Power Tuning HPC Jobs on Power-Constrained Systems

Neha Gholkar[1], Frank Mueller[1], Barry Rountree[2]
[1]North Carolina State University, USA, ngholka@ncsu.edu, mueller@cs.ncsu.edu
[2]Lawrence Livermore National Laboratory, USA, rountree4@llnl.gov

ABSTRACT

As we approach the exascale era, power has become a primary bottleneck. The US Department of Energy has set a power constraint of 20MW on each exascale machine. To be able achieve one exaflop under this constraint, it is necessary that we use power intelligently to maximize performance under a power constraint.

Most production-level parallel applications that run on a supercomputer are tightly-coupled parallel applications. A naïve approach of enforcing a power constraint for a parallel job would be to divide the job's power budget uniformly across all the processors. However, previous work has shown that a power capped job suffers from performance variation of otherwise identical processors leading to overall sub-optimal performance. We propose a 2-level hierarchical variation-aware approach of managing power at machine-level. At the macro level, *PPartition* partitions a machine's power budget across jobs to assign a power budget to each job running on the system such that the machine never exceeds its power budget. At the micro level, *PTune* makes job-centric decisions by taking the performance variation into account. For every moldable job, PTune determines the optimal number of processors, the selection of processors and the distribution of the job's power budget across them, with the goal of maximizing the job's performance under its power budget.

Experiments show that, at the micro level, PTune achieves a performance improvement of up to 29% compared to a naïve approach. PTune does not lead to any performance degradation, yet frees up almost 40% of the processors for the same performance as that of the naïve approach under a hard power bound. At the macro level, PPartition is able to achieve a throughput improvement of 5-35% compared to uniform power distribution.

1. INTRODUCTION

The supercomputing community is headed toward the era of exascale computing, which is slated to begin around 2020. Today's fastest supercomputer, Tianhe-2, consumes 17.8MW to deliver 33.86PFlops [2]. If we were to build an exascale machine with today's technology it would consume up to 350MW of power. A typical power plant generates 1GW of power, which is sufficient to power 700,000 homes [1]. The US DOE has set a power constraint of 20MW per future exascale systems to maintain a feasible electrical power demand. In order to get an exaflop under this constraint, we need at least an order of magnitude improvement in power efficiency with respect to today's systems [5, 7, 19, 31].

Exascale systems are expected to be power-constrained: the size of the machine will be limited by the amount of provisioned power. Existing best practice requires provisioning power based on the theoretical maximum power draw of the machine, despite the fact that only a synthetic workload comes close to this level of power consumption. One of the key contributions in the power-constrained domain is "hardware overprovisioning" [24]. The idea is to provision much less power per node and thus provision more nodes. The benefit is that all of the scarce resource (power) will be used. The drawback is that power must be carefully scheduled within the machine in order to approach optimal performance.

Fig. 1 depicts this foundational idea. Let the hardware overprovisioned system consist of N_{max} processors and let the power budgeted for this system be $P_{m/c}$ Watts. As shown in the figure, with $P_{m/c}$ Watts total system power, only a part of the system (say N_{alloc} where $N_{alloc} < N_{max}$ processors) can be utilized at peak power (collection of nodes in red). Another valid configuration is to utilize the entire system at low power. One of the several other intermediate configurations is to use medium power levels and utilize a portion of the system larger than that at peak power but smaller than that at low power. In each of these configurations, a machine's power budget is uniformly distributed across a varying number of processors, i.e, each processor is allocated approximately $\frac{P_{m/c}}{N_{alloc}}$ Watts of power. This is a naïve approach of enforcing a power budget. Depending on the application's characteristics (memory-, compute-, and communication-boundedness), different applications achieve optimal performance on different configurations. In a nutshell, power procured for a system must be managed as a malleable resource to maximize performance of an overprovisioned system under a power constraint.

To facilitate the selection of different power levels, hardware manufacturers are providing various features like power clamping and on-chip power measurement mechanisms (e.g.,

PACT '16, September 11-15, 2016, Haifa, Israel
© 2016 ACM. ISBN 978-1-4503-4121-9/16/09…$15.00
DOI: http://dx.doi.org/10.1145/2967938.2967961

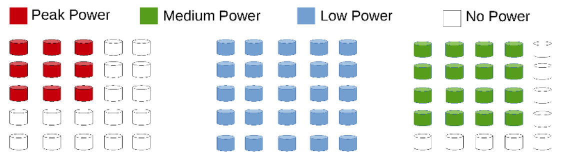

Figure 1: Hardware Overprovisioning under Power Constraint

Intel's Running Average Power Limit or RAPL [20]), as well as component power measurement infrastructures [17, 15]. In this work, we use RAPL to cap and measure the power consumption of the processors, where a processor is a single multi-core chip shipped by Intel.

When processors in the same stock keeping unit (SKU) operate at the same frequency, there is observable variation in power draw [6, 39, 16, 37]. When processors are capped at uniform power, this variation is expressed in terms of performance, i.e., the system is no longer homogeneous [27]. There are many potential root causes of this variation, including, but not limited to, process variation and thermal variation due to ambient machine room temperature.

We observe that these phenomena also translate into variation in the peak power efficiency of the processors. We define *power efficiency* as the performance achieved by a processor per Watt of the power consumed by it. Performance is quantified in terms of number of instructions retired per second (IPS). We make two important observations. First, not all processors are equally power efficient at a fixed power bound. Second, the most efficient processors are most efficient at lower power bounds whereas the least efficient processors are most efficient at higher power bounds. Based on these insights about variation across the system, we propose *power tuning* and *power partitioning* strategies that distribute the power budget non-uniformly across the system to achieve maximum performance irrespective of the cause of variation.

We propose a 2-level hierarchical solution that performs Power Partitioning (PPartition) at macro-level and Power Tuning (PTune) at micro-level to maximize the "science performed per Watt". PPartition aims at improving the throughput of the machine by (re-)partitioning a machine's power budget across jobs while scheduling them. PTune is a variation-aware power manager for moldable jobs whose number of processors can be chosen at dispatch time. For every job, PTune maximizes its performance under its job power budget (assigned by PPartition) by determining the following:

- The optimal selection of a subset of processors from the available ones; and
- the (non-uniform) distribution of the job's power budget across the selected processors.

PTune uses off-line characterization data to make these decisions before the beginning of job execution. PTune achieves a job performance improvement of up to 29% over uniform power. PTune does not lead to any performance degradation, yet frees up 40% of the resources compared to uniform power. PPartition and PTune together improve the throughput of the machine by 5-35% compared to conventional scheduling on an overprovisioned machine.

In this work, we make the following contributions:

- We characterize the performance of contemporary Intel processors with multiple codes at multiple power bounds. Performance variations of up to 30% are observed across these processors. Using this data we conduct a power efficiency study that forms the basis for the work.
- We propose *PTune*, a variation-aware job power tuner that optimizes a job for performance under a strict job power constraint.
- We propose *PPartition*, a variation-aware resource manager that manages a machine's power budget as a resource to maximize the throughput of the machine under a machine-level power constraint.
- We evaluate PTune and PPartition using three of the NAS Parallel benchmark codes [3] and a molecular dynamics proxy application, CoMD [30].

2. MOTIVATION

Research focus has only recently shifted from just performance to minimizing energy usage of supercomputers. Considering the US DOE mandate for a power constraint per exascale site, efforts need to be directed towards using all of the limited amount of power intelligently to maximize performance under this constraint.

As stated previously, uniform power capping is the naïve approach of enforcing a power constraint. In order to understand what happens under such a scheme, we characterized the performance of 600 Ivy Bridge processors on a cluster. We ran three of the NAS Parallel Benchmark (NPB) suite codes [3], viz., Embarrassingly Parallel (EP), Block Tridiagonal solver (BT), Scalar Penta-tridiagonal solver (SP), and CoMD, a molecular dynamics proxy application from the Mantevo suite [30] at several different processor power bounds on all the processors. The processor power bounds were set using RAPL. The results are depicted in Fig. 2. The x-axis represents operating power in Watts while the y-axis represents Instructions Retired per Second (IPS) in billions. A maximum performance of 77, 50, 80, and 60 billions IPS is achieved for CoMD, EP, BT and SP, respectively. The cluster becomes non-uniform under power bounds with performance variations of up to 30% across this cluster for these applications. The potential causes of variability are discussed next but are effectively irrelevant as our proposed methods are agnostic of specific causes. More significantly, our experiments will show that this variability in performance translates into variation in peak *power efficiency* of the processors, which we exploit.

Power Efficiency

Let *power efficiency* be defined as the number of instructions retired per second per Watt of operating power. Fig. 3 represents the power efficiency curves of the processors on the cluster for the same set of codes. The x-axis represents the operating power in Watts and the y-axis represents the power efficiency in billion IPS/W. The rainbow palette represents different processors, where each curve (or each color) in the plots corresponds to a unique processor.

We make the following observations from these experiments:

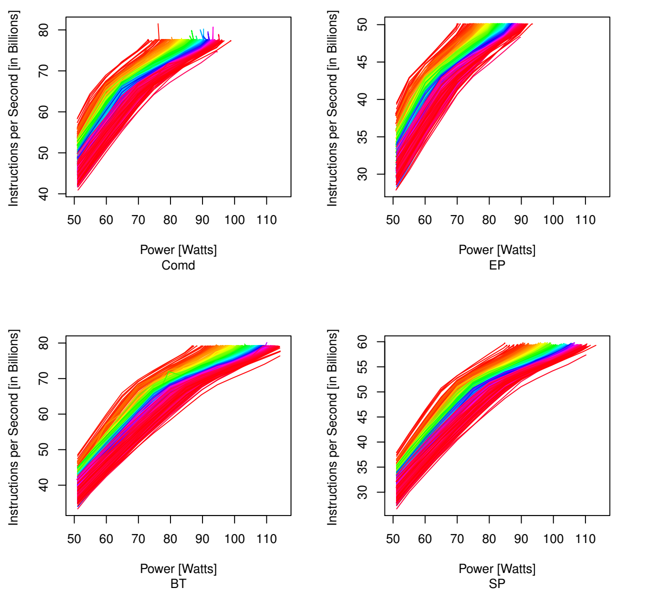

Figure 2: IPS vs. Power for each processor. Each rainbow line represent one processor. Curves in red (bottom) are least efficient, curves in orange (top) are most efficient.

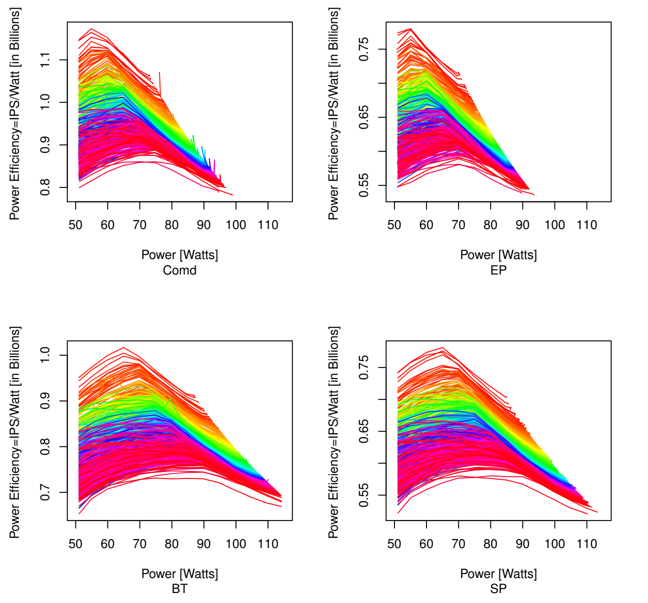

Figure 3: Power Efficiency in IPS/W vs. Operating power. One rainbow line per processor, red curves (bottom) are least, orange ones (top) most efficient.

- The power efficiency of a processor varies with its operating power and is non-monotonic. It is also workload-dependent.
- Peak power efficiency varies across processors.
- Most importantly, efficient processors are most efficient at lower power bounds whereas the inefficient processors are most efficient at higher power bounds. The "peak" of every curve is the point at which the processor achieves the maximum efficiency, i.e., maximum IPS/W. Orange curves (efficient processors) have peaks at lower power compared to the peaks of the red curves (less efficient processors) and the rest lie in between.

Fig. 4 depicts the results of our thermal experiments. The x-axis presents processor IDs (processors are sorted in the order of efficiency). The y-axis presents the measured temperature (triangles) of the processors normalized with respect to the maximum temperature and the unbounded power (crosses) of the processors also normalized with respect to the maximum power. In these experiments, the processors were not capped, and they achieved uniform performance. We observe that the temperature increases as we go from efficient to inefficient processors (left to right), as does the unbounded power. However, not all inefficient processors are hotter than the efficient ones. This shows that thermal variation may be one of the potential causes of variation in efficiency but there are other factors that counter the effect as we do not see a linear trend for temperature (in contrast to the linear trend of unbounded power). We believe that one of the contributing factors is process/manufacturing variation induced at the time of fabrication. In the end, our proposed mechanism is agnostic of the actual cause of variation, it simply exploits the fact that variation (due to whatever reason) exists.

In summary, there exists variation in power efficiency across processors. There is a unique local maximum in every power efficiency curve that occurs at disparate power levels for different processors. Starting from the minimum power, increasing the power assigned to a processor leads to increasing gains in IPS. However, increasing the power beyond the peak efficiency point of a processor leads to diminishing returns. Hence, when power is limited, processors should operate at power levels close to their peak efficiency to maximize the overall efficiency of the system. Since the peak efficiency points for efficient processors are at lower power levels than for the inefficient processors, the optimal configuration should select lower power levels for efficient processors and higher power levels for inefficient processors to maximize performance. On the contrary, a naïve / *uniform power* scheme caps all the processors at identical power bounds. Hence, it is sub-optimal.

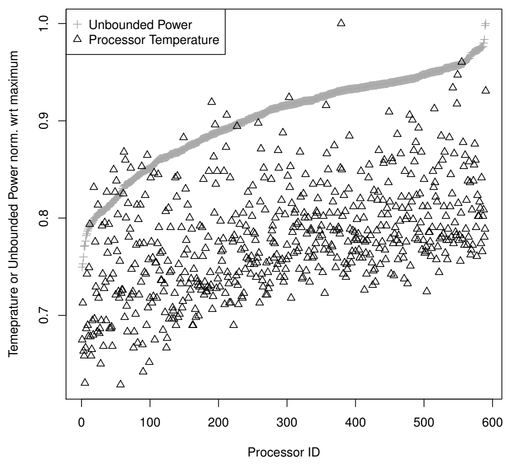

Figure 4: Temperature and unbounded power of processors. Processors are sorted by unbounded power consumption.

An optimal algorithm should aim at leveraging the non-uniformity of the cluster to maximize the performance of a job under its power constraint.

To this end, we propose *PTune*, a power-performance variation-aware power tuner that exactly does this for each job. For every job, given a power budget, it determines the following: (1) the optimal number of processors (say n_{opt}); (2) selection of n_{opt} processors; and (3) the power distribution (say p_k, where $1 \leq k \leq n_{opt}$) across the selected n_{opt} processors.

3. PROBLEM STATEMENT

The problem statement then is as follows: Given a machine level power budget, how should the machine's power be distributed across (a) jobs and (b) tasks within jobs on a given system, where (b) is discussed later. For (a), the process of making these decisions at the macro level of jobs is called *power partitioning*. Each job on the machine receives its own power partition.

We address the following questions:

1. How many partitions do we need at a time? I.e., determine how many jobs should be scheduled at a time.
2. What is the size of each of the power partitions? I.e., determine the power budget assigned to each job.

For (b), at the micro level, given a hard job-level power budget P_{Ji}, we need to determine the optimal number of processors, n_{opt}, with a power distribution (p_1, p_2, ... , $p_{(n_{opt}-1)}$, $p_{n_{opt}}$) such that performance of the job is maximized under its power budget. The constraint on the power distribution is expressed as

$$\sum_{k=1}^{n} p_k \leq P_{Ji}; min\_power \leq p_k \leq max\_power_k.$$

Here, min_power is the minimum power that needs to be assigned to a processor for reliable performance and max_power_k is the maximum power consumed by the k^{th} processor (uncapped power consumption) for an application. The performance of a job can be quantified in terms of number of instructions retired per second (IPS).[1] For a parallel application on n processors, the effective IPS is the aggregated IPS over n processors ($JobIPS_n$). Hence, the objective function is

$$Maximize(JobIPS_n).$$

A processor's IPS is a non-linear function of the power at which it operates. Each processor can be power bounded at several levels using the RAPL capping capabilities, which forces it to operate at various power levels within a fixed range. We know that unbounded power consumption is variable across processors while achieving the same unbounded (peak) performance for a given application. This is depicted in Fig. 5. The x-axis indicates the power at which the processor operates and the y-axis shows the IPS (in billions) of the processor of an application. Each solid curve corresponds to the most efficient processor while the dotted curve correspond to the least efficient processor. The following two observations are made from this data:

1. On a single processor, the performance (IPS) achieved at any fixed power level is different for different workloads.
2. The performance of an application on two different processors at any fixed power level is not the same.

[1]The general model holds for other performance metrics as well. We selected IPS here because it closely correlates to power in our experiments.

This means that when determining the optimal distribution of power across processors it is necessary to take the processor characteristics and the application characteristics into account. One solution may not fit all applications. The optimal configuration for an application on one set of processors may be different from that on another set of processors because of performance variations under a power cap.

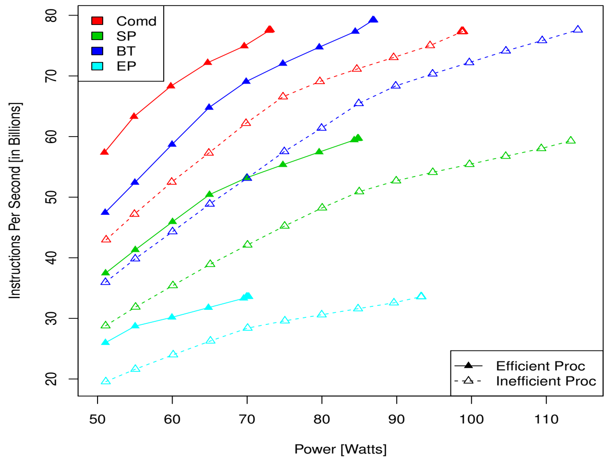

Figure 5: IPS vs. Power for efficient and inefficient processors

4. PROPOSED SOLUTION

We propose a 2-level hierarchical approach of managing power as a resource (see Fig. 6). The parameters of the model are described in Table 1. N_{max}, $P_{m/c}$ and n_{req} are the inputs to the model that we assume. n_{opt} is calculated once for every job at its dispatch time. N_{alloc}, P_{Ji} and p_k are re-calculated every time any job is dispatched. min_power is architecturally defined for every family of processors. Table 2 is populated off-line using the characterization data. We make the assumption that the power consumption of the interconnect is zero, i.e., interconnect power is beyond the scope, and so are task-to-node mapping effects on power. We only consider processor power in this work and assume moldable jobs. DRAM power could not be included due to motherboard limitations at the time of this work. We do not expect the users of the system to predict and request power in their job request. Power decisions are made by our system software (PTune and PPartition). Users may be allowed to influence these decisions by assigning priorities to their jobs.

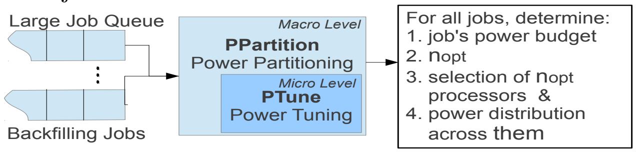

Figure 6: Hierarchical Power Manager

At the macro level, we propose *PPartition*, a technique of partitioning a machine's power budget across jobs while scheduling them. Once a job is dispatched by a conventional scheduler (e.g., slurm or Maui/pbs), PPartition calculates its power budget. If the required power is not available, it

Table 1: Model Parameters

Parameter	Description	PPartition	PTune
N_{max}	maximum number of processors on a machine	Input	N/A
N_{alloc}	number of processors already allocated to jobs	Output	N/A
$P_{m/c}$	power budget of the machine	Input	N/A
P_{m/c_unused}	unused power budget of the machine	Variable	N/A
n_{req}	number of processors requested by a job	Input	Input
n_{opt}	optimal number of processors for a job	Output	Output
n	number of processors for a job under its power budget	N/A	Variable
P_{Ji}	power budget of the i^{th} job	Output	Input
p_k	power cap of k^{th} processor within a job	N/A	Output
min_power	minimum processor power cap	Input	Input
max_power_j	maximum processor power cap of the j^{th} processors	Input	Input
power-ips table	characterization data Table 2	Input	Input

Table 2: power-ips lookup table (last metric in Tab. 1)

Power Cap[W]	IPS in Billions	Measure Power [W]
60	46.43	59.99
80	64.83	79.88
100	76.33	99.43
120	79.13	104.66

steals power from the previously scheduled jobs and provisions this power for the new job. If sufficient power cannot be obtained, PPartition overrides the conventional scheduler's decision based on free resources (nodes) and does not schedule this job until sufficient power is available.

At the micro-level, we propose *PTune*, a power balancing model that determines the distribution of a job's power budget (one job at a time) across an optimal selection of processors (among all free resources) to maximize the performance of a job under its power budget.

5. PTUNE

PTune shrinks the job's processor allocation by eliminating the less efficient processors that are expensive in terms of power to maximize the performance of a job under its power budget. Fig. 7 depicts the micro-level power tuner. For each job Ji, a power budget P_{Ji} is calculated at the macro level by PPartition. For every job with this assigned power budget, PTune answers the following questions:

1. How many (n_{opt}) and which processors should a job run on?

2. What should be the power ($p_1, ..., p_{n_{opt}}$) assigned to each of the n_{opt} processors?

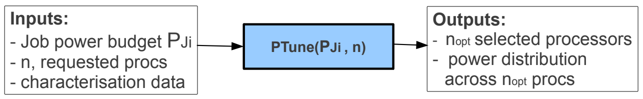

Figure 7: PTune

Let us start by addressing the first question. In order to use a processor, it needs to be assigned at least the minimum power (min_power) that is constant across all processors. The upper limit on the processor's power (max_power_k) is variable across processors.

Fig. 8 shows the maximum power consumption of 600 Ivy Bridge processors when they are not power capped. The unbounded performance is uniform across all the processors. The x-axis represents all the processor sorted by power consumption and the y-axis represents the maximum power consumption in Watts. The optimal configuration for maximum performance of a job under a strict power budget consists of the maximum number of most efficient processors (from the left) such that their aggregate power consumption does not exceed the job's power budget.

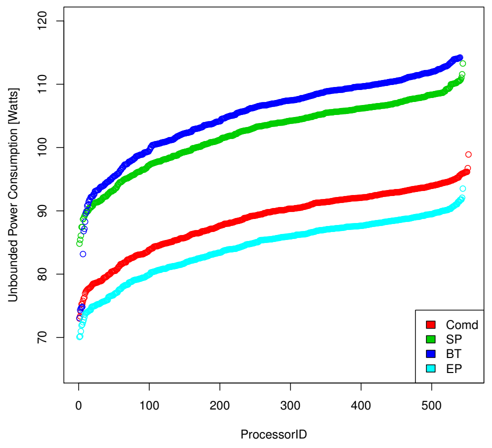

Figure 8: Unbounded power consumption of processors under uniform performance

5.1 Sort the Processors

The first step towards determining the optimal configuration is to sort the available processors by their relative power efficiency. This is equivalent to sorting them by their unbounded power consumption. Let the sorted set of processors be indexed by k.

We divide this distribution of processors into quartiles, viz., Q1, Q2, Q3 and Q4, in the order of efficiency and pick processors from one or more of these quartiles for evaluation purposes.

5.2 Bounds on Number of Processors

The lower bound on n, $n_\perp$, can be calculated by determining the maximum number of processors that can be capped at their maximum power, max_power_k, under the power budget. The selection of processors is reformed in the sorted order as described above. $n_\perp$ is given by the largest value of n that satisfies the following constraint:

$$P_{Ji} \geq \sum_{k=1}^{n} max\_power_k.$$

The upper bound on n, $n_\top$, represents the maximum number of processors that can be operated at min_power under the power budget. The bound $n_\top$ is calculated as follows:

$$n_\top = \frac{P_{Ji}}{min\_power}.$$

The processor count, n, is iterated from $n_\perp$ to $n_\top$, and in each step, the next efficient processor is added to the set of processors. Job-level performance, $JobIPS_n$, is calculated in each iteration by $DistributePower()$ for the power budget P_{Ji} and a given number of processors, n, where $n_\perp \leq n \leq n_\top$.

The optimal number of processors, i.e., n_{opt}, is the value of n at which a job's IPS is maximized.

$$JobIPS_{n_{opt}} = max(JobIPS_{n_\perp}, JobIPS_{(n_\perp+1)}, ..., JobIPS_{n_\top}).$$

PTune leads to $n_{opt} \leq n$. Thus, PTune tends to reduce the number of processors required for a moldable job The spare processors are returned back to the global pool of unused resources so that they can be utilized by other jobs.

5.3 Distribute Power: Mathematical Model

DistributePower(), takes three inputs, viz., the number of processors n, the job's power budget, P_{Ji}, and the power distribution across $n-1$ processors determined in the previous iteration. The output of this function is the maximum job IPS that can be achieved under P_{Ji} Watts with n processors. It also calculates the optimal power caps, (p_1, ..., p_n), for n processors, which forms an input for the next iteration. This can be mathematically expressed as follows:

$$DistributePower(n, P_{Ji}, (p_1, ..., p_n)) = DistributePower((n-1), P_{Ji} - p_n, p_1, ..., p_{(n-1)}) + getProcIPS(n, p_n).$$

The function getProcIPS(k,p_k) performs a look-up in Tab. 2 to return the expected performance (IPS) of the k^{th} processor when it is capped at p_k Watts.

5.4 Power Stealing and Shifting

DistributePower() consists of two main steps, viz. Power Stealing and Power Shifting.

Step 1: Power is stolen in discrete quantities ($delta\_power$) from the $n-1$ processors to provision power for the n^{th} processor (see Fig. 9). The victim/donor processor is the one that suffers minimum loss in IPS when $delta\_power$ is stolen from it. If the aggregate stolen power is at least min_power, an additional n^{th} processor is added to the processor set.

Step 2: Power is shifted from a donor to a receiver in discrete quantities, $delta\_power$, across the n processors. The victim/donor processor is identified in the same way as in step 1. The receiver is the processor that gains maximum IPS on receiving $delta\_power$.

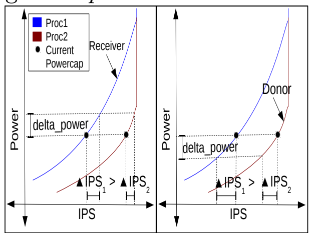

Figure 9: Donor and receiver of discrete power

6. PPARTITION

Fig. 10 depicts the macro-level power partitioning algorithm. The power partitioner co-operates with the conventional scheduler (simulated in R). PPartition receives information on the performance variations across processors. It always chooses the most efficient n_{req} processors of the available processors (or a subset thereof) to schedule a job. The conventional scheduler dispatches a job from the job queue when the requested number of processing resources are available. When job J_i is dispatched, its initial power budget, P_{Ji}, is calculated as follows:

$P_{Ji} \leftarrow P_{m/c} * \frac{n_{req}}{N_{max}}$

If the required power is available, PTune determines the optimal configuration for the job and the job is scheduled. It is important to note that even though the job power budget is proportionate to the number of requested processors, PTune schedules jobs on reduced number of processors. As a result, the machine's power depletes at a faster rate that the processing resources. If the available (unused) power is less than the calculated job power budget, power is stolen from already scheduled jobs. This is called power repartitioning (lower right blue/shaded box in Fig. 10) and detailed next.

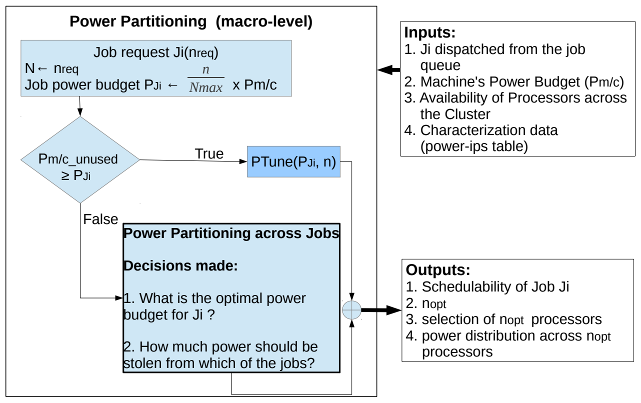
Power Partitioning (macro-level)
Job request Ji(nreq)
N ← nreq
Job power budget PJi ← n / Nmax x Pm/c
Pm/c_unused ≥ PJi
True
False
PTune(PJi, n)
Power Partitioning across Jobs
Decisions made:
1. What is the optimal power budget for Ji ?
2. How much power should be stolen from which of the jobs?
Inputs:
1. Ji dispatched from the job queue
2. Machine's Power Budget (Pm/c)
3. Availability of Processors across the Cluster
4. Characterization data (power-ips table)
Outputs:
1. Schedulability of Job Ji
2. nopt
3. selection of nopt processors
4. power distribution across nopt processors

Figure 10: PPartitioning: Repartitioning Power

Power Repartitioning

The power repartitioning algorithm is shown in Algorithm 1. As all of the machine power budget is already used up by the $N_{allocated}$ processors, a fair power share for the new job is calculated as

$P_{Ji} = P_{m/c} * \frac{n}{n+N_{allocated}}$, where n=$n_{req}$.

Algorithm 1 Repartitioning Power For Accommodating the i^{th} job

```
1: procedure PowerPartitioner($J_i$,$n_{req}$)
2:     $n_{opt} \leftarrow n_{req}$
3:     do
4:         $n \leftarrow n_{opt}$
5:         $P_{Ji} \leftarrow P_{m/c} * \frac{n}{n+N_{allocated}}$    ▷ Recompute $P_{Ji}$ proportional to the portion of busy processors requested
6:         $n_{opt} \leftarrow PTune(P_{Ji}, n)$    ▷ Recompute $n_{opt}$
7:     while $n_{opt} < n$
       ▷ Repartition power across jobs to provision power for the $i^{th}$ job
8:     for $k \leftarrow 1; k < i; k++$ do
9:         $power\_to\_be\_stolen[k] \leftarrow (P_{Ji} - P_{m/c\_unused}) * \frac{jobpowerbudgets[k]}{\sum jobpowerbudgets}$
10:        $total\_stolen\_power \leftarrow total\_stolen\_power + ShrinkPartition(power\_to\_be\_stolen[k], k)$
11:    end for
12:    if $total\_stolen\_power < P_{Ji}$ then    ▷ If enough power cannot be stolen, recompute $n_{opt}$
13:        $P_{Ji} \leftarrow total\_stolen\_power$
14:        $n_{opt} \leftarrow PTune(P_{Ji}, n)$
15:    end if
16: end procedure
```

The job is power tuned for the requested n_{req} (assigned to n) processors under P_{Ji} Watts calculated above. PTune gets rid of the unaffordable less efficient processors, if any, leading to $n_{opt} \leq n$. We recompute (in the while loop) the proportionate power the new job with n_{opt} processors should have due to power partitioning across all jobs. This new power budget, P_{Ji}, then becomes the base for another PTune, and so on, until the number of processors (monotonically decreasing) for the new job reaches a fixed point (stabilizes) in the while loop. The fixed point guarantees a fair power level (P_{Ji}) relative to other jobs, but we still need to find other jobs to steal just enough power for this job.

In the following for loop, power is stolen from each of the scheduled jobs in a proportionate manner to each other's power budget. This is accomplished by *ShrinkPartition*, which consists of (1) stealing just enough power and (2) power tuning for the remaining power of a job and the same number of processors (since we assume moldable but not malleable applications). Here, we steal as much power as possible while retaining heterogeneous power bounds across a job's processors to respect processor variations and thus ensure a high IPS under lower power budget.

The aggregate stolen power from other jobs is offered to the new job. If the stolen power is less than the fixed power level for the new job, which was P_{Ji}, then the new job needs to be tuned one more time. If the stolen power was sufficient for this last tuning step, the new job is scheduled and the power re-tuning decisions made by ShrinkPartition for the existing ones are enforced. If, however, the stolen power is insufficient (as determined by PTune when the power budget cannot accomodate more than $\frac{n}{2}$ processors), no power is redistributed, i.e., all jobs remain unchanged in their power settings and the new job is deferred until at least another job completes.

7. IMPLEMENTATION

We modified the libmsr [35] library to gather the processor characterization data. We implemented a power-performance profiler using the MPI profiling interface (PMPI) that invoked various subroutines of the libmsr library to assess the power and the performance of MPI applications. We captured several fixed counter values, power consumption, and completion times for each application on all the processors. The processor power consumption was measured using Intel's RAPL interface. This characterization data is made available to PTune and PPartition.

We assume that the jobs are moldable. Our power manager works in co-ordination with the conventional job scheduler. Once a job is dispatched by the conventional scheduler, the power manager (PPartition+PTune) determines its power budget, the selection of processors from those available, and the power distribution (or processor power caps) across them.

We assume a large job queue (> 384 processes) and a backfilling queue (< 48 processes). The conventional job scheduler schedules as many large jobs as it can on the machine before scheduling the backfilling jobs. We assume up to Nmax=550 nodes with 12 cores each (6600 processes). If the power manager decides to schedule the job, power distribution across its processors (and power repartitioning if required) is enforced using RAPL.

8. EXPERIMENTAL SETUP

Experiments were conducted on a 324-node Ivy Bridge cluster. Each node has two 12-core Intel(R) Xeon(R) CPU E5-2695 v2 @ 2.40GHz processors and 128 GB of memory. We used MVAPICH2 version 1.7. The codes were compiled with the Intel compiler version 12.1. The msr-safe kernel module provides direct access to Intel RAPL registers via libmsr [35]. We used the package (PKG) domain of RAPL that provided us the capability of capping power for each of the processors in an experiment. The scheduling environment was simulated in R.

We again used EP, BT, and SP from the NPB suite and CoMD from the Mantevo suite in their pure MPI versions. We exponentially increase the node count for our experiments. The inputs were weakly scaled for different node counts. We report performance in terms of completion time in seconds and power in Watt. The reported numbers are averages across ten runs.

9. RESULTS

Experiments were conducted for single job power tuning and multi-job power partitioning.

Variation under Power Caps: Sorting Required

We now exploit the observed variability in the unbounded power consumption of the processor chips, which translates into variation in performance under a power constraint. This variability may be caused by factors such as manufacturing/process variation (at CMOS/transistor level), ambient machine room temperature in different rack positions (higher/lower to the floor), or others. Yet, our method handles variation irrespective of its causes. Previous work [27] and Section 2 has already established that a cluster is not homogeneous under a power constraint because of such variation. We also observe that scheduling a job on different sets of fixed number of processors under a constant power budget leads to variation in the performance of a parallel job.

We present a selection of configurations to demonstrate this behavior in Figures 11 and 12. The x-axis represents the codes and the number of processors. The y-axis indicates the completion time in seconds. The codes are run on several combinations of processors from one or more quartiles of the processor distribution. The numbers on the top of the bars indicate percentage slowdown with respect to the baseline. The processors are uniformly capped at 51W in this set of experiments, i.e., they maintain a constant job power budget of 8KW, 16KW, and 32KW for 16, 32, and 64 processor experiments, respectively. The baseline for 16 processor experiments (Fig. 11) is the performance on the processors belonging to quartile Q1. For 32 and 64 processors (Fig. 12), the baseline is the performance on the processors belonging to Q1 and Q4 (also see legends). Q1 consists of the most efficient processors whereas Q4 consists of the least efficient processors. We observe a performance slowdown ranging from 2% to 18%. We observe that performance deteriorates as we include less efficient processors (Q2, Q3, Q4) in the mix. Hence, the optimal selection of n_{opt} processors should consist of the most efficient processors from the available ones.

PTune

We evaluate the effectiveness of PTune using the aforementioned codes. In Fig. 13, we present results for three different combinations of processors belonging to different quartiles. There are three data points corresponding to each code.

In the figures, n_{LOWER} (synonymous with $n_{\perp}$) is the maximum number of processors that can operate at maximum

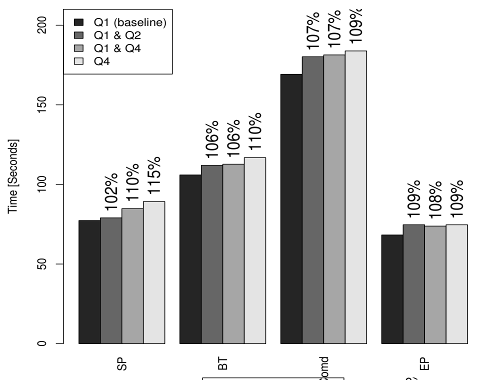

Figure 11: Performance variation on 16 processors

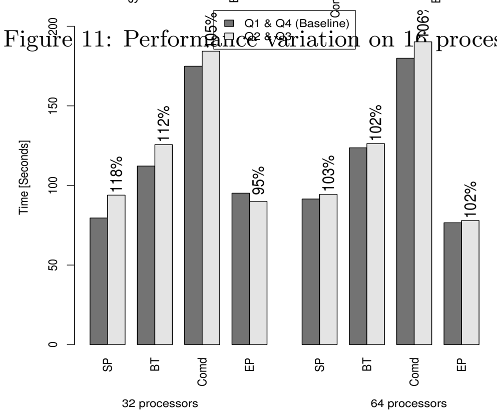

Figure 12: Performance variation on 16 and 32 processors

power such that their aggregate power does not violate the job level power constraint. This configuration most closely resembles the worst case power provisioning as processors are not power constrained. PTune is the data point corresponding to optimal configuration suggested by the power tuner. Uniform power corresponds to the naïve approach of distributing the job's power budget evenly across all processors in a job. This is the baseline configuration.

Performance

In Fig. 13, the y-axis represents performance (top graph) in terms of wall-clock time (in seconds) and the number of processors recommended by the power manager (bottom graph) over different codes and quartiles to which the processors belong (x-axis). The numbers on the bars indicate the runtime reduction and utilized number of processors relative to the baseline in percent.

We observe a performance improvement of up to 22%. The gains are dependent on the combination of processors from different quartiles as well as on workload. PTune is able to free up to 38% of the resources while achieving similar or higher performance than the baseline configuration.

Scalability

We evaluate PTune on up to 128 processors. Fig. 14 presents results addressing the scalability of PTune. PTune achieves performance improvements of as much as 29% with a minimum of 1%. More significantly, in case of the minimal performance improvement, PTune frees up 23% of the processors, which subsequently become available to the next scheduled jobs. We observe an error of less than 2% between the total job power consumption (measured via RAPL) of the PTune recommended configurations and the assigned job-level power budget across all experiments.

PPartition

In this section, we perform a macro-level evaluation of our 2-level model. We simulate the conventional scheduler that dispatches jobs from multiple queues, one at a time. Let np be the number of processes. The scheduler handles 3 queues, 1 large job queue ($np \geq 768$ or $n \geq 64$ processors), and two backfill queues ($np \leq 48$ or $n \leq 4$ processors, $48 < np < 768$ or $4 < n < 64$ processors). Larger jobs are scheduled first followed by backfilling jobs to improve the system utilization. We assume $Nmax = 550$ processors. Our job mix consists of 25% jobs from each EP, SP, BT, and CoMD.

We assume a hardware overprovisioned machine with a machine power budget $P_{m/c} = 28KW$. Fig. 15 depicts a scenario in which the job scheduler is oblivious of power management. The machine's power budget is uniformly distributed across all the processors. We call this naïve scheduling. The conventional scheduler schedules jobs as long as the required number of processors are available. Fig. 16 depicts the scenario when our power manager (PTune + PPartition) co-ordinates with the conventional scheduler to make variation-aware power and job scheduling decisions. The x-axes in both the plots represent job identifiers ordered by the time that they are dispatched by the conventional scheduler. We can see that the large 64 processor job is scheduled first followed by the backfilling jobs. The left y-axis denotes job performance in IPS. Each of the red, green and blue curves represents a job's performance as more and more jobs are scheduled over time (moving right along the x-axis).

Our scheduler starts with jobs at high power budget and, hence, high performance. But as more jobs are dispatched, power is stolen from the previously scheduled jobs. This leads to a drop in their performance. In return, we are able to schedule more jobs at the expense of the performance of already running jobs. In this scenario, our scheme is able to schedule 58 jobs whereas the power-oblivious scheduler is able to schedule only 36 jobs to run at the same time. This is because PTune schedules each job on a reduced number of processors compared to the naïve scheme. The performance of most of the first 36 jobs (that are scheduled under both the schemes) of our approach is at least as good as the naïve one. In addition to these jobs, our power control is able to schedule 22 more backfill jobs that further improve the overall throughput (SysIPS) of the machine (compared to the naïve approach) under the same power constraint.

The right y-axes depict the system's power consumption as a fraction of (normalized to) the overall provisioned system power, $P_{m/c}$, in one line graph (circles) and the system's performance ($SysIPS = \sum JobIPS_i$) normalized to the maximum in the other (crosses). Both graphs track each other closely, but under our power control, the machine power is fully utilized much earlier (after ≈ 10 jobs) whereas 36 jobs are required to reach this level in the naïve case. These initial jobs also achieve higher performance under our scheme (>1500 Billions IPS) than that in the naïve case (900 to 1500 Billion IPS for backfill jobs and 2100 Billion IPS for the large job) before the other jobs are scheduled, and these jobs would thus terminate earlier as they have progressed further under our power control compared to the naïve case. This shows that when there are fewer jobs running on a machine, our power manager is able to direct all machine power to jobs where it is needed to maxi-

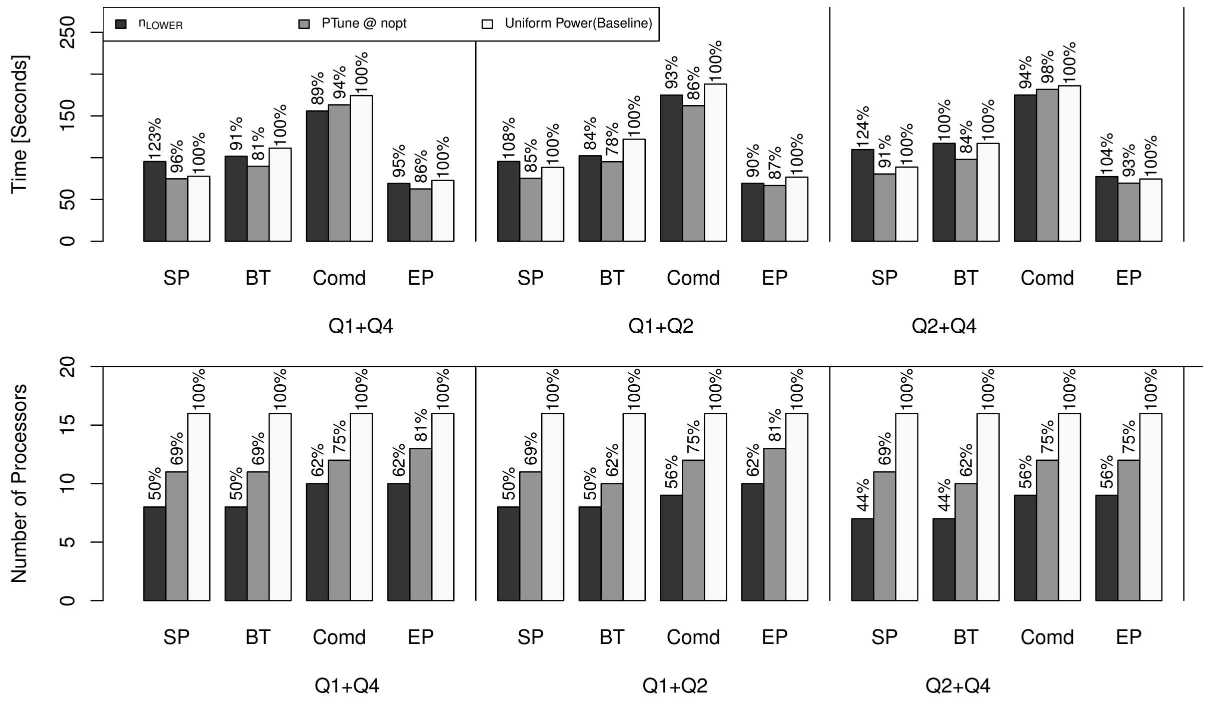

Figure 13: Evaluation of PTune on 16 processors from one or more quartiles

mize performance under a power constraint unlike the naïve approach.

Fig. 17 presents a comparison of the throughput of our scheme compared to three other naïve schemes. The x-axis denotes the machine's power budget and the y-axis depicts the throughput of the machine (SysIPS) normalized to a maximum throughput at 39KW (left bar per set). Uniform capping schemes assume that an appropriate number of randomly selected processors on the machine are already capped at $P_{m/c}/N_{max}$, 75W (mid-way between minimum power and TDP), and TDP, such that their aggregate power does not exceed the machine's power budget. The rest of the processors in these configurations are not available to the conventional scheduler in the naïve scheme. PTune+PPartition represents our model that makes variation-aware decisions about scheduling jobs across the entire machine under a machine-level power constraint. The percentages on top of bars indicate how much lower the throughput per naïve scheme is compared to our solution. Our model achieves 5-35% higher throughput.

Fig. 18 depicts the performance of all the jobs that are scheduled under schemes 1-4 (top left legend) indicated along the x-axis. The y-axis denotes job's performance normalized wrt. to the aggregate job performance (SysIPS) under the respective schemes. We see that our approach (scheme 1) is able to schedule a much larger number of jobs (denser clusters in plot) than the naïve scheduling policy (scheme 2) by trading off performance of some jobs.

10. RELATED WORK

Energy has been an important issue in high performance computing (HPC) for over a decade. Supercomputers as old as BlueGene/L have been built with the goal of maximizing power efficiency. Power-scalable clusters that are equipped with voltage and frequency scaling have existed for over a decade that enabled researchers to study the energy problem in HPC. Freeh et al. [14] investigated the energy-time trade off of MPI applications to prove that it is feasible to save energy by scaling the processor down to lower energy levels with or without time penalty depending on the application. Springer et al. [36] proposed a combined approach of performance modeling and performance prediction for minimizing the execution times of MPI applications under energy bounds. They used voltage and frequency scaling on single cores of a small cluster of up to 10 nodes for their experiments. In addition, there is abundant work presenting algorithms that use frequency and voltage scaling mechanisms for energy savings [23, 28, 29, 13, 4]. In contrast, our work uses power capping via the Intel RAPL interface. Totoni et al. [38, 22] presented an ILP-based runtime system that schedules work on a selective *subset* of cores of a single multi-core chip to meet the power or performance constraint. Within-die or core-to-core variation-aware DVFS schemes [37, 16] have been proposed for chip-multiprocessors. These schemes select optimal voltage and frequency set points for each of the cores to achieve improved power to performance ratio for the chip. Our work differs from this work in terms of granularity. We study variation across several processors or chips and not across cores of a single multi-core chip. We manage resources at processor (or chip) level. We use either *all or none* of the cores of the chips/multi-core processors on a machine. Our goal is to improve the performance of a parallel job scheduled on multiple processors under a strict power budget.

System-wide solutions for power constraint systems have been proposed that aim at increasing the throughput of systems by leveraging the idea of *hardware overprovisioning* [12, 24, 32, 33, 11]. Sarood et al. [34] proposed a scheme of determining an optimal number of nodes under strong scaling of applications executing on an overprovisioned system while

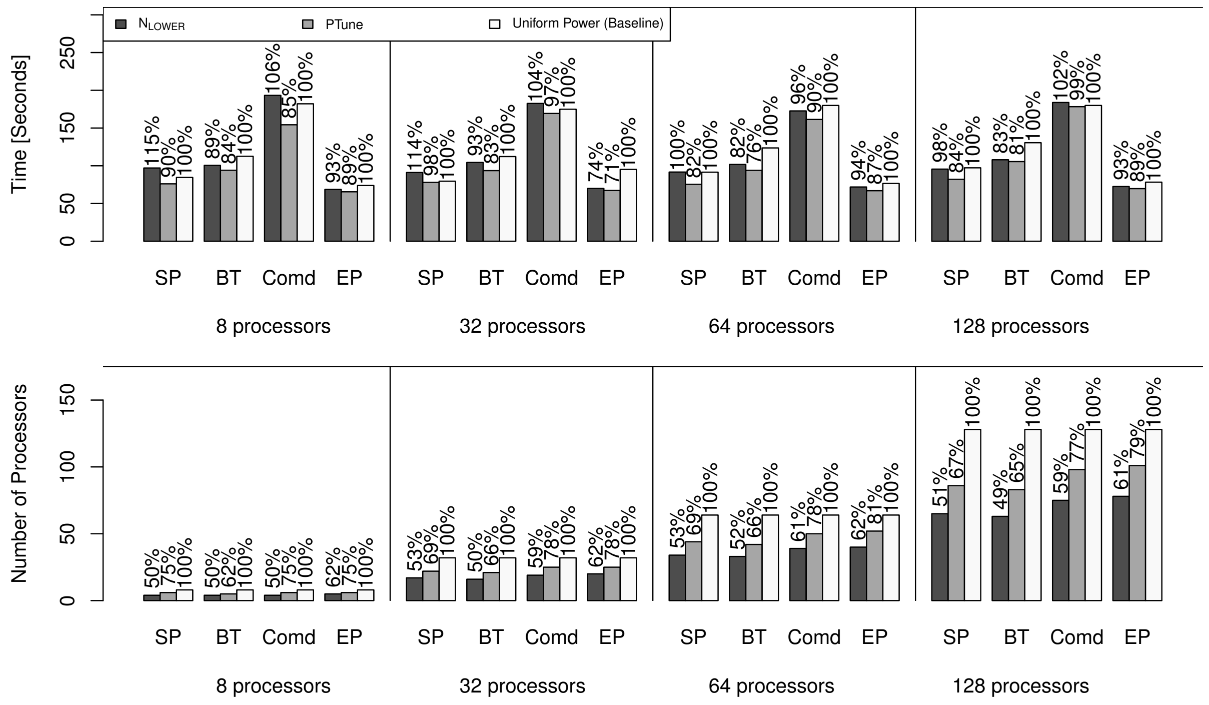
Figure 14: Evaluation of PTune on processors from Q1 and Q4 quartiles

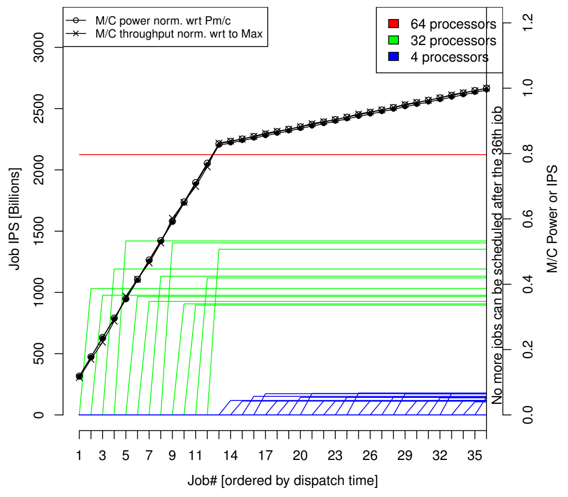
Figure 15: Uniform power distributed across the machine. $P_{m/c} = 28kW$

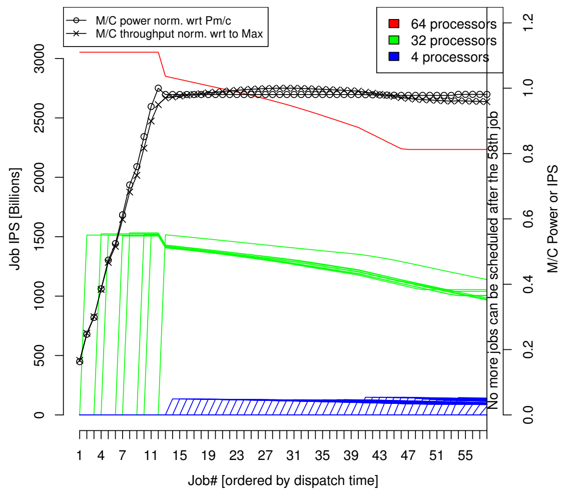
Figure 16: PPartition + PTune. $P_{m/c} = 28kW$.

distributing power between CPU and memory. Etinski et al. [10, 9] proposed the use of dynamic voltage and frequency scaling (DVFS) at the job scheduling-level to save energy and improve overall job performance. Patki et al. [25] proposed power-aware backfilling to improve the throughput of the system. Ellsworth et al. [8] presented a power scheduler that enforced a system-wide power bound by reallocating power across the cluster. Our work differs from all of the above because our approach takes the performance variation across processors of a cluster into account while scheduling and tuning jobs for performance.

Inadomi et al. [18] propose performance optimizations across inhomogeneous processors. It provides a detailed analysis of the phenomenon across multiple clusters and provides a set of simple algorithms for intelligent power balancing. These algorithms, while groundbreaking, suffered from two serious limitations. First, the processor power model assumed that CPU clock frequency increased proportionally with power. While that is a useful simplification, our work here shows that the story is not nearly so simple. Second, the algorithms assumed the ideal number of nodes to use was fixed a priori. In our approach, PPart and PTune jointly determine the ideal number of nodes and the power budget for every job at the time of scheduling. While the

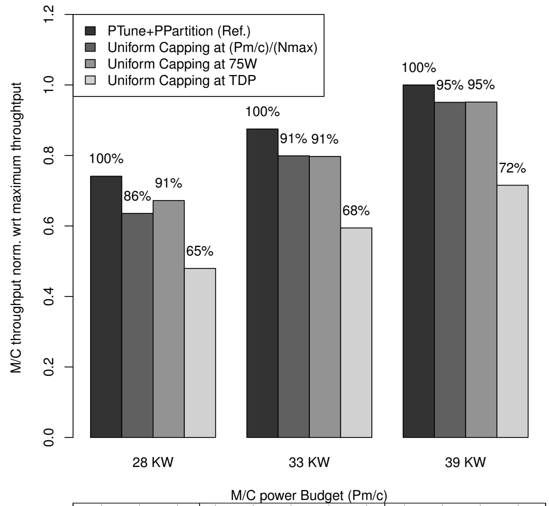

Figure 17: Throughput

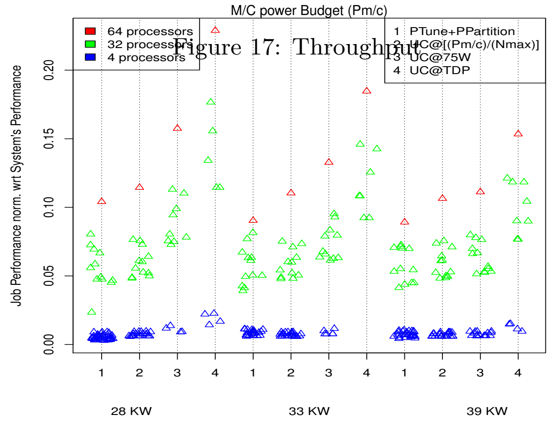

Figure 18: Job performance. A job is represented by a triangle

result of both of our approaches is a power schedule, we are solving a fundamentally different problem.

Kappiah et al. [21] presented a system that saves energy at the expense of execution time by scaling down the frequencies of the cores when they encounter slack time in an MPI application. Rountree et al. [28] used linear programming to establish a bound on optimal energy savings of an MPI application and presented a runtime algorithm to save energy in HPC applications with negligible delay [29]. Power conservation by means of turning off unwanted nodes is proposed in [26]. In the above presented solutions, authors used one core per node and their goal was to maximize energy savings with minimal impact on the execution time. In contrast to these solutions, we are intolerant to performance degradation. We use multicore processors and our goal is to minimize the completion time as long as we stay within the power budget.

11. SUMMARY

We presented a hierarchical variation-aware machine-wide solution for managing power on a hardware overprovisioned machine. It consists of a macro-level Power Partitioner that makes power and job scheduling decisions and a micro-level Power Tuner that determines the optimal processor selection and their power caps for a job such that its performance is maximized under a power constraint. PTune achieves up to 29% improvement in performance compared to uniform power capping. It does not lead to any performance degradation, yet frees up to 40% of resources compared to uniform power capping. PPartition is able to improve the throughput of the machine by 5-35% compared to naïve scheduling under the same machine power budget.

We established that under a power constraint, the variability in performance transforms into variation in peak power efficiency. We believe that this variation in power efficiency should be one of the primary considerations in the future power management research.

12. ACKNOWLEDGEMENTS

We extend our thanks to Livermore Computing for facilitating Dedicated Access Time on the LLNL cluster. This work was supported in part by NSF grants 1217748, 1058779, 0958311, and by Consortium for Advanced Simulation of Light Water Reactors (CASL). This material is also based upon work supported by the U.S. Department of Energy's Lawrence Livermore National Laboratory. Office of Science, under Award number DE-AC52-07NA27344 and supported by Office of Science, Office of Advanced Scientific Computing Research (LLNL-CONF-656877, LLNL-CONF-698261).

13. REFERENCES

[1] U.s. energy information administration. https://www.eia.gov/electricity/annual/html/epa_04_03.html.

[2] Top 500 list. http://www.top500.org/, June 2002.

[3] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, D. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. The NAS Parallel Benchmarks. *The International Journal of Supercomputer Applications*, 5(3):63–73, Fall 1991.

[4] P. E. Bailey, A. Marathe, D. K. Lowenthal, B. Rountree, and M. Schulz. Finding the limits of power-constrained application performance. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, pages 79:1–79:12, New York, NY, USA, 2015. ACM.

[5] K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, J. Hiller, S. Karp, S. Keckler, D. Klein, R. Lucas, M. Richards, A. Scarpelli, S. Scott, A. Snavely, T. Sterling, R. S. Williams, K. Yelick, K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, J. Hiller, S. Keckler, D. Klein, P. Kogge, R. S. Williams, and K. Yelick. Exascale Computing Study: Technology Challenges in Achieving Exascale Systems. 2008.

[6] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De. Parameter variations and impact on circuits and microarchitecture. In *Design Automation Conference, 2003. Proceedings*, pages 338–342, June 2003.

[7] B. Dally. Power and programmability: The challenges of exascale computing. In *Presentation at ASCR Exascale Research PI Meeting*, 2011.

[8] D. A. Ellsworth, A. D. Malony, B. Rountree, and M. Schulz. Pow: System-wide dynamic reallocation of

limited power in hpc. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '15, pages 145–148, New York, NY, USA, 2015. ACM.

[9] 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.

[10] M. Etinski, J. Corbalan, J. Labarta, and M. Valero. Utilization driven power-aware parallel job scheduling. *Computer Science - R&D*, 25(3-4):207–216, 2010.

[11] M. E. Femal and V. Freeh. Boosting data center performance through non-uniform power allocation. In *International Conference on Autonomic Computing*, pages 250–261, 2005.

[12] M. E. Femal and V. W. Freeh. Safe overprovisioning: using power limits to increase aggregate throughput. In *In International Conference on Power-Aware Computer Systems*, December 2005.

[13] V. Freeh, F. Pan, N. Kappiah, and D. K. Lowenthal. Using multiple energy gears in mpi programs on a power-scalable cluster. In *PPoPP*, pages 164–173, June 2005.

[14] V. Freeh, F. Pan, N. Kappiah, D. K. Lowenthal, and R. Springer. Exploring the energy-time tradeoff in mpi programs on a power-scalable cluster. In *IPDPS*, May 2005.

[15] R. Ge, X. Feng, S. Song, H.-C. Chang, and D. Li. Powerpack: Energy profiling and analysis of high-performance systems and applications. May 2010.

[16] S. Herbert and D. Marculescu. Variation-aware dynamic voltage/frequency scaling. In *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on*, pages 301–312, Feb 2009.

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

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

[19] InsideHPC. Power Consumption is the Exascale Gorilla in the Room. http://insidehpc.com/2010/12/.

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

[21] N. Kappiah, V. Freeh, and D. K. Lowenthal. Just in time dynamic voltage scaling: Exploiting inter-node slack to save energy in mpi programs. In *SC*, Nov 2005.

[22] A. Langer, E. Totoni, U. S. Palekar, and L. V. Kalé. Energy-efficient computing for hpc workloads on heterogeneous manycore chips. In *Proceedings of Programming Models and Applications on Multicores and Manycores*. ACM, 2015.

[23] M. Y. Lim, V. Freeh, and D. K. Lowenthal. Adaptive, transparent frequency and voltage scaling of communication phases in mpi programs. In *SC*, 2006.

[24] 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.

[25] T. Patki, D. K. Lowenthal, A. Sasidharan, M. Maiterth, B. Rountree, M. Schulz, and B. R. de Supinski. Practical Resource Management in Power-Constrained, High Performance Computing. In *HPDC*, 2015.

[26] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath. Load balancing and unbalancing for power and performance in cluster-based systems. In *Workshop on compilers and operating systems for low power*, 2001.

[27] 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*, pages 947–953. IEEE Computer Society, 2012.

[28] B. Rountree, D. K. Lowenthal, S. Funk, V. Freeh, B. R. de Supinski, and M. Schulz. Bounding energy consumption in large-scale mpi programs. In *SC*, pages 10–16, Nov 2007.

[29] B. Rountree, D. K. Lowenthal, M. Schulz, V. Freeh, and T. Bletsch. Adagio: Making dvs practical for complex hpc applications. In *ICS*, Nov 2009.

[30] Sandia National Laboratory. Mantevo project home page. https://software.sandia.gov/mantevo, June 2011.

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

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

[33] O. Sarood, A. Langer, A. Gupta, and L. V. Kale. Maximizing throughput of overprovisioned hpc data centers under a strict power budget. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '14, New Orleans, LA, 2014. ACM.

[34] O. Sarood, A. Langer, L. V. Kale, B. Rountree, and B. de Supinski. Optimizing Power Allocation to CPU and Memory Subsystems in Overprovisioned HPC Systems. In *Proceedings of IEEE Cluster 2013*, Indianapolis, IN, USA, September 2013.

[35] K. Shoga, B. Rountree, M. Schulz, and J. Shafer. Whitelisting msrs with msr-safe. In *3rd Workshop on Extreme-Scale Programming Tools at SC*, Nov. 2014. http://www.vi-hps.org/upload/program/espt-sc14/vi-hps-ESPT14-Shoga.pdf.

[36] R. Springer, D. K. Lowenthal, B. Rountree, , and V. Freeh. Minimizing execution time in mpi programs on an energy-constrained,power-scalable cluster. In *PPoPP*, May 2006.

[37] 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.

[38] E. Totoni, A. Langer, J. Torrellas, and L. Kale. Scheduling for hpc systems with process variation heterogeneity. In *Technical Report YCS-2009-443, Department of Computer Science, University of York*, 2014.

[39] L. Zhang, L. S. Bai, R. P. Dick, L. Shang, and R. Joseph. Process variation characterization of chip-level multiprocessors. In *Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE*, pages 694–697, July 2009.