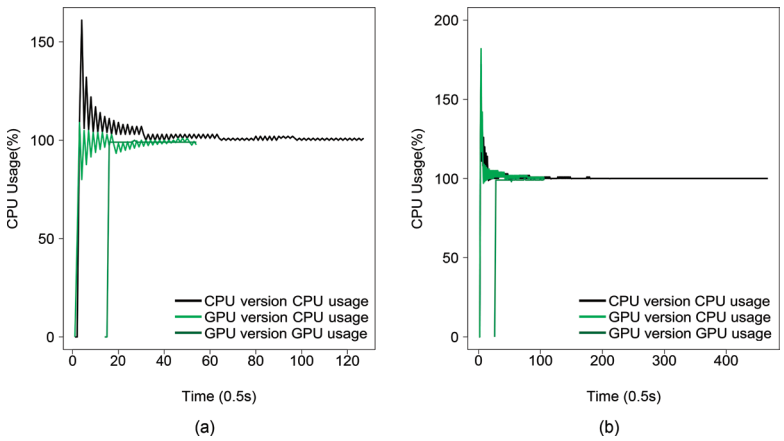

Figure 8.
The circuit application run on CPU-GPU heterogeneous server (a) CPU utilization of circuit (loops = 2 and pieces= 4), and (b) CPU utilization of circuit (loops = 4 and pieces= 8).

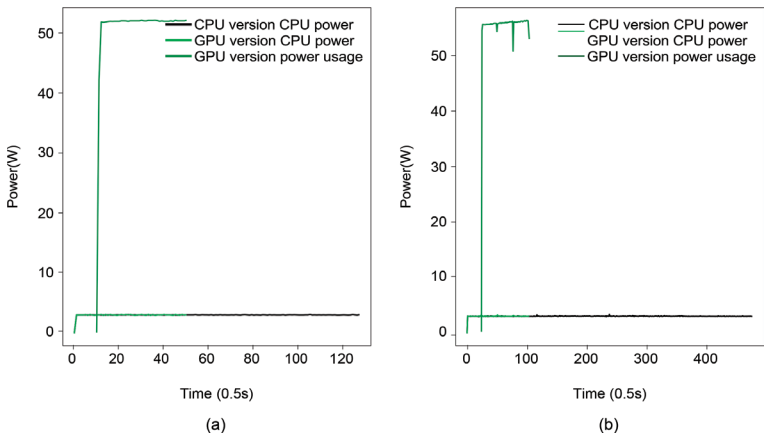

Figure 9.
Power consumption of circuit on the heterogeneous server. The CPU power usage is measured by PowerAPI, and the GPU power usage is measured by NVML (a) Power consumption of circuit (loops = 2 and pieces = 4), and (b) Power consumption of circuit (loops = 4 and pieces = 8).

has more capacity to handle more independent tasks and gain more throughput but consume more power.

5.3 Execution time and energy consumption of *Circuit*

Figure 10 depicts the execution time and energy consumption of the *Circuit*. Not surprisingly, the GPU version of *Circuit* runs on heterogeneous platform shorten the execution time but at the cost of consuming more power. For the problem size of 2 Loops and 4 Pieces, it takes 65.3 s for CPU version and 26.7 s for GPU version to execute, and consumes 389.6 J and 2164.1 J energy respectively. That means CPU version takes 1.45 times more execution time and saves 72.0% of the power compared to the GPU execution. In another situation for the problem size of 4 Loops and 8 Pieces, it takes 240.4 s and 1469.9 J for CPU version, and 53.9 s and 4782.6 J for GPU version to execute. That means CPU version of the circuit takes 3.46 times more execution time and saves 69.3% of the energy. This result indicates that Legion applications with large problem sizes should be delivered onto heterogeneous platform to reduce their execution time, which can lead to a slight increase in energy consumption.

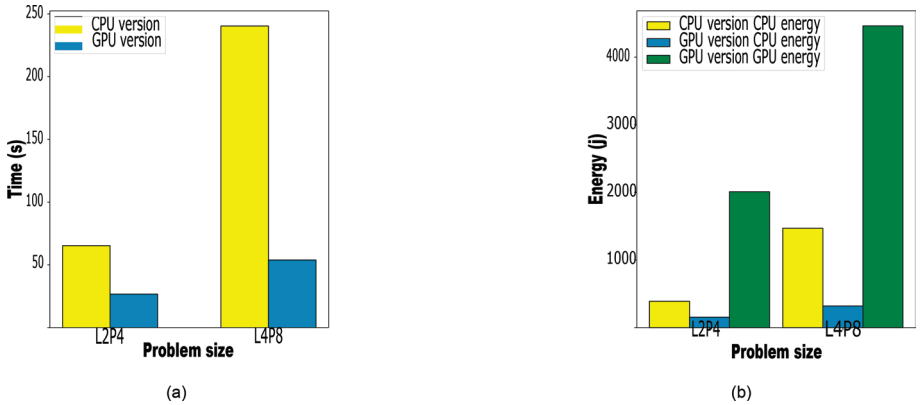

Figure 10.
Execution time and energy consumption of circuit on two platforms (a) Execution time of circuit, and (b) Energy consumption of circuit.

5.4 Influence of GPU frequency scaling

Dynamic voltage and frequency scaling (DVFS) is often used to find the best configuration for optimal energy and energy savings. **Figure 12** compares the performance of circuit running on GPU accelerator with different frequencies scaling, where **Figure 11** shows the power consumption of circuit running on different frequencies, **Figure 12a** compares the execution time. **Figure 12b** and **c** describe the energy consumption and the "FLOPS" which indicate the energy efficiency of the circuit application. From the figure we can see that the standard frequency which is 745 MHz of the Tesla K40c GPU is not the best setting for the Legion circuit. With the lowest frequency at 324 MHz, the circuit takes 2.21 times more execution time

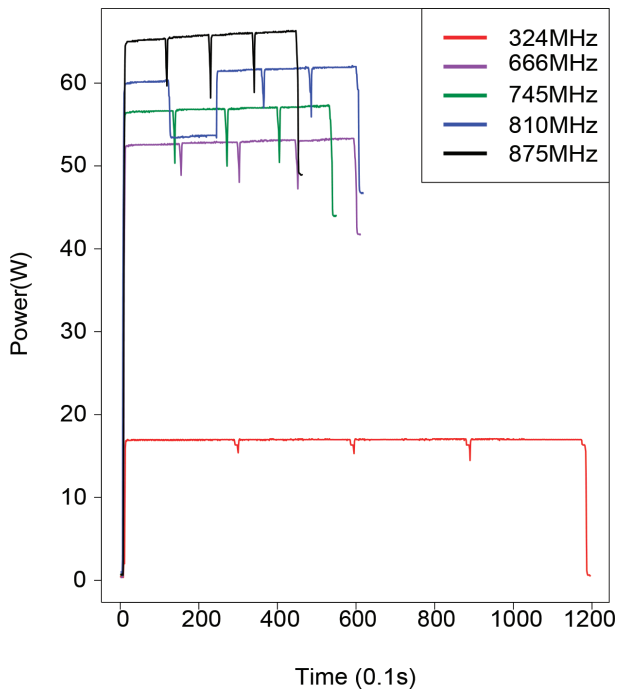

Figure 11.
Power consumption with GPU frequency scaling.

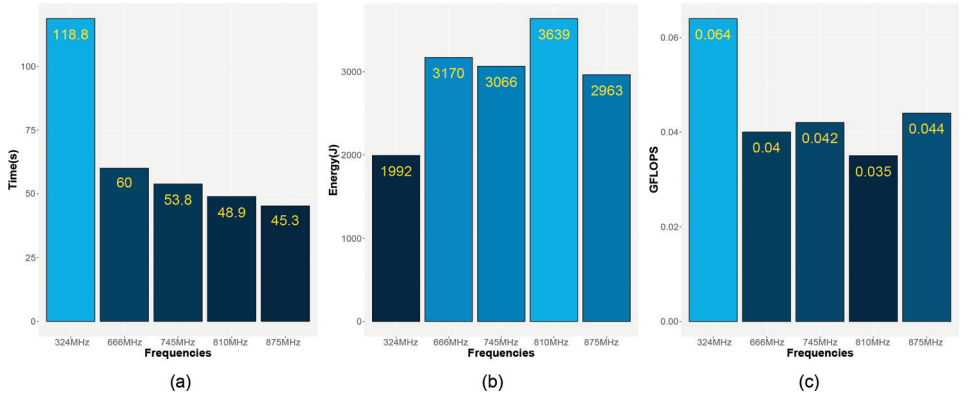

Figure 12.
Performance of circuit run at different GPU frequencies (loops = 4 and pieces = 8) (a) Execution time of different GPU frequency scaling, (b) Energy Consumption of different frequency GPU scaling, and (c) FLOPS of different frequency GPU scaling.

while saving 35% of power. Execution time is reduced by 18.8 with 3.4% energy savings when operating the circuit with the highest GPU frequency. In both cases the power consumption is lower than at the standard frequency. Both frequency settings provide good energy efficiency. The frequency selection depends on the power requirements.

To follow the advance of hardware technology, we not only test Legion applications on our Nvidia K40 GPU, but also run the Legion version of circuit on a new GPU, that is P100 GPU Accelerator(12 GB Card). We scale the frequency of P100 to its base frequency at 1126 MHz and its max frequency at 1303 MHz to evaluate the performance of circuit. **Figure 13** shows the performance of the Legion version of circuit with a workload of loops = 4 and pieces = 8. From **Figure 13a**, we can see if the frequency of P100 is set to 1303 MHz, the power consumption exceeds 100 W, while the power consumption is around 88 W, if the frequency is set to 1126 MHz. **Figure 13b–d** depict the execution time, energy consumption, and the processing power of P100 for Legion circuit respectively. Compared to Tesla K40c, there is a big improvement on performance, while the energy consumption has a significant drop. This is due to the reduced execution

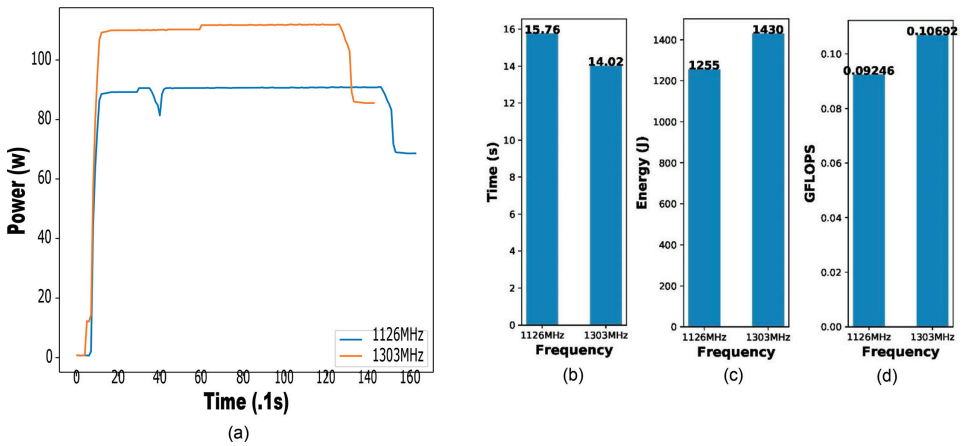

Figure 13.
Performance of circuit run at different GPU frequencies (loops = 4 and pieces = 8) (a) Power consumption of different GPU frequency scaling, (b) Execution time, (c) Energy, and (d) GFLOPS.

time. Hence, we expect that the Legion version of circuit will have a better performance on the latest GPUs.

6. Findings and discussion

The Legion -based MiniAero is a good example to highlight the Legion Helper's scalability problem. If the Legion helper is unable to isolate independent tasks quickly enough with the increased number of associated compute resources (such as CPU cores), this becomes a performance bottleneck. The resource utilization of Legion helper processes continues to increase and the throughput of compute tasks decreases. This leads to a longer execution time and reduced energy efficiency.

In cases where the Legion system and legacy applications have good scalability, energy and energy savings become more effective as more computational resources are used. The execution time of an application is significantly reduced, while the power consumption does not increase much, resulting in better energy efficiency.

Legion offers significant benefits through GPU computing. As GPU-mapped and scheduled tasks can be performed in parallel, performance enhancement and energy efficiency can be further improved.

7. Related works

Some new Legion program model features, model components, and how these components work were presented in [4]. A combination of static and dynamic checks to improve the solidity of the Legion system and a compositional parallel semantics are described in [12]. An event-based runtime system [13] is embedded in Legion asynchronously for heterogeneous and distributed storage architectures. Structure slicing [14] breaks the specification of data usage, identifies data parallelism, and reduces data movement. A highly productive programming language, Regent [10], which can be translated into Legion implementation, runs sequentially without explicit synchronization.

Power profiling in production computer systems provides valuable data and knowledge for the development of power simulators and resource scheduling policies. Fine-grained power profiling techniques measure the power consumption of individual hardware components such as CPU [15], memory [16], hard disk [17] and other devices [18]. In contrast, coarse-grained performance profiling aims to characterize system-wide performance dynamics, such as the macro stream framework [19]. Moreover, a power meter for virtualized environments was presented in [20]. CPU event counters and the Performance Programming Interface Library were used to estimate the power usage on a per-thread basis. Kamil et al. profiled HPC applications on multiple test platforms and projected the performance profiling results from a single node to a complete system [21]. Ge et al. investigated the influence of software and hardware configurations on system-wide power consumption [22]. They found that properties of HPC applications affect the power consumption of a system. Hackenberg et al. conducted a detailed analysis of Haswell's P-state and C-state transition latencies and the impact of Haswell's new power management mechanisms on memory bandwidth and performance reproducibility [23]. Our work differs from these previous efforts by measuring and analyzing the impact of new Haswell power management capabilities on the performance and performance of HPC codes.

Some researchers analyzed the power and energy efficiency of different types of applications run on HPC systems. Bari et al. investigated OpenMP's runtime configurations on power constrained systems at different power levels [24]. They found that a suitable selection of OpenMP's runtime parameters could improve the execution time and reduce the energy consumption of a parallel program by up to 67 and 72%, respectively. Qasem et al. [25] evaluated the impact of data layout and placement on the energy efficiency of heterogeneous applications by means of memory divergence, data access patterns, arithmetic intensities and data placement. They found that data layout and placement had a significant impact on the energy efficiency. Additionally, analytical models were developed to analyze energy efficiency in [26]. The models were able to support a priori selection of the operating frequency that leaded to a near optimal energy consumption for the execution of multi-threading applications. Meanwhile, Heinrich et al. aimed to predict the energy consumption of MPI applications by developing a computation model, a communication model, and an energy model which were integrated into the SimGrid simulation toolkit [27]. To improve the system performance by utilizing the available power budget more efficiently on multiple-node platforms, a hierarchical multi-dimensional power aware allocation framework was developed in [28] for power bounded parallel computing. The power allocation was performed using memory power-level settings, thread concurrency throttling, and core-thread affinity, and the scheduler outperformed other methods by 20% on average.

To control the power consumption of HPC systems, power limitation [29] is a promising and effective approach. System operators can balance the performance and power consumption of clusters by adjusting the maximum amount of power (also called the power budget) that clusters can consume. Pelly et al. presented a dynamic current sourcing and coverage method at the [30] Power Distribution Unit (PDU). They proposed using a heuristic policy to shift the capacity weakness to servers with increasing power requirements. Zhang et al. proposed a hybrid software/hardware power capping system and proved that their power cap outperforms the hardware power capping system provided by Intel and has the same reaction time [31]. For HPC jobs, many factors affect power consumption, including hardware configurations and resource usage. Femal et al. developed a hierarchical management policy to distribute the power budget to clusters [32]. Kim et al. investigated the relationship between CPU voltages and system performance and energy efficiency [33]. Utilizing Dynamic Voltage Scaling (DVS) technologies, a Task Planning Policy has been proposed that aims to minimize energy consumption while meeting specified performance requirements. Rountree et al. proposed guidelines for overprovisioning hardware with hardware-enforced performance limitations and system-wide performance reallocation in an application-independent manner [34, 35]. We have developed a complete system simulator, TracSim [36], which estimates the capacity of trapped energy under various power-limiting and job-planning guidelines.

8. Conclusion

In this chapter, we describe the power consumption, energy efficiency, performance, and resource usage of Legion runtime environment and applications. Our experimental results show that Legion offers favorable energy efficiency, although in some cases its scalability can be influenced by Legion Helpers. The Legion programming model is consistent with the massively parallel nature of the GPU design and shows good performance and energy efficiency for large problem-size applications.

Acknowledgements

This work is supported by the U.S. Department of Energy contract DE-AC52-06NA25396. This chapter has been assigned the LANL identifier LA-UR-16-25965.

Conflict of interest

The authors declare no conflict of interest.

Author details

Song Huang[1], Song Fu[1*], Scott Pakin[2] and Michael Lang[2]

1 University of North Texas, Denton, Texas, USA

2 Los Alamos National Laboratory, Los Alamos, New Mexico, USA

*Address all correspondence to: song.fu@unt.edu

References

[1] D. of Energy. Department of Energy Awards Six Research Contracts Totaling $258 million to Accelerate U.S. Supercomputing Technology; 2017

[2] Deng Q, Meisner D, Bhattacharjee A, Wenisch TF, Bianchini R. Coscale: Coordinating cpu and memory system dvfs in server systems. In: Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture. IEEE Computer Society; 2012. pp. 143-154

[3] Legion Project. http://legion.stanford.edu/. 2016

[4] Bauer M, Treichler S, Slaughter E, Aiken A. Legion: Expressing locality and independence with logical regions. In: High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for. IEEE; 2012. pp. 1-11

[5] Mucci PJ, Browne S, Deane C, Ho G. Papi: A portable interface to hardware performance counters. In: Proceedings of the Department of Defense HPCMP users Group Conference. Vol. 710. 1999

[6] David H, Gorbatov E, Hanebutte UR, Khanna R, Le C. Rapl: Memory power estimation and capping. In: Proceedings of the 16th ACM/IEEE International Symposium on Low Power Electronics and Design. ACM; 2010. pp. 189-194

[7] Bourdon A, Noureddine A, Rouvoy R, Seinturier L. PowerAPI: A software library to monitor the energy consumed at the process-level. ERCIM News. 2013;**92**:2013

[8] NVIDIA. Nvidia management library (nvml), 2016

[9] Mantevo Project. https://mantevo.org/packages/. 2016

[10] Slaughter E, Lee W, Treichler S, Bauer M, Aiken A. Regent: A high-productivity programming language for hpc with logical regions. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. ACM; 2015. p. 81

[11] Hollman DS, Hollman DS, et al. Lessons Learned from Porting the Miniaero Application to Charm++. Technical report. Sandia National Laboratories; 2015

[12] Treichler S, Bauer M, Aiken A. Language support for dynamic, hierarchical data partitioning. In: ACM SIGPLAN Notices. Vol. 48. ACM; 2013. pp. 495-514

[13] Aiken A, Bauer M, Treichler S. Realm: An event-based low-level runtime for distributed memory architectures. In: Parallel Architecture and Compilation Techniques (PACT), 2014 23rd International Conference on. IEEE; 2014. pp. 263-275

[14] Bauer M, Treichler S, Slaughter E, Aiken A. Structure slicing: Extending logical regions with fields. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE Press; 2014. pp. 845-856

[15] Magklis G, Scott ML, Semeraro G, Albonesi DH, Dropsho S. Profile-based dynamic voltage and frequency scaling for a multiple clock domain microprocessor. ACM SIGARCH Computer Architecture News. 2003;**31**(2):14-27

[16] Ye W, Vijaykrishnan N, Kandemir M, Irwin MJ. The design and use of simplepower: A cycle-accurate energy estimation tool. In: Proceedings of the 37th Annual Design Automation Conference. ACM; 2000. pp. 340-345

[17] Zedlewski J, Sobti S, Garg N, Zheng F, Krishnamurthy A, Wang RY, et al. Modeling hard-disk power consumption. In: FAST. Vol. 3. 2003. pp. 217-230

[18] Ye TT, Benini L, De Micheli G. Analysis of power consumption on switch fabrics in network routers. In: Design Automation Conference, 2002. Proceedings. 39th. IEEE; 2002. pp. 524-529

[19] Zhang Z, Fu S. Macropower: A coarse-grain power profiling framework for energy-efficient cloud computing. In: Performance Computing and Communications Conference (IPCCC), 2011 IEEE 30th International. IEEE; 2011. pp. 1-8

[20] Phung J, Lee YC, Zomaya AY. Application-agnostic power monitoring in virtualized environments. In: Cluster, Cloud and Grid Computing (CCGRID), 2017 17th IEEE/ACM International Symposium on. IEEE; 2017. pp. 335-344

[21] Kamil S, Shalf J, Strohmaier E. Power efficiency in high performance computing. In: Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on. IEEE; 2008. pp. 1-8

[22] Ge R, Feng X, Song S, Chang H-C, Li D, Cameron KW. Powerpack: Energy profiling and analysis of high-performance systems and applications. IEEE Transactions on Parallel and Distributed Systems. 2010;**21**(5):658-671

[23] Hackenberg D, Schöne R, Ilsche T, Molka D, Schuchart J, Geyer R. An energy efficiency feature survey of the intel Haswell processor. In: Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International. IEEE; 2015. pp. 896-904

[24] Bari MAS, Malik AM, Qawasmeh A, Chapman B. A detailed analysis of OpenMP runtime configurations for power constrained systems. In: 2017 Eighth International Green and Sustainable Computing Conference (IGSC). IEEE; 2017. pp. 1-8

[25] Qasem A, Teich S. Evaluating the impact of data layout and placement on the energy efficiency of heterogeneous applications. In: Green and Sustainable Computing Conference (IGSC), 2017 Eighth International. IEEE; 2017. pp. 1-8

[26] Rauber T, Runger G, Stachowski M. Model-based optimization of the energy efficiency of multi-threaded applications. In: 2017 Eighth International Green and Sustainable Computing Conference (IGSC). IEEE; 2017. pp. 1-6

[27] Heinrich FC, Cornebize T, Degomme A, Legrand A, Carpen-Amarie A, Hunold S, et al. Predicting the energy-consumption of mpi applications at scale using only a single node. In: Cluster Computing (CLUSTER), 2017 IEEE International Conference on. IEEE; 2017. pp. 92-102

[28] Zou P, Allen T, Davis CH IV, Feng X, Ge R. Clip: Cluster-level intelligent power coordination for power-bounded systems. In: Cluster Computing (CLUSTER), 2017 IEEE International Conference on. IEEE; 2017. pp. 541-551

[29] Choi J, Govindan S, Urgaonkar B, Sivasubramaniam A. Profiling, prediction, and capping of power consumption in consolidated environments. In: Modeling, Analysis and Simulation of Computers and Telecommunication Systems, 2008. MASCOTS 2008. IEEE International Symposium on. IEEE; 2008. pp. 1-10

[30] Pelley S, Meisner D, Zandevakili P, Wenisch TF, Underwood J. Power routing: Dynamic power provisioning

in the data center. In: ACM Sigplan Notices. Vol. 45. 2010. pp. 231-242

[31] Zhang H, Hoffmann H. Maximizing performance under a power cap: A comparison of hardware, software, and hybrid techniques. ACM SIGARCH Computer Architecture News. 2016;**44**(2):545-559

[32] Femal ME, Freeh VW. Boosting data center performance through non-uniform power allocation. In: Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on. IEEE; 2005. pp. 250-261

[33] Kim KH, Buyya R, Kim J. Power aware scheduling of bag-of-tasks applications with deadline constraints on Dvs-enabled clusters. In: CCGrid; 2007

[34] Ellsworth DA, Malony AD, Rountree B, Schulz M. POW: System-wide dynamic reallocation of limited power in HPC. In: Proc. of HPDC. 2015

[35] Patki T, Lowenthal DK, Rountree B, Schulz M, De Supinski BR. Exploring hardware overprovisioning in power-constrained, high performance computing. In: Proceedings of the 27th international ACM conference on International conference on supercomputing. ACM; 2013. pp. 173-182

[36] Zhang Z, Lang M, Pakin S, Fu S. Trapped capacity: Scheduling under a power cap to maximize machine-room throughput. In: Proceedings of the 2nd International Workshop on Energy Efficient Supercomputing. IEEE Press; 2014. pp. 41-50