Adaptive Configuration Selection for Power-Constrained Heterogeneous Systems

Peter E. Bailey, David K. Lowenthal
Department of Computer Science
The University of Arizona
{pbailey, dkl}@cs.arizona.edu

Vignesh Ravi
Advanced Micro Devices, Inc.
Vignesh.Ravi@amd.com

Barry Rountree, Martin Schulz,
Bronis R. de Supinski
Lawrence Livermore National Laboratory
{rountree4, schulzm, bronis}@llnl.gov

Abstract—**As power becomes an increasingly important design factor in high-end supercomputers, future systems will likely operate with power limitations significantly below their peak power specifications. These limitations will be enforced through a combination of software and hardware power policies, which will filter down from the system level to individual nodes. Hardware is already moving in this direction by providing power-capping interfaces to the user. The power/performance trade-off at the node level is critical in maximizing the performance of power-constrained cluster systems, but is also complex because of the many interacting architectural features and accelerators that comprise the hardware configuration of a node.**

The key to solving this challenge is an accurate power/performance model that will aid in selecting the right configuration from a large set of available configurations. In this paper, we present a novel approach to generate such a model offline using kernel clustering and multivariate linear regression. Our model requires only two iterations to select a configuration, which provides a significant advantage over exhaustive search-based strategies. We apply our model to predict power and performance for different applications using arbitrary configurations, and show that our model, when used with hardware frequency-limiting, selects configurations with significantly higher performance at a given power limit than those chosen by frequency-limiting alone.

When applied to a set of 36 computational kernels from a range of applications, our model accurately predicts power and performance; it maintains 91% of optimal performance while meeting power constraints 88% of the time. When the model violates a power constraint, it exceeds the constraint by only 6% in the average case, while simultaneously achieving 54% more performance than an oracle.

I. Introduction

Traditionally, supercomputers have been designed to maximize performance irrespective of power. This trend cannot continue; the US Department of Energy has a goal of 20 MW for an exascale supercomputer, similar to the power of today's petascale machines. Tianhe-2, the fastest supercomputer as of 2013, achieves a peak performance of 33.8 petaFLOP/s under load at 17.8 MW of power. Therefore, the goal of exascale performance at 20 MW implies at least a 26-fold improvement in power efficiency at scale.

To achieve this ambitious goal, exascale system designs are likely to include two things that are becoming commonplace. The first is accelerators (e.g., GPUs, Xeon Phi). Newer integrated CPU-GPU architectures, such as AMD's Kaveri, reduce GPU overhead and further improve power efficiency. The second trend we expect is for systems to include more hardware than can be powered fully simultaneously. In such a system, power constraints will be enforced by system-wide power policies and local power and thermal design points. Additionally, the system will selectively enable or disable hardware resources to direct power to performance-critical components.

Such power constraints will be passed down through the machine hierarchy to each rack, node, and core. Already, we see node-level architectures imposing power constraints and controlling power distribution; for example, some Intel processors support the running average power limit (RAPL) [1] interface, which allows the user or system software to set a power constraint, while AMD APUs provide BAPM [2] to balance power between the CPU and GPU on the chip.

Given a power constraint, maximizing performance is complex and requires finding appropriate node-level power-performance trade-offs. For example, how do different hardware-level configurations—where a configuration consists of a device selection (CPU or GPU), number of cores, voltage and frequency for both the CPU and GPU, and process/core mapping—affect performance over a wide range of parallel applications? This question can most effectively be answered using a model that predicts both power and performance across the entire configuration space for a range of applications. In this paper, we demonstrate the creation and application of such a model to choose node-level configurations that maximize performance under prescribed power constraints in a heterogeneous machine.

In this paper, we make the following contributions:

- We characterize the power and performance of a set of benchmarks on a single-chip heterogeneous processor, and demonstrate a method to build a prediction model from this characterization.
- We create a predictive model for estimating relative power and performance across available devices and configurations, using data from only two sample configurations (one per device) as input.
- We perform adaptive configuration selection using the model to maximize performance under imposed power constraints, and we compare our method to state-of-the-practice power allocation schemes.

Our model is a key ingredient to maximizing performance on a multi-node cluster. Before that problem can be attempted, an accurate node-level model must be developed. This work makes clear that device selection is important for performance and power. Our model predicts relative performance between devices and between configurations within a device, allowing a wide range of power limits. In addition, our model needs only two iterations of a kernel to find an effective configuration.

In the rest of this paper, Section II explores related work, Section III provides step-by-step analysis of our modeling process, and Section IV identifies our experimental setup and details data collection and model derivation. Next, Section V presents analysis and interpretation of model results. Finally, Section VI outlines future work.

This material is based upon work supported by the U.S. Department of Energy's Lawrence Livermore National Laboratory, Office of Science, under Award number DE-AC52-07NA27344 and supported by Office of Science, Office of Advanced Scientific Computing Research (LLNL-CONF-656877).

II. Related Work

A. Performance Modeling

Many researchers have studied performance prediction models for parallel applications on homogeneous platforms. The work of Curtis-Maury et al. [3]–[6] is closely related to ours. They implemented various online and offline configuration selection runtimes for OpenMP applications, employing dynamic concurrency throttling (DCT), dynamic voltage and frequency scaling (DVFS), and simultaneous multithreading (SMT) to maximize performance. These systems independently employed offline regression models and artificial neural networks (ANNs).

Cochran et al. and Reda et al., respectively [7], [8], demonstrated runtime systems for power-constrained platforms, employing DVFS and thread packing. However, they did not consider heterogeneous platforms, and, in [7], required physical instrumentation with a power meter to make power measurements. Additionally, their runtime overhead was comparatively large. In contrast, our system uses on-chip power estimates, and requires less than one millisecond to make each configuration selection.

Lee et al. created models based on regression and ANNs to predict performance for parallel applications on homogeneous cluster systems [9]. Their models were application-specific, and required substantial training of each application before predictions could be made. In contrast, our system may be used with any application.

Wang et al. used machine learning to select configurations for OpenMP applications on two significantly distinct multicore processors [10]. They compared their implementation to both regression-based and analytical performance prediction models, showing that their implementation required less training and made predictions with higher accuracy. However, they did not predict power, and only considered a single device at a time. Also, their implementation requires instrumentation by a compiler, and therefore access to source code.

Karami et al. [11] and Kerr et al. [12] created respective performance models for NVIDIA GPUs that used performance counter statistics and multiple regression, factors in common with our system. However, their models only apply to GPUs, and do not account for power or energy. In addition, the system of Kerr et al. requires not only execution of a kernel to make predictions, but also static analysis and data collection via simulation. Our system only requires modification of the runtime, and does not require source code or a simulator.

Luk et al. [13] built Qilin, a runtime system and programming framework for heterogeneous systems. Qilin features adaptive workload partitioning between devices. While the authors show that Qilin achieves energy savings over CPU-only and GPU-only configurations, Qilin is not power-aware.

In contrast to the above body of work, our work models both performance and power on heterogeneous systems with multiple devices and multiple implementations, and thus requires not only accurate CPU power and performance models, but also accurate GPU models. Our work has additional advantages over the above. First, modern machines have many more available configurations. In particular, the addition of accelerators provides opportunities for performance and power efficiency improvement, but such improvements are not automatic, and accelerators do not benefit all parallel code. Our model navigates this space of configurations and devices after only two iterations, efficiently determining which device/configuration to use for each kernel. Second, our model predicts power and performance for configurations on which it has not been trained. This significantly reduces the amount of training data required, and prepares our model for future many-core architectures for which training on all configurations will be infeasible. Finally, our combined model benefits from the close relationship between power and performance. Specifically, we use power measurements to predict performance and vice versa; power consumed at our sample configurations is a good predictor of performance in other configurations, while performance is a good predictor of power consumption. Also, running under a power constraint demands simultaneous accuracy of the power and performance models; if power or performance are mispredicted, the resulting configuration will use too much power, achieve suboptimal performance, or both.

B. Power-Performance Modeling

There are many other research thrusts involving power-performance prediction models. Li et al. implemented a hybrid MPI/OpenMP power-aware runtime system employing DCT and DVFS [14]. This system used online application samples with an offline model to optimize energy while maintaining performance. Springer et al. [15] demonstrated a system for MPI applications that, given an energy budget, selected an appropriate number of nodes and a per-phase DVFS setting to minimize application completion time. Wu et al. investigated online ways to control frequency/voltage in multiprocessors with multiple clock domains [16]. Hong et al. [17] created a combined power and performance model for GPUs. In practice, the model could be used to optimize performance under a power constraint. However, the model only addressed the GPU portion of a heterogeneous system. Our work differs from the above by respecting a power constraint in a heterogeneous system, as opposed to minimizing energy or predicting power in a homogeneous system.

Performance prediction is not only important to application execution; it is also important in architectural design. Li et al. used heuristics and hill-climbing to search a processor design space for power-efficient processor configurations for parallel applications [18]. In addition, Lee et al. used regression modeling to search a processor design space for power-efficient architectures [19]. Hsu and Feng [20] developed a model called *Beta* for predicting the effect of DVFS. Finally, several groups have developed an architectural performance model that takes into account the effect of multiple outstanding loads, for predicting performance under DVFS [21]–[23].

Isci et al. optimized simulated processor performance under a chip-level power budget for a workload of simultaneously executing serial SPEC benchmarks [24]. This work made a case for the use of per-core DVFS, a feature rarely available in multicore CPUs. However, the workloads of multiple, simultaneously executing serial applications are a limiting factor; modern processors routinely execute multiple parallel applications. Our system focuses on optimizing performance for one parallel application at a time; this is important because accurate single-application models are a necessary ingredient in multi-application optimization systems.

In between hardware and software, firmware plays an important role in power budgeting. Paul et al. implemented a firmware-based power control system for heterogeneous processors [25]. The system balanced power between a multicore CPU and an on-chip GPU, maximizing performance under a chip-wide power constraint by changing CPU and GPU DVFS states in response to measured application frequency sensitivity. This approach is application-independent and OS-independent, but is limited in power scalability. For example, it is unable to relocate a kernel from the CPU to the GPU or change the number of CPU cores in operation.

C. Power Modeling, Heuristics

Much attention has been paid to modeling only power, especially in server systems. Shen et al. [26] developed a scheme that attributes power consumption to individual tasks and allows for model error correction. Another example is the work of Bertran et. al. [27], who developed a performance counter-based model that can identify power phases.

Work that is somewhat related to ours includes run-time systems to manage power/energy automatically [20], [28], [29]. These systems are heuristic-based and generally do not use formal models.

III. Our Model

Our general approach is to classify kernels into a small number of clusters based on how each kernel's performance scales with changes in available power, then use those clusters to predict power and performance for arbitrary applications and their kernels. Our process is divided into two stages: offline and online. In the offline stage, we characterize the system with a small number of kernels (the training set), group kernels into clusters, and create regression models for each cluster. In the online stage, we use models from the offline stage to predict power and performance for previously unknown kernels and applications (the validation set). We note that the offline stage is conducted only once to characterize a new system, whereas the online stage is applied for each new application. The rest of this section details both the offline and online stages, which Figure 1 depicts in flowchart form.

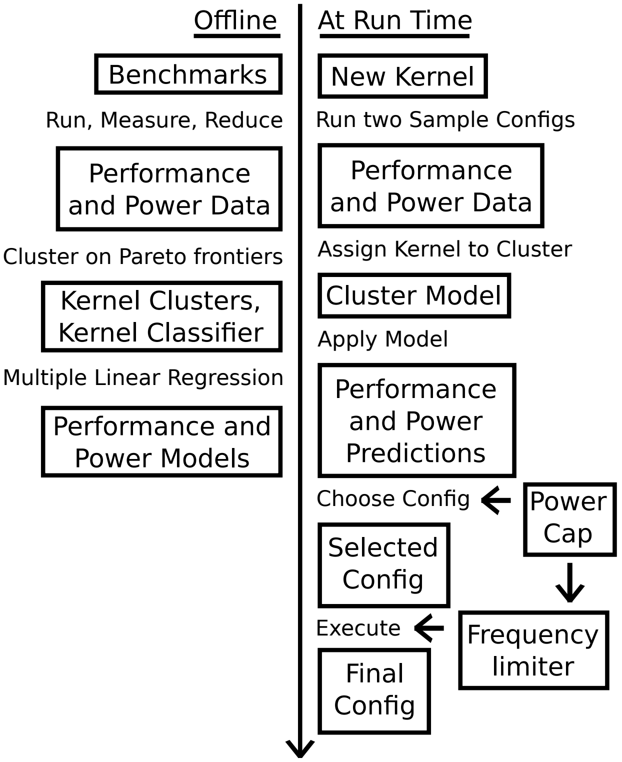

Fig. 1: Flow chart showing offline and online portions of our system.

A. Assumptions

Ideally, a single programming framework would achieve competitive performance on a range of device types, without requiring manual specialization of source code to specific devices. We are unaware of such a framework, so we choose a distinct implementation for each device: OpenMP on the CPU, and OpenCL on the GPU. OpenMP is simpler to use than OpenCL, and we found its performance superior to OpenCL on the CPU for all but two of the 36 kernels we tested. Additionally, there are multiple ongoing efforts to support OpenMP (or similar frameworks) on heterogeneous systems [30]–[32]. Use of such a system would remove the need for multiple implementations of an application.

Even though our applications support both OpenMP and OpenCL, we do not examine hybrid codes. This is because the lack of compiler/runtime support for hybrid codes requires the programmer to split kernel inputs and combine outputs. In many cases, hybrid execution actually decreases overall performance due to load imbalance or increased parallel overhead [13], [31]. Also, even if hybrid execution increases performance, it will strictly lower power-efficiency compared to the best single device, which is of primary concern in power-constrained environments. In the best possible case, hybrid execution will increase performance by a factor of two over the best single device, but will increase power consumption at least as much. Consequently, the benefit of hybrid execution in a power-constrained environment is often much lower than the best case. Similarly, we assume that kernels execute sequentially. All of our benchmark applications fit this condition.

B. Offline Stage: Clustering and Training

Our regression models are based on power and performance profiling data collected on a per-kernel basis from a set of training benchmarks (the training set). In this paper, we use a cross-validation scheme to select training kernels (see Section V-C); however, the training set could be composed of microbenchmarks or a standard benchmark suite. We obtain the profiling data through a combination of performance counters and an on-chip system management microcontroller. We use PAPI [33] to measure CPU counters and use the northbridge performance monitoring unit (PMU) directly for northbridge counters, which track memory-related events. The system management microcontroller provides real-time power estimates for two domains: the CPU cores and the northbridge and GPU together. We integrate the power estimates over time to obtain an average power estimate for each kernel. In addition to the two power domains, we track the following counts for each kernel execution: L2 data cache misses, L1 data cache misses, TLB misses, conditional branch instructions, vector instructions, stalled core cycles, total core cycles, reference cycles, idle FPU cycles, interrupts, and DRAM accesses. All such counts are normalized to one or more of core cycles, reference cycles, and instructions.

Power-performance Pareto frontiers play a key role in our modeling process. We derive Pareto frontiers from power and performance data for each kernel, comparing configurations within and between processors. An example frontier is depicted in Figure 2, showing that using the GPU results in better performance for higher power limits, while the CPU is able to reach lower power limits. Performance is normalized to that of the highest-performing configuration, and is kernel-specific. The distinct levels of GPU performance correspond to distinct GPU P-states. The differences in power are due almost entirely to CPU P-state selection. The line indicates the dependence of attainable performance on available power. It is important to note that with perfect knowledge of the machine and kernel in question, the majority of configurations would never be selected because the configurations along the frontier use less power for the same or greater performance than all other configurations. The configurations on the frontier are listed in Table I. Note that the first GPU configuration is at the GPU's lowest frequency (0.3 GHz), and that this kernel does not benefit from running the GPU at its highest frequency (0.8 GHz). Also, the GPU configurations have varying CPU frequencies; this reflects kernel launch overhead spent in the driver and OpenCL runtime, which run on the CPU.

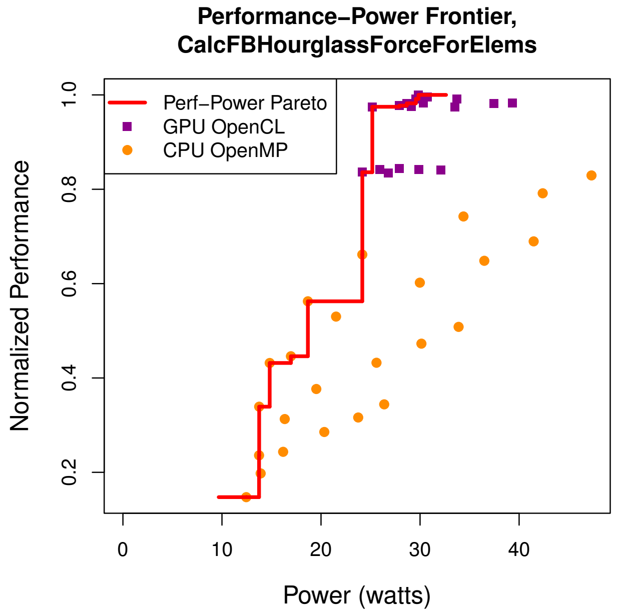

Fig. 2: A power-performance Pareto frontier from LULESH. Table I lists the configurations on the frontier.

Device	GPU f.	Threads	CPU f.	Power	Perf.*
CPU	0.3 GHz	1	1.4 GHz	12.5 w	0.15
CPU	0.3 GHz	2	1.4 GHz	13.7 w	0.24
CPU	0.3 GHz	3	1.4 GHz	13.8 w	0.34
CPU	0.3 GHz	4	1.4 GHz	14.8 w	0.43
CPU	0.3 GHz	3	1.9 GHz	17.0 w	0.45
CPU	0.3 GHz	4	1.9 GHz	18.7 w	0.56
CPU	0.3 GHz	4	2.4 GHz	24.2 w	0.66
GPU	0.3 GHz	1	1.4 GHz	24.2 w	0.84
GPU	0.6 GHz	1	1.4 GHz	25.2 w	0.97
GPU	0.6 GHz	1	1.9 GHz	27.9 w	0.98
GPU	0.6 GHz	1	2.4 GHz	28.7 w	0.98
GPU	0.6 GHz	1	3.3 GHz	29.6 w	0.99
GPU	0.6 GHz	1	3.7 GHz	29.8 w	1.00

TABLE I: Configurations on the power-performance Pareto frontier of the CalcFBHourGlass kernel from LULESH. All configurations for this kernel are shown in Figure 2. *Normalized performance

We use the frontiers to group kernels into clusters, with the key insight being that kernels with similar power and performance scaling behavior will generally have the same configurations on their respective frontiers, arranged in the same order. We first create a kernel dissimilarity matrix by performing pair-wise comparisons of all kernels' frontiers. For

Device	CPU frequency	CPU threads	GPU frequency
CPU	3.7 GHz	4	311 MHz
GPU	3.7 GHz	1	819 MHz

TABLE II: Sample configurations. When an unknown kernel is encountered, we run it with these configurations prior to making power and performance predictions.

each frontier comparison, we first select only the configurations that are present in both frontiers. Then, we compute the Kendall rank correlation coefficient [34] between the orders of the shared configurations within each frontier. The Kendall rank correlation coefficient indicates similarity or dissimilarity between the orders of two lists; if the lists are identical, the coefficient is 1, and if one list is the reverse of the other, the coefficient is -1.

From the resulting dissimilarity matrix, we perform relational clustering via the R Fossil package [35]. This groups the kernels into clusters according to similarities between the order of configurations along the kernels' respective power-performance frontiers. Each cluster contains kernels from at least three of the five benchmark/input combinations. For the benchmarks and kernels we tested, we found empirically that five clusters optimized the predictive ability of our system; using fewer clusters resulted in over-generalized models, and using more clusters resulted in over-specialized models.

It is unreasonable to expect a single regression model to make accurate predictions for all kernels, especially when using only the machine configuration space as input. For example, the kernels we tested show large variance in power consumption; even after selecting the best-performing configuration for each kernel, one kernel uses 19 watts, while another uses 55. Similarly, kernels vary in how performance scales with power consumption. One kernel's best performance is 367 times that of its worst, while another kernel spans a range of only 1.62 in performance. For this reason, we group kernels into clusters, and create power and performance models for each cluster.

In the regression models, we use sample configurations as a frame of reference for mapping performance on one configuration to performance on all other configurations. The sample configurations are listed in Table II. We chose these configurations to match common execution configurations in environments without power constraints. We note that the sample configuration iterations are part of normal application execution. It is important to limit the number of sample configurations; requiring more sample configurations leads to more time spent in configurations that are suboptimal from power or performance perspectives.

The performance models predict how performance scales from a sample configuration on the relevant device. That is, $P_{perf} = (a_1 \cdot x_1 + a_2 \cdot x_2 + ... + a_n \cdot x_n) \cdot S_{perf}$, where P_{perf} is the predicted performance, S_{perf} is the sample configuration performance on the same device, the a_i are the model coefficients, and the x_i are the configuration variables (frequency, number of cores, etc.) and their first-order interactions (i.e. frequency $\cdot$ cores). The power models directly predict power, however: $P_{power} = b_0 + b_1 \cdot x_1 + b_2 \cdot x_2 + ... + b_n \cdot x_n$, where P_{power} is predicted power, and the b_i are model coefficients, with b_0 as the intercept. Performance is generally a nonlinear function of our configuration variables. Our goal in using linear performance and power prediction models is to rank configurations in performance and power in a computationally efficient manner. We find that linear models satisfy this goal. This model formulation has an important consequence; once a new kernel is associated with a cluster, the only new information required to make power and performance predictions is the kernel's performance on the sample configurations.

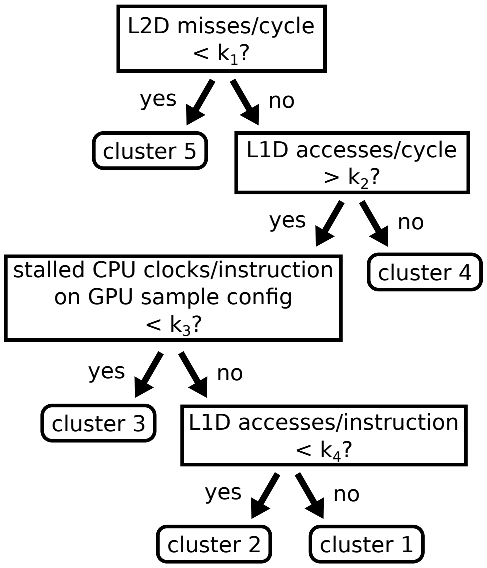

Fig. 3: An example cluster classification tree.

While the models and clusters are trained offline from known kernels, the purpose of our system is to make predictions for new kernels. However, there is asymmetry in information available regarding new kernels and those from the training set; those from the training set have run on all available configurations, while new kernels have only run once on each device. This asymmetry motivates the creation of a system to assign new kernels to existing clusters. For this purpose, we train a classification tree [36] on performance counter and power data from training kernels on the sample configurations. Figure 3 shows an example tree. In this tree, new kernels are classified into four existing clusters based on four normalized performance counter metrics. The constants k_1 to k_4 are determined empirically from the training kernels.

C. Online Stage: Predictions and Scheduling

Before power and performance predictions for a kernel can be made, the kernel must be assigned to one of the trained clusters via the classification tree. The inputs to the tree depend on having run the kernel on the sample configurations, so we use the first two iterations of the kernel to run on the sample configurations, with one iteration on each device (CPU and

GPU). Once the classification tree selects a cluster, we apply the selected cluster's models to predict power and performance for the new kernel at all machine configurations across all available devices.

From the predicted power and performance for all configurations for a new kernel, we derive a predicted Pareto frontier. The resulting frontier allows a scheduler to select specific devices and configurations depending on the scheduling goal at hand. In this paper, we focus on maximizing attainable performance under an imposed power constraint, but the predicted values could be used to select configurations for energy efficiency, energy-delay product, or any other scheduling goal. The use of a predicted Pareto frontier makes our system adaptable to dynamic power constraints, and avoids the need to examine predictions for all configurations when scheduling conditions change.

D. *Profiling Library*

In order to associate power and performance measurements with specific sections of application execution, we create an integrated profiling library. This also allows us to take into account the overheads of data transfer and kernel launch in the case of OpenCL and thread creation and synchronization in the case of OpenMP.

Our library is designed to provide a foundation for dynamic scheduling. A history of performance and power measurements is made accessible to the application or runtime, which facilitates online selections of device and configuration for a given kernel.

As our benchmarks are implemented in both OpenMP and OpenCL, we currently instrument source code by hand with profiling pragmas, which a source preprocessor then converts into profiling library calls. At run time, the application invokes the library to record samples of performance counters and power measurements to resident data structures, which are written to disk after the application completes. Such instrumentation could also be performed automatically by a compiler [37], or effected through dynamic library interposition, wrapping OpenCL API calls.

IV. Experimental Setup

A. *Test System*

The Trinity heterogeneous processor (APU) forms the basis of our test machine for this paper. It contains two out-of-order dual-core CPU compute units (also known as PileDriver modules or CUs) and a GPU. The cores within a dual-core module share the front-end and floating point units along with a 2M L2 cache. The CPU cores share a power plane, while the GPU is on a separate power plane. The GPU consists of 384 Radeon$^{\text{TM}}$ cores, each capable of one single-precision fused multiply-add computation (FMAC) operation per cycle. The GPU is organized as six SIMD units, each containing 16 four-way VLIW processing units. The memory controller is shared between the CPU and the GPU. More details on the Trinity processor are available [2].

The Trinity A10-5800k processor supports six software-visible DVFS states, or P-states, ranging from 1.4 to 3.7 GHz. P-states can be assigned per CU. However, since all compute units on the chip share a voltage plane, the voltage across all compute units is set by the CU with maximum frequency. Software-visible P-states are managed either by the OS through the Advanced Configuration and Power Interface (ACPI) specification [38] or by the hardware. The processor also supports higher frequencies, but we do not consider them, as we require direct control over CPU P-states.

The GPU has a power plane separate from that of the CPU in which voltage and frequency are controlled independently. For the rest of this paper, we consider three effective GPU P-states, at 311 MHz, 649 MHz, and 819 MHz, respectively.

B. *Benchmarks and Tools*

We evaluate our model on a suite of exascale proxy application benchmarks. LULESH, CoMD, and SMC were originally developed by the U.S. Department of Energy and subsequently ported to OpenCL/OpenMP by AMD and the Department of Energy. These benchmarks are representative of the class of workloads expected in an exascale supercomputer. LU is from the Rodinia benchmark suite [39], chosen for its relevance to the LINPACK benchmark [40] commonly used to rank supercomputer performance.

With the exception of LU, all of the benchmarks in our study are composed of multiple kernels. Our OpenCL version of LULESH [41], a shock hydrodynamics benchmark, contains 20 significant kernels. CoMD [42], a molecular dynamics benchmark, contains 7 significant kernels, while SMC, a combustion benchmark, contains 8. In total, our benchmarks contain 36 kernels. Running benchmarks with various inputs increases the variance in kernel behavior, and increases our benchmark/input combination count to 65.

We use the following tools in data acquisition, compilation, and benchmark execution: R 3.0.1, GCC 4.7, AMD OpenCL APP SDK 2.8.1, PAPI 5.2.0, and Ubuntu 12.04.3 with Linux 3.11.

C. *Overheads*

Our system is designed to maximize performance, so we are careful to reduce its offline and online overheads. In pursuit of this goal, our training kernels require less than two hours to run, and our offline data reduction and model construction process requires about ten minutes.

Data acquisition involves recording performance counter and power measurements at the start and finish of each kernel execution, which we find to add less than 50 microseconds. Our power measurement method involves sampling and accumulating an on-chip power estimate at 1 kHz, which incurs overhead of less than 10% in all cases. However, this method of power measurement is not necessary on architectures equipped with hardware- or firmware-based energy accumulators.

In the online case, the overheads of our system include tree classification and model application for a new kernel. Both of these rely on having run the kernel on the two sample configurations. Application of the tree classifier requires time on the order of the depth of the tree, and model application requires a simple matrix-vector product of the configuration space with the model coefficients. The online overheads are negligible because they are encountered only once per kernel; after the second iteration of a kernel, its configuration is fixed.

V. Experimental Evaluation and Results

A. *Methods for Limiting Power*

We compare our model against various state-of-the-practice methods for power limiting based on Intel's RAPL [1]. RAPL dynamically adjusts CPU core frequency to meet an imposed power constraint. Our test system is not equipped with RAPL, so we simulate its behavior. In addition to simulating frequency-limiting on the CPU, we also simulate it on the GPU.

We evaluate two basic frequency-limiting methods in detail, with one focused on the CPU (referred to as CPU+FL) and one focused on the GPU (GPU+FL). For the CPU, we enable all available cores, set the GPU to minimum frequency, and let the frequency limiter set CPU P-states in response to power constraints. For the GPU, we initially set CPU frequency to its minimum and GPU frequency to its maximum during kernel execution, then let the frequency limiter control GPU P-states in response to power constraints. If there is power headroom after setting the GPU P-state, we increase the CPU frequency as much as is possible without violating the power constraint.

We also evaluate the combination of our model with a frequency-limiting system, referred to as Model+FL.

B. *Evaluation Metrics*

We evaluate all of the aforementioned power-limiting methods by comparing them against an oracle with perfect knowledge. For each method, we compare configurations selected by the method with those selected by the oracle under various power constraints. The specific power constraints correspond to the power consumption levels at the configurations on the oracle-selected power-performance frontier for each kernel under comparison. We compare the method-selected configurations with the oracle-selected configurations by computing power and performance differences in two categories: when the method meets the power limit, and when the method does not meet power limit. For the remainder of the paper, we refer to these categories as under-limit and over-limit, respectively. A method may fail to meet a power constraint by selecting a configuration that cannot be sufficiently scaled via DVFS.

C. *Cross-Validation*

To verify that our model makes accurate predictions for new kernels, we perform leave-one-out cross-validation [43, Chapter 7] for the entire process across individual benchmarks. That is, for each benchmark, we form a training set that consists of kernels from other benchmarks. From kernels in the training set, we compute clusters, cluster models, and a classification tree, then apply them to kernels from the benchmark under validation. In doing so, we ensure that the model is always applied to as-yet-unseen benchmarks.

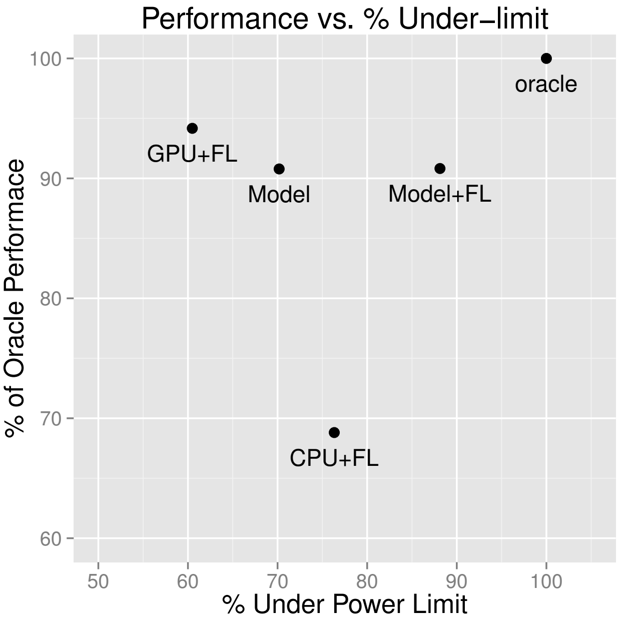

Fig. 4: Comparisons of our model and state-of-the-practice methods, normalized to an oracle.

D. *Results: Comparison of Power-limiting Methods*

Table III compares all the previously discussed methods (CPU+FL, GPU+FL, Model, and Model+FL) to an oracle with perfect knowledge of the test system and benchmark kernels. When combined with frequency-limiting, our model has a clear advantage over the other methods. Primarily, Model+FL both achieves near-optimal performance and meets power constraints more often than the other methods. In over-limit cases, Model+FL uses the least power, while still providing 54% more performance than the oracle.

Figure 4 compares each power-limiting method to an oracle in two metrics: percentage of time meeting the tested power constraints, and the percentage of optimal performance achieved in those cases. When combined with frequency-limiting, our model is closest to the oracle when considering both metrics together. GPU+FL achieves higher performance, but it meets power constraints only 60% of the time, whereas our model achieves high performance while meeting power constraints 88% of the time. In the absence of power constraints, running all kernels on the GPU yields high performance. In a power-constrained system, however, more care must be taken to select appropriate devices and configurations.

The values in our method comparisons are averaged across all kernels that compose each benchmark, weighted by how much of the benchmark time is spent in each kernel. Figure 5 compares the methods by their performance in under-limit cases. Model+FL has a clear advantage over the other methods in maintaining high performance across the set of

		Under-limit		Over-limit	
Method	% Under-limit	% Oracle Perf.	% Oracle Power	% Oracle Power	% Oracle Perf.
Model	70	91	94	112	139
Model+FL	88	91	91	106	154
GPU+FL	60	94	95	137	1723
CPU+FL	76	69	94	111	216

TABLE III: Comparisons of our model and state-of-the-practice methods, normalized to an oracle.

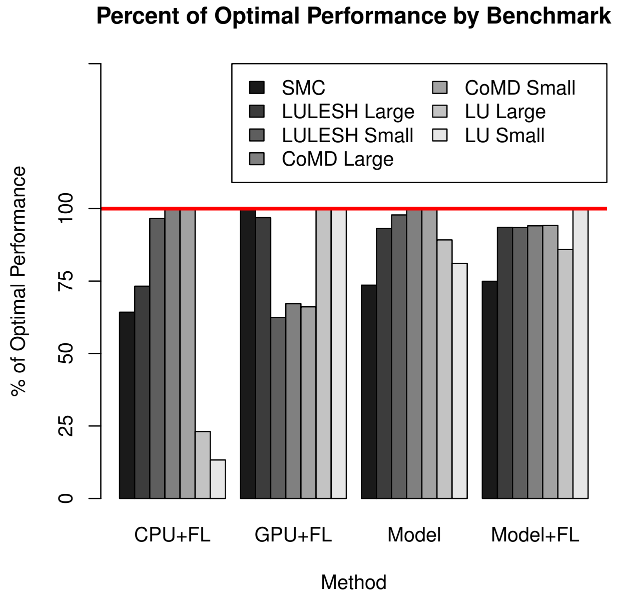

Fig. 5: Performance vs oracle in under-limit cases.

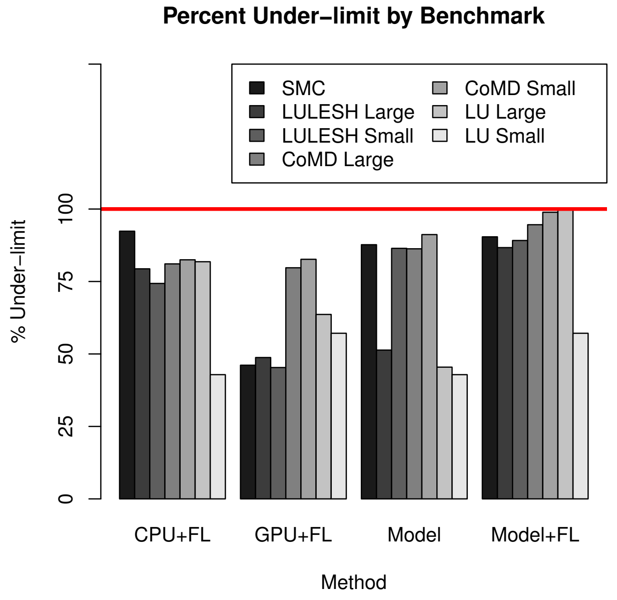

Fig. 6: Percent of cases under-limit.

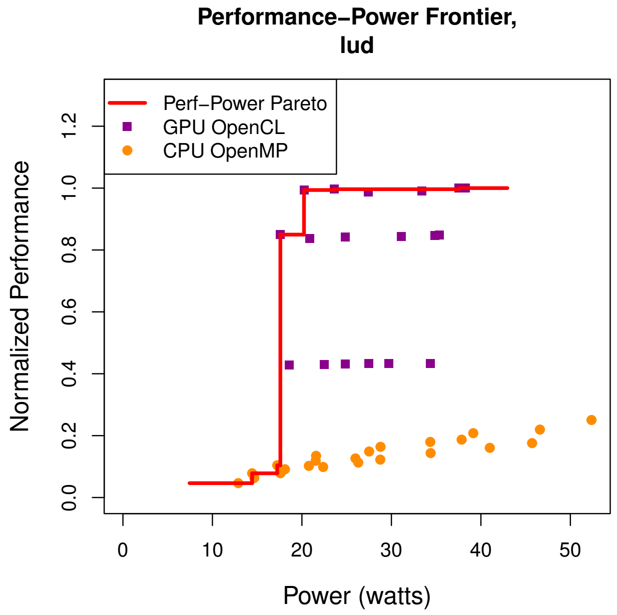

Fig. 7: Power-performance frontier of LU Small.

benchmarks. Over all benchmarks, Model+FL achieves a minimum of 74.9% of oracle performance, while the state-of-the-practice methods, CPU+FL and GPU+FL, achieve only 13.3% and 62.4% of oracle performance for their respective worst-case benchmarks. Similarly, Model+FL outperforms the other methods in respecting power constraints. Figure 6 indicates that Model+FL meets power constraints more often than all other methods for all benchmark/input combinations except SMC, where CPU+FL meets constraints more often, and LU Small, where Model+FL ties with GPU+FL at 57.1%. LU Small (Figure 7) is representative of multiple challenges encountered in attempting to meet power constraints. First, when available power changes from 17.2 watts to 17.6 watts, achievable normalized performance changes from 10.4% to 89.0% as a result of switching from the CPU to the GPU. Second, all configurations with three or four cores use more than 17.2 watts on LU Small, which requires the power-limiting method to not only select the correct device, but also select the correct number of cores, even when combined with frequency-limiting. These challenges hinder all of the tested methods from meeting power constraints in multiple kernels, including LU Small. Model+FL and GPU+FL both select the GPU at all power constraints, leaving them unable to meet

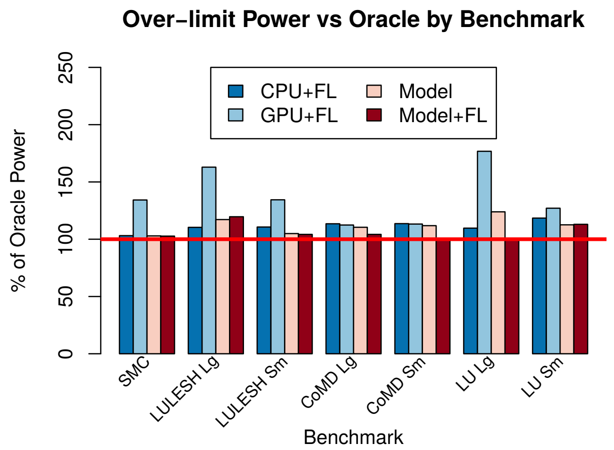

Fig. 8: Power vs oracle in over-limit cases.

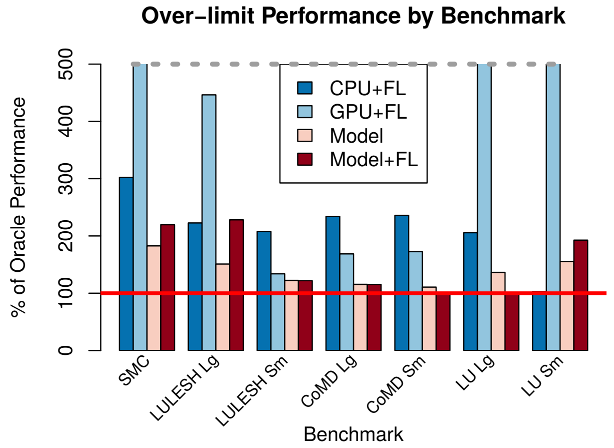

Fig. 9: Performance vs oracle in over-limit cases. It is possible to exceed oracle performance only when also exceeding oracle power. The clipped values from GPU+FL are 1218% for SMC, 9297% for LU Large, and 627% for LU Small.

constraints under 17.6 watts. CPU+FL always runs on four threads, thus violating the lower constraints.

Figure 8 details cases in which the methods do not meet power constraints. In these cases, Model+FL uses less power than the other methods for all of the benchmark/input combinations except LULESH Large, where CPU+FL uses 110% of oracle power vs 120% for Model+FL, and LU Small, where Model ties with Model+FL at 113%. Methods using frequency-limiting are not able to meet some power constraints because they select configurations with the inappropriate device (i.e. GPU instead of CPU), or they select configurations with too many cores.

Figure 9 shows the impact of exceeding power constraints for each method and benchmark. GPU+FL violates power constraints 40% of the time, which frequently results in higher power and performance than an oracle at the same power constraint. In the most extreme example, GPU+FL achieves 92 times better performance than an oracle on LU Large, but exceeds the power constraints by an average of 77%. Model+FL experiences a similar effect, but the magnitude is limited to a factor of 2.3 times oracle performance at an average of 20% over the power constraint for LULESH Small.

VI. Future Work

A few features would significantly improve the utility of our proposed model. One idea is to apply a variance-stabilizing transformation to model inputs and outputs during the training phase. This would give less weight to both very small and very large fitted model values [19].

Taking variance into account when predicting best configurations could also improve model accuracy when applied to new applications. If the confidence interval for a prediction is large, it may be wise to choose another configuration with smaller confidence interval and lower expected performance.

Few hardware features are exposed that directly affect power consumption, but one that we did not yet include in our machine configuration space is opportunistic overclocking. This feature allows the CPU to increase its frequency beyond user-selectable levels, but only when there is enough thermal headroom; if the chip is too hot, such frequency boosting will not engage.

Our system does not automatically differentiate between invocations of the same kernel with distinct data inputs or input sizes. For the purposes of this paper, we manually identified multiple input sizes, but an OpenCL runtime could use readily available information to accomplish this. For identifying use in distinct contexts, the runtime could use call stacks to differentiate between invocations of the same kernel from distinct points in the application [28], [44].

Processor power is a major component of system power, but not the only component. However, processor and memory power make up the majority of system power. Network power also contributes significantly to total system power [45] , but memory power is more volatile than network power and thus more amenable to our power management techniques. In future work, we intend to account for memory power in addition to processor power.

VII. Conclusions

In this paper, we demonstrate a power/performance model that selects efficient configurations under imposed power constraints. Our model is trained offline by characterizing the machine with a small set of benchmarks. Importantly, our model handles a much larger configuration space compared to prior work, including multiple devices and parallel implementations.

We show that our model is practical through a significant experimental study. Specifically, our model accurately predicts power and performance for a set of 36 kernels from a range of applications, and maintains 91% of optimal performance while meeting power constraints 88% of the time.

VIII. Acknowledgements

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. 1216829. This work was partially performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

References

[1] H. David, E. Gorbatov, U. Hanebutte, R. Khanna, and C. Le, "RAPL: Memory power estimation and capping," in *ACM/IEEE International Symposium on Low Power Electronics and Design*. ACM, 2010, pp. 189–194.

[2] S. Nussbaum, "Amd trinity fusion apu," in *Proceedings of the Hot Chips: A Symposium on High Performance Chips*, 2012.

[3] M. Curtis-Maury, J. Dzierwa, C. Antonopoulos, and D. Nikolopoulos, "Online power-performance adaptation of multithreaded programs using hardware event-based prediction," in *ACM International Conference on Supercomputing*, vol. 28, 2006, pp. 157–166.

[4] M. Curtis-Maury, K. Singh, S. McKee, F. Blagojevic, D. Nikolopoulos, B. De Supinski, and M. Schulz, "Identifying energy-efficient concurrency levels using machine learning," in *IEEE International Conference on Cluster Computing*. IEEE, 2007, pp. 488–495.

[5] M. Curtis-Maury, J. Dzierwa, C. Antonopoulos, and D. Nikolopoulos, "Online strategies for high-performance power-aware thread execution on emerging multiprocessors," in *IEEE International Parallel and Distributed Processing Symposium*, 2006.

[6] M. Curtis-Maury, A. Shah, F. Blagojevic, D. Nikolopoulos, B. de Supinski, and M. Schulz, "Prediction models for multi-dimensional power-performance optimization on many cores," in *International Conference on Parallel Architectures and Compilation Techniques*, 2008.

[7] R. Cochran, C. Hankendi, A. K. Coskun, and S. Reda, "Pack & cap: adaptive dvfs and thread packing under power caps," in *Proceedings of the 44th annual IEEE/ACM international symposium on microarchitecture*. ACM, 2011, pp. 175–185.

[8] S. Reda, R. Cochran, and A. K. Coskun, "Adaptive power capping for servers with multithreaded workloads," *IEEE Micro*, vol. 32, no. 5, pp. 0064–75, 2012.

[9] B. C. Lee, D. M. Brooks, B. R. de Supinski, M. Schulz, K. Singh, and S. A. McKee, "Methods of inference and learning for performance modeling of parallel applications," in *Proceedings of the 12th ACM SIGPLAN symposium on Principles and practice of parallel programming*. ACM, 2007, pp. 249–258.

[10] Z. Wang and M. F. O'Boyle, "Mapping parallelism to multi-cores: a machine learning based approach," *ACM Sigplan Notices*, vol. 44, no. 4, pp. 75–84, 2009.

[11] A. Karami, S. A. Mirsoleimani, and F. Khunjush, "A statistical performance prediction model for opencl kernels on nvidia gpus," in *Computer Architecture and Digital Systems (CADS), 2013 17th CSI International Symposium on*. IEEE, 2013, pp. 15–22.

[12] A. Kerr, E. Anger, G. Hendry, and S. Yalamanchili, "Eiger: A framework for the automated synthesis of statistical performance models," in *High Performance Computing (HiPC), 2012 19th International Conference on*. IEEE, 2012, pp. 1–6.

[13] C.-K. Luk, S. Hong, and H. Kim, "Qilin: exploiting parallelism on heterogeneous multiprocessors with adaptive mapping," in *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*. IEEE, 2009, pp. 45–55.

[14] D. Li, B. de Supinski, M. Schulz, K. Cameron, and D. Nikolopoulos, "Hybrid MPI/OpenMP power-aware computing," in *IEEE International Parallel and Distributed Processing Symposium*, 2010, pp. 1–12.

[15] R. Springer, D. Lowenthal, B. Rountree, and V. Freeh, "Minimizing execution time in MPI programs on an energy-constrained, power-scalable cluster," in *Symposium on Principles and Practice of Parallel Programming*, 2006, pp. 230–238.

[16] Q. Wu, P. Juang, M. Martonosi, and D. W. Clark, "Formal online methods for voltage/frequency control in multiple clock domain microprocessors," *SIGARCH Comput. Archit. News*, vol. 32, no. 5, pp. 248–259, Oct. 2004.

[17] S. Hong and H. Kim, "An integrated gpu power and performance model," in *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3. ACM, 2010, pp. 280–289.

[18] J. Li and J. Martinez, "Dynamic power-performance adaptation of parallel computation on chip multiprocessors," in *IEEE International Conference High-Performance Computer Architecture*, 2006, pp. 77–87.

[19] B. Lee and D. Brooks, "Accurate and efficient regression modeling for microarchitectural performance and power prediction," in *ACM SIGOPS Operating Systems Review*, vol. 40, no. 5. ACM, 2006, pp. 185–194.

[20] C.-H. Hsu and W. Feng, "Effective dynamic-voltage scaling through CPU-boundedness detection," in *Fourth IEEE/ACM Workshop on Power-Aware Computing Systems*, Dec. 2004.

[21] B. Rountree, D. Lowenthal, M. Schulz, and B. De Supinski, "Practical performance prediction under dynamic voltage frequency scaling," in *International Green Computing Conference*, Jul 2011.

[22] G. Keramidas, V. Spiliopoulos, and S. Kaxiras, "Interval-Based Models for Run-Time DVFS Orchestration in Superscalar Processors," in *International Conference on Computing Frontiers*, 2010.

[23] S. Eyerman and L. Eeckhout, "A Counter Architecture for Online DVFS Profitability Estimation," *IEEE Transactions on Computers*, 2010.

[24] C. Isci, A. Buyuktosunoglu, C. Cher, P. Bose, and M. Martonosi, "An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget," in *IEEE/ACM International Symposium on Microarchitecture*, 2006, pp. 347–358.

[25] I. Paul, V. Ravi, S. Manne, M. Arora, and S. Yalamanchili, "Coordinated energy management in heterogeneous processors," in *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2013, p. 59.

[26] K. Shen, A. Shriraman, S. Dwarkadas, and X. Zhang, "Power and energy containers for multicore servers," in *ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, Mar. 2013.

[27] R. Bertran, M. Gonzalez, X. Martorell, N. Navarro, and E. Ayguade, "Decomposable and responsive power models for multicore processors using performance counters," in *International Conference on Supercomputing*, Jun. 2010.

[28] B. Rountree, D. K. Lowenthal, B. de Supinski, M. Schulz, and V. W. Freeh, "Adagio: Making DVS practical for complex HPC applications," in *International Conference on Supercomputing*, Yorktown Heights, N.Y., USA, Jun. 2009.

[29] K. Cameron, X. Feng, and R. Ge, "Performance-constrained, distributed DVS scheduling for scientific applications on power-aware clusters," in *Supercomputing*, Nov. 2005.

[30] J. C. Beyer, E. J. Stotzer, A. Hart, and B. R. de Supinski, "Openmp for accelerators," in *OpenMP in the Petascale Era*. Springer, 2011.

[31] T. R. Scogland, B. Rountree, W.-c. Feng, and B. R. de Supinski, "Heterogeneous task scheduling for accelerated openmp," in *Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*. IEEE, 2012, pp. 144–155.

[32] S. Wienke, P. Springer, C. Terboven, and D. an Mey, "Openacc–first experiences with real-world applications," in *Euro-Par 2012 Parallel Processing*. Springer, 2012, pp. 859–870.

[33] S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci, "A portable programming interface for performance evaluation on modern processors," *International Journal of High Performance Computing Applications*, vol. 14, no. 3, pp. 189–204, 2000.

[34] M. G. Kendall, "A new measure of rank correlation," *Biometrika*, vol. 30, no. 1/2, pp. 81–93, 1938.

[35] M. J. Vavrek, "fossil: palaeoecological and palaeogeographical analysis tools," *Palaeontologia Electronica*, vol. 14, no. 1, p. 1T, 2011, r package version 0.3.0.

[36] L. B. J. F. R. Olshen and C. J. Stone, "Classification and regression trees," *Wadsworth International Group*, 1984.

[37] B. Mohr, A. Malony, S. Shende, F. Wolf *et al.*, *Towards a performance tool interface for OpenMP: An approach based on directive rewriting*. Forschungszentrum, Zentralinst. für Angewandte Mathematik, 2001.

[38] *Advanced Configuration and Power Interface (ACPI) Specification*, Std. [Online]. Available: http://www.acpi.info/spec.htm

[39] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*. IEEE, 2009, pp. 44–54.

[40] J. J. Dongarra, *LINPACK users' guide*. Siam, 1979, no. 8.

[41] I. Karlin, “Lulesh programming model and performance ports overview,” Lawrence Livermore National Laboratory (LLNL), Livermore, CA, Tech. Rep., 2012.

[42] (2013) Comd. [Online]. Available: https://github.com/exmatex/CoMD

[43] T. Hastie, R. Tibshirani, and J. J. H. Friedman, *The elements of statistical learning*. Springer New York, 2001, vol. 1.

[44] M. C. Huang, J. Renau, and J. Torrellas, “Positional adaptation of processors: application to energy reduction,” in *Computer Architecture, 2003. Proceedings. 30th Annual International Symposium on*. IEEE, 2003, pp. 157–168.

[45] D. Economou, S. Rivoire, C. Kozyrakis, and P. Ranganathan, “Full-system power analysis and modeling for server environments.” International Symposium on Computer Architecture-IEEE, 2006.