TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 12/12 pp370–383
DOI: 10.26599/TST.2020.9010012
Volume 26, Number 3, June 2021

More Bang for Your Buck: Boosting Performance with Capped Power Consumption

Juan Chen*, Xinxin Qi, Feihao Wu, Jianbin Fang, Yong Dong, Yuan Yuan, Zheng Wang, and Keqin Li

Abstract: Achieving faster performance without increasing power and energy consumption for computing systems is an outstanding challenge. This paper develops a novel resource allocation scheme for memory-bound applications running on High-Performance Computing (HPC) clusters, aiming to improve application performance without breaching peak power constraints and total energy consumption. Our scheme estimates how the number of processor cores and CPU frequency setting affects the application performance. It then uses the estimate to provide additional compute nodes to memory-bound applications if it is profitable to do so. We implement and apply our algorithm to 12 representative benchmarks from the NAS parallel benchmark and HPC Challenge (HPCC) benchmark suites and evaluate it on a representative HPC cluster. Experimental results show that our approach can effectively mitigate memory contention to improve application performance, and it achieves this without significantly increasing the peak power and overall energy consumption. Our approach obtains on average 12.69% performance improvement over the default resource allocation strategy, but uses 7.06% less total power, which translates into 17.77% energy savings.

Key words: energy efficiency; high-performance computing; performance boost; power control; processor frequency scaling

1 Introduction

High-Performance Computing (HPC) systems capability is increasingly constrained by their power consumption, and this will become worse due to the end of Dennard scaling[1,2]. However, HPC users want a higher performance to run the more complex model or larger datasets. As such, there is a vital need to find ways to boost the efficiency of HPC applications without increasing power consumption significantly.

Power and performance optimization for HPC systems is certainly not a new research topic. There is considerable work on designing innovative compute architecture for better energy efficiency for HPC systems[3–11]. Other work utilizes a software-based resource scheduling approach by carefully determining the computation resource settings, such as the number of assigned compute nodes and processor frequency to match the workload to improve application performance under a power constraint[12,13]. A key advantage of a software-based approach is that it is readily deployable on existing hardware as no hardware modification is required[14,15].

The allocation of computing resources, such as computing nodes or Control Processing Unit (CPU) cores, is essential for power and performance optimization. Current resource allocation strategies

• Juan Chen, Xinxin Qi, Feihao Wu, Jianbin Fang, Yong Dong, and Yuan Yuan are with the College of Computer, National University of Defense Technology, Changsha 410073, China. E-mail: juanchen@nudt.edu.cn; qixinxin19@nudt.edu.cn; wufeihao16@nudt.edu.cn; j.fang@nudt.edu.cn; yongdong@nudt.edu.cn; yuanyuan@nudt.edu.cn.
• Zheng Wang is with the College of Computer, University of Leeds, London LS2 9JT, UK. E-mail: z.wang5@leeds.ac.uk.
• Keqin Li is with the School of Science and Engineering, State University of New York, New York, NY 12561, USA. E-mail: lik@newpaltz.edu.
∗ To whom correspondence should be addressed.
Manuscript received: 2020-03-25; accepted: 2020-04-02

aimed at maximizing system use, i.e., no additional computing nodes would be allocated unless the processor cores of existing nodes have been fully utilized. Despite these techniques, they do not offer the optimal performance for memory-bound applications by using as few computing nodes as possible to limit the overall power consumption. As we can see in this paper, maximizing processor core usage can lead to serious memory contention, which in turn leads to sub-optimal performance. Because the memory sub-system is now becoming a bottleneck of HPC systems[16, 17], we need a better resource allocation strategy to effectively tackle the memory contention problem to achieve higher performance.

This work aims at developing a new approach for HPC resource allocation, specifically targeting memory-bound data-parallel applications. Our insight is that instead of maximizing processor utilization, one can allocate additional computing nodes to reduce memory contention. If we can do this, we can then free some processor cores (as well as reduce the number of parallel processes) on a node to reduce memory contention to improve memory-bound workload performance.

However, translating this high-level idea into a practical system is non-trivial. A key challenge is to determine the optimal number of computing nodes to be allocated to an application, for a given number of user-requested processes. If we allocate too few or too many computing nodes, we either will be unable to gain much, or the extra power consumption will outweigh the benefit. Also, our scheduling decision would need to ensure that additional computing nodes will not generate substantially more power consumption as compared to the default approach. To overcome these challenges, we need a new technique to accurately model the application behavior to drive a precise resource allocation strategy.

In this paper, we present a novel HPC resource allocation strategy, which aims to boost the application performance without significantly compromising the power consumption budget for memory-bound workloads. We achieve this by firstly characterizing the application workload behavior based on offline profiling information (Section 3.1). The workload characteristics and profiling information are used to decide how many computing nodes the target application will be run. We then propose a new technique for modeling the subtle interactions between application performance, power consumption, compute nodes, memory bandwidth congestion, and CPU clock frequency. The model is then used as a method to change the number of allocated computing nodes and the CPU frequency to fit the hardware configuration to the workload behavior. Specifically, if the target application is perceived to be a memory-bound workload, we reduce the CPU frequency to lower the system's power consumption, as the bottleneck is on memory assessment.

We apply our approach to 12 representative Message Passing Interface (MPI) benchmarks from the NAS parallel benchmark and HPC Challenge (HPCC) benchmark suites. Our evaluation platform is an HPC cluster with 20 cores per node. Experimental results show that our approach achieves on average a 12.69% performance improvement over the conventional resource allocation strategy, but uses 7.06% less total power, which translates into 17.77% energy savings.

This paper makes the following technical contributions:

(1) We show that by allocating additional computing nodes with appropriate CPU frequency settings, one can boost application performance without increasing power consumption of the system for memory-bound workloads (Section 2.2).

(2) We present a novel technique for capturing interactions between workload characteristics and computation resource allocation and CPU settings (Section 3.1).

(3) We present a novel algorithm for scheduling and configuring resource allocation at the application level, showing how performance can be enhanced without infringing the overall power consumption constraint (Section 3.2).

2 Background and Overview

2.1 Problem scope

Our work tackles the question of resource allocation for distributed HPC workloads. We target typical HPC environments where the user job is submitted to a job queue. As part of the job submission, the user may specify the number of processes required. We are interested in developing a resource scheduler for mapping requested resources (i.e., the number of parallel processes) to compute nodes in this work. A default strategy for the problem would be to allocate the minimal number of compute nodes to run one parallel process on a physical core. However, as we will show in this paper, such an approach often leads to sub-optimal performance for memory-intensive applications. Our goal is to

determine the optimal number of compute nodes and the processor clock frequency of each compute node to reduce the running time of memory-bound applications, while at the same time to cap the peak power and energy consumption as the default strategy.

2.2 Motivating example

Considering scheduling bt and cg from the NAS MPI parallel benchmark suite on an HPC cluster where each computing node has 20 processor cores (see Section 4.1), in this example, we assume that a user requests to run bt and cg with 169 and 128 processes respectively and both benchmarks run with the class D input.

2.2.1 Room for performance improvement

Figure 1 shows how the number of compute nodes affects application performance. The baseline here is to maximize the processor utilization, i.e., using the minimum number of compute nodes where each parallel process runs on a physical processor core. With this baseline strategy, on our evaluation platform, we would allocate 9 and 7 compute nodes respectively to bt and cg. In this example, we experimentally allocate one and two extra compute nodes to the applications but keep the number of parallel processes unchanged (which will be evenly distributed across compute nodes).

While running the same number of parallel processes on more distributed machines incurs a higher communication latency, both applications benefit from improved parallel processes distribution. We believe that the performance improvement is due to less memory contention between parallel processes. To verify our hypothesis, we use Intel VTune Amplifier to profile how often CPU was stalled on the main memory (DRAM) on a single computing node. This lower-is-better metric is known as DRAM bound, which captures the computing node's RAM contention. Figure 2 shows how DRAM bound on a single node and the total memory traffic under different node allocation policies. Using more compute nodes can lead to a lower DRAM bound value,

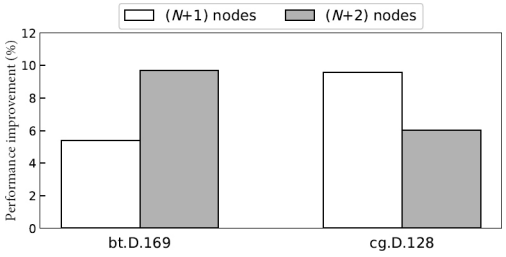

Fig. 1 Performance improvement when adding additional computing nodes without using extra parallel processes.

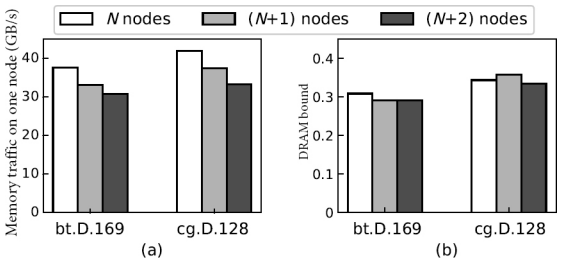

Fig. 2 How memory traffic on the master node changes when the number of computing nodes increases.

which suggests that increasing the number of compute nodes for these two applications can reduce the memory contention and the frequency of CPU stalls. The less memory contention and CPU stalls thus lead to improved running times. However, the sweet spot for computing node allocations varies across applications. For bt, using two additional nodes leads to better performance, but this would lead to slowdown for cg when using just one additional node.

2.2.2 Power & energy consumption

Figure 3 compares the total peak power of different number of nodes and different frequencies. The red dash line denotes the target total power value of 2.6 GHz under N nodes. Program bt consumes 2076 W at 2.6 GHz under nine nodes ($N = 9$). When one and two nodes are added, the total peak power increases to 2184 and 2305 W, respectively. To constrain the total power consumption, we have to lower the CPU frequency. Frequencies 2.4 and 2.2 GHz are set for one and two increased nodes, respectively. This diagram shows that although using additional compute nodes results in a significant increase in total power, by scaling down processor frequencies the total power can be constrained to be at the original level. Figure 4 compares energy consumption when the program runs with the same number of processes but uses different computing nodes and CPU frequencies. Without changing the CPU frequency, using more computing nodes is likely to lead

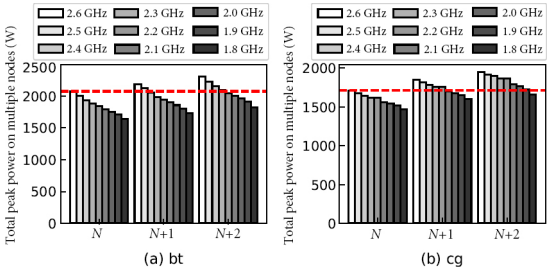

Fig. 3 Total peak power on multiple nodes changes when the number of nodes increases. The red dash line denotes the target total power value of 2.6 GHz under N nodes.

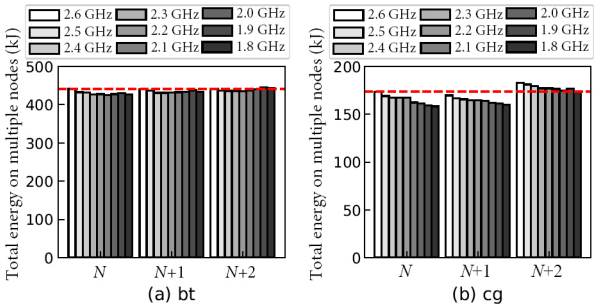

Fig. 4 Total energy consumption on different CPU frequencies when using different number of compute nodes. The red dash line denotes the target total energy consumption of 2.6 GHz under N nodes.

to an increase in energy consumption. However, we can achieve faster job completion time with less energy consumption by carefully tuning the CPU frequency and the additional number of computing nodes. The optimum CPU frequency, however, depends on the number of computing nodes and the program.

To summarize, running examples have shown that achieving better performance without increasing the power consumption and energy usage by the use of additional compute nodes is possible. However, the optimum number of compute nodes and the CPU frequency setting depend on both user-specific resource requirements and workload characteristics. In the remainder of this paper, we show how an adaptive resource scheduler can be developed to perform compute resource allocation to boost the application performance without increasing the total power consumption and energy consumption.

2.3 Overview of our approach

Our approach is implemented on an HPC cluster as part of the central job scheduler used. This is completely automated, and no modification to the application source code is required. To determine the compute resources to use for a job that is ready to run, our approach takes as input the program binary, input data, and user-supplied requirement the number of parallel processes in our case. It then determines the number of compute nodes to be provided to the job and the CPU's clock frequency to be used across allocated machines.

To determine the optimal hardware resource allocation and frequency settings, we first profile the application as part of the strategy for resource allocation to capture the memory and CPU characteristics of the target application (Section 3.1). Specifically, VTune is used to collect the memory trace and performance metrics of the target application from a single node, and we use Intel Running Average Power Limit (RAPL) to take power measurements from all computing nodes. Then, under different allocation strategies, the profiling information is used to estimate the performance gain and energy consumption to find the optimum setting (Section 3.1). Finally, we actively analyze the instantaneous power readings to dynamically adjust the CPU frequency to cap the peak power of each computing node (Section 3.2) .

3 Our Approach

The heart of our approach is a set of functions for modeling how the resource allocation and frequency setting affect the application's performance and power consumption.

3.1 Power and performance modeling

The overall goal of our power-performance model is to improve the program performance without increasing the total power consumption. It achieves this by finding the relationship of energy consumption of allocated computing nodes, CPU frequencies, and parallel execution time. The application execution time, T, and the energy consumption of N allocated computing nodes depend on three parameters: the number of computing nodes, N, the processor frequency, f_i, and the application characteristics (memory-bound or not in this paper). With these notations in place, our optimization goal can be formulated as

$$\begin{aligned} \min \quad & T(N, \Delta N, f_i, B_N) \\ \text{s.t} \quad & \Delta P^+ + \Delta P^- \leqslant 0 \\ & \Delta P^+ \geqslant 0 \\ & \Delta P^- \leqslant 0 \end{aligned} \tag{1}$$

where ΔP^+ denotes the increased power due to ΔN nodes in addition to the one given by the default resource allocation policy. Specifically, the default policy would assign N computing nodes with all processor cores of a node to be completely used and run at the highest CPU frequency $f_{\max}$. We use ΔP^- to denote the power savings resulted from the scaling down of the CPU frequency.

Table 1 lists all the parameters used by our model and their descriptions. $P^{\text{cpu}}(f_i)$ is the energy consumption of a single CPU on one computing node, which depends on the processor frequency, f_i. P^{mem} is the energy consumption of one computing node for the memory sub-system. We observe that P^{mem} is relatively stable (with a variance of less than 2%) for a given program, regardless of the number of computing nodes to be used.

Table 1 Parameters used by our power-performance model.

Notation	Description
c	Number of processor cores on each computing node
n	Number of processes
N	Number of assigned computing nodes with the default resource allocation strategy
ΔN	Number of computing nodes to be increased
$b_i(t)$	Memory traffic of computing node i at time t
$B_N(t)$	Average value of $b_i(t)$ for N nodes, $0 \leqslant i \leqslant N$
B_N	Maximum value of $B_N(t)$, $0 \leqslant t \leqslant T$ where T represents program running time on N nodes
B	Physical memory bandwidth on a single computing node
α	Threshold for the ratio of memory traffic to memory bandwidth
$f_{\max}$	Maximum CPU clock frequency
$f_{\min}$	Minimum CPU clock frequency
f_i	Processor clock frequency, which satisfies $f_{\min} \leqslant f_i \leqslant f_{\max}$
$P^{\text{cpu}}(f_i)$	Power consumption of one CPU core with clock frequency f_i
$P^{\text{cpu}}_{\text{idle}}$	Power consumption for each CPU core in the idle state, which equals $k \cdot P^{\text{cpu}}(f_{\max})$
P^{mem}	Power consumption of DRAM on one computing node
P^{other}	Power consumption of compoents other than the CPU and DRAM on one computing node

As such, we set P^{mem} to be a constant value for a given program.

3.1.1 Memory behaviour characterization

Suppose we allocate N computing nodes to run a parallel program. Assume at time t ($0 \leqslant t \leqslant T$), the memory traffic on each node is $b_1(t), b_2(t), \ldots, b_N(t)$, respectively. The average memory traffic per node, $B_N(t)$, can then be calculated as

$$B_N(t) = \frac{\sum_{i=1}^{N} b_i(t)}{N} \tag{2}$$

We use the maximum value of $B_N(t)$ across all sample periods to denote the peak memory traffic of the target program. The peak memory traffic, B_N, is denoted as

$$B_N = \max_{0 \leqslant t \leqslant T} B_N(t) \tag{3}$$

Let B denote the memory bandwidth of a computing node. We consider the closer the peak memory traffic, B_N, to the bandwidth, B, the more memory-intensive the application. We consider an application to be memory-bound if B_N/B is greater than a threshold, α. In this work, we only try to provide memory-bound applications with extra computing nodes to reduce memory contention. Given this setting, one may come up with the following intuitive condition:

$$N \cdot B_N = (N + \Delta N) \cdot B_{(N+\Delta N)} \tag{4}$$

However, Eq. (4) is not satisfied with a memory-bound program. As Fig. 2 suggests, the more nodes, the heavier the total memory traffic will be. Note that the total memory traffic of N nodes is less than that of $N+1$ nodes. In other words, $N \cdot B_N$ is usually smaller than the demand for the total memory traffic under memory-bound cases. When sufficient computing nodes are provided (ΔN^{perf}), we can minimize the memory contention. As such, we have the following formulation:

$$N \cdot \left(\frac{\text{bound}}{\alpha} \cdot B_N \right) = (N + \Delta N^{\text{perf}}) \cdot B_{(N+\Delta N^{\text{perf}})} \tag{5}$$

where bound denotes the degree of memory contention. We use VTune in this work to count the bound value for each program, but other profiling tools can also be used. For example, running sp with 169 processes on 9 of our computing nodes gives a bound value of 0.401. As such, the performance improvement, ΔN^{perf}, as a result of using additional computing nodes can be modeled as

$$\Delta N^{\text{perf}} = \frac{N \cdot (\frac{\text{bound}}{\alpha} \cdot B_N)}{B_{(N+\Delta N^{\text{perf}})}} - N.$$

3.1.2 Performance modeling

For memory-bound programs, the increased memory traffic per node after adding extra computing nodes can be calculated as

$$B_{N+\Delta N^{\text{perf}}} = \alpha \cdot B \tag{6}$$

The maximum number of extra computing nodes, ΔN^{perf}, that will be profitable sill, can be defined as

$$\Delta N^{\text{perf}} = \frac{N \cdot \left(\frac{\text{bound}}{\alpha} \cdot B_N \right)}{\alpha \cdot B} - N \tag{7}$$

3.1.3 Power modeling

Since the current multi-core design often does not provide voltage scaling on a per-processor core basis, using extra computing nodes will increase the energy consumption even some of the cores are not utilized, if we do not scale down the CPU frequency. Recall that our goal is to ensure the total power consumption of $(N + \Delta N)$ computing nodes does not surpass that of N computing nodes with a CPU frequency of $f_{\max}$. This constraint can be defined as

$$\begin{aligned} &n \cdot P^{\text{cpu}}(f_{\max}) + (N \cdot c - n) \cdot P^{\text{cpu}}_{\text{idle}} + P^{\text{mem}} \cdot N + \\ &P^{\text{other}} \cdot N \geqslant n \cdot P^{\text{cpu}}(f_{\text{mid}}) + ((N + \Delta N) \cdot c - n) \times \\ &P^{\text{cpu}}_{\text{idle}} + P^{\text{mem}} \cdot (N + \Delta N) + P^{\text{other}} \cdot (N + \Delta N) \end{aligned} \tag{8}$$

where

$$P_{\text{idle}}^{\text{cpu}} = k \cdot P^{\text{cpu}}(f_{\max}).$$

Here, the left term of Formula (8) represents the default allocation policy for N nodes. The right term represents our strategy for using $(N + \Delta N)$ nodes. In Formula (8), we assume ΔN numbers of nodes will lead to $\Delta N \times c \times P_{\text{idle}}^{\text{cpu}}$ power consumption for idle cores. We also assume the memory power consumption is a constant value, P^{mem}. In order to keep the total power consumption at a similar level as the N allocation scheme, all n busy processor cores frequencies have to be scaled down from $f_{\max}$ to f_{mid}. As such, we take the average of $f_{\max}$ and $f_{\min}$ as f_{mid}.

Using Formula (8), we have the maximum value of ΔN, namely ΔN^{power}, as

$$\Delta N^{\text{power}} = \frac{n \cdot (P^{\text{cpu}}(f_{\max}) - P^{\text{cpu}}(f_{\text{mid}}))}{c \cdot k \cdot P^{\text{cpu}}(f_{\max}) + P^{\text{mem}} + P^{\text{other}}} \quad (9)$$

Take the power consumption into consideration, the number of extra computing nodes to use will need to fall into the range of $[0, \Delta N^{\text{power}}]$.

3.1.4 Putting together

Considering both the power and performance constraints, the additional profitable computing nodes ΔN will be

$$\Delta N \in [0, \Delta N^{\text{perf}}] \cap [0, \Delta N^{\text{power}}] \quad (10)$$

To maximize the performance boost, we take the maximum number of this interval in Formula (10). Then, we can achieve the optimal ΔN^*:

$$\Delta N^* = \max\{[0, \Delta N^{\text{perf}}] \cap [0, \Delta N^{\text{power}}]\} \quad (11)$$

In Formula (8), let $\Delta N = \Delta N^*$, we have

$$\begin{aligned} &n \cdot P^{\text{cpu}}(f_{\max}) + (N \cdot c - n) \cdot P_{\text{idle}}^{\text{cpu}} + P^{\text{mem}} \cdot N + \\ &P^{\text{other}} \cdot N \geqslant n \cdot P^{\text{cpu}}(f_i) + ((N + \Delta N^*) \cdot c - n) \times \\ &P_{\text{idle}}^{\text{cpu}} + P^{\text{mem}} \cdot (N + \Delta N^*) + P^{\text{other}} \cdot (N + \Delta N^*) \end{aligned} \quad (12)$$

where $P^{\text{cpu}}(f_{\text{mid}})$ is replaced by $P^{\text{cpu}}(f_i)$. Then, we have

$$P^{\text{cpu}}(f_i) \leqslant \frac{n \cdot P^{\text{cpu}}(f_{\max}) - \Delta N^* (c \cdot P_{\text{idle}}^{\text{cpu}} + P^{\text{mem}} \cdot P^{\text{other}})}{n} \quad (13)$$

Let the initial optimal processor frequency f^* be defined in such a way that

$$P^{\text{cpu}}(f^*) = \max\{P^{\text{cpu}}(f_i)\} \quad (14)$$

The adjustable frequency range for a parallel program running on our supercomputer system is $[f_{\min}, f_{\max}]$ with 0.1 GHz as the scale. In Table 2, the initial optimal frequency f^* can be found out to make $P^{\text{cpu}}(f^*)$ most approach to $\max\{P^{\text{cpu}}(f_i)\}$. In the section of experiments, we show the detailed value for Table 2 for our experimental platform.

Table 2 Model parameters for single-core power estimation.

CPU frequency (GHz)	$P^{\text{cpu}}(f_i)$ (W)
$f_{\max}$	$P^{\text{cpu}}(f_{\max})$
$f_{\max} - \Delta f$	$P^{\text{cpu}}(f_{\max} - \Delta f)$
$f_{\max} - 2 \times \Delta f$	$P^{\text{cpu}}(f_{\max} - 2 \times \Delta f)$
...	...
$f_{\min}$	$P^{\text{cpu}}(f_{\min})$

Besides the optimal ΔN^* and f^*, we also calculate power consumption of a single CPU after adding nodes P_{f_i}, as shown in Eq. (14). With f^* as the initial frequency and P_{f_i} as a power limit value, we use RAPL to perform power capping. RAPL provides a series of Model Specific Register (MSR) interfaces, such as power limit, time window, clamp bit, etc., to perform power limiting by specifying these interfaces.

3.2 Resource allocation

Our resource allocation scheme is described in Algorithm 1. This algorithm obtains nearly optimal values for a given program, ΔN and f^*, for parallel performance boost without any increase in total power.

The algorithm's input parameters include the maximum/minimum CPU frequencies $f_{\max}/f_{\min}$, physical memory bandwidth B, the threshold for memory traffic to physical memory bandwidth ratio α, the default allocated node number N, the number of processes n, and the number of cores per node c. The outputs are the optimum number of increased computing nodes for the given program ΔN^* and

Algorithm 1 Resource scheduling algorithm

Input: Processor frequency levels $f_{\min}, \ldots, f_{\max}$; physical memory bandwidth B; memory bandwidth bound ratio α; for a given parallel program, the number of assigned computing nodes N under the default resource allocation strategy; number of processes n; and number of cores per node c;

Output: Number of increased computing nodes ΔN^*; the uniform processor frequency f^*; and the target power P_{target} for RAPL.

1: Measure actual power $P^{\text{cpu}}(f_i)$ with different frequencies and $P_{\text{idle}}^{\text{cpu}}$;
2: Run the program and get profiling data by VTune and then determine the values of B_N and bound;
3: Calculate ΔN^{perf} by Eq. (7);
4: During the program runs, we use RAPL to measure one-node memory power P^{mem} and a single-node peak power P_N;
5: Calculate ΔN^{power} by Eq. (9);
6: Calculate ΔN^* by Eq. (11);
7: $P_{\text{target}} = P_N \times \frac{N}{N + \Delta N^*}$;
8: Get $P^{\text{cpu}}(f^*)$ by Eq. (14) and choose a most appropriate clock frequency f^* by Table 2 as the initial frequency;
9: End.

uniform processor frequency f^*.

Our resource allocation scheme works as follows:

Firstly, we measure all the platform related parameters including all power values in Table 2 under different frequency levels and $P_{\text{idle}}^{\text{cpu}}$. Even if at the same frequency level, the actual power consumption of different applications could be different. Actual power is closely related to the utilization rate of CPUs. The power values listed in Table 2 are the power measured under the nearly maximum processor utilization rate.

Secondly, we run the program and obtain corresponding parameters by performance profile data and power profile data. Performance related profile data for the given program include B_N and bound. Power related profile data refer to P^{mem} and the single-node peak power P_N. P_N is used to calculate the objective total power. By Eqs. (7) and (9), we can calculate ΔN^{perf} and ΔN^{power}. The optimal number of increased nodes ΔN^* is obtained by Eq. (11). At the same time, by Eq. (14), $P^{\text{cpu}}(f^*)$ is calculated. According to Table 2, the initial optimal frequency f^* is decided.

Characterizing the applications via profiling does not restrict our approach, as most scientific computing applications usually run many times. Even if the profile-based approach consumes a great deal of time on data profiling, we can still benefit from the later process.

Finally, we get the target power P_{target}, $P_N \times \frac{N}{N+\Delta N^*}$, which is used to control power by RAPL as the given power limit value.

According to this algorithm, to obtain a better optimal frequency setting, we run the program again for real-time power limiting. We use RAPL to ensure that the average power does not surpass P_{target} by automatically adjusting the frequency. In this case, a good starting point could be provided by the initial optimal f^* given by our model. In Section 5, we show how our model can effectively support this endeavor.

4 Experimental Setup

4.1 Platform and benchmarks

Hardware. We evaluate our approach with 64 computing nodes on an HPC cluster. Each of the computing nodes has 64 GB DDR4 RAM and two Intel Haswell 10-core E5-2660v3 processors running at 2.6 GHz. The multi-core processor supports Dynamic Voltage and Frequency Scale (DVFS) with 15 states, from 1.2 to 2.6 GHz at a step of 0.1 GHz. Thermal Design Power (TDP) for this processor is 105 W. Each computing node supports RAPL for power measurement. We disable hyperthreading to obtain stable performance.

Benchmarks. We use 12 MPI benchmarks from the NAS parallel benchmark suite and HPCC[18] suite. Table 3 lists the benchmarks used in our evaluation.

Software. Each computing node runs CentOS 7.4 with Linux kernel 3.10. We rely on the local Operating System (OS) to schedule processes and do not bind tasks to specific cores. All the benchmarks are compiled with gcc 4.8.5 with "-O3" as the compiler option and run with openmpi 4.0.0.

4.2 Evaluation methodology

Memory tracing. We use VTune[19] to collect the memory trace with a sample rate of one second by running the program on the master node of a cluster. We take the weighted average of the memory histogram produced by VTune as the memory traffic value, i.e., B_N

Table 3 Algorithm results for about ten compute nodes.

Benchmark	Type	N	ΔN^*	f^*	Performance boost (%)	Power saving (%)	Energy saving (%)
sp.D.169	Memory-bound	9	2	1.2	13.98	2.75	16.22
RandomAccess.160	Memory-bound	8	1	1.2	9.80	0.35	11.27
mg.D.128	Memory-bound	7	1	1.9	18.63	0.40	18.73
STREAM.160	Memory-bound	8	1	1.7	10.19	2.42	8.84
DGEMM.160	Memory-bound	8	2	1.5	18.82	2.12	19.85
cg.D.128	Memory-bound	7	2	1.2	–3.34	2.90	–0.29
bt.D.169	Memory-bound	9	1	1.8	–1.98	4.27	2.26
lu.D.128	Memory-bound	7	0	2.6	0	0	0
ft.D.128	CPU-bound	7	0	2.6	0	0	0
FFT.160	CPU-bound	8	0	2.6	0	0	0
PRANTS.160	CPU-bound	8	0	2.6	0	0	0
ep.D.128	CPU-bound	7	0	2.6	0	0	0

in Eq. (3). For example, Fig. 5 shows the memory traffic histogram of benchmark sp with $B_N = 51.76$ GB/s. Later we show that this strategy is effective in capturing the memory characteristics of a program.

Energy measurement. We use powergov[20] to measure processor power consumption and memory power consumption. Powergov uses RAPL to modelthe power consumption for processor and memory. We use mlc[21] to measure memory bandwidth and memory access delay under the various memory traffic conditions. We measure the CPU consumption under different frequency settings on a per-core level. Table 4 gives the measured per-core power consumption under different CPU frequencies. For memory power consumption, our experiments on real hardware show that the memory power consumption is constant as the CPU frequency changes. Power consumption of the rest system component, i.e., P^{other}, is around 25 W, which is obtained through RAPL.

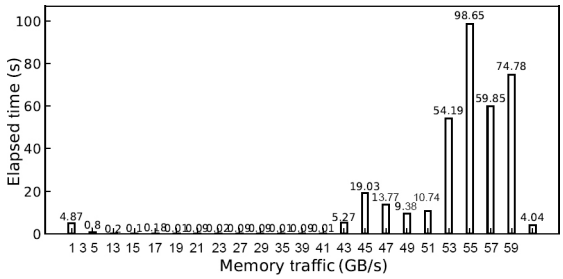

Fig. 5 Memory traffic histogram for benchmark sp.

Table 4 Single-core power and corresponding CPU frequencies.

CPU frequency (GHz)	$P^{\text{cpu}}(f_i)$	CPU frequency (GHz)	$P^{\text{cpu}}(f_i)$
2.6	7.8	2.5	7.7
2.4	7.5	2.3	7.3
2.2	7.1	2.1	7.0
2.0	6.8	1.9	6.7
1.8	6.4	1.7	6.2
1.6	6.1	1.5	6.0
1.4	5.9	1.3	5.8
1.2	5.7		

Performance report. We run each model on each input to collect performance profile data and power profile data until the 95% confidence bound per model per input is less than 5%. On average, we run each benchmark three times for each evaluation setting and remove obvious outliers. Then, we report the average performance across multiple runs.

5 Experimental Result

5.1 Overall result

Table 3 shows the performance improvement over the default resource allocation strategy when using our approach. It also gives the number of compute nodes and CPU frequency settings given by our analysis. Here, column "ΔN^*" shows the number of additional nodes for each benchmark.

Overall, our approach significantly improves the performance of memory-bound applications. It achieves on average 9.44% improvement for the seven memory-bound benchmarks, but using less 2.17% total power consumption. This translates to an average 10.98% reduction in energy consumption. Our approach improves the performance for most of the memory-bound applications (up to 18.82%) without incurring power consumption increase.

We also evaluated our approach using a larger dataset with a larger number of parallel processes using 30 computing nodes. The results are given in Table 5 where our approach delivers a performance improvement of up to 12.69% for the seven memory-bound benchmarks, with an average total savings of 7.06% for power and

Table 5 Algorithm results for about 30 compute nodes.

Benchmark	Type	N	ΔN^*	f^*	Performance boost (%)	Power saving (%)	Energy saving (%)
sp.E.529	Memory-bound	27	4	1.9	10.36	8.02	16.23
RandomAccess.512	Memory-bound	26	1	1.2	21.11	11.19	30.37
mg.E.512	Memory-bound	26	5	1.8	13.00	8.72	20.45
STREAM.512	Memory-bound	26	3	1.9	13.70	0.35	14.73
DGEMM.512	Memory-bound	26	4	2.2	10.24	–1.21	8.24
cg.E.512	Memory-bound	26	3	1.9	3.14	11.18	13.25
bt.E.529	Memory-bound	27	2	2.0	17.31	11.20	24.50
lu.E.529	Memory-bound	27	0	2.6	0	0	0
ft.E.512	CPU-bound	26	0	2.6	0	0	0
FFT.512	CPU-bound	26	0	2.6	0	0	0
PRANTS.512	CPU-bound	26	0	2.6	0	0	0
ep.E.529	CPU-bound	27	0	2.6	0	0	0

17.77% energy consumption.

5.2 Impact of using more nodes

Figure 6 shows using more nodes to run programs can reduce the execution time when the CPU frequency is fixed at 2.6 GHz. The performance improvement depends on the degree of memory contention. The higher the memory contention is, the better the performance improvement could be achieved by adding more nodes. However, using more nodes will increase the total power consumption.

Figure 7 shows the details of the increase of power consumption in percentage when all processor frequencies are 2.6 GHz.

5.3 Bandwidth comparison

We compare the sum of memory traffic on N compute nodes and that on $(N + \Delta N)$ compute nodes in Fig. 8.

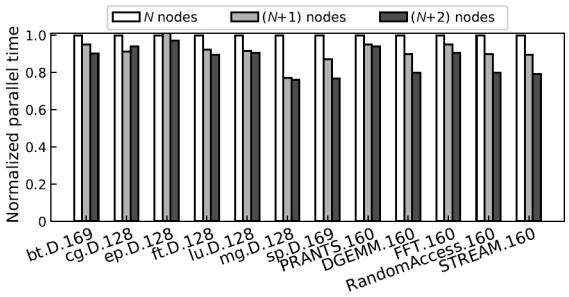

Fig. 6 Parallel time reduction due to add one or more nodes.

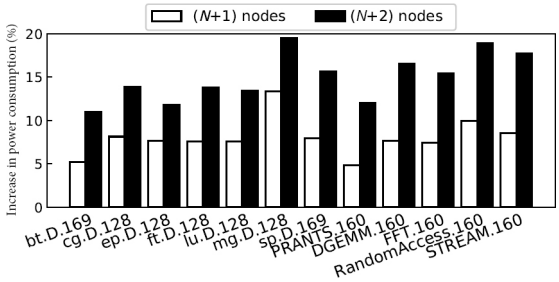

Fig. 7 Details of the increase of power consumption in percentage when all processor frequencies are 2.6 GHz.

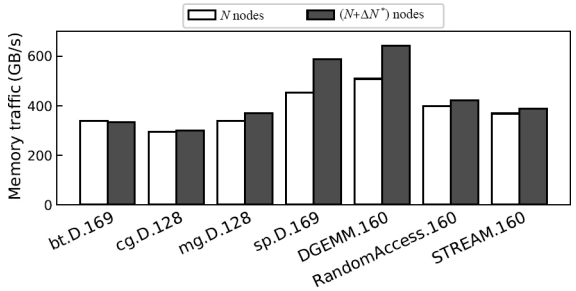

Fig. 8 Memory traffics of N nodes and $(N+\Delta N^*)$ nodes.

All the memory traffic values in Fig. 8 adopt the weighted memory bandwidth based on VTune memory traffic histogram (Fig. 5). For each group of columns, the left one represents the memory traffic sum on N nodes and the right one shows the memory traffic sum on $(N + \Delta N^*)$ nodes. ΔN^* is determined by model PPC. For memory-bound applications, memory bandwidth congestion on N nodes is usually more serious than that on $(N + \Delta N^*)$ nodes. So, memory traffic sum we measured on N nodes is usually lower than memory bandwidth the program actually demands. With the increase of nodes, memory bandwidth congestion starts to be relieved. Memory traffic sum we measured on $(N + \Delta N^*)$ nodes will be closer to memory bandwidth demands, larger than memory traffic sum we measured on N nodes.

5.4 Result using power limit

After calculating the optimal number of compute nodes ΔN^* and the optimal frequency value f^* with our approach, we run seven memory-bound applications with the fixed frequency f^* and get the experimental results as Fig. 9 and mark Case 0. Regarding the optimal frequency value f^* as the initial frequency of power capping, seven memory-bound applications are executed under RAPL power control (where the frequency is dynamically scaled) and the experimental results are recorded as Case 1. The frequency value under the

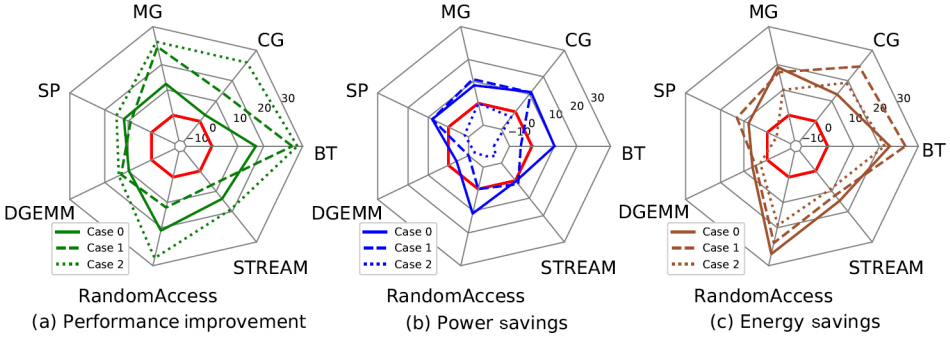

Fig. 9 Improvement of performance, and power and energy savings of NAS benchmarks. The further away from the centre, the better the improvement.

default resource allocation strategy $f_{\max}$ is regarded as the initial frequency of power capping and seven memory-bound programs are executed under RAPL power control. The experimental results are recorded as Case 2.

Figure 9 shows the performance, power, and energy consumption changes of seven applications under three scenarios (Case 0, Case 1, and Case 2) compared with the default resource allocation strategy. The further away from the center, the better performance improvement, power savings, and energy savings an application has. Combining these three graphs, we find that compared with Case 1, Case 2 improves the performance most, but it causes the worst power consumption, which is not in line with the original intention of this paper. The purpose of this paper is to improve the performance of the applications under power constraints. However, to improve the performance of the program at the cost of causing, a lot of power consumption is not what we advocate. Unlike Case 2, Case 1 strictly complies with the power constraints, and the percentage of power consumption reduction tends to zero, which maximizes the performance of the application within the power constraints.

Cases 0 and 1 are the recommended methods. Both of them use the resource allocation strategy proposed in this paper to find the optimal number of compute nodes and frequency. The only difference is that RAPL has been carried on to power capping in Case 1, which achieves higher performance improvement. It is worth emphasizing that both Cases 0 and 1 have applied our approach proposed in this paper, which shows that this algorithm is effective and necessary.

5.5 Using up all the cores of $N + \Delta N^*$ compute nodes

Because our approach adds ΔN^* nodes based on the default resource allocation strategy, the number of idle cores increases while the number of processes remains unchanged, which reduces the utilization of processors. Can we further utilize all the processor cores (by increasing the number of processes) to further improve program performance? Experimental results show that for benchmark sp, RandomAccess, DGEMM, and bt, using up all the cores of $N + \Delta N^*$ nodes leads to 7.77%, 2.98%, 9.65%, and 4.54% performance improvement respectively compared to our approach (Case 1). Power increases by 3.15%, –0.09%, 3.37%, and 4.15%, respectively. Energy reduces by 0.06%, 0.52%, 6.21%, and 0.36%, respectively. But for STREAM, the performance decreases by 21.30% and the energy consumption increases by 16.11%. This result shows that the number of used processor core has a great impact on the performance of memory-bound programs. However, our approach can achieve the performance improvement of memory-bound programs under power constraints.

5.6 Compared to the optimal performance

In this experiment, we enumerate all ΔN values from 1 to 4 and f^* values from 1.2 to 2.6 GHz to find the optimal ΔN^* and f^* for performance improvement and power control. This optimal result by traversal methods is denoted as optimal value in Fig. 10. Figures 10 and 11 compare the performance improvement by our approach and that by optimal setting using 10 and 30 nodes, respectively. Table 6 lists two groups of values for ΔN^* and f^* with our approach and optimal setting, respectively. We can find the performance improvement by our algorithm is competitive compared to that by optimal setting. The performance improvement difference between them for the whole 12 benchmarks is 1.3% on average in Fig. 10 and 1.5% in Fig. 11, respectively.

The power savings comparison of our approach and optimal setting by traversal methods is shown in Figs. 12 and 13. The power savings difference between them for

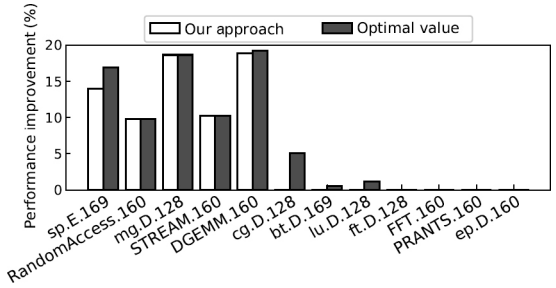

Fig. 10 Performance improvement comparison of our approach and optimal setting for 10 nodes.

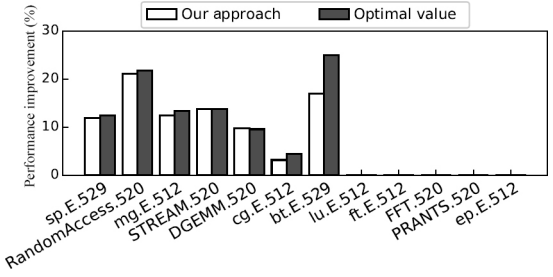

Fig. 11 Performance improvement comparison of our approach and optimal setting for 30 nodes.

Table 6 Extra nodes ΔN^* and CPU frequency f^* given by our approach and optimal settings.

Benchmark	Type	N	ΔN^* by ours	ΔN^* by optimal setting	Frequency by ours (GHz)	Frequency by optimal setting (GHz)
sp	Memory-bound	9	2	2	1.5	1.7
RandomAccess	Memory-bound	8	1	1	1.8	1.8
mg	Memory-bound	7	1	1	1.8	1.8
STREAM	Memory-bound	8	1	1	1.8	1.9
DGEMM	Memory-bound	8	2	2	1.2	1.4
cg	Memory-bound	7	2	1	1.8	2.1
bt	Memory-bound	9	1	1	2.3	2.4
lu	Memory-bound	7	0	1	2.6	2.3
ft	CPU-bound	7	0	0	2.6	2.6
FFT	CPU-bound	8	0	0	2.6	2.6
PRANTS	CPU-bound	8	0	0	2.6	2.6
ep	CPU-bound	7	0	0	2.6	2.6

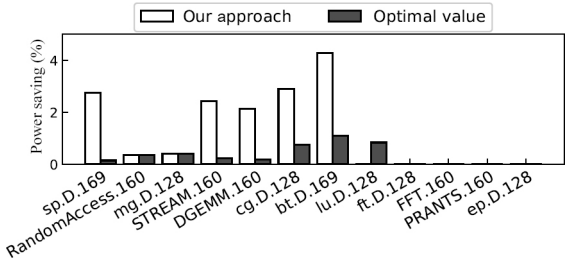

Fig. 12 Power savings given by our approach and optimal settings for using 10 computing nodes.

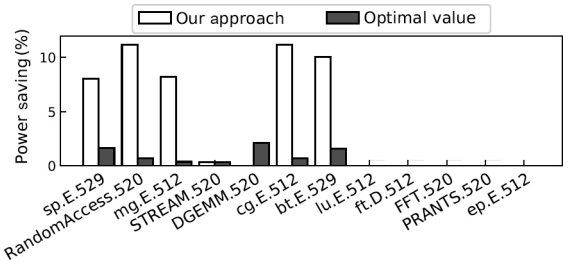

Fig. 13 Power savings given by our approach and optimal settings for using 30 computing nodes.

12 benchmarks for 10 nodes is 0.9% on average. By our approach, programs mg and DGEMM get more power savings compared to optimal settings. That is because the processor frequency by our approach is a little lower than that by the optimal setting.

6 Related Work

Energy consumption has become one of the most important concerns in computing systems and HPC in particular[13]. Several researchers have developed techniques and systems to save energy with a slight increase in execution time. Reference [22] used Duty Cycle Modulation technology to save energy in MPI applications and Ref. [23] used DVFS technology to save energy in OpenMP applications. Reference [24] found that combining DVFS and Duty Cycle Modulation can get more energy savings. These approaches saved processor energy consumption by scaling down processor clock frequency with a modest increase in execution time. The increase in execution time depends on the accuracy of processor idle time prediction. References [25, 26] used Near Threshold Computing (NTC) to save processor energy consumption. Except for saving processor energy consumption, Refs. [27, 28] focused on how to reduce memory energy consumption.

Besides low-power techniques, power-constrained problems for compute nodes are also focused. References [29, 30] reasonably allocated power to CPU and memory for performance improvement with the power limits. The main idea is that the power demands of the processor and memory are different for different applications. According to applications' characteristics, they allocated power to CPU and memory to satisfy their demand for performance and power. Furthermore, Refs. [31, 32] focused on a cluster. Firstly, when the power of a cluster is limited, they needed to set the number of active nodes according to an application's scalability. Secondly, they needed to allocate the power to compute nodes, and also allocate the power to processor and memory in one node ultimately. Finally, they improved the application's performance with power constraints.

Processor overclocking has been used to improve energy efficiency. For some applications, Ref. [30] found using turbo technology can achieve better performance under power constraints. It changes the clock speed of each socket (include turbo frequency), core use per socket, hyperthreading, the number of sockets in use, and the number of memory controllers in use to improve

performance. Reference [33] found F-overclocking technology can achieve greater energy efficiency than DVFS, low voltage technology, and baseline. Reference [34] also found Turbo Boost Technology would enhance the application's energy efficiency. This prior work is thus complementary to our work and can be used to control the CPU frequency when adding extra nodes.

7 Conclusion

This paper has presented a novel resource allocation scheme for HPC workloads, specifically targets memory-bound data-parallel applications. Our approach exploits a key observation that to improve performance, by reducing the number of parallel processes on a single host, one can reduce the memory contention. Unlike prior work that aims to maximize the system utilization, our approach judiciously allocates additional computing nodes to run a fewer number of parallel processes on a single node. Furthermore, to cap the total power consumption, our approach automatically determines the best CPU frequency to suit the CPU performance with the memory throughput. We propose a set of analytical models to estimate the profitability of using additional compute nodes based on profiling information. We evaluate our approach by applying it on a high-performance cluster to 12 MPI benchmarks. Experimental findings show that our approach improves the performance of seven memory-bound applications by 12.69% on average, but using 7.06% less overall power consumption, which translates into 17.77% energy savings when compared to the default resource allocation strategy.

Acknowledgment

This work was supported in part by the Advanced Research Project of China (No. 31511010203) and the Research Program of NUDT (No. ZK18-03-10).

References

[1] R. H. Dennard, F. H. Gaensslen, H. N. Yu, V. L. Rideout, E. Bassous, and A. R. LeBlanc, Design of ion-implanted MOSFET's with very small physical dimensions, *IEEE Journal of Solid-State Circuits*, vol. 9, no. 5, pp. 256–268, 1974.

[2] M. Bohr, A 30 year retrospective on Dennard's MOSFET scaling paper, *IEEE Solid-State Circuits Society Newsletter*, vol. 12, no. 1, pp. 11–13, 2007.

[3] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen, Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction, in *Proc. 36th Annu. IEEE/ACM Int. Symp. Microarchitecture* (*MICRO 36*), San Diego, CA, USA, 2003, pp. 81–92.

[4] R. Kumar, V. Zyuban, and D. M. Tullsen, Interconnections in multi-core architectures: Understanding mechanisms, overheads and scaling, *ACM SIGARCH Computer Architecture News*, vol. 33, no. 2, pp. 408–419, 2005.

[5] R. Kumar, D. M. Tullsen, N. P. Jouppi, and P. Ranganathan, Heterogeneous chip multiprocessors, *Computer*, vol. 38, no. 11, pp. 32–38, 2005.

[6] T. Heath, B. Diniz, E. V. Carrera, W. Meira, and R. Bianchini, Energy conservation in heterogeneous server clusters, in *Proc. 10th ACM SIGPLAN Symp. Principles and Practice of Parallel Programming*, Chicago, IL, USA, 2005, pp. 186–195.

[7] Y. M. Li, K. Skadron, D. Brooks, and Z. G. Hu, Performance, energy, and thermal considerations for SMT and CMP architectures, in *Proc. 11th Int. Symp. High-Performance Computer Architecture*, San Francisco, CA, USA, 2005, pp. 71–82.

[8] A. Lukefahr, S. Padmanabha, R. Das, F. M. Sleiman, R. Dreslinski, T. F. Wenisch, and S. Mahlke, Composite cores: Pushing heterogeneity into a core, in *2012 45th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Vancouver, Canada, 2012, pp. 317–328

[9] T. S. Muthukaruppan, M. Pricopi, V. Venkataramani, T. Mitra, and S. Vishin, Hierarchical power management for asymmetric multi-core in dark silicon era, in *2013 50th ACM/EDAC/IEEE Design Automation Conf.* (*DAC*), Austin, TX, USA, 2013, pp. 1–9.

[10] J. Meng, K. Kawakami, and A. K. Coskun, Optimizing energy efficiency of 3-D multicore systems with stacked DRAM under power and thermal constraints, in *Proc. 49th Annu. Design Automation Conf.*, San Francisco, CA, USA, 2012, pp. 648–655.

[11] T. Cao, S. M. Blackburn, T. J. Gao, and K. S. McKinley, The Yin and Yang of power and performance for asymmetric hardware and managed software, in *2012 39th Annu. Int. Symp. Computer Architecture* (*ISCA*), Portland, OR, USA, 2012, pp. 225–236.

[12] N. Gholkar, F. Mueller, and B. Rountree, Power tuning HPC jobs on power-constrained systems, in *Proc. 2016 Int. Conf. Parallel Architectures and Compilation*, Haifa, Israel, 2016, pp. 179–191.

[13] T. Patki, D. K. Lowenthal, A. Sasidharan, M. Maiterth, B. L. Rountree, M. Schulz, and B. R. de Supinski, Practical resource management in power-constrained, high performance computing, in *Proc. 24th Int. Symp. High-Performance Parallel and Distributed Computing*, Portland, OR, USA, 2015, pp. 121–132.

[14] C. Isci, A. Buyuktosunoglu, C. Y. Cher, P. Bose, and M. Martonosi, An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget, in *2006 39th Annu. IEEE/ACM Int. Symp. Microarchitecture* (*MICRO'06*), Orlando, FL, USA, 2006, pp. 347–358.

[15] S. Pagani, J. J. Chen, and M. M. Li, Energy efficiency on multi-core architectures with multiple voltage islands, *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 6, pp. 1608–1621, 2015.

[16] S. W. Williams, A. Waterman, and D. A. Patterson, Roofline: An insightful visual performance model for multicore architectures, *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[17] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick, The Landscape of Parallel Computing Research: A View from Berkeley, Electrical Engineering and Computer Sciences, Tech. Rep. UCB/EECS-2006-183, University of California at Berkeley, Berkeley, CA, USA, 2006.

[18] P. R. Luszczek, D. H. Bailey, J. J. Dongarra, J. Kepner, R. F. Lucas, R. Rabenseifner, and D. Takahashi, The HPC Challenge (HPCC) benchmark suite, in *Proc. 2006 ACM/IEEE Conf. Supercomputing* (*SC'06*), Tampa, FL, USA, 2006, p. 213.

[19] R. Jeffrey, Intel® VTune™ Amplifier, https://software.intel.com/en-us/articles/intel-system-studio-intel-vtune-amplifier-platform-profiler-overview, 2018.

[20] M. Dimitrov, Intel® Power Governor, https://software.intel.com/en-us/articles/intel-power-governor, 2012.

[21] V. Viswanathan, Intel® Memory Latency Checker v3.8, https://software.intel.com/en-us/articles/intelr-memory-latency-checker, 2013.

[22] B. Rountree, D. K. Lowenthal, B. R. de Supinski, M. Schulz, V. W. Freeh, and T. Bletsch, Adagio: Making DVS practical for complex HPC applications, in *Proc. 23rd Int. Conf. Supercomputing*, New York, NY, USA, 2009, pp. 460–469.

[23] W. Wang, A. Porterfield, J. Cavazos, and S. Bhalachandra, Using per-loop CPU clock modulation for energy efficiency in OpenMP applications, presented at the 2015 44th Int. Conf. Parallel Processing, Beijing, China, 2015, pp. 629–638.

[24] S. Bhalachandra, A. Porterfield, S. L. Olivier, and J. F. Prins, An adaptive core-specific runtime for energy efficiency, peesented at 2017 IEEE Int. Parallel and Distributed Processing Symp. (IPDPS), Orlando, FL, USA, 2017, pp. 947–956.

[25] I. Stamelakos, S. Xydis, G. Palermo, and C. Silvano, Variation-aware voltage island formation for power efficient near-threshold manycore architectures, presented at the 2014 19th Asia and South Pacific Design Automation Conf. (ASP-DAC), Singapore, 2014, pp. 304–310.

[26] U. R. Karpuzcu, A. Sinkar, N. S. Kim, and J. Torrellas, EnergySmart: Toward energy-efficient manycores for near-threshold computing, presented at 2013 IEEE 19th Int. Symp. High Performance Computer Architecture (HPCA), Shenzhen, China, 2013, pp. 542–553.

[27] R. Begum, D. Werner, M. Hempstead, G. Prasad, and G. Challen, Energy-performance trade-offs on energy-constrained devices with multi-component DVFS, presented at 2015 IEEE Int. Symp. Workload Characterization, Atlanta, GA, USA, 2015, pp. 34–43.

[28] Q. X. Liu, M. Moreto, J. Abella, F. J. Cazorla, and M. Valero, DReAM: An approach to estimate per-task DRAM energy in multicore systems, *ACM Transactions on Design Automation of Electronic Systems*, vol. 22, no. 1, p. 16, 2016.

[29] A. Tiwari, M. Schulz, and L. Carrington, Predicting optimal power allocation for CPU and DRAM domains, in *2015 IEEE Int. Parallel and Distributed Processing Symp. Workshop*, Hyderabad, India, 2015, pp. 951–959.

[30] H. Z. Zhang and H. Hoffmann, Maximizing performance under a power cap: A comparison of hardware, software, and hybrid techniques, in *Proc. 21st Int. Conf. Architectural Support for Programming Languages and Operating Systems* (*ASPLOS'16*), Atlanta, GA, USA, 2016, pp. 545–559.

[31] P. F. Zou, T. Allen, C. H. Davis, X. Z. Feng, and R. Ge, CLIP: Cluster-level intelligent power coordination for power-bounded systems, presented at the 2017 IEEE Int. Conf. Cluster Computing (CLUSTER), Honolulu, HI, USA, 2017, pp. 541–551.

[32] T. Patki, D. K. Lowenthal, B. Rountree, M. Schulz, and B. R. de Supinski, Exploring hardware overprovisioning in power-constrained, high performance computing, in *Proc. 27th Int. ACM Conf. Int. Conf. Supercomputing* (*ICS'13*), Eugene, OR, USA, 2013, pp. 173–182.

[33] D. Lo and C. Kozyrakis, Dynamic management of TurboMode in modern multi-core chips, presented at 2014 IEEE 20th Int. Symp. High Performance Computer Architecture (HPCA), Orlando, FL, USA, 2014, pp. 603–613.

[34] H. B. Jang, J. Lee, J. Kong, T. Suh, and S. W. Chung, Leveraging process variation for performance and energy: In the perspective of overclocking, *IEEE Transactions on Computers*, vol. 63, no. 5, pp. 1316–1322, 2014.

Juan Chen received the PhD degree from National University of Defense Technology, China in 2007. She is now an associate professor at National University of Defense Technology, China. Her research interests focus on supercomputer systems and energy-efficient software optimization method.

Xinxin Qi received the BS degree from Sun Yat-Sen University, China in 2019, and now is a master student at National University of Defense Technology. Her research interests include high performance computing and energy efficiency computing.

Feihao Wu received the MS degree from National University of Defense Technology in 2018. His research interests include the large scale parallel numerical simulation and energy efficiency computing.

Jianbin Fang is an assistant professor in computer science at NUDT. He obtained the PhD degree from Delft University of Technology in 2014. His research interests include parallel programming for many-cores, parallel compilers, performance modeling, and scalable algorithms.

Yong Dong received the PhD degree from National University of Defense Technology, China in 2012. He is now an associate professor at National University of Defense Technology, China. His main research interests include supercomputer systems and storage systems.

Yuan Yuan received the PhD degree from National University of Defense Technology, China in 2011. He is now an associate professor at National University of Defense Technology, China. His research interests include supercomputer systems and HPC monitoring and diagnosis.

Zheng Wang received the PhD degree in computer science from University of Edinburgh, UK in 2011. He is an associate professor at University of Leeds. His research interests include the boundaries of parallel program optimisation, systems security, and applied machine learning. He received four best paper awards for his work on machine learning based compiler optimisation (PACT'10, CGO'17, PACT'17, and CGO'19).

Keqin Li received the PhD degree in computer science from the University of Houston, USA in 1990. He is a SUNY distinguished professor of computer science at the State University of New York and a distinguished professor at Hunan University, China. His current research interests include cloud computing, fog computing and mobile edge computing, energy-efficient computing and communication, embedded systems and cyber-physical systems, heterogeneous computing systems, big data computing, high-performance computing, CPU-GPU hybrid and cooperative computing, computer architectures and systems, computer networking, machine learning, and intelligent and soft computing. He has published over 710 journal articles, book chapters, and refereed conference papers, and has received several best paper awards.