CLIP: Cluster-Level Intelligent Power Coordination for Power-Bounded Systems

Conference Paper · September 2017

DOI: 10.1109/CLUSTER.2017.98

Pengfei Zou
Clemson University
18 PUBLICATIONS **284** CITATIONS
SEE PROFILE

Tyler Allen
Clemson University
9 PUBLICATIONS **52** CITATIONS
SEE PROFILE

Claude Harrison Davis IV
Los Alamos National Laboratory
1 PUBLICATION **12** CITATIONS
SEE PROFILE

Xizhou Feng
Clemson University
37 PUBLICATIONS **2,124** CITATIONS
SEE PROFILE

CLIP: Cluster-Level Intelligent Power Coordination for Power-Bounded Systems

Pengfei Zou, Tyler Allen, Claude H. Davis IV, Xizhou Feng, and Rong Ge
School of Computing, Clemson University
{pzou, tnallen, clauded, xizhouf, rge}@clemson.edu

***Abstract*—High performance computing systems will need to operate with certain power budgets while maximizing performance in the exascale era. Such systems are built with power aware components, whose collective peak power may exceed the specified power budget. Cluster level power bounded computing addresses this power challenge by coordinating power among components within compute nodes and further adjusting the number of participating nodes. It offers more space to increase system performance by utilizing the available power budget more efficiently within and across the nodes.**

In this paper, we present the design of a hierarchical multi-dimensional power aware allocation framework, CLIP, for power bounded parallel computing on multicore-based computer clusters. The framework satisfies the specified power bound by managing the power distribution among nodes at the cluster level, and among sockets, cores and NUMA memory modules at the node level. The power allocation is enforced with multiple complementary power management techniques, including memory power level setting, thread concurrency throttling, and core-thread affinity. We present an application characterization method based on applications' scalability and an associated performance model, which can accurately determine the optimal number of participating compute nodes and components, and their power distribution for given applications. Experimental results on a Haswell-based computer cluster show that the proposed scheduler outperforms compared methods by over 20% on average for various power budgets.

***Index Terms*—Power-bounded computing, resource coordination, performance analysis, multicore computing, cluster.**

I. Introduction

Power has become a critical constraint for the evolution of large scale High-Performance Computing (HPC) systems and commercial data centers. This constraint spans almost every level of computing technologies, from IC chips all the way up to data centers. Power constraints can be induced by physical barriers, technical difficulties, and economical burdens [12, 23]. Resultantly, future supercomputers must operate more efficiently and intelligently under strict power budgets. For example, the U.S. Department of Energy sets the power budget for exascale computing as 20 megawatts [27]. Such reality forces the HPC community to transform its goal from maximizing performance without power limit to improving performance within similar power budgets. The increasing demands for computing and memory performance from scientific and big data applications makes the problem even more challenging.

Optimally managing power for HPC workloads requires an intelligent strategy to control the number of participating nodes and allocate the available power budget to different subsystems (CPU, uncore and memory) within nodes. Inappropriately assigning nodes can either cause inefficient utilization of the available power or lead to subsystems running at ineffective power levels, thereby delivering inferior performance. For example, if fewer nodes are assigned, each node gets excessive power budget than demand and applications' parallelism can't be fully explored at the cluster level. Contrarily, if more nodes are assigned, each node receives insufficient power to coordinate the constituent components at their optimal states, leading to significant performance degradation. System power management is challenging, requiring careful balance between the cluster, node, and component levels to avoid power waste and performance degradation.

Maximizing application performance under limited power budgets should explore applications' characteristics [40] and be application-aware [15]. For example, node-level concurrency varies significantly with applications; simply using the same CPU and memory power budgets cannot efficiently translate the supplied power to performance for every application. Figure 1 shows that application-aware power distribution and resource allocation on a single node can improve the performance of NPB-SP by up to 75%. Application-aware strategies would have a larger impact at the cluster level. The key to such strategies lies in the accurate application characterization, and associated resource concurrency configuration and power distribution.

In this paper, we study the problem of power-bounded computing on multicore-based systems and present the Cluster Level Intelligent Power (CLIP) coordination framework. CLIP employs application-aware power bounded scheduling for parallel applications on clusters built of NUMA multicore nodes. It characterizes the scalability of parallel applications and their power demands, and accordingly recommends the optimal application execution configuration and power distribution. The framework implementation is hierarchical and consists of two levels: the cluster level determines the number of nodes and the power budget for each node; the node level selectively activates the CPU cores and distributes the available power budget to the CPU and memory within nodes. The framework uses light-weight off-line profiling for application characterization, and classifies workloads into three categories. It delivers desirable performance and meets the power budget with four steps:

1) Identify the number of participating nodes based on

2168-9253/17 $31.00 © 2017 IEEE
DOI 10.1109/CLUSTER.2017.98

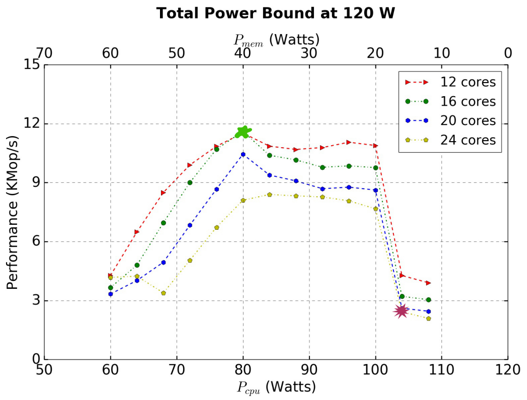


Fig. 1: Performance impacts of resource coordination for a power budget of 120 Watts. This figure reveals significant performance variations with different CPU-memory power allocations and CPU core assignments.

application scalability and power demand on each node.

2) Allocate a per-node power budget between CPU and DRAM based on the application's features.
3) Choose core and memory affinity based on application memory access intensity.
4) Identify the optimal number of active cores based on the application's scalability at the node level.

Overall, the major findings and contributions of this work include:

- We show that power-aware hardware and workload execution management improves both performance and power efficiency for power-constrained systems.
- We propose an application-aware power coordination method, which comprises application characterization and performance modeling. This method can identify a (near) optimal configuration without exhaustively searching the configuration space.
- We implement the power coordination framework and evaluate it on real systems with multiple applications. Experimental results show the framework performs close to the optimal solution under various power budgets.
- We present several findings and insights on concurrency configurations for high performance power-bounded computing.

II. Application Scalability and Impact of Power Budget

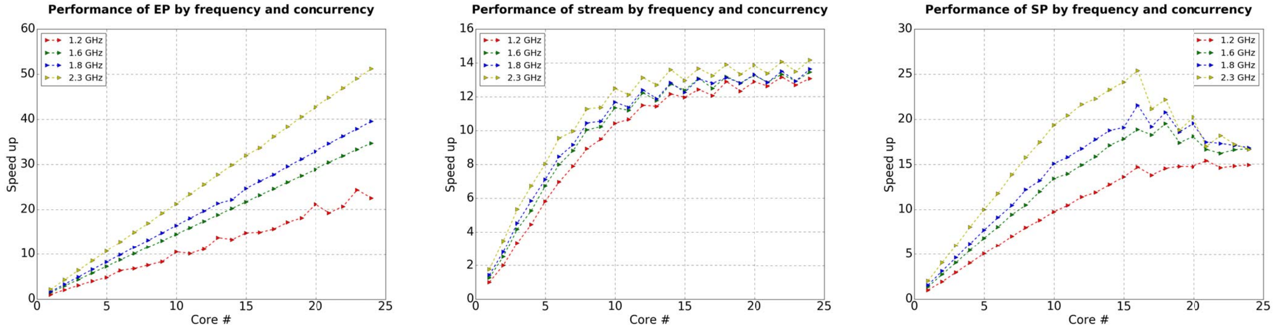
Application performance is dependent on the usage and optimization of concurrent computational resources. HPC system resources are rapidly increasing in multiple dimensions including the number of nodes, the number of cores per processor, and the size of shared memory. While today's large-scale systems have been intensively exploited with parallel computing technologies, workloads can not fully use the underlying hardware at scale due to algorithm design, workload partition, and data movement and communication. Furthermore, power capping causes performance degradations when the power budget is insufficient to support the concurrency configuration.

Scalability describes how application performance changes on parallel computing systems. It is measured with speedup $S(n) = \frac{\text{perf}(n)}{\text{perf}(1)}$ with the number of utilized cores n. Similarly, scalability can describe how application performance changes with processor's speed and be measured with $S(\text{freq}) = \frac{\text{perf(freq)}}{\text{perf}(f_{\text{lowest}})}$. Here $freq$ is the processor frequency. Empirically, $S(\text{freq}) \propto \text{f}req$ holds for most applications, while $S(n) \propto n$ is true only for ideal or embarrassingly parallel applications.

Figure 2 illustrates how applications perform with various processor frequency and processor count on a node with 24 cores. There are three types of scalability trends on parallel architectures, which we denote as *linear*, *logarithmic*, and *parabolic*. As the number of cores increases, application performance increases with both *linear* and *logarithmic* trends. The difference is that the growth rate is constant with the *linear* trend and but reduces with the *logarithmic* trend. The *parabolic* trend is much different where performance decreases after a certain number of cores and using all cores could deliver a significantly degraded performance. We observe that there is an inflection point in each of the non-linear curves in Figures 2b and 2c, which separates the scalability trends into two segments. Both *logarithmic* and *parabolic* can be approximated by a piecewise model, i.e., a linear piecewise segment ($S(n) \propto n\,(n \leq NP) \mid NP$ is the inflection point), and the remaining segment.

A power budget imposed on the system has different performance impacts on these three types of applications. Figure 3 shows the performances of EP, Stream, and SP benchmarks under various power budgets. For a linear application like EP in Figure 3a, its performance is the best when all cores participate in execution. Thus, we do not consider decreasing the concurrency unless the power budget is lower than the the lower bound of the acceptable power range [15], which is not recommended for running applications. For logarithmic applications, the optimal concurrency decreases with the power budget, as shown in Figure 3b. Using less cores could significantly improve the performance of these applications if the power budget is acceptable yet very limited. For parabolic applications, introducing a power budget exacerbates the performance loss of the all-core configuration, as seen in Figure 3c. Because of this behavior, coordinating the concurrency is even more important for parabolic applications for performance improvement. From these results, we can conclude that coordinating thread concurrency at the node level is necessary.

We must also consider distributing the power budget at the cluster level in addition to the node level as the total power budget is imposed on the entire system across the nodes. Concurrency configuration under different power budgets is essential at the node level, and this is the foundation to obtaining the optimal performance at the cluster level. Power coordination between main components has a great performance impact on the entire system [15]. If the node power budget



(a) (b) (c)

Fig. 2: Scalability trends of linear (2a), logarithmic (2b), and parabolic (2c). The performance of linear applications increases linearly with concurrency and processor frequency. The performance of logarithmic applications increases linearly until an inflection point, after which performance growth drops. The performance of parabolic applications increases linearly when concurrency is less than the global maximum. Beyond the global maximum, increasing concurrency causes performance degradation.

is less than the lower bound of the acceptable power range, performance decreases significantly and the performance loss can outweigh the gain on the power savings. Contrarily, if an excessive node power budget is allocated, fewer nodes can be activated to execute applications and exploit algorithm parallelism. We must consider resource allocation at the cluster level in addition to the node level, and decide how to best distribute the power budget to the needed nodes in the cluster.

III. Methodology

The CLIP framework consists of two levels: the cluster level and the node level. At the *node level*, we select the resource coordination configuration and predict the efficient power range. The selection of the node level configuration integrates the classification of applications, the selection of the number of threads, and their mapping and available power range on the node. The *cluster level* handles power and node allocation and coordination. Figures 4 and 5 show the details of the two levels in the CLIP framework.

A. Node-Level Application-Aware Configuration Selection

The goal of node level configuration selection is to estimate the performance of hybrid MPI/OpenMP applications on the node level and offer prerequisite knowledge to support for cluster level power coordination. The configuration specifies how many OpenMP threads to run, how to map these threads across NUMA architecture, and how to allocate the power among the main computer components (CPU and DRAM) under various power budgets. With knowledge of the correlation between performance and configuration under multiple power budgets, the cluster can determine the number of nodes and coordinate the power across available nodes from the total cluster power budget.

Determining the number of threads and their socket affinity is the key to finding the optimal configuration under power budget. As demonstrated in Section II, there are three different workload scalability trends due to their various scalable abilities: *linear*, *logarithmic*, and *parabolic*. It is hard to achieve high accuracy by simply deriving a linear regression model to describe the relationship of scalability to the number of threads as in [25, 6]. The main issue is that hardware evolution causes the old methods to lose precision. In order to cover the diversity of workload scalability in modern hardware environments, we devise the performance prediction model in two steps. The first step employs the classification model to predict the scalability trend as one of the three types listed previously. The second step trains each type of workload independently and infers the corresponding function.

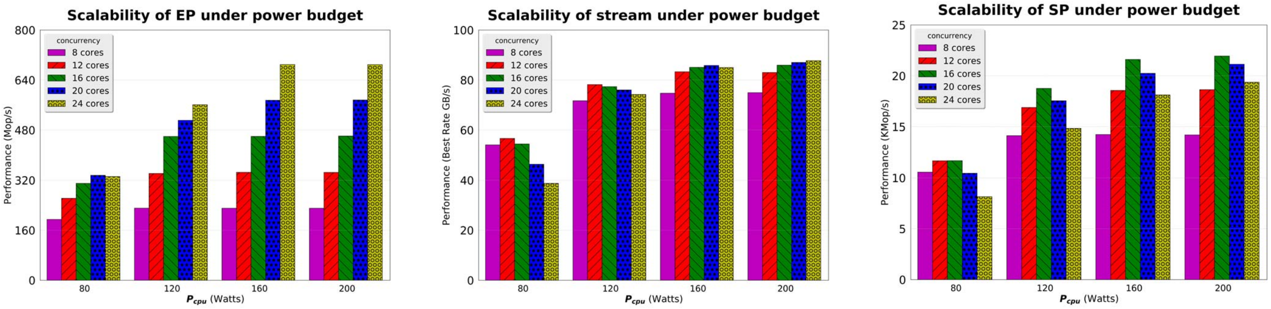
1) Scalability Trend Classification: The classification model predicts the scalability trend by using input from the sample configurations in the profiling stage. We simply compare the performance under two profile stages: Perf_{all} and $\text{Perf}_{\text{half}}$ denoting the performance with all and half of the available cores respectively. By collecting multiple benchmarks, we classify the applications with $\frac{\text{Perf}_{\text{half}}}{\text{Perf}_{\text{all}}} < 0.7$ as linear type; the applications with $\frac{\text{Perf}_{\text{half}}}{\text{Perf}_{\text{all}}} \geq 0.7$ & $\frac{\text{Perf}_{\text{half}}}{\text{Perf}_{\text{all}}} < 1$ as logarithmic type and applications with $\frac{\text{Perf}_{\text{half}}}{\text{Perf}_{\text{all}}} \geq 1$ as parabolic type.

2) Performance Prediction Model:

a) Linear Type: To estimate the performance for linear type applications, we model the run time of the target configuration $Time_t$ as a linear function:

$$Time_t = \sum_{i=1}^{m} (Time_i \cdot \alpha_{(t,i)}) + \lambda_t \tag{1}$$

The terms $\alpha_{(t,i)}$ and λ_t describe the relationship between the target configuration run time and the sample configuration run time. m denotes the number of sample configurations in the profile stage. For the linear type, we only need to profile half-core and all-core sample configurations to implement the equation. $\alpha_{(t,i)}$ scales up the recorded execution time, $Time_i$, on the sample configurations and reflects the scalability based on hardware event rates.



(a)

(b)

(c)

Fig. 3: Performance impact of processor power budget for *linear* (3a), *logarithmic* (3b), and *parabolic* (3c) applications. The maximum concurrency is optimal for linear applications unless the power budget is very low. The optimal concurrency of logarithmic application varies with the power budget. For parabolic applications, the performance gap between the optimal concurrency and maximum concurrency increases when the power budget is decreased.

b) Non-Linear Types: By definition, it is not possible to derive a linear function to describe the relationship between scalability and concurrency for both logarithmic and parabolic applications. Instead, we model the non-linear functions using a piecewise function composed of two linear segments.

From Figure 2b, the speedup of these workloads increases linearly while the thread number is less than or equal to a specific number NP, the inflection point of the function. When the thread number is larger than NP, the trend can also be approximated with another linear model with a smaller slope to describe the trend. As a result, we can divide a curve into two parts and conclude a linear piecewise function to simplify the problem without impacting the accuracy significantly. Similarly, we can also use two linear functions to describe the trends for the parabolic type as shown in Figure 2c.

We use multivariate linear regression (MLR) to predict the inflection point NP for applications that have been classified and verified. The MLR model is sufficiently accurate for our collected data as exploited in [25, 6]. More sophisticated machine learning methods may generate overfit and decrease the accuracy because the amount of data collected is insufficient. The MLR model utilizes the event rates and the manually identified inflection point NP for model training; the events table is listed in Table I. These events are related to applications' memory access patterns and are able to identify which concurrency level can cause performance stagnancy or loss.

Logarithmic Functions. After obtaining the inflection point NP, the two segment linear functions can be derived as:

$$\begin{cases} Time_t = \sum_{i=1}^{m}(Time_i \cdot \alpha_{(t,i)}) + \lambda_t & if\ \ t \leq NP \\ Time_t = \sum_{i=1}^{m'}(Time_i \cdot \alpha'_{(t,i)}) + \lambda'_t & if\ \ t > NP \end{cases} \tag{2}$$

Equation 2 illustrates two slope and intercept parameters to represent the scalability growth difference. Since power increases close to linearly with the participating cores count [16], it is not sufficient to run the application with concurrency in the latter segment while power is not sufficient to keep all running cores at the highest frequency. As seen in Figure 2, frequency significantly impacts the application's performance. Therefore, we would prefer high frequency to high concurrency for logarithmic applications. Thus, this model offers support for efficiently exploiting cluster-level power allocation on each node.

Parabolic Functions. The function for the former segment is derived as:

$$Time_t = \sum_{i=1}^{m}(Time_i \cdot \alpha_{(t,i)}) + \lambda_t \qquad if\ \ t \leq NP \tag{3}$$

For parabolic applications, it is more urgent to find the optimal concurrency level no matter if a power budget is imposed. Simply using all cores or reducing the core count without considering application characteristics could degrade performance significantly. Since the latter segment consumes more power and obtains lower performance, we disregard the prediction for the $n > NP$ segment.

TABLE I: The Haswell hardware events used in sample configurations for prediction.

Predictor	Description
Event0	Instruction Cache (ICACHE) Misses
Event1	Memory Access Read Bandwidth
Event2	Memory Access Write Bandwidth
Event3	L3 Cache Miss from Local DRAM
Event4	L3 Cache Miss from Remote DRAM
Event5	Cycles Active
Event6	Instructions Retired
Event7	Performance ratio by full cores and half cores

B. Cluster-Level Power Allocation

Power coordination at the cluster level needs to not only consider how many nodes should be involved in the computation of the scheduled task, but also consider how to allocate power to each applied node. In order to achieve an optimal solution, we first use the prediction model to obtain the number of threads in node level; after that, we could speculate the power consumption range of the CPU and memory on the node with different frequencies according to the power model. The next step is to determine the number of nodes by predicting the performance with different configurations for the given cluster power budget. Lastly, coordinating the power budget on each node by considering the manufacture variability can further decrease the impact of inter node unbalance and improve the application's performance [20].

1) Cluster Power Allocation: To determine the number of nodes, we need have a good understanding of how the power capping on each node would affect the performance. As demonstrated by previous research [15, 16], each application has its specific power levels for CPU and memory. P_{cpu,L_1}, P_{mem,L_1} are the CPU and memory power consumption when the CPU runs at the highest CPU frequency. A given node power budget larger than $P_{cpu,L_1} + P_{mem,L_1}$ causes a waste of power at the node level and decreases the scalability at the cluster level. We can also obtain P_{cpu,L_2}, P_{mem,L_2} by executing application with the lowest CPU frequency. For a given node power budget less than $P_{cpu,L_2} + P_{mem,L_2}$, the performance of the specific node would be impacted significantly. Thus, we can obtain several options for the node count, each corresponding to a node power budget falling in the range of $[P_{cpu,L_2} + P_{mem,L_2}, P_{cpu,L_1} + P_{mem,L_1}]$. For each application, the scheduler could choose the best number n of nodes (MPI process) that is easier to pair with the data decomposition.

The specific power for the components of each node can be derived from the following equations. First, the total cluster power budget can be distributed as:

$$P(job) = P_1 + P_2 + ... + P_k + ... + P_n \tag{4}$$

Next, we decompose the power of node $P_k (k \in [1, n])$ as the aggregated sum of components types, including processor, memory, and others, i.e.:

$$P_k = P_{ProcT} + P_{MemT} + P_{OtherT} \tag{5}$$

The power of processor P_{ProcT} can be formulated as the sum of individual processors $P_{proc,i}$, which is composed of a base power $P_{pbase,i}$ and a set of core powers P_{c_j}. Subsequently, we model the power of each core for the specific workload w. NS and NC are denoted as the number of sockets (processors) and the number of activated cores on the processor respectively.

$$P_{ProcT} = \sum_{i=1}^{NS} P_{proc,i} \tag{6}$$

$$P_{proc,i} = P_{p_{base},i} + \sum_{j=1}^{NC} P_{c_j}(w) \tag{7}$$

The i_{th} processor power is divided as the base power and the load power by activated cores in Equation 7.

Similarly, the power of memory P_{MemT} can be typed as the sum of memory components' power which is broken down with a base power $P_{m_{base},i}$ and an activity power $P_{m_{load},i}(w)$:

$$P_{MemT} = \sum_{i=1}^{NS} P_{mem,i} \tag{8}$$

$$P_{mem,i} = P_{m_{base},i} + P_{m_{load},i}(w) \tag{9}$$

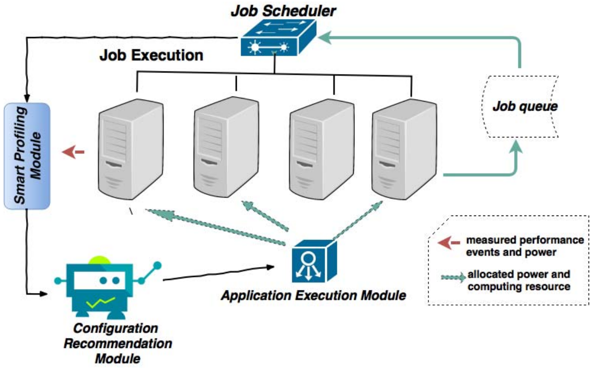

Job Scheduler
Job Execution
Job queue
Smart Profiling Module
measured performance events and power
allocated power and computing resource
Application Execution Module
Configuration Recommendation Module

Fig. 4: **Overview of CLIP**. The target clusters consist of a number of nodes built of multiple multicore-based processors with NUMA architecture. The configuration recommendation module chooses the number of nodes and determines the node power budget P_{node} based on measured performance events and power consumption.

2) Inter Node Power Coordination based on Manufacture Variability: The manufacture variability is one of the key factors that impact the performance across nodes. Inadomi's work [20] demonstrated that manufacture variability can cause significant imbalance among nodes and increase the synchronization cost. To address this problem, we adopt the same method [20] to handle the variability. However, our experimental nodes are quite homogeneous, thus we only coordinate power between nodes when the manufacture power variability exceeds a threshold.

IV. System Design

A. CLIP Overview

We build a framework named CLIP to support application-aware power coordination on multicore-based clusters with NUMA architecture. As illustrated in Figure 4, CLIP includes an intelligent, experimental profiling module, a data-driven execution configuration recommendation module, an application execution module, and several helper tools to provide a user-friendly convenient power-bounded computing environment.

B. CLIP Components

1) Smart Profiling Module: This module reduces the time required to create an experimental profile. It runs several iterations of the specified task application with sufficient power and utilizing all cores. Compared to a full run, which is usually hundreds or thousands of iterations for most HPC applications, smart profiling with a few iterations incurs minimal overhead [31]. The collected data is used to distinguish mapping preference as in [16] and determine the core affinity for the half-core profile configuration. After that, we can derive the scalability trend type of the task. The number of subsequent profiling depends on the classification results. By exploiting the smart profiling module, we can gather all the data for predicting the performance with high accuracy using only two or three total sample configuration profiles.

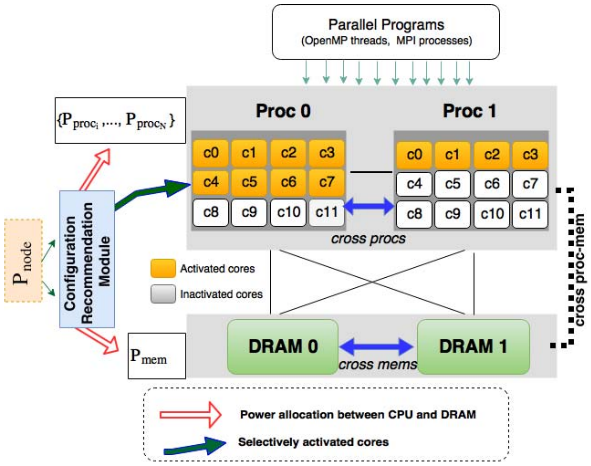

Parallel Programs
(OpenMP threads, MPI processes)
{P proc_i ,..., P proc_N }
Proc 0
Proc 1
c0
c1
c2
c3
c4
c5
c6
c7
c8
c9
c10
c11
cross procs
cross proc-mem
P node
Configuration Recommendation Module
Activated cores
Inactivated cores
P mem
DRAM 0
DRAM 1
cross mems
Power allocation between CPU and DRAM
Selectively activated cores

Fig. 5: **The architecture of CLIP at the node level**. Processors and memory can be capped with specific power levels and per-core DVFS is available. The Configuration Recommendation Module selectively activates cores and decides the power allocation between CPU and DRAM,

2) Configuration Recommendation Module: The configuration recommendation module implements the decision engine which takes a four-tuple (program, problem size, power budget, and profile data) as input and returns a parallel workload execution configuration for each task to the application execution module. By collecting data from the smart profiling module, the configuration recommendation module first determines the optimal power range of each node and decides the number of nodes for the given task. Then, the configuration recommendation module classifies the trend of the application and selects the optimal concurrency under the power budget.

3) Application Execution Module: The application execution module provides a user interface, which takes a program and checks whether the program has been recorded in our knowledge database. In the case of a negative response, the application execution module calls the smart profiling module and inputs the gathered data to the recommendation module. Following that, the application execution module creates a script to launch the job with the execution configuration on a power-bounded multicore cluster through our job scheduler.

4) System Interface Helper Tools: It includes several customized system tools such as a power meter reader, a RAPL and DVFS power controller, and a performance event collector. The tool set interacts with system kernels, libraries, modules, and runtime environments. It automates the collection and recording of performance and power data for jobs managed by the smart profiling module and application execution module.

C. Power Bounded Scheduling Algorithm

Algorithm Descriptions: As illustrated in Algorithm 1, the proposed scheduler acts in two steps:

1) The scheduler searches for the given job in the knowledge database to decide if it is necessary to start smart profiling. By smart profiling or searching from the knowledge database, the system is able to acquire the optimal power range $[P_{cpu_{L_o}} + P_{mem_{L_o}}, P_{cpu_{H_o}} + P_{mem_{H_o}}]$ for each node. After that, the system inputs the profile data and the given power budget recommendation to decide the number of nodes and the power budget for each node.
2) The scheduler inputs the power budget for each node and the profile data for each applications to the recommendation module and gets the suggested power budget for the CPU and memory, the number of activated cores, and the optimal core affinity.

Algorithm 1 CLIP (Cluster level intelligent power coordination system).

```
function CLIP($App$, $C$)

    Input: $P_{ub}$: the total power budget for the cluster;
    Input: $App$: the application under study
    Input: $C$: the cluster with $N_{total}$ nodes
    Input: $N_{total}$: the total number of nodes in the cluster $C$
    Output: $N_{nodes}$: suggested number of active compute nodes
    Output: $P_{cpu_{run_i}}$: suggested CPU power for node $i$
    Output: $P_{mem_{run_i}}$: suggested memory power for node $i$
    Output: $N_{cores}$: suggested number of active cores on each node
    Output: $Map$: suggested mapping affinity

[$P_{cpu_{H_o}}, P_{cpu_{L_o}}, P_{mem_{H_o}}, P_{mem_{L_o}}$, Profile] $\leftarrow SmartProf(App)$

[$N_{core}, Map$] $\leftarrow$ Recommendation (Profile)

    if $App$ has a set of predefined number of processes $N_{def_1}, ..., N_{def_n}$
then
        if $N_{def_k} \leq P_{ub}/(P_{cpu_{L_o}} + P_{mem_{L_o}}) < N_{def_{k+1}}$ then
            $N_{nodes} \leftarrow N_{def_k}$
            $P_{node} \leftarrow P_{ub}/N_{nodes}$
            for every node $i$ to be activated do
                [$P_{cpu_{run_i}}, P_{mem_{run_i}}$] $\leftarrow P_{node}$ (Equation 5)
            end for
        end if
    else
        if $P_{ub} > N_{total} * (P_{cpu_{H_o}} + P_{mem_{H_o}})$ then
            $N_{nodes} \leftarrow N_{total}$
        else
            $N_{nodes} \leftarrow P_{ub}/(P_{cpu_{H_o}} + P_{mem_{H_o}})$
        end if
        for every node $i$ to be activated do
            $P_{cpu_{run_i}} \leftarrow P_{cpu_{H_o}} + P_{cpu_{v_i}}$
            $P_{mem_{run_i}} \leftarrow P_{mem_{H_o}} + P_{mem_{v_i}}$
        end for
    end if
end function
```

V. Experimental Results and Discussions

A. Experimental Testbed

Our experimental platform is an 8-node cluster. Each node has two 12-core Intel(R) Xeon(R) CPU E5-2670v3 @ 2.30GHz processors and 128 GB DDR4 DRAM evenly distributed between two NUMA sockets. The codes are compiled with openmpi 2.0.1. We use RAPL [21] to cap and measure power for the processors and memory modules in our experiments.

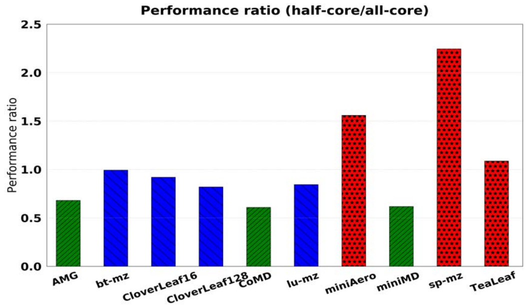


Fig. 6: Parallel speedup ratio (half-core/all-core) comparison. Green: linear applications; Blue: logarithmic applications; and Red: parabolic applications.

We use the parallel hybrid MPI/OpenMP benchmarks listed in Table II in our evaluation. These benchmarks have different workload characteristics spanning from highly scalable, compute-intensive workloads to moderately scalable, memory-intensive workloads. We include two different parameters for the CloverLeaf benchmark to show that the input parameters may affect the power coordination decision for some applications.

B. Application Classification and Prediction

We profile the power and performance for each benchmark with no more than three sample configuration executions. First, we run the application using all available cores and measure the memory bandwidth and cross-NUMA access intensity to determine the core affinity. The next step is to run the application with half of the available cores using a core affinity mapping pattern decided by first step. Using the first two configurations, we are able to identify the scalability trends and predict the inflection point. The last step uses the predicted configuration and measures the events and power again to deduct the model.

1) Classification: Figure 6 visualizes the benchmark speedup ratio of half-core to all-core when no power bound is imposed. The figure reveals that AMG, miniMD, and CoMD benchmarks are linear type applications and their performances grow fast with the core counts. Using the half-core configuration only achieves about a half of the performance delivered by the all-core configuration. BT-MZ, LU-MZ and CloverLeaf are logarithmic type applications, for which performance still increases by scaling from the half-core configuration to the all-core configuration but the performance gain becomes less. TeaLeaf, miniAero and SP-MZ are parabolic type applications and their performances drop from the half-core configuration to the all-core configuration.

To be specific, BT-MZ is the multi-zone version of BT benchmark. The stagnant scalability of BT-MZ for size C beyond half-core is due to function *exch_qbc*, which significantly affects the performance of the BT-MZ version. Thus, we change the concurrency setting phase-by-phase for the BT benchmark to increase performance.

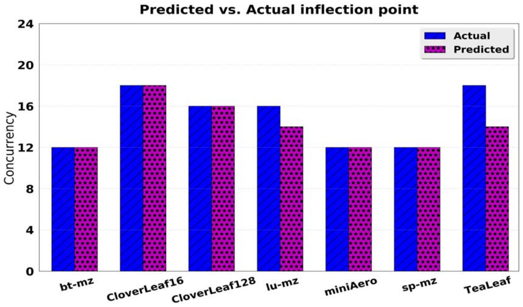


Fig. 7: Predicted and actual inflection points comparison. Logarithmic applications achieve high power efficiency at the inflection point. Parabolic applications achieve the highest performance and power efficiency at the inflection point.

2) Model training and prediction: In order to cover applications with various characteristics, we select benchmarks from NAS Parallel Benchmarks (NPB) [1], HPC Challenge Benchmark (HPCC) [28], UVA STREAM [32], PolyBench [34] and others to help us train the parameters and predict inflection points. Model evaluation results in Figure 7 show the predicted inflection points and the comparison to the actual values through an exhaustive search. We observe that applications perform worse with an odd-value concurrency than with a close even-value concurrency in general. Even-value concurrency allocates resources evenly and thus improves performance, especially for logarithmic and parabolic applications. Thus, we floor the predicted results to an even number. The results indicate that the predictions are strong for most applications and only underestimate for LU-MZ and TeaLeaf.

C. Clip Performance Evaluation

In order to evaluate the performance of CLIP, we compare it with three other coordination methods under various given power budgets.

- **All-In**. This utilizes all supplied nodes. It allocates 30 watts to memory and the remaining power to CPU on each node without considering the cluster power budget. All of the cores participate in application execution. To be specific, allocating 30 watts to memory meets most applications' memory power requirement and won't cause very significant degradation for extremely memory intensive applications' performance.
- **Lower Limit**. This method ensures that no nodes participating in the computation are allocated a budget less than a preset value, i.e., 180 Watts. If the total power budget cannot allocate every node more than 180 watts, the scheduler decreases the number of active nodes. Additionally, this method utilizes all cores on each active node and allocates 30 watts to memory.
- **Coordinated** [15]. This method ensures that the nodes participating in computation are allocated a budget no less than a preset value specific to the application [15]. It

TABLE II: List of Benchmarks Used in This Study

Benchmark	Description	Parameters	Workload Pattern	Scalability Type
BT-MZ	Block Tri-diagonal solver	C	compute	logarithmic
LU-MZ	Lower-Upper Gauss-Seidel solver	C	compute/memory	logarithmic
SP-MZ	Scalar Penta-diagonal solver	C	compute/memory	parabolic
CoMD	classical molecular dynamics	-n 240 240 240	compute	linear
AMG	algebraic multigrid solver	-n 300 300 300	compute/memory	linear
miniAero	mini version to solve the compressible Navier-Stokes equations	default	compute	parabolic
miniMD	force computations	default	compute	linear
TeaLeaf	solves the linear heat conduction equation	Tea10.in	compute/memory	parabolic
CloverLeaf	solves the compressible Euler equations on a Cartesian grid	clover128_short.in	computer/memory	logarithmic
CloverLeaf	solves the compressible Euler equations on a Cartesian grid	clover16.in	computer/memory	logarithmic

coordinates power between CPU and memory according to the power model. The **Coordinated** method executes applications at the highest possible concurrency.

- **CLIP**. It guarantees that the participating nodes are allocated with a budget no less than the lower bound of the acceptable power range for the specified application. Therefore, it decreases the node count to ensure each node has a reasonable power budget if the cluster power budget is not sufficient for all supplied nodes. Besides, **CLIP** changes the concurrency on each node according to the application scalability type and total power budget, and also coordinates power allocation between CPU and memory.

Figures 8 and 9 summarize our comparison of the four methods. In the comparison, we use the relative performance based on the **All-In** method without a power bound. The two figures support the following observations:

1) **CLIP** achieves similar performance as **All-In** for most of the applications under study, and outperforms $\geq 40\%$ for miniMD and sp-mz applications of the parabolic type, when there is no specified power bound.
2) **CLIP** performs close to the optimal for all the tested benchmarks if the power budget is unlimited or high.
3) **CLIP** outperforms **All-In**, **Coordinated**, **Low-Limit** for most cases, specially for logarithmic and parabolic applications.
4) **CLIP** defends **Coordinated** for parabolic applications (SP-MZ, miniAero and TeaLeaf) by up to 60% overall. When the thread count exceeds optimal, these parabolic applications experience a worsened performance but consume more power. Carefully distributing resources for such applications significantly improves performance.
5) **CLIP** outperforms **Coordinated** for logarithmic when the power budget is low. logarithmic applications is common among big data applications that require higher memory bandwidth. This observation confirms the hypothesis that classifying applications and setting corresponding configurations is beneficial for power-bounded computing.

To conclude, the experimental results confirm that the proposed resource coordination method provides an efficient and effective solution for power-bounded cluster computing, and our framework and implementation are applicable to real systems and applications.

VI. Related Work

Power is consistently one of the top challenges for HPC evolving from a soft concern several years ago to a strict bound today [14]. Accordingly, the objectives and approaches of HPC power management have continuously evolved to fit the changing requirements of HPC. With the looming approach of the exascale era, power increasingly limits the scalability and operation of high performance computing systems [35].

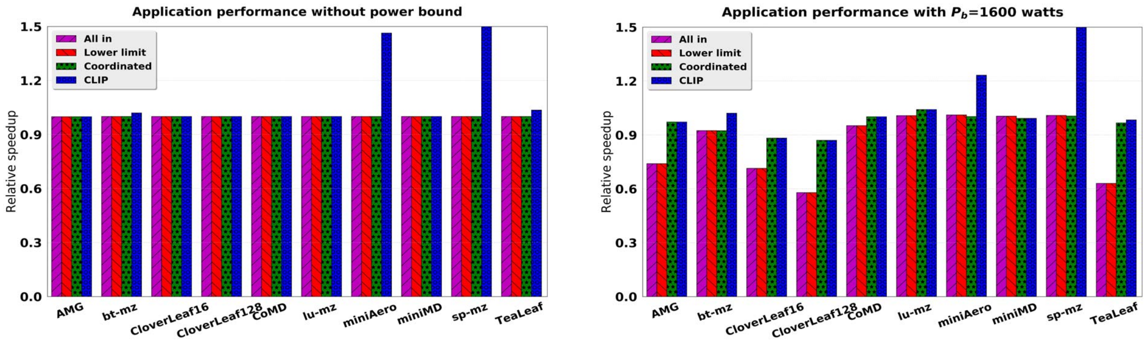
Power bounded computing is far different from earlier power aware computing [10, 26, 39] or power capping and budgeting [41, 36, 19, 5]. Power aware computing aims to reduce power consumption or save energy with little performance impact by integrating methods utilizing component performance-power states. Power capping and budgeting technology mainly focus on peak power and thermal emergency control in commercial data centers [19]. Because both the hardware architecture and the workloads (usually being sequential and independent) running in commercial data centers dramatically differ from HPC and the parallel workloads, most of the power capping and budgeting techniques are not applicable in an HPC environment directly.

Recently, many power aware, power capping, and power budget ideas have been expanded into HPC area by researchers. These shared hardware and software technologies could be divided into these categories:

1) Software control of the component power states such as applying DVFS and clock-throttling to limit the power of CPU [42, 24, 22, 3, 2, 13, 9], memory [7], or both [18, 4, 37, 39].
2) Software control of workload execution such as controlling the level of concurrency [38, 5, 30].
3) Hardware-based power capping by using RAPL [21, 8].
4) Hardware-based power coordination between main computer components [40, 15].
5) Combined software and hardware methods [43, 31].

In this paper, we focus on utilizing software and hardware more efficiently by adapting concurrency management and coordinating power between CPU and memory on the cluster.

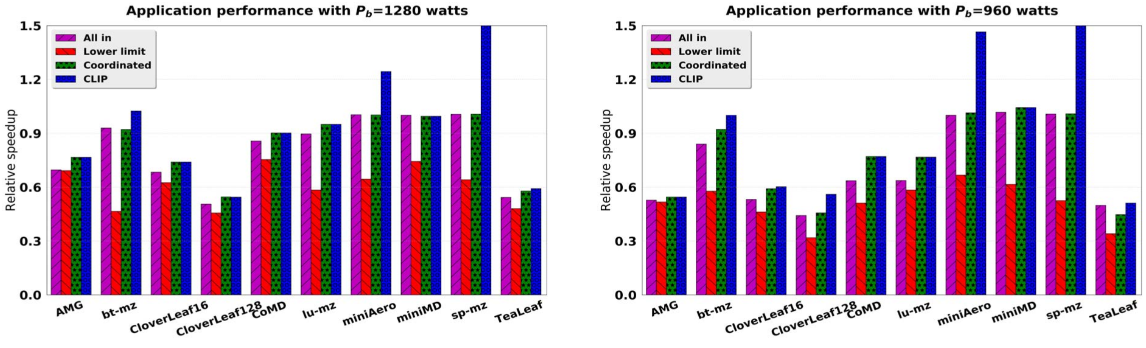
Recognizing the importance of accurately achieving the near optimal concurrency and power budget range, other researchers have also studied the prediction of the optimal number of threads [6, 25] and optimal power allocation between CPU and memory [40]. Both ACTOR [6] and Li et al. [25] use hardware events rate to predict the optimal



(a)

(b)

Fig. 8: Performance Comparison of different power allocation methods under high power budget



(a)

(b)

Fig. 9: Performance comparison of different power allocation methods under low power budget

concurrency; however, the new features of current many-core hardware architecture make it infeasible to use these prediction methods directly.

Cluster-level power coordination generally considers the number of active nodes and power distribution among these nodes for a given total budget. Existing work takes the effect of manufacturing variability into account to eliminate the load imbalance [11, 17, 29]. Hardware overprovisioning [33] gives promising exploratory results from dynamically changing the node count, concurrency parameters on the node level and changing the power budget for each node, but fails to conclude with a straightforward solution. $Conductor$ [31] exhaustively searches available configurations to find the optimal thread concurrency, without discerning the optimal number of nodes. In addition, $Conductor$ profiles applications with different configurations on multiple nodes, with which the impact of communication on parameterization is unknown. POWshed [11] shifts power to more power-intensive applications to improve throughput without exploring concurrency throttling.

Our work differs from these prior studies [11, 17, 29, 31] in three main aspects. First, our study reveals the different impacts on applications with different types of scalability. Such findings are helpful for designing power management methods. Second, we develop an intelligent configuration recommendation system, which suggests application execution configurations and power coordination between CPU and memory without exhaustive searching. Third, our approach considers both node-level and cluster-level techniques and integrates them tightly to improve application performance on NUMA multicore-based systems under power bound.

VII. Conclusion

In this paper we present CLIP, a framework for cluster-level power-bounded resource coordination on NUMA multicore-based systems. It utilizes a novel approach to identify application scalability and explores multi-dimensional power management techniques to improve performance under a power budget. Experimental evaluations demonstrate that the scheduler is able to identify near-optimal configurations for given power budgets.

To the best of our knowledge, this is the first power bounded scheduler that simultaneously considers inter-node scalability, node-level concurrency, and main component power coordination. As power has been a scarce resource, power-bounded research expands the tools to address power challenges in the exascale era. Our evaluation shows that CLIP outperforms prior work significantly for complex applications. The average improvements are close to 20% under low power budget. By implementing the scheduler and evaluating it in a real cluster, this work reveals some key insights about power bounded computing on large scale systems and provides a solution with a low overhead. One limitation of this work is that CLIP doesn't directly support jobs launched with predefined node and core counts. We plan to develop a runtime system to address this issue and accommodate the needs in the future.

Acknowledgment

The work is supported in part by the U.S. National Science Foundation under Grants CCF-1551511 and CNS-1551262.

References

[1] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, et al. The nas parallel benchmarks. *The International Journal of Supercomputing Applications*, 5(3):63–73, 1991.

[2] K. J. Barker, D. J. Kerbyson, and E. Anger. On the feasibility of dynamic power steering. In *Proceedings of the 2Nd International Workshop on Energy Efficient Supercomputing*, E2SC '14, pages 60–69, Piscataway, NJ, USA, 2014. IEEE Press.

[3] J. M. Cebrián, J. L. Aragon, and S. Kaxiras. Power Token Balancing: Adapting CMPs to Power Constraints for Parallel Multithreaded Workloads. In *2011 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 431–442. IEEE, 2011.

[4] M. Chen, X. Wang, and X. Li. Coordinating processor and main memory for efficientserver power control. In *Proceedings of the International Conference on Supercomputing*, ICS '11, pages 130–140, New York, NY, USA, 2011. ACM.

[5] 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*, pages 175–185. ACM, 2011.

[6] M. Curtis-Maury, F. Blagojevic, C. D. Antonopoulos, and D. S. Nikolopoulos. Prediction-based power-performance adaptation of multithreaded scientific codes. *IEEE Transactions on Parallel and Distributed Systems*, 19(10):1396–1410, 2008.

[7] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu. Memory power management via dynamic voltage/frequency scaling. In *Proceedings of the 8th ACM International Conference on Autonomic Computing*, ICAC '11, pages 31–40, New York, NY, USA, 2011. ACM.

[8] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu. Memory power management via dynamic voltage/frequency scaling. In *Proceedings of the 8th ACM International Conference on Autonomic Computing*, ICAC '11, pages 31–40, New York, NY, USA, 2011. ACM.

[9] D. De Sensi, M. Torquati, and M. Danelutto. A reconfiguration algorithm for power-aware parallel applications. *ACM Trans. Archit. Code Optim.*, 13(4):43:1–43:25, Dec. 2016.

[10] Q. Deng, D. Meisner, A. Bhattacharjee, T. Wenisch, and R. Bianchini. CoScale: Coordinating CPU and Memory System DVFS in Server Systems. In *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 143–154, Dec 2012.

[11] D. A. Ellsworth, A. D. Malony, B. Rountree, and M. Schulz. Dynamic Power Sharing for Higher Job Throughput. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15. ACM, 2015.

[12] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger. Dark Silicon and the End of Multicore Scaling. In *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*, pages 365–376. IEEE, 2011.

[13] M. Etinski, J. Corbalan, J. Labarta, and M. Valero. Parallel job scheduling for power constrained HPC systems . *Parallel Computing*, 38(12):615 – 630, 2012.

[14] R. Ge, X. Feng, and K. W. Cameron. Improvement of power-performance efficiency for high-end computing. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 11 - Volume 12*, IPDPS '05, pages 233.2–, Washington, DC, USA, 2005. IEEE Computer Society.

[15] R. Ge, X. Feng, Y. He, and P. Zou. The case for cross-component power coordination on power bounded systems. In *2016 45th International Conference on Parallel Processing (ICPP)*, pages 516–525, Aug 2016.

[16] R. Ge, P. Zou, and X. Feng. Application-aware power coordination on power bounded numa multicore systems. In *accepted to the 46th International Conference on Parallel Processing*. IEEE, 2017.

[17] N. Gholkar, F. Mueller, and B. Rountree. Power tuning hpc jobs on power-constrained systems. In *Proceedings of the 2016 International Conference on Parallel Architectures and Compilation*, PACT '16, pages 179–191, New York, NY, USA, 2016. ACM.

[18] H. Hanson, W. Felter, W. Huang, C. Lefurgy, K. Rajamani, F. Rawson, and G. Silva. Processor-Memory Power Shifting for Multi-Core Systems. In *4th Workshop on Energy Efficient Design*, 2012.

[19] H. Hanson, S. Keckler, S. Ghiasi, F. Rawson, J. Rubio, et al. Power, performance, and thermal management for high-performance systems. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–8. IEEE, 2007.

[20] Y. Inadomi, T. Patki, K. Inoue, M. Aoyagi, B. Rountree, M. Schulz, D. Lowenthal, Y. Wada, K. Fukazawa, M. Ueda, M. Kondo, and I. Miyoshi. Analyzing and mitigating the impact of manufacturing variability in power-constrained supercomputing. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, pages 78:1–78:12, New York, NY, USA, 2015. ACM.

[21] Intel. Volume 3B: System Programming Guide, Part 2. *Intel 64 and IA-32 Architectures Software Developers Manual*, June 2013.

[22] C. Isci, A. Buyuktosunoglu, C.-Y. 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 MICRO*, pages 347–358, 2006.

[23] V. Kontorinis, L. E. Zhang, B. Aksanli, J. Sampson, H. Homayoun, E. Pettis, D. M. Tullsen, and T. S. Rosing. Managing Distributed UPS Energy for Effective Power Capping in Data Centers. In *Computer Architecture (ISCA), 2012 39th Annual International Symposium on*, pages 488–499, June 2012.

[24] C. Lefurgy, X. Wang, and M. Ware. Power Capping: a Prelude to Power Shifting. *Cluster Computing*, 11(2):183–195, 2008.

[25] D. Li, B. R. de Supinski, M. Schulz, K. Cameron, and D. S. Nikolopoulos. Hybrid mpi/openmp power-aware computing. In *2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS)*, pages 1–12, April 2010.

[26] X. Li, R. Gupta, S. V. Adve, and Y. Zhou. Cross-Component Energy Management: Joint Adaptation of Processor and Memory. *ACM Transactions on Architecture and Code Optimization (TACO)*, 4(3), Sept. 2007.

[27] R. Lucas, J. Ang, K. Bergman, S. Borkar, W. Carlson, L. Carrington, G. Chiu, R. Colwell, W. Dally, J. Dongarra, et al. Top ten exascale research challenges. *DOE ASCAC subcommittee report*, pages 1–86, 2014.

[28] P. R. Luszczek, D. H. Bailey, J. J. Dongarra, J. Kepner, R. F. Lucas, R. Rabenseifner, and D. Takahashi. The hpc challenge (hpcc) benchmark suite. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, SC '06, New York, NY, USA, 2006. ACM.

[29] M. Maiterth, M. Schulz, D. Kranzlmller, and B. Rountree. Power Balancing in an Emulated Exascale Environment. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1142–1149, May 2016.

[30] A. Marathe, P. E. Bailey, D. K. Lowenthal, B. Rountree, M. Schulz, and B. R. de Supinski. A Run-time System for Power-constrained HPC Applications. In *High Performance Computing*, pages 394–408. Springer, 2015.

[31] A. Marathe, P. E. Bailey, D. K. Lowenthal, B. Rountree, M. Schulz, and B. R. de Supinski. A run-time system for power-constrained hpc applications. In *International Conference on High Performance Computing*, pages 394–408. Springer, 2015.

[32] J. D. McCalpin. Stream benchmark. *Link: www. cs. virginia. edu/stream/ref. html# what*, 22, 1995.

[33] T. Patki, D. K. Lowenthal, B. Rountree, M. Schulz, and B. R. De Supinski. Exploring hardware overprovisioning in power-constrained, high performance computing. In *Proceedings of the 27th international ACM conference on International conference on supercomputing*, pages 173–182. ACM, 2013.

[34] L.-N. Pouchet. Polybench: The polyhedral benchmark suite. *URL: http://www. cs. ucla. edu/pouchet/software/polybench*, 2012.

[35] V. Sarkar, S. Amarasinghe, D. Campbell, et al. Exascale Software Study:

Software Challenges in Extreme Scale Systems. *DARPA Information Processing Techniques Office, Washington DC*, 14:159, 2009.

[36] O. Sarood, A. Gupta, and L. Kalé. Temperature Aware Load Balancing for Parallel Applications: Preliminary Work. In *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW)*, pages 796–803. IEEE, 2011.

[37] O. Sarood, A. Langer, L. Kalé, B. Rountree, and B. De Supinski. Optimizing Power Allocation to CPU and Memory Subsystems in Overprovisioned HPC Systems. In *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER)*, pages 1–8. IEEE, 2013.

[38] A. Sharifi, A. K. Mishra, S. Srikantaiah, M. Kandemir, and C. R. Das. Pepon: Performance-aware hierarchical power budgeting for noc based multicores. In *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*, PACT '12, pages 65–74, New York, NY, USA, 2012. ACM.

[39] V. Sundriyal and M. Sosonkina. Joint Frequency Scaling of Processor and DRAM. *The Journal of Supercomputing*, pages 1–21, 2016.

[40] A. Tiwari, M. Schulz, and L. Carrington. Predicting Optimal Power Allocation for CPU and DRAM Domains. In *Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International*, pages 951–959, May 2015.

[41] X. Wang and Y. Wang. Coordinating Power Control and Performance Management for Virtualized Server Clusters. *Parallel and Distributed Systems, IEEE Transactions on*, 22(2):245–259, 2011.

[42] Y. Wang, K. Ma, and X. Wang. Temperature-Constrained Power Control for Chip Multiprocessors with Online Model Estimation. In *ACM SIGARCH Computer Architecture News*, volume 37, pages 314–324. ACM, 2009.

[43] H. Zhang and H. Hoffmann. Maximizing Performance Under a Power Cap: A Comparison of Hardware, Software, and Hybrid Techniques. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '16, pages 545–559, New York, NY, USA, 2016. ACM.