Analyzing and Mitigating the Impact of Manufacturing Variability in Power-Constrained Supercomputing

Yuichi Inadomi[1], Tapasya Patki[2], Koji Inoue[1], Mutsumi Aoyagi[1],
Barry Rountree[3], Martin Schulz[3], David Lowenthal[2], Yasutaka Wada[4],
Keiichiro Fukazawa[5], Masatsugu Ueda[1], Masaaki Kondo[6], Ikuo Miyoshi[7]

[1]Kyushu University
[2]University of Arizona
[3]Lawrence Livermore National Laboratory
[4]Meisei University
[5]Kyoto University
[6]University of Tokyo
[7]Fujitsu Ltd.
Contact Email: inadomi@soc.ait.kyushu-u.ac.jp

ABSTRACT

A key challenge in next-generation supercomputing is to effectively schedule limited power resources. Modern processors suffer from increasingly large power variations due to the chip manufacturing process. These variations lead to *power inhomogeneity* in current systems and manifest into *performance inhomogeneity* in power constrained environments, drastically limiting supercomputing performance. We present a first-of-its-kind study on manufacturing variability on four production HPC systems spanning four microarchitectures, analyze its impact on HPC applications, and propose a novel variation-aware power budgeting scheme to maximize effective application performance. Our low-cost and scalable budgeting algorithm strives to achieve performance homogeneity under a power constraint by deriving application-specific, module-level power allocations. Experimental results using a 1,920 socket system show up to 5.4X speedup, with an average speedup of 1.8X across all benchmarks when compared to a variation-unaware power allocation scheme.

Categories and Subject Descriptors

C.4 [**Computer Systems Organization**]: Performance of Systems

General Terms

Algorithms, Measurement, Performance

Keywords

Power-constrained HPC, Performance Modeling

SC '15, November 15-20, 2015, Austin, TX, USA

ACM 978-1-4503-3723-6/15/11 ...$15.00.
http://dx.doi.org/10.1145/2807591.2807638.

1. INTRODUCTION

Future High-Performance Computing (HPC) systems are expected to be significantly power constrained. For example, the U.S. Department of Energy has set a target of achieving an exaflop under 20 MW in the next decade [2, 4, 45]. Optimizing application performance under a power bound has thus become an important research area in the semiconductor as well as supercomputing communities.

With the end of Dennard Scaling, it is difficult to improve processor performance without affecting power [16], and the leaps needed for achieving exascale performance with tight power constraints are pushing technology advances to their limit. As a consequence, the fabrication process now leads to some processors being less power efficient than others even when they have the exact same architectural specifications [7, 14, 22, 52]. This trend is intensifying as we scale out. We refer to this variation in processor power consumption as *manufacturing variability*.

Because of this manufacturing variability, *modules* (individual processors and associated DRAM) in current HPC systems are already *inhomogeneous* from the point of view of power. Early results on 64 processors have shown a 10% power variation for identical workloads at equivalent performance [41]. In power-limited systems, where enforcing hardware power caps is necessary, this power variation turns into CPU frequency variation, directly impacting application execution times and causing *performance inhomogeneity* [41]. This raises several new concerns; for example, a perfectly load balanced application will now experience load imbalance, and application performance will depend significantly on the physical processors allocated to it during scheduling.

In this paper, we first study this phenomenon of power inhomogeneity and its impact on HPC application performance and then propose a low-cost, variation-aware power budgeting algorithm that improves performance under a power constraint. We make the following contributions:

- We present a first-of-its-kind study on processor manufacturing variability and quantify it on four production HPC systems with four different underlying architectures (Intel Sandy Bridge and Ivy Bridge, IBM Blue-

Gene/Q, and AMD Piledriver) using three power measurement techniques (Intel's RAPL, IBM BlueGene/Q EMON, and PowerInsight). Overall, we observe up to 23% variation in processor (CPU) power.

- We analyze HPC applications on a large-scale, 1,920-module Intel Ivy Bridge system and show that under a power constraint, the variation in module power (CPU and associated DRAM) can result in up to 64% performance variation across application ranks.
- We design a low-cost, scalable, variation-aware power budgeting algorithm that improves application performance under a power constraint by determining module-level power allocations to ensure performance homogeneity.
- We implement this algorithm and propose two possible power management strategies: one with *power capping* and the other with *frequency selection*. The former specifies the amount of power assigned to each module via RAPL, whereas the latter directly controls the CPU frequency by using `CPUfrequtils`.
- Our practical, large-scale experiments demonstrate that the proposed implementations can achieve a maximum speedup of 5.4X and an average speedup of 1.8X on HPC applications when compared to a naive, variation-unaware power allocation scheme.

The paper is organized as follows. Section 2 explains manufacturing variability and discusses existing power management techniques in HPC. Section 3 presents the details of the architectures and benchmarks used in our experiments. Section 4 analyzes the extent of power variation and its impact on HPC application performance and Section 5 proposes our novel power budgeting scheme. In Section 6, we report our evaluation results and demonstrate the efficiency of the proposed framework. Section 7 concludes this paper.

2. BACKGROUND AND RELATED WORK

2.1 Manufacturing Variability

Over the past several decades, significant processor level performance improvements were obtained by increasing the transistor density on a processor die (Moore's Law). This was accomplished by shrinking the feature size of transistors, which involved scaling down the associated threshold voltage and current. Shrinking a transistor below a certain level leads to increased subthreshold conduction, leakage currents and heat dissipation, making it impossible to improve processor performance without affecting its power consumption and reliability [16, 17]. This also makes lithography challenging, as distortions in channel lengths and film thickness may occur and can lead to threshold voltage (and eventually, CPU frequency and power) variations [8, 22, 52]. These variations can be random (usually dopant variations) or systematic and can occur in a single die (*within-die*) or between multiple dies (*die-to-die*). Other factors such as temperature and supply voltage can cause additional variations [7, 14]. Despite the ongoing research in fabrication mechanisms and newer technologies such as power gating, these manufacturing variations in CPU frequencies and power are expected to worsen [28, 44]. Similar variations owing to the fabrication process are found in DRAM chips.

Most vendors address variation in CPU frequency by using *frequency binning*—processors with the same performance characteristics are placed in the same bin (typically, HPC systems obtain all their processors from the same bin). Currently, vendors do not deploy *power binning*, which is why we observe power inhomogeneity in existing large-scale supercomputers. However, when we start constraining power in hardware, the variation in power translates to variation in CPU frequencies, affecting performance drastically [41, 50].

2.2 Power-constrained supercomputing

Managing power efficiently and improving performance under a power constraint has been the focus of recent research in the semiconductor industry as well as in the supercomputing community. The semiconductor industry is focusing on the development of better hardware, such as non-volatile memories, low-power interconnects [24, 37] and more power-efficient processor architectures [10]. Another direction for research is near-threshold voltage computing, which currently presents performance and reliability challenges [31, 32]. In the HPC community, a key area of research is *hardware overprovisioning* [39, 46], in which a system is designed to have more capacity than what it can fully power with the given overall power resource. Such overprovisioned systems can be reconfigured dynamically based on workload characteristics. Power-aware resource management techniques for conventional [6, 18, 19] as well as overprovisioned systems [40, 47] are being actively developed. Additionally, selecting efficient application configurations and using techniques such as dynamic concurrency throttling are being widely studied [5, 11]. Prior research in this domain focused on energy efficiency and includes studies to save energy by trading execution times as well as techniques to achieve optimal energy savings without affecting performance [9, 21, 23, 30, 42, 43].

More recently, runtime systems to redistribute power intelligently within and between applications have been proposed [15, 33, 35]. However, these do not address manufacturing variability. Totoni et. al [51] have discussed power-aware scheduling for process variation heterogeneity with Integer Linear Programming on 36-core chip multiprocessors (CMPs) with the Charm++ [29] platform. While this is an interesting approach, it is a simulation study on 100 CMPs and has high overhead because it involves solving an NP-hard ILP for each scheduling decision. Additionally, it may lead to increased resource fragmentation and unfair sharing of the power resource in a real system [20, 40]. Our algorithm uses a low-cost single-module test run of the application along with manufacturing variability data from large-scale production systems and can work in conjunction with existing as well as future resource managers, making it more scalable and efficient from the point of view of deployment on real production systems.

3. SETUP FOR STUDYING VARIATION

As we move towards power limited environments, detecting and balancing manufacturing variability will play an important role in the optimization of HPC application performance. Before we can discuss the necessary mechanisms and techniques to counteract these variations, as we will do in Section 5, we must first understand the impact of manufacturing variability. We therefore begin by presenting an extensive study to determine the impact of variation in power

on HPC application performance. The following section details the techniques used to measure and control power and describes the systems and benchmarks used in our study.

3.1 Power Measurement Techniques

Several portable vendor-specific mechanisms to measure and manage module power have been developed. Some of these mechanisms use sensors and some others use performance counter based models to estimate component power. The techniques used in our study include Intel's Running Average Power Limit (RAPL), Penguin Computing's PowerInsight (PI) and IBM's Environmental Monitor (EMON). We discuss these techniques in detail below. Currently, RAPL is the only technique that allows us to constrain and manage power. Table 1 summarizes these techniques and shows the granularity of measurement available in each case.

We primarily focus on module power (CPU sockets and DRAM) in this work, mostly because it can be controlled dynamically in HPC systems and can be used to constrain and manage power at a fine granularity. Other components, such as interconnects, do not provide this dynamic control and account for static or base power consumption.

3.1.1 RAPL

RAPL is a model-based power management interface that was introduced with the Intel Sandy Bridge microarchitecture [12, 27]. It supports on-board power measurement and *hardware power capping* across two main domains—*Package* (PKG or CPU) and *DRAM*. RAPL is implemented with the help of programmable Machine Specific Registers (MSRs) and we program these by using the libMSR library [49]. To enforce a power cap using RAPL, users can specify a power bound and a time window, and the hardware ensures that the average power over the time window does not exceed the specified bound in the requested power domain. Intel also has an MSR to control its Turbo Boost feature [26], which can be used for overclocking the processor dynamically. The operating CPU frequency in Turbo mode depends on the workload and the ambient temperature and cannot be controlled directly in software.

Although the specification of RAPL covers DRAM power capping, motherboards supporting this functionality rarely exist in large-scale supercomputers. Since DRAM power is strongly correlated with CPU performance, it can be managed by capping the CPU power. We thus restrict power capping to the CPU domain in this paper[1].

3.1.2 BlueGene/Q EMON

Each rack of a BG/Q machine houses two midplanes, eight link cards, and two service cards. Each midplane has 16 node boards, each of which has 32 compute cards (nodes) connected by a 5D-torus. Power measurement is supported at the node board level. A compute card has 17 active cores, 16 of which are application cores. Measurements are supported across seven different domains, the main ones being chip cores (CPU) and chip memory.

To facilitate power measurement, each node board is connected to an FPGA over the EMON bus, which in turn is connected to two Direct-Current Assemblies (DCAs). The DCAs have a microcontroller that periodically calculates instantaneous power [53–55] and communicates with the FPGA over the I2C bus. The FPGA can relay data to the BG/Q compute nodes over the EMON bus. IBM provides an EMON API to access power consumption data.

[1]Our work can be easily extended to include memory-level power capping when it becomes available in HPC production systems.

Table 1: Power Measurement Techniques

Technique	Reported	Granularity	Power Capping
RAPL	Average	1 ms	Yes
PowerInsight	Instantaneous	1 ms (or less)	No
BGQ EMON	Instantaneous	300 ms	No

3.1.3 PowerInsight (PI)

PI is an architecture-independent, sensor-based technique to monitor node power [13, 25]. The PI architecture consists of three components: (1) a harness with sensor modules for measurement; (2) a cape, or carrier board, with three Analog-to-Digital-Converters (ADC) connected to the sensor modules, and; (3) a BeagleBone core with an ARM Cortex A8 processor, 256 MB of memory, and USB/Ethernet connectivity to the primary node being measured.

Each sensor module is an Allegro ACS713 hall effect current sensor and a voltage divider. Typically, the BeagleBone is plugged in to the motherboard using the USB port which provides console connectivity to the node. CPU and DRAM power consumption data can be measured by using the *getRawPower* software utility, or with the PI-API.

3.2 Systems and Architectures

We use four supercomputing systems (Cab, Vulcan, Teller, and HA8K) for our analysis in this paper, spanning four underlying micro-architectures. The first two systems are located at the Lawrence Livermore National Laboratory (LLNL), and the other two are located at Sandia National Laboratory (SNL) and Kyushu University. Vulcan is the 9th fastest supercomputer in the world, and the HA8K system is a part of the 49th-fastest supercomputer in the world, QUARTETTO. Table 2 provides details about the computing resources and power measurement support in these systems. For the Cab system at LLNL, DRAM power measurement was not available due to BIOS restrictions.

3.3 Benchmarks

The benchmarks used in this work are described below. We used the MPI versions of these codes in this paper. For the HA8K system, we used Intel compiler (version 15.0.1) with Intel MPI (version 5.0). We used GNU compiler (version 4.7) with OpenMPI (version 1.6) on Teller, GNU compiler (version 4.4.7) with BG/Q MPI on Vulcan, and the Intel compiler (version 14.0.3) with MVAPICH2 on Cab.

3.3.1 *DGEMM and *STREAM

These two benchmarks are from the HPC Challenge benchmark suite [34]. *DGEMM is a compute-bound, embarrassingly parallel matrix multiplication subroutine from the BLAS library and is also the main kernel for the High Performance Linpack (HPL) benchmark. We used a thread-parallelized version of this code in the Intel Math Kernel Library with a matrix size of $12,288 \times 12,288$. The *STREAM benchmark is used to measure sustainable memory bandwidth and executes simple vector operations. We optimized

Table 2: Architectures Under Consideration

Site	Node Micro-Architecture	Total Nodes	Procs. Per Node	Cores Per Proc.	CPU Frequency	Memory Per Node	TDP	Power Msrmt.
Cab (LLNL)	Intel E5-2670 Sandy Bridge	1,296	2	8	2.6 GHz	32 GB	115 W	RAPL
BG/Q Vulcan (LLNL)	IBM PowerPC A2	24,576	1	16 (compute)	1.6 GHz	16 GB	Unreported (Max 100 kW per rack)	EMON
Teller (SNL)	AMD A10-5800K Piledriver	104	1	4	3.8 GHz	16 GB	100 W	PI
HA8K (Quartetto) Kyushu Univ.	Intel E5-2697v2 Ivy Bridge	960	2	12	2.7 GHz	256 GB	130 W	RAPL

the original HPCC code to facilitate the use of the AVX instructions and used OpenMP in addition to MPI. Each vector is 24 GB and does not exceed the capacity of the DRAM module.

3.3.2 NAS Parallel Benchmarks: EP, BT and SP

Embarrassingly Parallel (EP) is a simple kernel from the NAS Parallel Benchmark (NPB) suite [1] that is used to generate independent Gaussian Random variates using the Marsaglia polar method. Block Tri-diagonal solver (BT) and Scalar Penta-diagonal solver (SP) are two other pseudo applications from NPB. We used the MPI version with Class D for EP, and the multizone hybrid MPI + OpenMP versions with Class E for BT and SP.

3.3.3 MHD

Magneto-Hydro-Dynamics (MHD) simulation is used understand the global configuration and dynamics of space plasma, which is essential for forecasting space weather. MHD equations are derived from the moments of Vlasov and Maxwell equations. We choose a three-dimensional MHD code that uses the Modified Leapfrog (MLF) method to solve partial differential equations iteratively [38].

3.3.4 mVMC

mVMC-mini (mVMC) is a mini-application included in the FIBER benchmark suite [36] developed by the RIKEN Advanced Institute for Computational Science. It uses Monte Carlo sampling to analyze the structure of strongly correlated electron systems. We use the middle-scale benchmark setting provided by the application developers.

4. ANALYZING VARIATIONS

In this section, we present detailed CPU power and performance measurements across three HPC production systems (Cab, Vulcan and Teller) and establish that power inhomogeneity exists irrespective of the underlying micro-architecture. We then examine module-level (CPU and DRAM) power variation in specific HPC applications on HA8K. Finally, with the help of RAPL power caps, we study the impact of module-level power variation on application performance in power-constrained scenarios.

4.1 CPU Power Inhomogeneities

Figure 1 shows power and performance data for a single-socket EP benchmark on the Cab, Vulcan and Teller systems. We choose single-socket EP for several reasons. First, it is an embarrassingly parallel, CPU-bound benchmark with no communication or synchronization overheads. Second, most of its working set fits in cache, which allows us to analyze CPU power and performance in isolation. Third, EP exhibits no per-run noise (less than 0.5% variation over 15 iterations on the same socket), which lets us conclude that our results depict nothing but manufacturing variability.

The performance data in Figure 1 is represented as a percentage slowdown compared to the fastest socket in the system, and the power data is reported as a percentage increase when compared to the most power-efficient socket. Additionally, processors are sorted by performance characteristics. Turbo Boost and Turbo Core have been enabled on supporting architectures [3, 26]. Power caps have not been enforced on the Cab system. Note that power capping is not supported on Vulcan and Teller. DRAM power readings were unavailable on Cab due to BIOS restrictions. We thus focus only on CPU power in this subsection.

For the Cab and Vulcan systems, we use 2,386 processors (1,193 nodes) and 1,536 processors (48 node boards) respectively and observe almost no performance variation. Maximum variation in power on Cab is 23%, and maximum variation in power on Vulcan is 11%. As explained in the previous subsection, this is expected when processors belong to the same frequency bin. On the Teller system, we obtain data for 64 processors, and observe both power and performance variation. Maximum power variation is 21%, and the maximum performance variation is 17%. We also notice a small negative correlation between performance and power—processors that consumed more power performed better. We believe this could be because of a different binning strategy that we are unaware of. Our key observation from Figure 1 is that manufacturing variability affects CPU power consumption significantly across different computing platforms and does not necessarily correlate with performance, making it extremely difficult to make application-level power and performance predictions.

4.2 Module Power Inhomogeneities

From this section onward, all our results in this paper are from the HA8K system. This is because we were able to use CPU power capping and measure DRAM power on this system. Additionally, we now conduct 1,920-module multi-node experiments. Due to space limitations, we only analyze two applications (*DGEMM and MHD) in this subsection. We observe similar trends for other benchmarks.

Figure 2 depicts results of the 1,920-module experiments of *DGEMM and MHD on the HA8K system with and with-

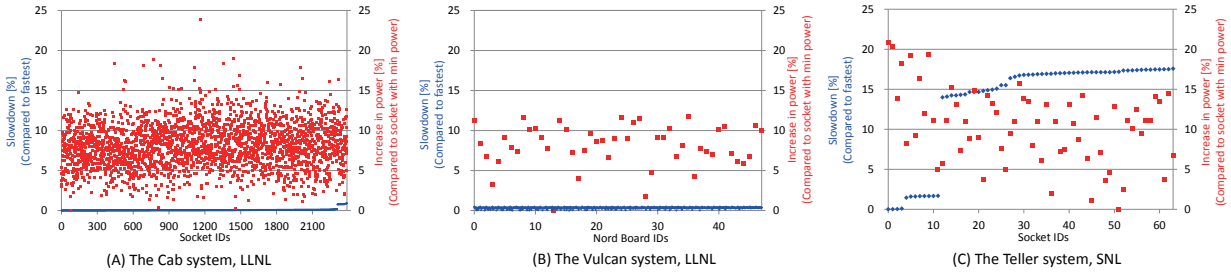

Figure 1: Processor Power and Performance Variation on Cab, Vulcan and Teller Systems

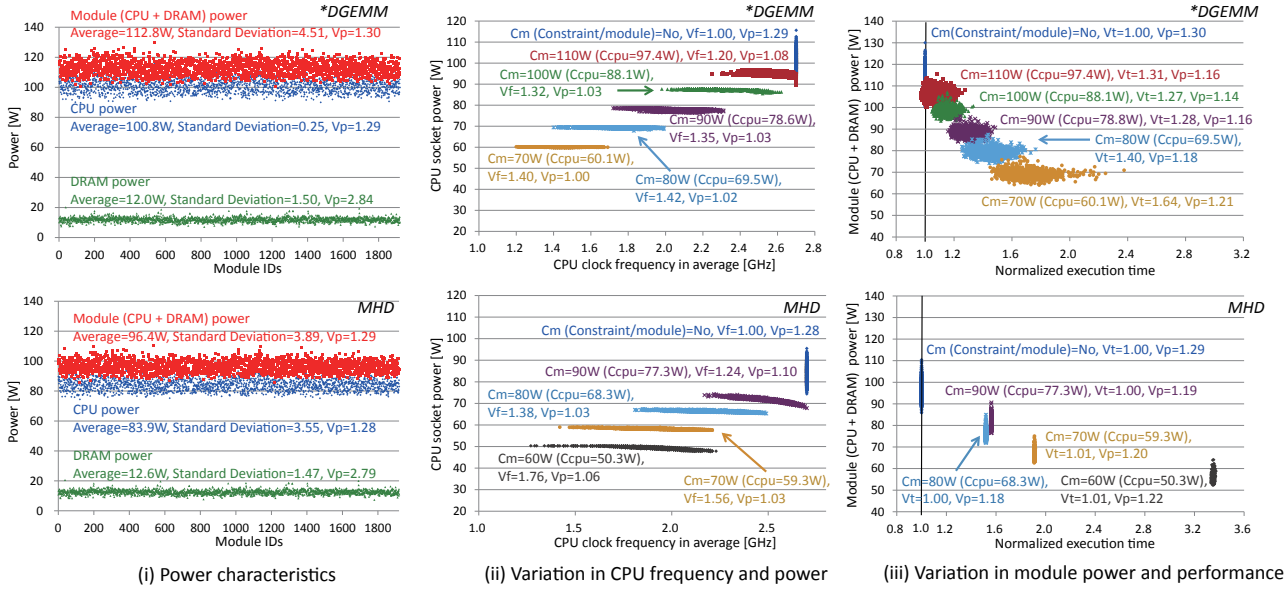

Figure 2: Module Power and Performance Variation the HA8K System

out power capping. We summarize graph notations in Table 3. The two graphs grouped as (i) report power consumption of each module and show a breakdown of CPU and DRAM power. *Vp*, or the worst-case power variation is calculated by dividing the maximum power value with the minimum power value in the appropriate set. We observe that *Vp* values at the module level are about 1.3, which means that there is a 30% difference in power consumption across modules even when they are running identical codes. For all our benchmarks, *Vp* values for module power ranged from 1.2 to 1.5. We also observe that DRAM power variation was significantly higher, with *Vp* values of about 2.8.

4.3 Impact on Application Performance

We now analyze how variation in power affects the CPU clock frequency when a power constraint is enforced. We use RAPL to enforce power caps, which adjusts the average power consumption of the CPU to that of the specified constraint for each time step (user-specified, typically 1 ms) by internally using techniques such as dynamic voltage and frequency scaling.

Table 3: Terminology

ID	Description
Cs	System-level power constraint
Cm	Module-level power constraint[2]
Ccpu	CPU power cap (determined statically)
Vp	Worst-case power variation
Vf	Worst-case CPU frequency variation
Vt	Worst-case execution time variation

The graphs grouped as (ii) in Figure 2 depict results of CPU frequency variation for *DGEMM and MHD. The x-axis is the average CPU frequency for a module across all

[2]For a naive scheme (defined in Section 6), and for Figure 2, $Cm = Cs/n$, where n is the number of modules. In case of our power budgeting algorithm though, each module will have a unique power allocation, P_{module} based on the application's characteristics and the manufacturing variability in the system, such that the average P_{module} across n modules is Cm.

RAPL time steps during the application's execution. The y-axis is the CPU power consumed. Here, *Cm* shows the power constraint applied to each module, and *Ccpu* is the CPU power cap enforced with RAPL. We analyze the CPU and DRAM power characteristics offline and determine *Ccpu* for each scenario. Similar to *Vp*, *Vf* is the worst-case variation in CPU frequency across the modules.

We observe that variation in power (when no power capping is enforced) translates to variation in CPU frequency under a power cap, leading to performance inhomogeneity in the system. This is a serious issue for HPC applications and future power-constrained systems. We also observe that the performance variation tends to become worse as we restrict power further. For instance, in *DGEMM, reducing the module power budget from 110 W to 70 W increases *Vf* from 1.20 to 1.40, indicating a 40% difference in CPU frequencies across modules at 70 W. In MHD, applying a 60 W module power cap results in 1.75X difference in CPU frequencies between the fastest and slowest CPUs. Note that this scenario makes a perfectly load-balanced application exhibit load imbalance under a power constraint. It is unclear how naturally load-imbalanced HPC applications will react to this phenomenon.

Finally, we consider the impact of variation in power on actual execution time of applications. The CPU-boundedness, memory characteristics and synchronization characteristics of an application will determine how much the overall performance impact will be. This is shown in the graphs grouped by (iii) in Figure 2. The x-axis shows the execution time of the application's MPI processes under a power constraint normalized to the execution time of the corresponding MPI processes when power was unconstrained. The y-axis denotes module power. *Vt* denotes the worst-case execution time variation across all MPI ranks.

There are two important observations. First, for *DGEMM, power capping results in up to 64% variation in per-rank performance ($Cm = 70W$), resulting in poor application performance. MHD, on the other hand, shows a different trend. There is no serious execution time variation despite the large amount of variation in CPU frequencies as discussed earlier. We attribute this to the periodic synchronizations in MHD, which tend to hide the variation in execution time across MPI ranks. Figure 3 shows 64-module MHD data and we observe that constraining power indeed affects CPU wait times and degrades performance. The x-axis represents the cumulative time spent by each rank of MHD in `MPI_Sendrecv` when exchanging data with neighboring ranks. *Vt* values are very high because for one process, the `MPI_Sendrecv` overhead is very small. For the remaining benchmarks, we observe that *STREAM has trends similar to *DGEMM, while NPB-BT, NPB-SP and mVMC are more similar to MHD.

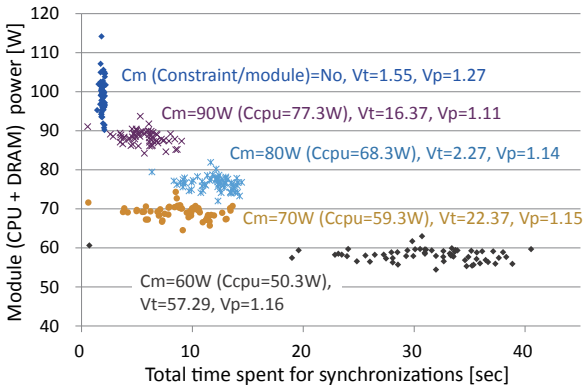

Figure 3: Module Power and Performance Variation in Synchronized Executions

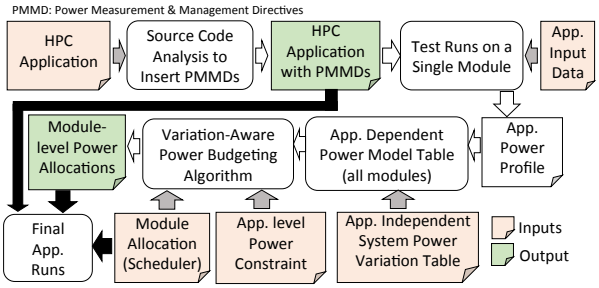

Figure 4: Framework of proposed power budgeting

5. VARIATION-AWARE POWER BUDGETING

As the previous section has shown, manufacturing variability can have a significant impact on application performance in power limited environments. We therefore need new mechanisms that can help detect and balance power to even out such variations. In this section, we present our variation-aware power budgeting algorithm that addresses this challenge. It is based on a Power Variation Table (PVT) that has to be constructed once per system and which then serves as the basis to estimate power variations in any target application. Once this table is computed, our power budgeting framework follows the workflow in Figure 4. The inputs to this framework include an HPC application, its associated data, an application-level power constraint imposed externally by the system, a list of modules (that is, physical processors) that were allocated by the job scheduler to the application, and the aforementioned PVT that describes the manufacturing variability on the given system. The output is a module-level power allocation to be applied to the given list of modules in order to maximize effective performance. A summary of the steps used in the framework is given below, and detailed explanations are provided in the subsections that follow.

1) Inserting Power Directives: First, Power Measurement and Management Directives (PMMDs) are inserted in the HPC application to facilitate analysis. We use the compiler-based instrumentation system from the TAU toolkit [48] and define the region of interest by inserting PMMDs just after `MPI_Init` and just before `MPI_Finalize`.

2) Single-module test run: We conduct two low-cost, single-module test runs of the application, one at the maximum CPU frequency and the other at the minimum CPU frequency, and measure the CPU and DRAM power.

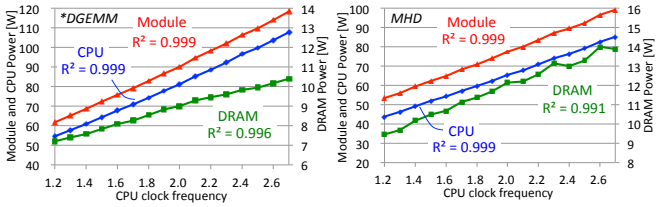

Figure 5: Power vs. CPU frequency on 64 HA8K modules

3) Power model calibration: We use the information from the single-module test run and a given, precomputed system-level PVT to create an application-dependent, variation-aware Power Model Table (PMT).

4) Power budgeting algorithm: We use the application's PMT and the given module list to determine the module-level power allocations that maximize the application's performance under the specified application-level power constraint. This includes determining module-level CPU frequencies and deciding a power constraint for each module in order to realize that frequency.

5) Final application run: The HPC application annotated with the PMMDs is executed on the given module list by using the module-level power allocations determined by the variation-aware power budgeting algorithm, using two implementation strategies: Power Capping (PC) with RAPL and Frequency Selection (FS) with `cpufrequtils`.

5.1 Budgeting Algorithm

In the following subsections, we detail Step 4, which is variation-aware power budgeting. We explain how we model power and determine the module-level power allocations given an application-level power constraint.

5.1.1 Modeling Power

In order to assign the appropriate amount of power to each module, we first need to develop a simple and accurate model to predict the power consumption of an application. Our model is based on the assumption that power consumption for both CPU and DRAM is proportional to the CPU frequency, which we have validated from the data we gathered (see Figure 5, which shows power consumption on y-axes and the CPU frequency on the x-axis).

Based on this assumption, for each new application, we conduct two single-module test runs at the maximum and the minimum CPU frequencies available (f_{max} and f_{min}). We represent the associated CPU and DRAM power consumption by P^{cpu}_{max}, P^{cpu}_{min}, P^{dram}_{max}, and P^{dram}_{min}. Then, for a known CPU frequency, f, we can predict the CPU, DRAM and module power (P^{cpu}, P^{dram},P^{module}) as shown below. Here, the coefficient α ($0 \leq \alpha \leq 1$) is a key parameter that is used to control the power-performance tradeoff.

$$f = \alpha(f_{max} - f_{min}) + f_{min} \tag{1}$$

$$P^{cpu} = \alpha(P^{cpu}_{max} - P^{cpu}_{min}) + P^{cpu}_{min} \tag{2}$$

$$P^{dram} = \alpha(P^{dram}_{max} - P^{dram}_{min}) + P^{dram}_{min} \tag{3}$$

$$P^{module} = P^{cpu} + P^{dram} \tag{4}$$

5.1.2 Module-Level Power Allocation

We assume that for most HPC applications, increasing the CPU frequency improves performance and we see this assumption hold for the vast majority of codes. Note that applications have already been tuned for memory performance and scalability, because the users have to determine the right number of modules (physical processors) to execute them when they submit their job to the scheduler. The variation-aware power budgeting algorithm strives to maximize effective performance under a power constraint, and its objective thus is:

To determine the maximum application-specific coefficient α such that the total power consumption across all modules does not exceed the given application-level power constraint, P_{budget}.

Given N modules and their associated module power at the maximum and minimum CPU frequencies ($P^{module}_{max,i}$ and $P^{module}_{min,i}$), we know that:

$$\sum_{i=1}^{N} \left(\alpha \left(P^{module}_{max,i} - P^{module}_{min,i} \right) + P^{module}_{min,i} \right) \leq P_{budget} \tag{5}$$

Hence, the maximum α for an application becomes:

$$\alpha \leq \frac{P^{budget} - \sum_i^N P^{module}_{min,i}}{\sum_i^N \left(P^{module}_{max,i} - P^{module}_{min,i} \right)} \tag{6}$$

Note that α is set to 1.0 when we do not have any power constraints, so that we can apply the f_{max} frequency. Additionally, it is important to note that α is application-specific and is common across all modules in order to ensure consistent performance.

After we determine α, the optimum power budget P^{module}_i, assigned to module i, is defined by the following equation.

$$P^{module}_i = \alpha \left(P^{module}_{max,i} - P^{module}_{min,i} \right) + P^{module}_{min,i} \tag{7}$$

The associated CPU power cap for the module is:

$$P^{cpu}_i = P^{module}_i - P^{dram}_i \tag{8}$$

(9)

5.2 Power Model Calibration

As discussed in Section 5.1, obtaining the maximum α under a given power constraint is the key challenge in our power budgeting scheme. To solve the optimization problem described in Equation (6), we need four application-specific parameters for each module—$P^{cpu}_{max,i}$, $P^{cpu}_{min,i}$, $P^{dram}_{max,i}$, and $P^{dram}_{min,i}$. The parameters depend on both application characteristics and manufacturing variability.

Because it is impractical to execute the application on all installed modules to obtain this information, we derive this information by using a previously measured, system-level Power Variation Table (PVT) and two single-module application runs. We refer to this step as *power model calibration.*

Figure 6 depicts this step. The PVT consists of N entries, each of which stores *variation scales* associated with the CPU and DRAM in each module in order to represent the degree of inter-module variation. The PVT is generated when the system is installed by executing representative microbenchmarks on each module. The power parameters, $P^{cpu}_{max,i}$, $P^{cpu}_{min,i}$, $P^{dram}_{max,i}$, and $P^{dram}_{min,i}$ are measured for each module, and the variation scales are obtained by dividing

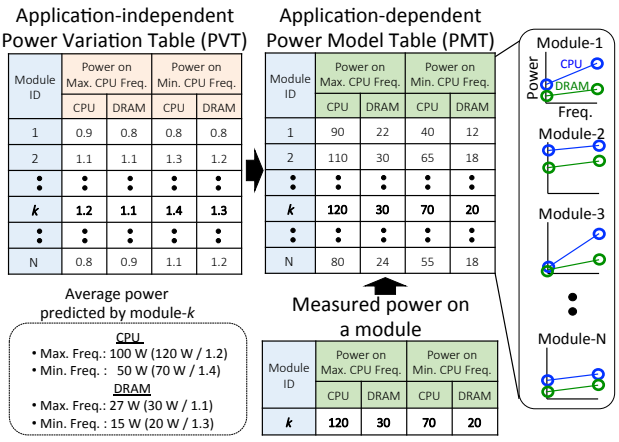

Figure 6: Power model calibration

each of these module power values by the respective average. For example, `Module-k` in Figure 6 has a variation scale of 1.2 for $P^{cpu}_{max,i}$. This indicates that `Module-k` consumes 1.2 times more maximum CPU power than the average across all modules. As the PVT is generated only once when the system is installed, it does not incur any overhead in the variation-aware power budgeting algorithm.

In order to generate the application-dependent Power Model Table (PMT), two single-module test runs are executed at the maximum and minimum CPU frequencies. We can then predict the four parameters for all modules by first determining the system-level average values for the parameters from the single-module tests, and then multiplying these average values with the variation scales from the PVT. For example, in Figure 6, if `Module-k` is used for the application test runs, and the measured CPU power with the maximum CPU frequency is 120 W, we first determine that the system-level average for the same is 100 W (because the variation scale for `Module-k` is 1.2), and then predict $P^{cpu}_{max,i}$ for all modules by multiplying the average by the corresponding variation scale. In this case, `Module-1`'s $P^{cpu}_{max,1}$ will be 90 W (100 W $\times$ 0.9). Once we have the PMT, we can derive module-level power constraints by using Equation 9.

5.3 Implementation and Accuracy

After we obtain module-level power allocations and CPU frequencies, we use the PMMDs to manage HPC application power. We have two simple implementations—Power Capping (PC) and Frequency Selection (FS). As explained earlier, our goal is to apply the best *common* CPU frequency that maximizes the application's performance under a given power constraint. PC attempts to *indirectly* control the CPU frequency by *directly* limiting the CPU power consumption. It is guaranteed that PC will never exceed the CPU power constraint because RAPL enforces strict power caps.

RAPL attempts to dynamically optimize the CPU frequency when a power cap is enforced, leading to CPU frequency throttling. This dynamic behavior does not guarantee consistent performance across modules, and may introduce load balancing concerns by itself. To address this issue, we propose the FS implementation, where a static frequency as determined by Equation (1) is applied to the modules. FS *directly* applies the determined CPU frequency by using `cpufrequtils`, and *indirectly* manages power consumption. While this guarantees consistent performance, it has the potential to violate the derived CPU power cap.

Table 4: Power constraints on *HA8K*

Cs [KW]	211	192	173	154	134	115	96
Ave. *Cm* [W]	110	100	90	80	70	60	50
**DGEMM*	✓	✓	✓	✓	✓	–	–
**STREAM*	•	✓	✓	✓	–	–	–
MHD	•	•	✓	✓	✓	✓	–
NPB-BT	•	•	•	✓	✓	✓	✓
NPB-SP	•	•	•	✓	✓	✓	✓
mVMC	•	•	•	✓	✓	✓	–

In our experiments, we use *STREAM as a microbenchmark to generate the PVT. The primary reason for choosing *STREAM was that it exhibited both memory and CPU boundedness, and consumed relatively large amounts of memory as well as CPU power. This allows us to capture power characteristics for both components in a fair manner at the same time with a single benchmark. For most of our benchmarks, the prediction error between the generated application-specific PMT and the measured power consumption for that application across all modules is under 5%. The exception was NPB-BT, which has a prediction error of about 10%.

6. EVALUATION

For our evaluation, we assume that the HPC applications under consideration are running on a dedicated system (HA8K with 1,920 modules) under different global power constraints without any other interference. This allows us test the limits of proposed variation-aware budgeting algorithm. Even though our framework supports multiple applications, determining what fraction of system power each application should be allocated is an orthogonal problem, which is part of our future work. Table 4 shows the applications and the system-level power constraints (Cs) that we consider. As discussed in Section 4, Cm is the expected average power constraint per module (and not the actual derived module-level power constraint), presented here to help readers understand the degree of variation-aware power capping intuitively. We had limited allocated time on HA8K, so we focused on specific, interesting scenarios in our experiments, which are marked with check-marks ("✓") in Table 4. The entries with bullet-marks ("•") represent scenarios which are not sufficiently power constrained from the point of view of the application's power profile resulting in no performance improvements (as no power capping is required), and the entries with hyphens ("–") indicate scenarios where the system is extremely power limited and the modules under consideration cannot be operated even with the minimum CPU frequency (f_{min}). We consider the following power allocation schemes in our evaluation.

- *Naïve* : This is an application-independent, variation-unaware power budgeting scheme that distributes power uniformly across all modules. For the PMT, the $P^{cpu}_{max,i}$ and $P^{dram}_{max,i}$ entries are the TDP values based on the architecture (130 W and 62 W respectively for our system).

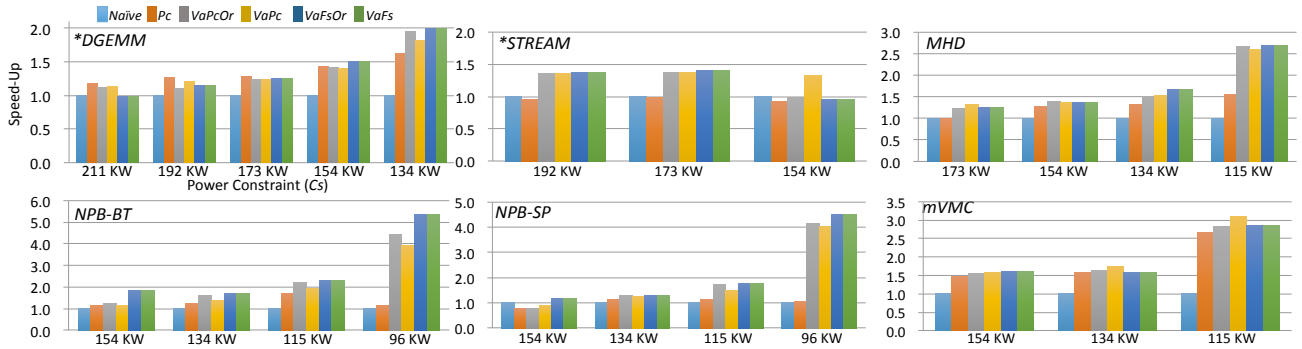

Figure 7: Speedup Compared to the Naïve Budgeting Scheme

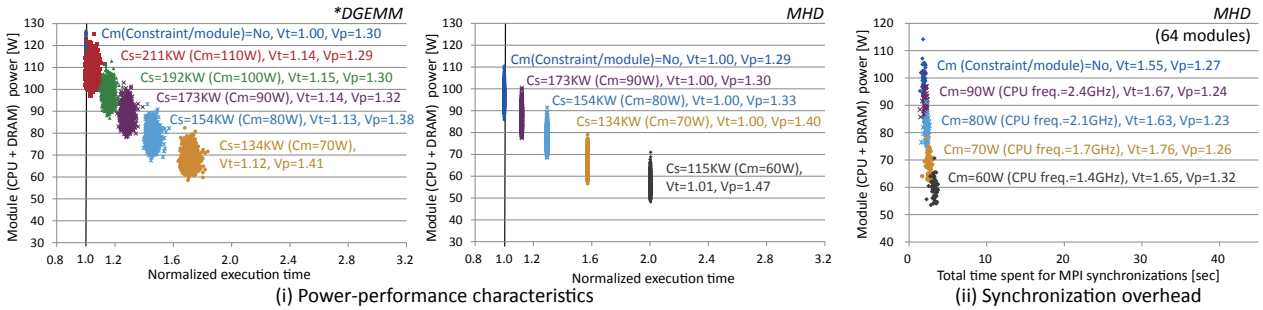

Figure 8: Detailed Power and Performance Results for the VaFs Budgeting Scheme

The $P^{cpu}_{min,i}$ and $P^{dram}_{min,i}$ entries are determined empirically. We note that for all our benchmarks, rapid degradation in performance occurs when the power allocated to the CPU goes below the threshold of 40 W. We assume this to be $P^{cpu}_{min,i}$. $P^{dram}_{min,i}$ is determined by measuring and averaging the DRAM power consumed at $P^{cpu}_{min,i}$, which in our case is 10 W. We use these values as input to Equation 6 to determine the appropriate α. *Naïve* is the baseline scheme in our evaluation.

- *Pc* : This is an application-dependent, variation-unaware scheme implemented via RAPL Power Capping (PC). In this case, we use the application-specific average values across all modules for to generate the PMT. As a result, power is distributed uniformly among the modules.

- *VaPc* : This is the application-dependent, variation-aware scheme implemented with PC using RAPL, as discussed in Section 5.

- *VaPcOr* : This is the same as *VaPc* except that we assume a perfect model calibration (oracle calibration). We obtain the PMT based on a complete execution of the HPC application on all modules.

- *VaFs* : This is the application-dependent, variation-aware scheme implemented with Frequency Selection (FS) using `cpufrequtils`, as discussed in Section 5.

- *VaFsOr* : This is the same as *VaFs* except that the model has a perfect calibration of CPU frequencies.

6.1 Performance Results

Figure 7 reports the speedup of all the aforementioned schemes when compared to *Naïve*. The FS-based scheme, *VaFs*, performs better than the other schemes in general. However, as discussed in Section 5.3, it may not adhere strictly to the derived CPU power constraint. With *VaFs*, we see a maximum performance improvement of 5.40X (for the NPB-BT benchmark with 96 KW), and an average improvement of 1.86X across all benchmarks. For *VaPc*, which strictly adheres to the given CPU power constraint, the maximum performance improvement is 4.03X (for NPB-SP at 96 KW), and the average improvement across all benchmarks is 1.72X.

VaFs almost always does better than *VaPc*, except for *STREAM with a 154 KW constraint and mVMC with a 115 KW constraint. Note that *VaPc* also outperforms the oracle results from *VaPcOr* in these two cases. We believe this can be attributed to the dynamic behavior of RAPL, which seems to be optimizing for performance in some way differently when different power caps are enforced. Another possible reason could be power and frequency mispredictions from our algorithm. Detailed analysis for these two cases is part of our ongoing work.

Figure 8 (i) shows the power-performance characteristics after applying *VaFs*. First, we compare the results for *DGEMM in Figure 8 (i) with Figure 2 (iii) from Section 4. We can observe that *VaFs* reduces the variation in execution time (Vt) by increasing the variation in power consumption (Vp), which is expected from this scheme. For instance, if we focus on the 70 W module power constraint, Vt and Vp

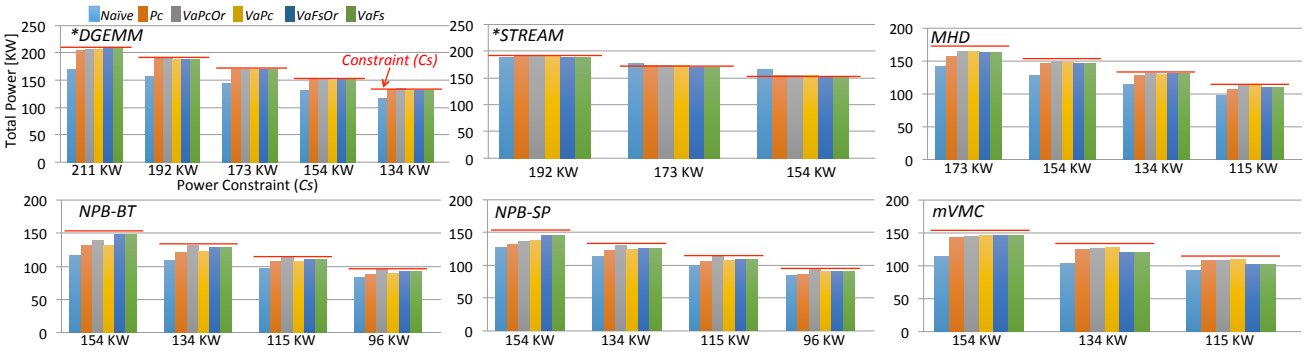

Figure 9: Total Power Consumption for All Budgeting Schemes

change from 1.64 to 1.12 and from 1.21 to 1.41, respectively. Figure 8 (i) also shows the results for MHD, where we can see a similar trend. Figure 8 (ii) depicts the cumulative synchronization time of each MPI process executed with *VaFs* on 64 modules, and we see that the variation problem in terms of synchronization overhead as explained in Figure 3 has been addressed, resulting in better performance.

Next, we compare the variation-aware schemes *VaPc* and *VaFs* to the variation-unaware conventional approach *Pc*. Except for *DGEMM, the proposed variation-aware approaches result in better performance. *Pc* does not work well when tight power constraints are imposed, particularly for *NPB-BT* and *NPB-SP* with power limitation of 96 KW. As explained in Section 4, tightening power restriction worsens the power variation problem, and *Pc* cannot address this issue as it is variation-unaware.

Finally, we discuss the effect of prediction-based power model calibration by comparing the proposed scheme *VaPc* to the oracle *VaPcOr*. We observe that *VaPc* can achieve similar performance improvements except for *NPB-BT* and **DGEMM* with the 134 KW power constraint. Recall from Section 5.3 that *NPB-BT* is the worst benchmark in terms of the power prediction accuracy. The mispredictions directly affect the power cap enforced by *VaPc*, thus impacting the CPU frequency and performance. In this paper we used only one microbenchmark (*STREAM) to generate the application-independent PVT. An approach to improve the prediction accuracy is to use micro-benchmarks with different characteristics to generate several PVTs, and then choose a suitable PVT based on the test runs.

6.2 Power Results

Figure 9 shows the total average power consumption measured via RAPL. The red lines in each graph show the enforced power constraint. We have confirmed that all schemes adhere to the power constraint in our results, except the *Naïve* scheme for **STREAM*. The main reason why *Naïve* cannot meet the power constraint is because it underestimates DRAM power as it does not take the application characteristics into account.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we first presented a detailed study of manufacturing variability on four production systems spanning different underlying architectures and established that variation in power can lead to up to 64% performance variation across HPC application ranks in power-limited environments. Then, we designed a low-cost, scalable, variation-aware power budgeting framework and implemented it using two techniques: power capping and frequency selection. Our experimental results on a large-scale, 1,920 module production system show up to 5.4X improvement in HPC application performance under a power constraint, and an average improvement of 1.8X across all benchmarks when compared to a variation-unaware, naive power allocation scheme.

In our present work, we focused on HPC applications executing on a dedicated 1,920-module system. Future research includes analyzing multiple applications under a system-level power constraint and optimizing for overall system throughput. This involves integrating our work with a power-aware resource manager such as RMAP [40], which can determine application-level power constraints and physical node allocations in a fair yet intelligent manner by using hardware overprovisioning [39,46]. We also want explore dynamic reallocation of power within and between HPC applications by analyzing their phase behavior in order to improve system throughput and power efficiency further.

8. ACKNOWLEDGMENTS

We extend our thanks to Livermore Computing and RIIT of Kyushu University for providing us the resources and support to conduct the large-scale power measurements presented in this paper. We also want to thank Timothy Meyer and Neha Gholkar for their initial help with gathering data on the Teller and Vulcan systems. Part of this work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-CONF-669812). This material was also based upon work supported by National Science Foundation under Grant No. 1216829. Additionally, this work was supported by the Japan Science and Technology Agency (JST) CREST program, *A Power Management Framework for Post Peta-Scale Supercomputers.*

9. REFERENCES

[1] NASA Advanced Supercomputing Division, NAS Parallel Benchmark Suite v3.3. 2006. `http://www.nas.nasa.gov/Resources/Software/npb.html`.

[2] 2013 Exascale Operating and Runtime Systems. Technical report, Advanced Science Computing

Research (ASCR), February 2013. http://science.doe.gov/grants/pdf/LAB13-02.pdf.

[3] AMD. AMD Turbo CORE Technology. http://www.amd.com/us/products/desktop/processors/phenom-ii/Pages/phenom-ii-key-architectural-features.aspx.

[4] S. Ashby, P. Beckman, J. Chen, P. Colella, B. Collins, D. Crawford, J. Dongarra, D. Kothe, R. Lusk, P. Messina, T. Mezzacappa, P. Moin, M. Norman, R. Rosner, V. Sarkar, A. Siegel, F. Streitz, A. White, and M. Wright. The Opportunities and Challenges of Exascale Computing. 2010.

[5] P. Bailey, D. Lowenthal, V. Ravi, B. Rountree, M. Schulz, and B. de Supinski. Adaptive Configuration Selection for Power-Constrained Heterogeneous Systems. In *International Conference on Parallel Processing*, ICPP '14, 2014.

[6] D. Bodas, J. Song, M. Rajappa, and A. Hoffman. Simple Power-aware Scheduler to Limit Power Consumption by HPC System Within a Budget. In *Proceedings of the 2nd International Workshop on Energy Efficient Supercomputing*, pages 21–30. IEEE Press, 2014.

[7] S. Borkar. Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation. *Micro, IEEE*, 25(6):10–16, Nov 2005.

[8] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De. Parameter Variations and Impact on Circuits and Microarchitecture. In *Proceedings of the 40th annual Design Automation Conference*, pages 338–342, June 2003.

[9] K. W. Cameron, X. Feng, and R. Ge. Performance-constrained, Distributed DVS Scheduling for Scientific Applications on Power-aware Clusters. In *Supercomputing*, November 2005.

[10] N. P. Carter, A. Agrawal, S. Borkar, R. Cledat, H. David, D. Dunning, J. B. Fryman, I. Ganev, R. A. Golliver, R. C. Knauerhase, R. Lethin, B. Meister, A. K. Mishra, W. R. Pinfold, J. Teller, J. Torrellas, N. Vasilache, G. Venkatesh, and J. Xu. Runnemede: An Architecture for Ubiquitous High-Performance Computing. In *19th IEEE International Symposium on High Performance Computer Architecture, HPCA 2013, Shenzhen, China, February 23-27, 2013*, pages 198–209, 2013.

[11] M. Curtis-Maury, F. Blagojevic, C. D. Antonopoulos, and D. S. Nikolopoulos. Prediction-Based Power-Performance Adaptation of Multithreaded Scientific Codes. *IEEE Trans. Parallel Distrib. Syst.*, 19(10):1396–1410, Oct. 2008.

[12] H. David, E. Gorbatov, U. Hanebutte, R. Khanna, and C. Le. RAPL: Memory Power Estimation and Capping. In *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design*, ISLPED '10, pages 189–194, 2010.

[13] D. DeBonis, J. H. Laros III, and K. Pedretti. Qualification for PowerInsight Accuracy of Power Measurements.

[14] S. Dighe, S. Vangal, P. Aseron, S. Kumar, T. Jacob, K. Bowman, J. Howard, J. Tschanz, V. Erraguntla, N. Borkar, V. De, and S. Borkar. Within-Die Variation-Aware Dynamic-Voltage-Frequency-Scaling With Optimal Core Allocation and Thread Hopping for the 80-Core TeraFLOPS Processor. *Solid-State Circuits, IEEE Journal of*, 46(1):184–193, Jan 2011.

[15] D. A. Ellsworth, A. D. Malony, B. Rountree, and M. Schulz. POW: System-wide Dynamic Reallocation of Limited Power in HPC. June 2015.

[16] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger. Power Challenges May End the Multicore Era. *Commun. ACM*, 56(2):93–102, Feb. 2013.

[17] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger. Dark Silicon and the End of Multicore Scaling. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, pages 365–376, New York, NY, USA, 2011. ACM.

[18] M. Etinski, J. Corbalan, J. Labarta, and M. Valero. Optimizing Job Performance Under a Given Power Constraint in HPC Centers. In *Green Computing Conference*, pages 257–267, 2010.

[19] M. Etinski, J. Corbalan, J. Labarta, and M. Valero. Linear Programming Based Parallel Job Scheduling for Power Constrained Systems. In *International Conference on High Performance Computing and Simulation*, pages 72–80, 2011.

[20] D. Feitelson, U. Schwiegelshohn, and L. Rudolph. Parallel Job Scheduling - A Status Report. In *In Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, 2004.

[21] R. Ge, X. Feng, W. Feng, and K. W. Cameron. CPU MISER: A Performance-Directed, Run-Time System for Power-Aware Clusters . In *International Conference on Parallel Processing*, 2007.

[22] L. R. Harriott. Limits of lithography. *Proceedings of the IEEE*, 89(3):366–374, 2001.

[23] C. Hsu and W. Feng. A Power-Aware Run-Time System for High-Performance Computing. In *Supercomputing*, November 2005.

[24] J. Hu, C. J. Xue, Q. Zhuge, W.-C. Tseng, and E.-M. Sha. Towards Energy Efficient Hybrid On-chip Scratch Pad Memory with Non-volatile Memory. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*, pages 1–6. IEEE, 2011.

[25] J. H. L. III, P. Pokorny, and D. Debonis. PowerInsight - A commodity power measurement capability. In *IGCC'13*, pages 1–6, 2013.

[26] Intel. Intel Turbo Boost Technology 2.0. http://www.intel.com/content/www/us/en/architecture-and-technology/turbo-boost/turbo-boost-technology.html.

[27] Intel. Intel-64 and IA-32 Architectures Software Developer's Manual, Volumes 3A and 3B: System Programming Guide. 2011.

[28] S. Jilla. Minimizing The Effects Of Manufacturing Variation During Physcial Layout. http://chipdesignmag.com/display.php?articleId=2437.

[29] L. Kalé and S. Krishnan. CHARM++: A Portable Concurrent Object Oriented System Based on C++. In *Proceedings of OOPSLA'93*. ACM Press, September 1993.

[30] N. Kappiah, V. W. Freeh, and D. K. Lowenthal. Just

In Time Dynamic Voltage Scaling: Exploiting Inter-Node Slack to Save Energy in MPI Programs. *Journal of Parallel and Distributed Computing*, 68:1175–1185, 2008.

[31] T. Karnik, M. Pant, and S. Borkar. Power management and delivery for high-performance microprocessors. In *The 50th Annual Design Automation Conference 2013, DAC '13, Austin, TX, USA, May 29 - June 07, 2013*, page 159, 2013.

[32] H. Kaul, M. Anders, S. Hsu, A. Agarwal, R. Krishnamurthy, and S. Borkar. Near-threshold Voltage (NTV) Design: Opportunities and Challenges. In *The 49th Annual Design Automation Conference 2012, DAC '12*, June 2012.

[33] Z. Liu, J. Lofstead, T. Wang, and W. Yu. A case of system-wide power management for scientific applications. In *Cluster Computing (CLUSTER), 2013 IEEE International Conference on*, September 2013.

[34] P. Luszczek, D. Bailey, J. Dongarra, J. Kepner, R. Lucas, R. Rabenseifner, and D. Takahash. HPC Challenge Benchmark Suite. `ttp://icl.cs.utk.edu/`pcc/index.html.

[35] A. Marathe, P. Bailey, D. K. Lowenthal, B. Rountree, M. Schulz, and B. R. de Supinski. A Run-time System for Power-constrained HPC Applications. In *In International Supercomputing Conference (ISC*, July 2015.

[36] N. Maruyama, S. Suzuki, K. Mikami, Y. Komuro, S. Takizawa, and M. Matsuda. Fiber Miniapp Suite. `fiber-miniapp.github.io`.

[37] S. Miwa, S. Aita, and H. Nakamura. Performance Estimation of High Performance Computing Systems with Energy Efficient Ethernet Technology. *Computer Science - Research and Development*, 29:161–169, August 2014.

[38] T. Ogino, R. J. Walker, and M. Ashour-Abdalla. A Global Magnetohydrodynamic Simulation of the Magnetopause when the Interplanetary Magnetic Field is Northward. *IEEE Transaction on Plasma Science*, 20:817–828, December 1992.

[39] T. Patki, D. K. Lowenthal, B. Rountree, M. Schulz, and B. R. de Supinski. Exploring Hardware Overprovisioning in Power-constrained, High Performance Computing. In *International Conference on Supercomputing*, pages 173–182, 2013.

[40] T. Patki, A. Sasidharan, M. Maiterth, D. Lowenthal, B. Rountree, M. Schulz, and B. de Supinski. Practical Resource Management in Power-Constrained, High Performance Computing. In *High Performance Parallel and Distributed Computing (HPDC)*, June 2015.

[41] B. Rountree, D. H. Ahn, B. R. de Supinski, D. K. Lowenthal, and M. Schulz. Beyond DVFS: A First Look at Performance under a Hardware-Enforced Power Bound. In *IPDPS Workshops (HPPAC)*, pages 947–953. IEEE Computer Society, 2012.

[42] B. Rountree, D. Lowenthal, B. de Supinski, M. Schulz, V. Freeh, and T. Bletch. Adagio: Making DVS Practical for Complex HPC Applications. In *International Conference on Supercomputing*, June 2009.

[43] B. Rountree, D. Lowenthal, M. Schulz, and B. de Supinski. Practical Performance Prediction Under Dynamic Voltage Frequency Scaling. In *International Green Computing Conference*, July 2011.

[44] S. Samaan. The Impact of Device Parameter Variations on the Frequency and Performance of VLSI Chips. In *Computer Aided Design, 2004. ICCAD-2004. IEEE/ACM International Conference on*, pages 343–346, Nov 2004.

[45] V. Sarkar, W. Harrod, and A. Snavely. Software Challenges in Extreme Scale Systems. In *Journal of Physics, Conference Series 012045*, 2009.

[46] O. Sarood. *Optimizing Performance Under Thermal and Power Constraints for HPC Data Centers*. PhD thesis, University of Illinois, Urbana-Champaign, December 2013.

[47] O. Sarood, A. Langer, A. Gupta, and L. V. Kale. Maximizing Throughput of Overprovisioned HPC Data Centers Under a Strict Power Budget. In *Supercomputing*, Nov. 2014.

[48] S. Shende and A. D. Malony. The Tau Parallel Performance System. *IJHPCA*, 20(2):287–311, 2006.

[49] K. Shoga, B. Rountree, and M. Schulz. Whitelisting MSRs with msr-safe. November 2014.

[50] R. Teodorescu and J. Torrellas. Variation-Aware Application Scheduling and Power Management for Chip Multiprocessors. In *Computer Architecture, 2008. ISCA '08. 35th International Symposium on*, pages 363–374, June 2008.

[51] E. Totoni, A. Langer, J. Torrellas, and L. Kale. Scheduling for HPC Systems with Process Variation Heterogeneity. January 2015.

[52] J. Tschanz, J. Kao, S. Narendra, R. Nair, D. Antoniadis, A. Chandrakasan, and V. De. Adaptive Body Bias for Reducing Impacts of Die-to-die and Within-die Parameter Variations on Microprocessor Frequency and Leakage. *Solid-State Circuits, IEEE Journal of*, 37(11):1396–1402, Nov 2002.

[53] S. Wallace, V. Vishwanath, S. Coghlan, Z. Lan, and M. E. Papka. Measuring power consumption on IBM Blue Gene/Q. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International*, pages 853–859. IEEE, 2013.

[54] S. Wallace, V. Vishwanath, S. Coghlan, J. Tramm, Z. Lan, and M. Papkay. Application power profiling on IBM Blue Gene/Q. In *Cluster Computing (CLUSTER), 2013 IEEE International Conference on*, pages 1–8. IEEE, 2013.

[55] K. Yoshii, K. Iskra, R. Gupta, P. Beckman, V. Vishwanath, C. Yu, and S. Coghlan. Evaluating power-monitoring capabilities on IBM Blue Gene/P and Blue Gene/Q. In *Cluster Computing (CLUSTER), 2012 IEEE International Conference on*, pages 36–44. IEEE, 2012.