Exploring Hardware Overprovisioning in Power-Constrained, High Performance Computing

Tapasya Patki
Dept. of Computer Science
The University of Arizona
tpatki@cs.arizona.edu

David K. Lowenthal
Dept. of Computer Science
The University of Arizona
dkl@cs.arizona.edu

Barry Rountree
Lawrence Livermore
National Laboratory
rountree@llnl.gov

Martin Schulz
Lawrence Livermore
National Laboratory
schulzm@llnl.gov

Bronis de Supinski
Lawrence Livermore
National Laboratory
bronis@llnl.gov

ABSTRACT

Most recent research in power-aware supercomputing has focused on making individual nodes more efficient and measuring the results in terms of flops per watt. While this work is vital in order to reach exascale computing at 20 megawatts, there has been a dearth of work that explores efficiency at the whole system level. Traditional approaches in supercomputer design use *worst-case* power provisioning: the total power allocated to the system is determined by the maximum power draw possible per node. In a world where power is plentiful and nodes are scarce, this solution is optimal. However, as power becomes the limiting factor in supercomputer design, worst-case provisioning becomes a drag on performance.

In this paper we demonstrate how a policy of *overprovisioning* hardware with respect to power combined with intelligent, hardware-enforced power bounds consistently leads to greater performance across a range of standard benchmarks. In particular, leveraging overprovisioning requires that applications use effective *configurations*; the best configuration depends on application scalability and memory contention. We show that using overprovisioning leads to an average speedup of more than 50% over worst-case provisioning.

1. INTRODUCTION

The reality of power-limited supercomputing has begun to transform the community's understanding of performance. The still highly influential standard has been flops measured on the Linpack benchmark [4]. *Flops per watt*, epitomized by the Green 500 [11] list of supercomputers, is a new metric that may soon carry more importance. However, flops-per-watt varies widely across applications, and designing a system that optimizes it for any single benchmark leads to inefficient and suboptimal performance for other benchmarks.

The architecture community has addressed a similar design problem on single nodes (and embedded systems) by a process of *overprovisioning* hardware [19]. For example, in recent chip designs (e.g., Nehalem), more cores exist on the processor than can simultaneously run at the highest CPU clock frequency (due to power constraints). The user (or system, or hardware) has a choice among several different combinations of clock frequency and active core counts, with smaller core counts allowing higher frequencies.

The U.S. Department of Energy's goal of exascale computing with 20 megawatts or less means that the power that can physically be brought to the machine room will constrain future systems. Further, typical estimates are that each MW-year costs $1M so supercomputing centers can only afford a limited amount of power. Thus, we should also consider overprovisioning for high-performance computing (HPC).

With such constraints, we can guarantee full power to a restricted number of nodes (*worst case provisioning*) or we must limit the power to more nodes (*overprovisioning*). While an overprovisioned supercomputer cannot execute each node at full power, it could achieve better overall performance in terms of application execution time than a system with fewer fully-powered nodes. However, to achieve better performance on overprovisioned systems, we require new strategies to manage applications such that the best overall performance is achieved—while not exceeding the system-wide power bound. In particular, we must understand how different applications behave under enforced power limitations and how application configurations influence the utilization of the available power. For instance, under a power bound, an application's scalability characteristics determine whether we want to use fewer nodes at higher power per node or more nodes at lower power per node. Future systems are even likely to go a step further and allow power to be directed to individual components; e.g., if the code is CPU-bound, then cores can be run faster while reducing power to memory and other node componenents.

In this paper we provide, as far as we know, the first extensive study on real hardware that explores how HPC applications behave under a power bound and how application configurations influence the performance in overprovisioned systems. We analyze strongly-scaled HPC benchmarks on the

rzmerl cluster at LLNL, which has 162 Intel Sandy Bridge nodes. We use Intel's RAPL interface to enable and to enforce power limits and to study their effects on performance using several benchmarks.

This paper makes the following contributions:

- We study the effect of a power bound on a set of high performance computing applications;
- We show that the optimal configuration for a given application depends on its parallel efficiency and memory intensity as well as the particular power bound;
- We show that using overprovisioning leads to an average speedup of more than 50% over worst-case provisioning on four applications (BT-MZ, LU-MZ, SP-MZ, and SPhot).

The rest of this paper is organized as follows. Section 2 provides the overall approach for our study. Section 3 introduces Intel's RAPL interface for power clamping, Sections 4, 5, and 6 describe our experimental details, baseline power results, and multiple-node results, respectively. Next, Section 7 describes related work, Section 8 discusses future opportunities in overprovisioned supercomputing, and Section 9 provides concluding remarks.

2. OVERPROVISIONING

As mentioned above, an overprovisioned system is one in which we cannot simultaneously power all components at peak power. Overprovisioned systems will become more common as we hit what Venkatesh et al. term the *utilization wall* [35]. We expect that as transistor switching speeds increase, we will not be able to dissipate the heat at the same rate. Thus, the fraction of the chip that can be executed at full speed is monotonically decreasing; the part of the chip that must remain unused is referred to as *dark silicon* [2]. This fact has given rise to dynamic overclocking features in modern processor architectures, such as Intel's Turbo Boost [15] and AMD's Turbo CORE [5]. Processors are now designed to be *overprovisioned* with respect to power— the CPU frequency at which a core executes depends on the number of active cores, and not all cores can simultaneously run at the highest frequency.

The same idea should be applied to supercomputing. Exascale systems will have a power budget; the current bound set by DoE is 20 MW. Overprovisioning in the supercomputing context means that not all nodes in the facility can execute at peak power simultaneously. As opposed to worst-case provisioning, overprovisioning is advantageous because it can allow, for example, both highly scalable and less scalable jobs to perform well. In particular, an application's scalability determines whether we should use fewer nodes at higher power per node or more nodes at lower power per node. Also, when only a few nodes in the application are on the critical path, running all nodes at peak power might be wasteful. In addition, depending on an application's CPU and memory usage, we can choose to use fewer cores per node or to allocate component power within a node (that is, power to the packages and the memory subsystem) based on utilization. While overprovisioning means that the supercomputing center buys more compute capacity can be used, it allows the user to customize the system to an application and thereby to achieve better performance.

As overprovisioning becomes more common, the HPC community will need to address the performance challenges that arise. Power must be scheduled more carefully among the hardware resources in an HPC cluster. These resources include the racks in the machine room, the nodes within the racks, the components within a node and the interconnect; and, a fixed system-level power-constraint on the cluster will hierarchically translate to a node-level power-constraint, possibly in a non-uniform manner.

In order to explore overprovisioning and its effect on application performance, we address the following question in this paper: given a machine with n nodes and c cores per node, a cluster-level power bound P on the machine, and a strongly-scaled HPC application, how important is choosing the optimal *configuration*? We define a configuration as: (1) a value for n, (2) a value for c, (3) an amount of power p to be allocated to each node. The constraint is that the total power consumed must be no more than the bound P, and the goal is to minimize application runtime.

We explore this issue through a series of experiments on a 162 node Intel Sandy Bridge cluster at LLNL and use Intel's RAPL interface to enforce various power bounds on a wide range of HPC applications and benchmarks. Thus, the cluster is effectively an overprovisioned system.

3. INTEL'S RAPL INTERFACE: RUNNING AVERAGE POWER LIMIT

The Intel Sandy Bridge processor family supports on-board power measurement and power clamping at a fine granularity through a novel interface named Running Average Power Limit (RAPL) [16]. The architecture has four power *domains—Package* (PKG), *Power Plane 0* (PP0), *Power Plane 1* (PP1) and *DRAM*. The PKG domain represents the processor die, and the DRAM domain includes directly attached memory. PP0 covers mostly the cores, while PP1 covers uncore devices (like the off-chip last-level cache or the Intel QuickPath Interconnect). The Sandy Bridge architecture has two processor *models*, the *client* (family=0x06, model=0x2A) and the *server* (family=0x06, model=0x2D). The client model supports the PKG, PP0 and the PP1 domains, and the server model supports the PKG, PP0 and DRAM domains.

A series of Machine Specific Registers (MSRs) implement RAPL. To access these MSRs, developers can use the *msr* kernel module. This module exports a file interface that can be used to read from or write to the MSRs given appropriate permissions using the `readmsr` and `writemsr` instructions. The RAPL interface includes the `MSR_RAPL_POWER_UNIT` read-only register that exposes the units for power, energy and time in Watts, Joules and Seconds respectively, at architecture-specific precision. We use server model Sandy Bridge nodes with units of 0.125W, 0.0000152J and 0.000977 seconds.

Power Measurement:. Each domain has a 32-bit, read-only `ENERGY_STATUS` MSR, which is updated approximately every millisecond. Because the unit of joules is small, this is expected to roll over within hours. Average power can be calculated using this MSR and the elapsed time.

Power Clamping. Each domain has a `POWER_LIMIT` MSR that can specify a time window and a maximum average-power value. The hardware ensures that the average power over the specified time window does not exceed the power value. The `POWER_INFO` set of registers provide information on the *thermal specification power*, the lowest power bound and the largest time window supported. For our system, for the PKG domain, the lowest power bound is 51W, the thermal specification is 115W, the largest possible time window is 0.0459 seconds, and the maximum power rating is 180W.

The `librapl` *library.* To analyze HPC applications under a power bound, we developed the *librapl* library to facilitate safe user-space access of Intel's RAPL MSRs. This library uses the MPI profiling layer to intercept `MPI_Init()` and `MPI_Finalize()` calls to set up the necessary MSRs. Thus, we do not modify application source code. The library can also sample MPI programs at a desired time interval and intercept every MPI call in the application to gather timing and energy/power information. It further infers the operating frequency for each core using the `APERF` and `MPERF` MSRs [16]. Our library can be downloaded from `https://github.com/tpatki/librapl` and is currently in use at Purdue University, the University of Illinois at Urbana-Champaign, LLNL, and the University of Arizona.

4. EXPERIMENTAL AND APPLICATION DETAILS

In all results in this paper, we report power values in Watts and timing information in seconds. We conduct our experiments on the *rzmerl* cluster at LLNL, which is a 162-node Sandy Bridge cluster. Each node is a 062D server model with two PKG domains, and 8 cores per package. The memory per node is 32GB. The clock speed is 2.6 GHz, and the maximum turbo frequency is 3.3 GHz. The cluster has a 32-node per job limit. We use MVAPICH2 version 1.7 and compile all codes with the Intel compiler version 12.1.5. OpenMP threads were scheduled using the scatter policy by setting the `KMP_AFFINITY` environment variable. Power clamping for PP0 and DRAM was disabled by default. We run all experiments with power clamping on the PKG domain with a single clamping window and the shortest possible time window (0.000977 seconds). Thus, we avoid any potential power spikes that might result when using larger time windows. We use four PKG power caps: 51 W; 65 W; 80 W; and 95 W. We run each configuration at least three times to eliminate noise. We disable Turbo Boost when power capping.

We also run experiments with turbo enabled without using any RAPL-enforced power bounds. A cap of 115 W in our results represents this *turbo mode*. 115 W corresponds to the thermal limit on the PKG and, when we use turbo mode, this thermal limit becomes our power cap by default.

Our experiments use a hybrid MPI/OpenMP model. Hybrid models have low intra-node communication overhead and are more flexible in terms of configurations that we can test. Also, future architectures are likely to have many integrated cores on a single chip (for instance, Intel's MIC [14]). Because MPI processes have significant memory and communication overhead, the hybrid model is more likely than a flat MPI model.

HPC Applications. We used four HPC applications: SPhot [1] from the ASC Purple suite [20] and BT-MZ, SP-MZ, and LU-MZ from the NAS suite [3]. SPhot is a 2D photon transport code that uses a Monte Carlo approach to solve the Boltzmann transport equation by mimicking the behavior of photons as they are born in hot matter and move through different materials. SPhot is a CPU-bound, embarrassingly parallel application. For our multiple-node experiments, Nruns was set to 8192. For the single-node experiments, Nruns was set to 1024. The NAS Multi-zone parallel benchmarks (NAS-MZ) are derived from Computational Fluid Dynamics (CFD) applications and are designed to evaluate the hybrid model. We use all three NAS-MZ benchmarks: Block Tri-diagonal solver, or BT-MZ; Scalar Penta-diagonal solver, or SP-MZ; and Lower-Upper Gauss-Seidel solver, or LU-MZ. We use the Class C inputs.

Synthetic Benchmarks. In order to cover the extreme cases in the application space, we developed four MPI/OpenMP synthetic benchmarks. These tests are (1) CPU-bound and scalable (SC); (2) CPU-bound and not scalable (NSC); (3) Memory-bound and scalable (SM); (4) Memory-bound and not scalable (NSM). The CPU-bound benchmarks run a simple spin loop, and the Memory-bound benchmarks do a vector copy in reverse order. We control scalability by adding communication using `MPI_Alltoall()` (i.e., fewer calls to `MPI_Alltoall()` means better scalability).

5. BASELINE POWER RESULTS

We first present details on the effect of power on our applications to understand the impact of power limitations. We run our benchmarks on one node, varying the core count from 4 to 16, to study application characteristics and their impact on configurations.

Figure 1 shows the average PKG and DRAM power measured across the two sockets of the node for our benchmarks, at 4 and 16 cores and in turbo mode. We observe that some applications are more memory-intensive than others and hence consume more DRAM power. Examples of these are BT-MZ, SP-MZ, LU-MZ, and SM. At 16 cores per node, SP-MZ used 10.3% of its socket power for memory. Similarly, BT-MZ and LU-MZ used about 7-8% of their socket power for memory. SPhot, on the other hand, is relatively CPU intensive. We also observe that moving from 4 to 16 cores affects PKG power more than DRAM power. For instance, SPhot used 44.6 W of PKG power at 4 cores and 82.1 W of PKG power at 16 cores; DRAM power increased from 4.9 W to 5.7 W. Similarly, for BT-MZ, PKG power increased by 93% from 54.5 W to 105.7 W, but DRAM power only increased by 31%.

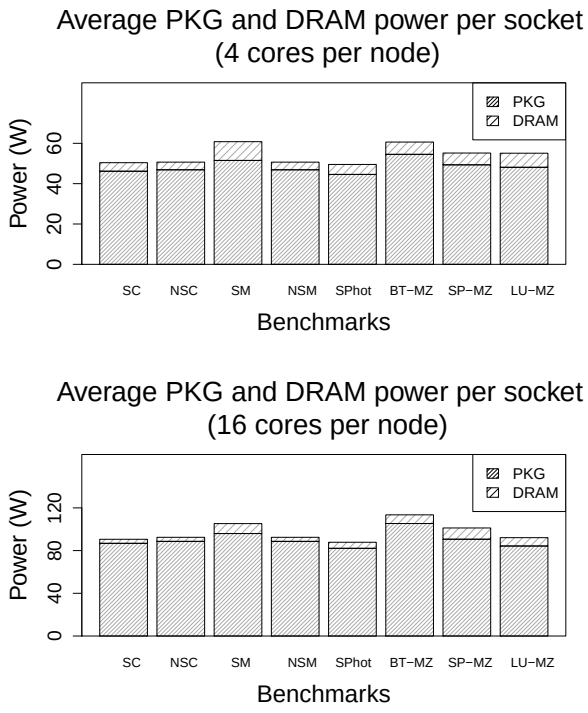


Figure 1: Average PKG and DRAM power consumption per socket

Figure 1 also shows that applications do not always use their allocated power. While the thermal limit on the PKG in turbo mode is 115 W, none of the applications actually use this much power, even when using all 16 cores. Most applications need between 80 and 90 W, except for BT-MZ, which uses nearly 106 W when running 16 cores. This important observation calls for efficient power allocation.

Figure 2 shows the impact of varying the PKG power bound on the node on application performance at 4 and 16 cores per node for SPhot and SP-MZ. At 4 cores per node, neither of the applications uses more than 51 W of PKG power; thus, in an ideal situation, when Turbo Boost is disabled, application performance should be unaffected as we increase power from 51 W to 100 W. We observe a slight slowdown in performance of about 1-2% for our applications from 51 W to 100 W with 4 cores per node. Since we run each experiment at least three times to eliminate noise, overhead introduced by Intel's RAPL algorithm (which is different at different power bounds) may cause this slowdown. At 16 cores per node, our applications benefit from higher power up to 85 W. Performance improved by 21.3% for SP-MZ and by 17.1% for SPhot. In turbo mode (115 W), all applications perform significantly better as they run at a higher CPU frequency. Performance for SPhot improved more than SP-MZ, primarily because SPhot is CPU-bound and less memory intensive.

Next, we run single-node and multiple-node experiments with Turbo Boost to determine its effect on configurations. We collect samples every second with *librapl* and measure

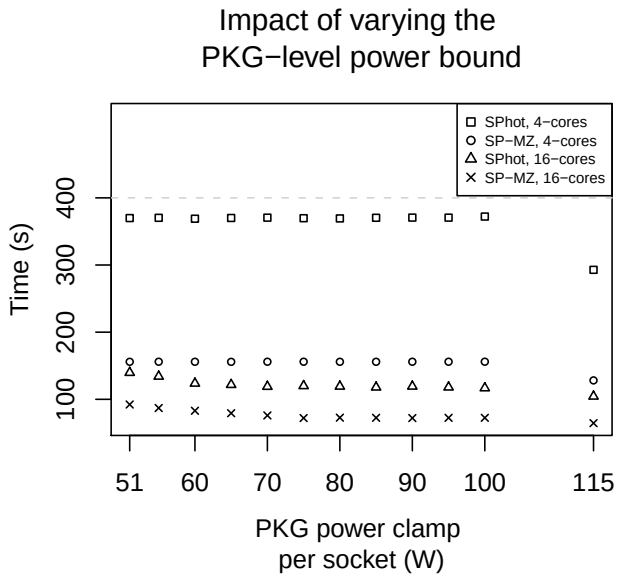


Figure 2: Impact of varying the PKG power clamp on execution time.

the frequency ratio by reading the `APERF` and `MPERF` MSRs [16]. Figure 3 shows our results on a single-node with varying core count on our CPU-bound, scalable synthetic benchmark. We multiply the median frequency ratio value from the samples with the maximum non-turbo frequency (2.6 GHz) to determine the effective turbo frequency. Frequency ratios vary little across our samples. Our results indicate that the effective turbo frequency depends on the number of active cores. Intel documentation [15] confirms this observation and also mentions that the turbo frequency varies with temperature. We run our experiments at LLNL, where the machine room temperature was fairly constant over time and do not encounter any variation in the turbo frequency that could be attributed to temperature.

We also run multiple-node experiments with Turbo Boost enabled and collect frequency samples every second with *librapl*. In these experiments, we use the same number of cores per node. All nodes reach the same turbo frequency, which is stable throughout the application's execution. All nodes engage in turbo mode similarly.

Summary. Power utilization varies between applications and power affects them differently. Thus, multiple-node overprovisioning could improve performance.

6. MULTIPLE NODE OVERPROVISIONING

This section presents and analyzes multiple-node results of the HPC applications and synthetic benchmarks that we listed in Section 4. Because we could not feasibly run every possible configuration and because *rzmerl* has a 32-node limit per job, we run experiments with 8 to 32 nodes and 4 to 16 cores per node, in increments of 2. For the scope of this paper, we assume uniform power allocation per node and that the applications are perfectly load-balanced.

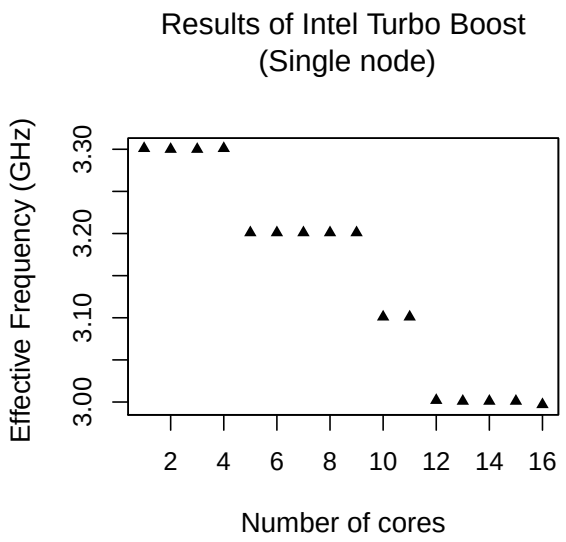


Figure 3: Turbo Boost on a single node with the CPU-bound, scalable benchmark.

6.1 Configurations

As discussed in Section 2, we define a configuration as: (1) a value for number of nodes, n, (2) a value for number of cores per node, c, and (3) power allocated per node, p, in Watts. We denote a configuration as $(n \times c, p)$. Because power capping is unavailable on DRAM and PP0, we use p to represent the PKG power that is allocated to the node. We measure DRAM power along with PKG power, and our results report their sum on each socket across all nodes when we report total power.

For comparison purposes, we define four special configurations: *packed-max*; *packed-min*; *spread-max*; and *spread-min*. The term *packed* denotes that a configuration uses all cores on one node before using an additional node, while *spread* denotes that processes are spread as evenly as possible across all nodes, with 4 being the fewest cores per node used. When we append *max* is appended, we use the maximum power (i.e., turbo mode) on each node, while *min* means we use the minimum power (i.e., clamp at 51 Watts). In all four special configurations, we continue to add cores and/or nodes until we reach the global power bound (or until we cannot add more nodes).

6.2 Results

We run our benchmarks on multiple nodes under various overprovisioned scenarios. Our cluster of 32 nodes consumed about 6350 W of power when running all cores as fast as possible. We investigate four overprovisioned scenarios; in each we have 32 nodes at our disposal, but power limits of 2500 W, 3000 W, 3500 W, and 4000 W. For comparison purposes, we also look at the case with unlimited power.

Figure 4 displays the potential speedup that overprovisioning can provide. This graph compares the best performance of any configuration that is under the respective power bound, compared to using *packed-max* (which is worst-case provisioning). If no bar exists then *packed-max* is optimal.

The average speedup of BT-MZ, LU-MZ, SP-MZ, and SPhot when using overprovisioning compared to worst-case provisioning is 73.8%, 55.6%, 67.2%, and 50.9% for a 2500 W, 3000 W, 3500 W, and 4000 W bound, respectively. Clearly, overprovisioning can lead to large performance improvements. For three cases, using a configuration other than *packed-max* is best even with unlimited power due to memory contention that degrades performance when using all cores.

Overprovisioning essentially allows applications to utilize the machine as a reconfigurable architecture, which allows for better performance than worst-case provisioning. For example, consider the global power bound of 3500 W. Here, SP-MZ executes 2.49 times faster when using the optimal configuration of $(26 \times 12, 80)$ than the *packed-max* configuration associated with worst-case provisioning, $(20 \times 16, 115)$. With worst-case provisioning, the facility would only have 20 nodes. We study the *potential* speedup that overprovisioning can support. We leave the question of *how* to determine configurations that realize this speedup for future work.

6.2.1 Comparing Different Configurations

Figure 5 shows in-depth results for three of our power bounds (2500 W, 3500 W, and unlimited) for all applications. The y-axis represents performance of the four canonical configurations, normalized to the performance of the optimal configuration under the global power bound (higher is better). The x-axis lists our eight benchmarks. Each figure includes a table that provides the actual configurations. The figure also contains the total power consumed (packages and memory power) and the time taken.

First, the figure shows some applications perform best using *packed* configurations compared to the *spread* configurations and that execution time can vary substantially between *packed* and *spread* configurations. This trend can be observed across all the benchmarks as well as power bounds. For example, at a global power bound of 2500 W, the *spread-min* configuration for SPhot runs 64.5% slower than *packed-min*. For SP-MZ at the same power bound, the *spread-max* configuration runs 2.16 times faster than the corresponding *packed-max* configuration. For SC, the *spread-min* runs more than twice as slow when compared to the *packed-min* configuration. In addition, for NSC, *spread-max* runs 2.36 times slower than *packed-max*.

The difference between the *packed-max* and *packed-min* configurations can also be significant. For example, with SPhot the *packed-max* configuration runs 29.8% slower than *packed-min* at 2500 W because the *packed-min* configuration utilizes more nodes and thus more total cores at lower power per node, as can be seen from the corresponding table. For SP-MZ and BT-MZ, the execution time difference between the *spread-max* and *spread-min* configurations is negligible. However, for LU-MZ this difference is 8% under a power bound of 2500 W and 16% under a power bound of 3500 W. When we vary the power bound, the execution time difference as well as the trend followed by the canonical and optimal configurations varies. At each global power bound, the configurations associated with the same canonical form

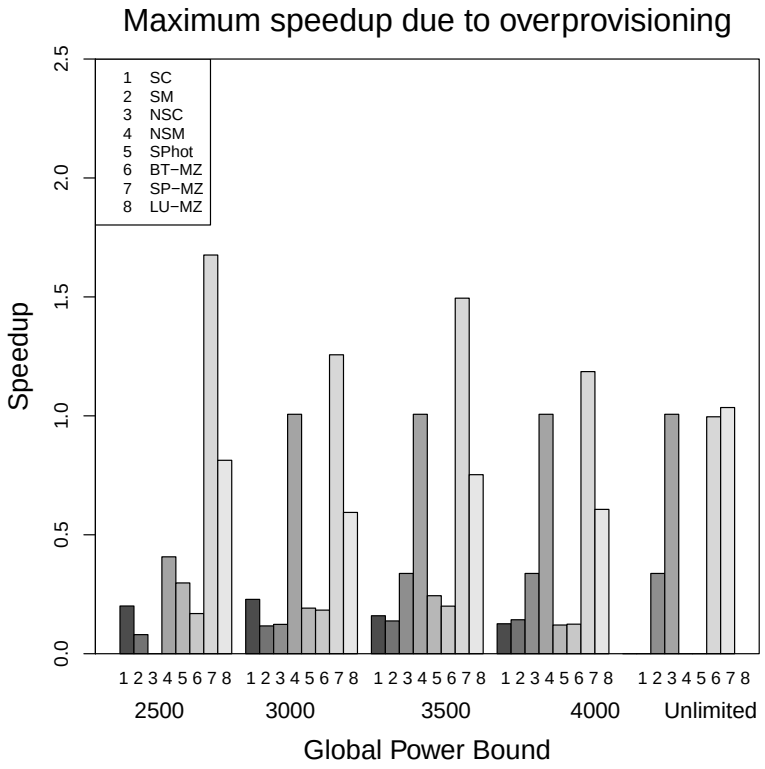


Figure 4: Speedup due to overprovisioning

can be different with an increase in the nodes and cores. For example, with SP-MZ *packed-max* is $(12 \times 16, 115)$ at 2500 W and $(20 \times 16, 115)$ at 3500 W.

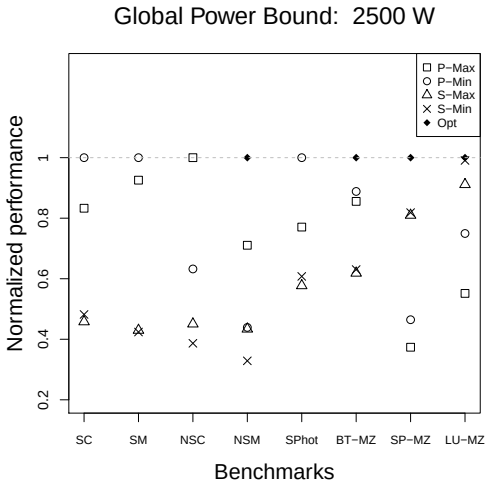
Second, the best configuration is not always one of *packed-max*, *packed-min*, *spread-max*, or *spread-min*. For example, at 2500 W, BT-MZ, SP-MZ, LU-MZ and NSM have an optimal configuration that is different than the four canonical configurations. As the corresponding table shows, the optimal configuration for SP-MZ was $(22 \times 8, 80)$, which was 22% faster than the fastest canonical configuration, $(28 \times 4, 51)$. Running fewer nodes at higher power per node performs better than running more nodes at lower power per node in this case, as opposed to SPhot.

Third, the best configuration for an application depends on the particular global power bound. For instance, at a power bound of 2500 W, the best configuration for SP-MZ is $(22 \times 8, 80)$. On the other hand, when the power bound is 3500 W, it is $(26 \times 12, 80)$—one expects more nodes with a higher power bound, but the number of cores per node is different.

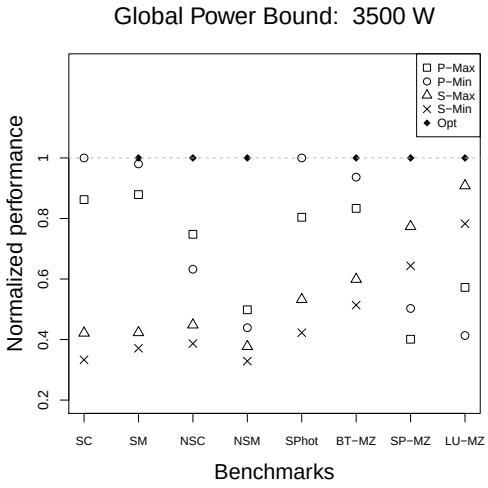
Application characteristics determine whether better execution times result from using more nodes, with fewer cores per node and at lower power; or fewer nodes, with more cores per node, and at a higher power. For instance, SPhot and BT-MZ always perform better when running more cores per node, and fewer nodes at higher power (i.e., *packed*), primarily because they are more CPU intensive than SP-MZ and LU-MZ. The *spread* configurations perform better for SP-MZ and LU-MZ because they are more memory intensive, and using more cores per node for such applications causes memory contention. For example, at 2500 W and 3500 W, the *spread* configurations are close to optimal for LU-MZ. Application scalability also plays an important role in determining the right configuration. SP-MZ, LU-MZ, NSC, and NSM have an optimal configuration that is neither *packed* nor *spread*, but somewhere in between because running as few as 4 cores per node and as many nodes as possible under the power bound increases communication.

6.2.2 Using Fewer Cores

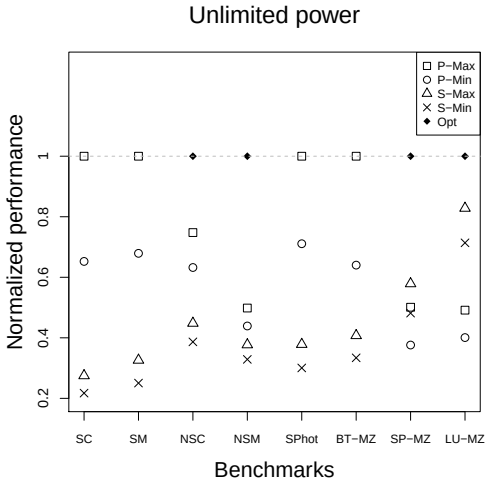
Cases exist in which using *fewer* total cores results in better execution time. This result is not unexpected, since similar results have been shown for worst-case provisioning [9]. For example, as the table corresponding to unlimited power shows, both SP-MZ and LU-MZ perform better with fewer total cores with the same PKG-level power bound. With SP-MZ, the *spread-max* configuration, $(32 \times 4, 115)$ is about 15% faster than the *packed-max*, $(32 \times 16, 115)$ (which runs four times as many cores). Two reasons lead to this behavior: first, in turbo mode, running fewer cores per node results in a higher effective turbo frequency; and, second, memory contention at 16 cores causes performance degradation. However, the optimal configuration is $(32 \times 14, 115)$, which uses more total cores than the *spread-max* configuration, but fewer cores per node than the *packed-max* configuration. The result is significantly better memory performance. The optimal configuration runs 72% faster than the fastest canonical one, which cannot be attributed to turbo



Bmark	Configuration $(n \times c, p)$	Total Power (W)	Time (s)
SPhot	$(12 \times 16, 115)$	2181.29	74.27
	$(22 \times 16, 51)$	2388.51	57.24
	$(24 \times 4, 115)$	2459.74	99.18
	$(32 \times 4, 51)$	2472.33	94.19
	$(22 \times 16, 51)$	2388.51	57.24
SP-MZ	$(12 \times 16, 115)$	1974.66	13.88
	$(20 \times 16, 51)$	2179.92	11.16
	$(22 \times 4, 115)$	2449.71	6.40
	$(28 \times 4, 51)$	2337.83	6.34
	$(22 \times 8, 80)$	2452.81	5.19
LU-MZ	$(12 \times 16, 115)$	2227.95	25.29
	$(20 \times 16, 51)$	2350.36	18.61
	$(22 \times 4, 115)$	2409.92	15.31
	$(28 \times 4, 51)$	2383.34	14.08
	$(22 \times 8, 80)$	2412.42	13.95



Bmark	Configuration $(n \times c, p)$	Total Power (W)	Time (s)
SPhot	$(18 \times 16, 115)$	3263.74	49.52
	$(32 \times 16, 51)$	3477.62	39.81
	$(32 \times 4, 115)$	3252.82	74.79
	$(32 \times 4, 51)$	2472.33	94.19
	$(32 \times 16, 51)$	3477.62	39.81
SP-MZ	$(20 \times 16, 115)$	3240.22	9.10
	$(32 \times 16, 51)$	3494.16	7.26
	$(32 \times 4, 115)$	3431.63	4.72
	$(32 \times 4, 51)$	2647.40	5.67
	$(26 \times 12, 80)$	3497.16	3.65
LU-MZ	$(18 \times 16, 115)$	3308.11	16.46
	$(30 \times 16, 51)$	3379.74	22.78
	$(32 \times 4, 115)$	3492.11	10.37
	$(32 \times 4, 51)$	2730.48	12.03
	$(32 \times 8, 95)$	3497.34	9.42



Bmark	Configuration $(n \times c, p)$	Total Power (W)	Time (s)
SPhot	$(32 \times 16, 115)$	5768.41	28.30
	$(32 \times 16, 51)$	3477.62	39.81
	$(32 \times 4, 115)$	3252.82	74.79
	$(32 \times 4, 51)$	2472.33	94.19
	$(32 \times 16, 115)$	5768.41	28.30
SP-MZ	$(32 \times 16, 115)$	4978.38	5.45
	$(32 \times 16, 51)$	3494.16	7.26
	$(32 \times 4, 115)$	3431.63	4.72
	$(32 \times 4, 51)$	2647.40	5.67
	$(32 \times 14, 115)$	5590.13	2.73
LU-MZ	$(32 \times 16, 115)$	5487.03	17.48
	$(32 \times 16, 51)$	3608.65	21.43
	$(32 \times 4, 115)$	3492.11	10.37
	$(32 \times 4, 51)$	2730.48	12.03
	$(32 \times 8, 115)$	4458.88	8.59

Figure 5: Multiple node experiments at a global power bound of 2500 W (top), 3500 W (middle) and unlimited (bottom). The left-hand side shows normalized performance (to the optimal) per benchmark, and the right-hand side shows the configuration and execution time for each point for SPhot, SP-MZ and LU-MZ. The last (shaded) row shows the optimal configuration.

mode because having 14 or 16 active cores per node results in the same effective turbo frequency.

In the case of LU-MZ, though, the effect of turbo mode as well as reduced memory contention can be seen more prominently. The optimal configuration, $(32 \times 8, 115)$ runs more than twice as fast as the *packed-max*, and 17% faster than the fastest canonical configuration, *spread-max*.

7. RELATED WORK

To the best of our knowledge, our work is the first study of HPC on power-constrained clusters and the first to explore optimal configurations under power bounds in the context of overprovisioned systems. Little work exists on HPC under a power bound. Rountree et al. [28] were the first to explore the idea using a dynamic, hardware-enforced power-bound in the HPC environment and to propose RAPL as an alternative to DVFS. They also examined power variation across processors and demonstrated that variation in processor power translates to variation in processor performance under a power bound. Springer et al. [32] studied computing under an energy bound for HPC applications, but this older work used DVFS instead of power clamping; they also used only eight single-core nodes and thereby avoided any scaling issues due to limited on-node memory bandwidth. Femal and Freeh [10] developed a technique for safe overprovisioning, but they focused on sets of CGI programs executed by web servers in data centers. In contrast, our work is focused on improving the performance of a single HPC program that has access to the entire cluster.

Curtis-Maury introduced Dynamic Concurrency Throttling, which controls the number of active threads that execute parallel regions to optimize for power and performance dynamically [7–9]. Li et al. [21] further extended this work to hybrid programming models. While these teams looked at configurations for general power minimization, they did not consider and study the impact of hardware-enforced or system-level power bounds on configurations.

Prior work has explored saving power/energy under a time bound in both the HPC and real-time communities. In the HPC arena, work has used linear programming to find near-optimal energy savings with zero time increase [29] as well as a run-time system that implements these ideas [27]. In the real-time community, several have used mixed integer linear programming to solve the DVFS scheduling problem [17,30, 33, 34] but are limited to a single processor. Several other real-time approaches save energy [24–26, 36, 37].

Additionally, there also is work in DVFS to trade execution time for lower power/energy [6, 13] as well as several analytic models to predict or to understand energy consumption in the context of scalability [12, 22, 31]. There are several other DVFS algorithms, including thrifty barriers [23], CPUMiser [12], and Jitter [18].

Overall, while significant work exists on optimizing or minimizing power, our work is the first to study the impact of power bounds, not only within a node, but also at larger-scale installations. Thus, we provide a first peek into the behavior of future generations of supercomputing systems, where such a scenario will be reality.

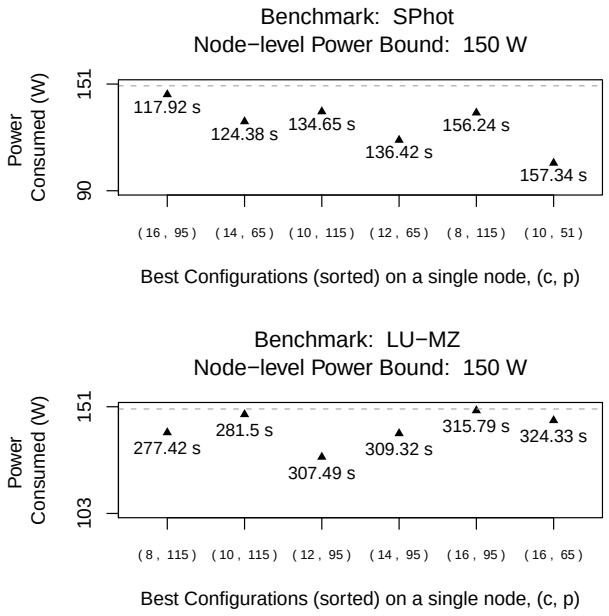


Figure 6: Configurations for SPhot and LU-MZ on a single node.

8. FUTURE WORK

Future work will proceed in multiple directions. One avenue is to investigate nonuniformity in power allocation. In this paper we studied *uniform* overprovisioned supercomputing. That is, while we showed that using more nodes with reduced power often leads to better performance, each node has a uniform configuration. While common for today's HPC applications, future applications are expected to be more heterogeneous.

In particular, assigning nonuniform power per node may improve performance, especially if the application exhibits load imbalance. However, a naive power assignment to each node (e.g., proportional increase in power based on amount of work) may produce poor results. For example, consider results on a single node in Figure 6. For SPhot, using more cores at lower power is profitable. On the other hand, for LU-MZ, running fewer cores at higher power is better.

Clearly, if we allocate power to nodes in a nonuniform manner, we should understand how power bounds impact performance through a node-level power-performance model. In past work we have used nonuniform DVFS settings to save energy with negligible performance loss [27].

A second avenue will develop models to predict the optimal configuration given a system-level power-constraint and a strongly-scaled application. We envision involve multiple steps, including (1) developing a single-node model to predict the optimal number of cores and power allocation for the components within a node (packages and memory subsystem) and (2) a model to allocate inter-node power based on the critical path of the application and its load imbalance.

A third avenue is to study the impact of using dynamic overclocking techniques such as Turbo Boost. We will experiment with clamping DRAM power and explore how best to utilize clamping in the PKG domain. RAPL is one of the technologies that will eventually enable us to schedule power more intelligently on a power-constrained cluster.

9. CONCLUSION

In this paper we identified an emerging problem in the HPC arena: how to leverage overprovisioning in HPC installations. With power becoming a first order design constraint on our road to exascale, such overprovisioned systems will be commonplace, i.e., we will no longer be able to fully power all nodes all the time. In such a scenario, we need to understand how we can configure our applications to best exploit the overall system while adhering to a global power bound.

Our experiments show that the optimal configuration under a cluster power bound for a strongly-scaled HPC application performs much better than the configuration corresponding to the naive, worst-case provisioned scenario. We also presented some early results on package power clamping and discussed its impact on application performance as well as on configurations. Overall, our experiments show that we can obtain substantial application speedup when carefully exploiting overprovisioned systems—over 50% on average.

10. ACKNOWLEDGEMENTS

We would like to extend our thanks to Livermore Computing and their support staff for providing us with the appropriate permissions required to access the MSRs. 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.

11. REFERENCES

[1] SPhot- Monte Carlo Transport Code, Lawrence Livermore National Laboratory.

[2] Ucsd center for dark silicon. `http://darksilicon.org/`.

[3] NASA Advanced Supercomputing Division, NAS Parallel Benchmark Suite v3.3, 2006.

[4] Top500 Supercomputer Sites, November 2012.

[5] AMD. AMD Turbo CORE Technology.

[6] K. W. Cameron, X. Feng, and R. Ge. Performance-constrained distributed DVS scheduling for scientific applications on power-aware clusters. In *Supercomputing*, Seattle, Washington, November 2005.

[7] Matthew Curtis-Maury, Filip Blagojevic, Christos D. Antonopoulos, and Dimitrios S. Nikolopoulos. Prediction-based power-performance adaptation of multithreaded scientific codes. *IEEE Trans. Parallel Distrib. Syst.*, 19(10):1396–1410, October 2008.

[8] Matthew Curtis-Maury, James Dzierwa, Christos D. Antonopoulos, and Dimitrios S. Nikolopoulos. Online power-performance adaptation of multithreaded programs using hardware event-based prediction. In *International Conference on Supercomputing*, New York, NY, USA, 2006. ACM.

[9] Matthew Curtis-Maury, Ankur Shah, Filip Blagojevic, Dimitrios S. Nikolopoulos, Bronis R. de Supinski, and Martin Schulz. Prediction models for multi-dimensional power-performance optimization on many cores. In *International Conference on Parallel Architectures and Compilation techniques*, New York, NY, USA, 2008. ACM.

[10] Mark E. Femal and Vincent W. Freeh. Safe overprovisioning: using power limits to increase aggregate throughput. In *International Conference on Power-Aware Computer Systems*, Dec 2005.

[11] Wu-Chun Feng and Kirk W. Cameron. The Green500 list: Encouraging sustainable supercomputing. *IEEE Computer*, 40(12):50–55, 2007.

[12] 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*, Xi'An, China, 2007.

[13] Chung-Hsing Hsu and Wu-Chun Feng. A power-aware run-time system for high-performance computing. In *Supercomputing*, November 2005.

[14] Intel. Intel Many Integrated Core Architecture.

[15] Intel. Intel Turbo Boost Technology 2.0.

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

[17] Tohru Ishihara and Hiroto Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In *International Symposium on Low power Electronics and Design*, pages 197–202, 1998.

[18] Nandani Kappiah, Vincent W. Freeh, and David K. Lowenthal. Just in time dynamic voltage scaling: Exploiting inter-node slack to save energy in MPI programs. In *Supercomputing*, November 2005.

[19] Rakesh Kumar, Dean M. Tullsen, Norman P. Jouppi, and Parthasarathy Ranganathan. Heterogeneous chip multiprocessors. *IEEE Computer*, 38(11):32–38, Nov 2005.

[20] Lawrence Livermore National Laboratory. The ASCI Purple benchmark codes. `http://www.llnl.gov/asci/purple/benchmarks/limited/code_list.html`.

[21] Dong Li, Bronis R. de Supinski, Martin Schulz, Dimitrios S. Nikolopoulos, and Kirk W. Cameron. Strategies for energy efficient resource management of hybrid programming models. *IEEE Transaction on Parallel and Distributed Systems*, 2012.

[22] Jian Li and José F. Martìnez. Dynamic power-performance adaptation of parallel computation on chip multiprocessors. In *12th International Symposium on High-Performance Computer Architecture*, Austin, Texas, February 2006.

[23] Jian Li, José F. Martínez, and Michael C. Huang. The thrifty barrier: Energy-aware synchronization in shared-memory multiprocessors. In *10th International Symposium on High Performance Computer Architecture*, Madrid, Spain, February 2004.

[24] Bren Mochocki, Xiaobo Sharon Hu, and Gang Quan. A realistic variable voltage scheduling model for real-time applications. In *Proceedings of the 2002 IEEE/ACM International Conference on Computer-Aided Design*, 2002.

[25] Bren Mochocki, Xiaobo Sharon Hu, and Gang Quan. Practical on-line DVS scheduling for fixed-priority real-time systems. In *11th IEEE Real Time and*

Embedded Technology and Applications Symposium, 2005.

[26] M. Angels Moncusí, Alex Arenas, and Jesus Labarta. Energy aware EDF scheduling in distributed hard real time systems. In *Real-Time Systems Symposium*, December 2003.

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

[28] Barry Rountree, Dong H. Ahn, Bronis R. de Supinski, David K. Lowenthal, and Martin Schulz. Beyond DVFS: A First Look at Performance under a Hardware-Enforced Power Bound. In *IPDPS Workshops*, pages 947–953. IEEE Computer Society, 2012.

[29] Barry Rountree, David K. Lowenthal, Shelby Funk, Vincent W. Freeh, Bronis de Supinski, and Martin Schulz. Bounding energy consumption in large-scale MPI programs. In *Supercomputing*, November 2007.

[30] H. Saputra, M. Kandemir, N. Vijaykrishnan, M.J. Irwin, J.S. Hu, C.-H. Hsu, and U. Kremer. Energy-conscious compilation based on voltage scaling. In *Joint Conference on Languages, Compilers and Tools for Embedded Systems*, 2002.

[31] Rob Springer, David K. Lowenthal, Barry Rountree, and Vincent W. Freeh. Minimizing execution time in MPI programs on an energy-constrained, power-scalable cluster. In *ACM Symposium on Principles and Practice of Parallel Programming*, March 2006.

[32] Robert C. Springer IV, David K. Lowenthal, Barry Rountree, and Vincent W. Freeh. Minimizing execution time in MPI programs on an energy-constrained, power-scalable cluster. In *ACM Symposium on Principles and Practice of Parallel Programming*, March 2006.

[33] Vishnu Swaminathan and Krishnendu Chakrabarty. Investigating the effect of voltage-switching on low-energy task scheduling in hard real-time systems. In *Asia South Pacific Design Automation Conference*, January 2001.

[34] Vishnu Swaminathan and Krshnendu Chakrabarty. Real-time task scheduling for energy-aware embedded systems. In *IEEE Real-Time Systems Symposium*, November 2000.

[35] Ganesh Venkatesh, Jack Sampson, Nathan Goulding, Saturnino Garcia, Vladyslav Bryksin, Jose Lugo-Martinez, Steven Swanson, and Michael Bedford Taylor. Conservation cores: reducing the energy of mature computations. In *ASPLOS*, 2010.

[36] Yumin Zhang, Xiaobo Sharon Hu, and Danny Z. Chen. Task scheduling voltage selection for energy minimization. In *Proceedings of the 39th annual Design Automation Conference*, 2002.

[37] Dakai Zhu, Rami Melhem, and Bruce R. Childers. Scheduling with dynamic voltage/speed adjustment using slack reclamation in multi-processor real-time systems. *IEEE Transactions on Parallel and Distributed Systems*, 2003.