Ge R, Feng X, Zou P *et al.* The paradigm of power bounded high-performance computing. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 38(1): 87-102 Jan. 2023. DOI: 10.1007/s11390-023-2885-7

The Paradigm of Power Bounded High-Performance Computing

Rong Ge[1], Xizhou Feng[2], Pengfei Zou[3], and Tyler Allen[4]

[1] *School of Computing, Clemson University, Clemson, SC 29634, U.S.A.*

[2] *Meta Platform, Inc., Menlo Park, CA 94025, U.S.A.*

[3] *Amazon, Inc., Seattle, WA 98170, U.S.A.*

[4] *University of North Carolina at Charlotte, NC 27599, U.S.A.*

E-mail: rge@clemson.edu; fengx@meta.com; pzou@g.clemson.edu; t.allen@uncc.edu

Received October 4, 2022; accepted January 2, 2023.

Abstract Modern computer systems are increasingly bounded by the available or permissible power at multiple layers from individual components to data centers. To cope with this reality, it is necessary to understand how power bounds impact performance, especially for systems built from high-end nodes, each consisting of multiple power hungry components. Because placing an inappropriate power bound on a node or a component can lead to severe performance loss, coordinating power allocation among nodes and components is mandatory to achieve desired performance given a total power budget. In this article, we describe the paradigm of power bounded high-performance computing, which considers coordinated power bound assignment to be a key factor in computer system performance analysis and optimization. We apply this paradigm to the problem of power coordination across multiple layers for both CPU and GPU computing. Using several case studies, we demonstrate how the principles of balanced power coordination can be applied and adapted to the interplay of workloads, hardware technology, and the available total power for performance improvement.

Keywords power bounded computing, cross-component power coordination, hierarchical power allocation

1 Introduction

Modern computer systems are increasingly bounded by the available or permissible power at multiple levels ranging from components and machines to clusters and data centers①. For example, computer components such as GPU cards and CPUs must operate within their thermal design power, and exascale systems are imposed with a power budget of 20–30 megawatts[1]. Such power bounds are set due to physical, technical, and economical reasons. Nevertheless, system performance must continue to increase, which makes power a deciding factor for the deliverable performance and scalability of applications and systems.

Meanwhile, emerging workloads such as large neural networks have created an insatiable compute and power demand②. The larger the network, the more accurate the model, and the more the power demand. The GPT-3's deep learning neural network has over 175 billion machine learning parameters, 17x larger than the previous largest trained Turing NLG language model. Motivated to increase model accuracy, DNN training does not necessarily prioritize power and performance efficiency[2]. Such workloads typically have a low system utilization of 52% on average on highly optimized systems[2]. Indiscriminately running the components at full speeds not only wastes the limited available power but also suffers suboptimal performance. To avoid this situation, it is mandatory to allocate power to nodes and components based on applications' needs. Left unaddressed, energy consumption will exceed supply③. In short, power bound-

Perspective

Special Issue in Honor of Professor Kai Hwang's 80th Birthday

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

①Arrhenius equation for reliability. http://www.jedec.org/standards-documents/dictionary/terms/arrhenius-equation-reliability, Jan. 2023.

②Bailey B. AI power consumption exploding. https://semiengineering.com/ai-power-consumption-exploding/, Dec. 2022.

③SRC. Decadal plan for semiconductors. https://www.src.org/about/decadal-plan/decadal-plan-full-report.pdf, Jan. 2023.

ed computing is not only urgent but also a reality.

Sustaining performance growth given a power bound has motivated research in computer architecture, system management, and their work in tandem. The paradigm of power bounded computing applies to system management. Specifically, it considers power as a scarce resource and aims to maximally use the available power budget to increase application performance and system throughput. The system over-provisions hardware components but dynamically manages them to operate within the given power bound, thus ensuring power is maximally translated to performance. This paradigm relies on the availability of power-aware components that support a set of power-performance states and can transition from one state to another as instructed by a user or a program.

We apply the paradigm of power bounded computing for both homogeneous and heterogeneous power constrained computing. The paradigm can be applied hierarchically, supporting multiple levels including clusters, nodes, accelerator devices, and components, as well as job co-scheduling. We describe some core technologies we have developed under this paradigm: node level cross-component power coordination[3–5], CLIP (cluster level intelligent power coordination)[6], and job co-running and resource sharing[7, 8]. These technologies tailor power allocation to applications and available power budgets, and significantly improve performance in comparison to the default settings.

2 The Power Bounded Computing Problem

Power bounded computing is built upon the premise that every compute node in the system can and will operate under a given power budget. By bounding per-node power consumption, a large-scale system can reconfigure itself based on its current workload to achieve better performance under the same power budget. For simplicity, we focus on processors and DRAM modules on hosts and discrete GPU accelerators, i.e., CPUs and memory, and GPU Streaming Multiprocessing cores (SMs) and global memory. A reason for this simplification is that the power consumption of other components is typically smaller and has smaller variations.

Given a system M and a system power budget P_b, the objective of power bounded computing (denoted PBC) is to maximize application performance and system throughput under total system power budget P_b with available hardware in machine M. Because the system power is allocated hierarchically, we formulate the PBC problem at both the system level and the nodal level.

The system level power bounded computing problem is formulated as follows: given a job queue Q with a set of parallel workloads $\{W_1, W_2, ..., W_n\}$, a machine M and a total power bound $P_{\rm b}$, find an optimal power, hardware and job allocation such that:

1) $STP(a^*, Q, M) = \max_{\alpha\in\mathcal{A}} STP(\alpha, Q, M)$, and
2) $\sum_{m\in M} P^*_m \leqslant P_{\rm b}$.

Here, STP is system throughput, α is hardware and resource allocation tuple (J, Hw, P) (i.e., how workloads (J) are assigned to nodes (Hw) with corresponding power budgets (P)), and $\mathcal{A}$ is the space that comprises all possible values of α. The system can distribute power across multiple levels, from a single component to an entire node. We define a component as power boundable if the component can and will always operate under the power cap allocated to it.

The node-level power bounded computing problem is formulated as follows: given a parallel workload W, a node nd comprising a set of K power boundable components $C_1, C_2, \ldots, C_K$, and a total power bound P_{nd}, find the upper bound of the achievable performance $\texttt{perf}_{\max}$ and the corresponding power allocation tuple $\alpha^* = (P^*_1, P^*_2, \ldots, P^*_K)$ such that:

1) $\texttt{perf}_{\max} = \max_{\alpha\in\mathcal{A}} \texttt{perf}(\alpha, W, nd)$,
2) $\alpha^* = \arg\max_{\alpha\in\mathcal{A}} \texttt{perf}(\alpha, W, nd)$, and
3) $\sum_{i=1}^{K} P^*_i \leqslant P_{nd}$.

Here, α is a power allocation tuple and $\mathcal{A}$ is the space that comprises all possible values of α within the node. The performance metric, `perf`, can have different definitions depending on both the application and the user's demand. Example metrics include compute rate, performance-to-power ratio, and system throughput.

3 Nodal Level Cross-Component Coordination

The first principle of power bounded computing is balancing power budget allocation among all components subject to an overall power cap with a system-wide optimization objective instead of an individual optimization goal. To understand how this principle will be applied to real systems, we look at a high-level picture of power consumption on today's high-end systems and their building blocks.

First, these systems and building blocks normally have a significantly larger power envelope than what

they actually consume at run-time. For example, a compute node of the Frontier supercomputer integrates 8 CCDs, 4 high-end GPUs with 2 GCDs, 512 GB DDR4 memory, and 512 GB HBM2e memory. Its total power envelope exceeds 3 000 W. If we allocate power based on these envelopes, inefficient use will lead to significant waste.

Second, these systems have multiple major power consumers like CPUs, GPUs, and memory whose power budgets must be simultaneously managed to maximize performance given a limited system power. Memory may consume more power than processors on big-memory systsems[④], and high-end GPUs consume comparable or more power than CPUs. Traditional methods that focus on a single dominant component would lead to suboptimal performance and power waste.

Third, hardware resources are generally overprovisioned but most applications will not fully utilize all of them all the time. For example, CPU algorithms do not use GPUs while GPU algorithms typically only use CPUs for kernel offloading and data management. Activating components and running them at full speed without disparity not only wastes the limited available power but also suffers suboptimal performance.

Finally, to maintain a high performance given a power budget, a balanced power allocation between components must be robust; otherwise a small power shift away from a balanced configuration may lead to significant performance loss.

To develop effective and robust technology for power bounded computing, we start from the node level and study how to achieve robust balanced power allocation between processing units and memory modules, i.e., CPUs and main memory, and GPUs and GPU global memory. Our main focuses include understanding the dynamics between processor-memory power allocation and application performance, identifying the patterns of power allocation impacts, and developing optimal power coordination strategies. Our research[3, 4] leads to the following findings.

1) Cross-component power coordination can improve performance significantly, e.g., 35% for GPU computing and more for CPU computing. These results highlight the impacts and urgency of applying power bounded computing principles to HPC (high performance computing) systems.

2) There exist distinct patterns of performance and power dynamics which can be categorized over both CPU and GPU computing. These patterns guide designing optimal cross-component power allocation algorithms on modern HPC.

3) Different workloads considerably vary in characteristics while sharing common patterns, suggesting the need to integrate application awareness into power scheduling and management.

4) Heuristic algorithms can quickly pinpoint near-optimal cross-component power allocations with lightweight application profiling.

3.1 Categorizing Power Allocation Effects

To understand the implication of bounding component power budget, we consider coordinated power allocation between processing units and memory, where each includes all the processors/cores and memory devices respectively. We denote the total power constraint P_{nd}, processing power P_{cpu}, memory power P_{mem}, and the power allocation $\alpha = (P_{\text{cpu}}, P_{\text{mem}})$. We note $P_{nd} \geqslant P_{\text{cpu+mem}}$ holds for all power allocations.

Given a power budget, application performance differs significantly by changing the power distribution between processors and memory. As shown in Fig.1, there exist six categories of power allocation scenarios with regard to their performance effects.

Category 1: *Adequate Power Allocation for Both CPUs and Memory.* Both CPUs and memory receive power allocation exceeding their maximum power demands, and an application can achieve its maximum performance. The actual power consumption of each component stays constant, and the sum may be less than P_{b}.

Category 2: *Adequate Memory Power, Lightly Bounded CPU Power.* The power allocated to CPUs is lightly bounded but adequate to operate all processor cores at a performance state. The actual CPU power closely matches the allocated power budget. Meanwhile, the actual DRAM power stays near the maximum value even though DRAM receives a higher power allocation. As CPU power budget decreases, application performance decreases monotonously and gradually.

Category 3: *Adequate CPU Power, Bounded Memory Power.* CPUs receive more power budget than

[④]HPE has constructed the largest singlememory computer system ever built. https://www.forbes.com/sites/aarontilley/2017/05/16/hpe-160-terabytes-memory/?sh=265e05a3383f, Jan. 2023.

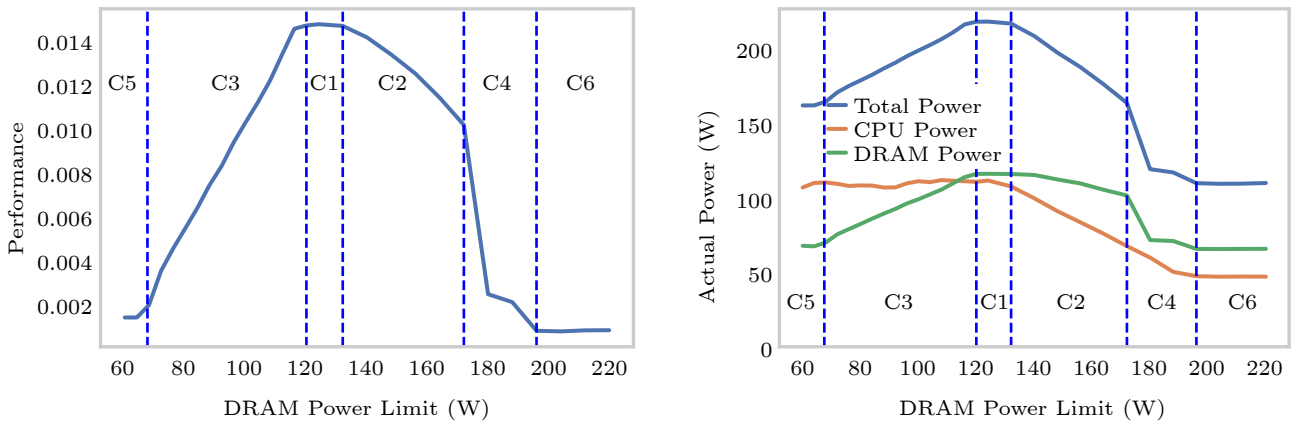


Fig.1. Categories of power allocation effects[3, 4]. (a) Performance of hpcc.StarRandomAccess vs DRAM power on IvyBridge. (b) Power of hpcc.StarRandomAccess vs DRAM power on IvyBridge. The plots of (a) application performance and (b) actual power consumption for different power allocations between processors and memory modules visually reveal six categories of power allocation scenarios for $P_{nd} = 240$ W. Ci means category i.

needed. Their actual power stays relatively constant and is slightly smaller than the maximum demand, regardless if more budget is allocated. Meanwhile, DRAM receives inadequate allocations, and its actual consumption is close to the budget. In scenario 3, application performance is bounded by memory performance; increasing power allocation to memory dramatically improves application performance.

Category 4: *Adequate Memory Power, Significantly Inadequate CPU Power.* When CPU power is significantly under-budgeted while DRAM is over-budgeted, the application performance drops sharply from those in categories 2 and 3. In category 4, memory consumes much less power than its allocation, mainly due to the fact that CPUs make less frequent memory request.

Category 5: *Adequate CPU Power, Minimum Memory Power.* The actual CPU power is close to the maximum CPU power required by a given workload (108 W in the test case used in Fig.1).

Category 6: *Adequate Memory Power, Minimum CPU Power.* CPUs receive a minimal or close to minimal power allocation. In this scenario, hardware overrides the software power allocation and CPUs consume a constant power, i.e., 48 W. Meanwhile, memory receives an excessive power budget. This scenario cannot ensure the system power bound and often delivers the worst performance.

3.1.1 *Power Allocation Categories on GPU Computing*

We have observed similar power allocation patterns for allocating power between GPU devices and memory for GPU computing. However, because GPU supports a smaller range of power management and uses different power capping mechanisms, the dynamics of GPU cross-component power allocation and categories have some unique features[3] as shown in Fig.2.

First, fewer categories appear in the application profiles, e.g., categories 1, 3 and 2 on Titan XP and category 3 on Titan V. GPU hardware prohibits categories (4 & 5 & 6) which would deliver an unacceptable low performance, by disallowing low power caps on SMs and memory. Further, the largest performance difference among all power allocations is about 30%.

Second, unlike independent management of processors and DRAM on the host, where the unused power budget on one component is simply wasted, the GPU power capping automatically reclaims and shifts it to another component, e.g., from DRAM to SMs. As a result, the intersections of categories are different from those for CPU computing, and the actual total power consumption always matches the set power cap, unless the cap exceeds applications' demands.

Third, with the new SM and HBM2 technologies, Titan V has a smaller total and DRAM power range than Titan XP.

3.2 Category Based Power Allocation Strategies

After analyzing the performance impacts of different power allocations, we found that given a power budget, the optimal cross-component power alloca-

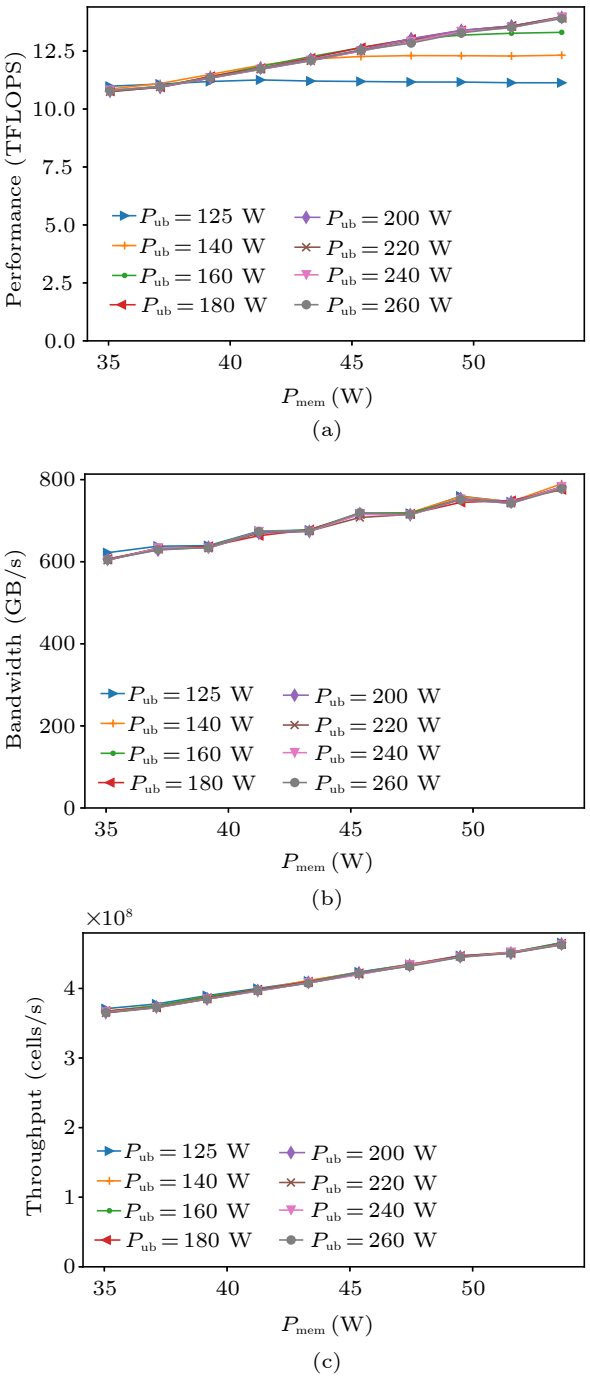


Fig.2. Performance trends as memory power allocation increases under various total power caps on Titan V[3]. The memory power is estimated using memory frequency setting and empirical power models built from experiment data on the card. (a) SGEMM on Titan V. (b) STREAM on Titan V. (c) Cloverleaf on Titan V.

tion measured by delivered performance provides a balanced interaction between compute and memory access, while other allocations bound at least one of them.

To study the effect of power bound, we define the maximum capacity $R_{\max}$ for a component K and the allocated capacity R_P when the component operates given a power budget P_K. A component reaches its maximum capacity when it receives adequate power while all other components are not subject to any power constraint. In other words, whether a component is bounded by power can be measured by such component's utilization rate—the ratio of the component's actual delivered rate R to its maximum capacity $R_{\max}$.

With the optimal power allocation, the utilization rate is high, close to 100% for both compute and memory access. In contrast, when processors are under powered, processor capacity utilization is high but memory capacity utilization is low, indicating that application execution is bounded by computing. Similarly, when memory is under powered, the application is bounded by memory access. Fig.3 illustrates this concept by examining the performance of DGEMM and STREAM benchmarks under different utilization rates for CPUs and memory on an Ivy-Bridge system.

Different applications have different demands for compute and memory access, and thus different compute intensities—the ratio of the computation rate to the memory bandwidth on the same system. As a result, their optimal power allocations differ. DGEMM is compute intensive and has a high power demand for CPUs. In contrast, STREAM is memory intensive, requiring more power allocation for memory access.

3.2.1 *Finding the Optimal Power Allocation*

The optimal cross-component power allocation is specific to the given power budget. It is located at category 1 given sufficient power, and usually at the intersection of two neighboring scenarios given smaller power budgets. As the power budget decreases, the optimal allocation is at the intersection of categories 2 and 3, and further moves to the intersection of categories 3 and 4. Table 1 summarizes the location of the optimal allocation for varying power budgets.

From the optimal cross-component power allocation, a shift in either direction causes performance degradation. However, shifting in one direction degrades performance more. We mark the critical component as the one that, if under powered, drastically degrades the application performance. The existence of a critical component suggests that a power allocation strategy ensures the power budget for the criti-

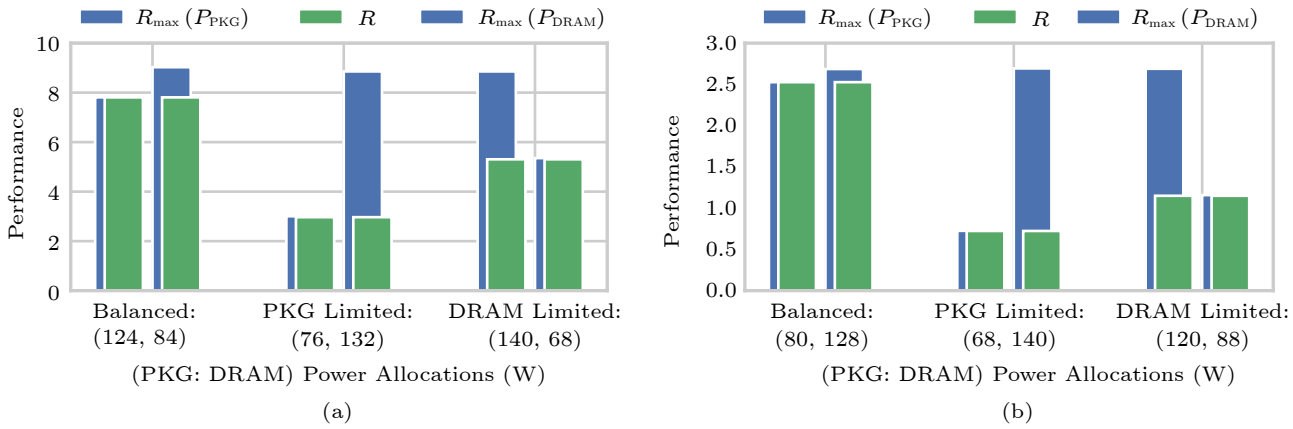


Fig.3. Balanced vs unbalanced allocations. Balanced compute and memory access for a given total power budget of 208 W[3]. (a) DGEMM on IvyBridge. (b) STREAM on IvyBridge.

Table 1. Categorical Inter-Domain Interaction Allocation and Critical Component vs Power Budget[3]

P_{ub}	Valid Allocation Scenario Category	Optimal Allocation	
		Intersection	Critical Component
Large	1, 2, 3, 4, 5, 6	1	None
↓	2, 3, 4, 5, 6	2/3	DRAM
↓	3, 4, 5, 6	3/4	CPU
↓	4, 5, 6	4/6	DRAM
Small	5, 6	5/6	CPU

cal component and approaches the optimal allocation from the scenario (underlined in Table 1) that better preserves the performance. We would like to reiterate that very small power budgets should not be allocated for running new jobs, due to unacceptable low power efficiency and performance.

3.2.2 *Category-Based Heuristic Power Coordination*

The power allocation categories lead to the design of heuristic power allocation methods for the problem of cross-component coordination. Such methods eliminate the need for exhaustive or fine-grain profiling to search the optimal power allocation for any given power budget. The existence of critical power levels provides two important heuristics. First, the power budget given to a computer system must be greater than a threshold to fall into category 1, 2, or 3 and operate in a productive manner. Second, given a power budget that is above this threshold, the critical power values dictate the set of valid power allocation scenarios and corresponding optimal cross-component allocations.

Using these heuristics, we have developed a category-based power coordination method called COORD[3] shown in Algorithm 1. In COORD, we assume dedicated execution environments where only one job runs on the system simultaneously, which is true on traditional high-performance computing systems. We also consider fixed total power budgets and distributions across components prior to job execution. Essentially, COORD breaks the set of possible power budgets into four subsets: 1) adequate budgets for both components to operate at their highest performance states; 2) adequate budgets only for one component to operate at its highest performance state (in this case we prioritize memory power allocation as it has a greater impact on performance); 3) neither component has the adequate budget to run at its highest performance state (in this case we proportionally allocate power between processors and memory); and 4) both components must be throttled down to satisfy the power limit (the algorithm rejects to allocate power to run the job due to the expected poor performance).

Empirically, COORD ensures 1) the system meets the power limits; and 2) the power allocation achieves the best or close-to-best application performance given a power budget. The propositions are confirmed by our experimental results.

Algorithm Adjustments for GPU Computing. Because GPU computing has a smaller power allocation space and the hardware automatically excludes unacceptable low power budgets, COORD can be simplified to use fewer parameters.

3.2.3 *Evaluation*

We have evaluated the COORD algorithms on multiple platforms using various applications[3] and the results are summarized in Fig.4.

Algorithm 1. Category-Based Heuristic Power Coordination[3]

procedure COORD(P_{b})
  $status \leftarrow Success$
  if $P_{\mathrm{b}} \geqslant P_{\mathrm{cpu},\,L_1} + P_{\mathrm{mem},\,L_1}$ **then** ▷ adequate power for both
    $P_{\mathtt{cpu}} \leftarrow P_{\mathrm{cpu},\,L_1}$
    $P_{\mathtt{mem}} \leftarrow P_{\mathrm{mem},\,L_1}$
    $status \leftarrow Hint\colon\ power\ surplus$
  else if $P_{\mathrm{b}} \geqslant P_{\mathrm{cpu},\,L_2} + P_{\mathrm{mem},\,L_1}$ **then** ▷ adequate power for one
    $P_{\mathtt{mem}} \leftarrow P_{\mathrm{mem},\,L_1}$
    $P_{\mathtt{cpu}} \leftarrow (P_{\mathrm{ub}} - P_{\mathtt{mem}})$
  else if $P_{\mathrm{b}} \geqslant P_{\mathrm{cpu},\,L_2} + P_{\mathrm{mem},\,L_2}$ **then** ▷ inadequate power
    $Pd_{\mathtt{cpu}} \leftarrow P_{\mathrm{cpu},\,L_1} - P_{\mathrm{cpu},\,L_2}$
    $Pd_{\mathtt{mem}} \leftarrow P_{\mathrm{mem},\,L_1} - P_{\mathrm{mem},L_2}$
    $percent_{\mathtt{cpu}} \leftarrow 1.0 \times Pd_{\mathtt{CPU}}/(Pd_{\mathtt{CPU}} + Pd_{\mathtt{mem}})$
    $P_{prop} \leftarrow P_{\mathrm{b}} - (P_{\mathrm{cpu},\,L_2} + P_{\mathrm{mem},\,L_2})$
    $P_{\mathtt{cpu}} \leftarrow P_{\mathrm{cpu},\,L_2} + percent_{\mathtt{cpu}} \times P_{prop}$
    $P_{\mathtt{mem}} \leftarrow (P_{\mathrm{b}} - P_{\mathtt{cpu}})$
  else
    $status \leftarrow Warning\colon\ budget\ too\ small!$
  end if ▷ power budget too small
  return $(P_{\mathtt{cpu}}, P_{\mathtt{mem}}, status)$
end procedure

Fig.4 shows that the power allocation found by COORD differs from the best power allocation by less than 5% for large power caps (preferred), less than 9.6% on average for all power caps for all CPU benchmarks, and less than 2% for GPU benchmarks. Given a power budget greater than the applications' maximal power demand, COORD delivers the same or similar performance as the best allocation for most cases.

In addition, COORD only allocates to components adequate powers that are lower than those set in sweeping experiments. One noteworthy observation is that COORD outperforms the default NVIDIA GPU power capping method by up to 33% for the applications under study. Such gain comes from the fact that COORD is aware of applications and available power budgets, while the default uses the same strategy to distribute power between GPU SMs and global memory.

4 CLIP: Cluster Level Intelligent Power Coordination

Building upon node-level power coordination, we apply the paradigm of power bounded computing to computer clusters. In a cluster, optimally managing power for HPC workloads requires an intelligent strategy to control the number of participating nodes in addition to allocating the available power budget to different subsystems (CPU-core, CPU-uncore and memory) within nodes. With a number of nodes as an additional dimension, cluster-level power bounded computing offers more space to increase system performance but also brings new challenges. Inappropriate node assignment can either cause inefficient utilization of the available power or lead to subsystems running at ineffective power levels, thereby delivering an inferior performance. Managing power at the cluster level requires striking a balance between clusters, nodes, and components.

We have developed a Cluster Level Intelligent Power (CLIP) coordination framework to address the challenges in cluster-level power bounded computing. CLIP employs application-aware power bounded scheduling for parallel applications on clusters built of NUMA multicore nodes. It characterizes the scalability of parallel applications and their power demands and accordingly recommends the optimal application execution configuration and power distribution. The framework implementation is hierarchical and consists of two levels: the cluster level determines the number of nodes and the power budget for each node; the node level selectively activates the CPU cores and distributes the available power budget to the CPU

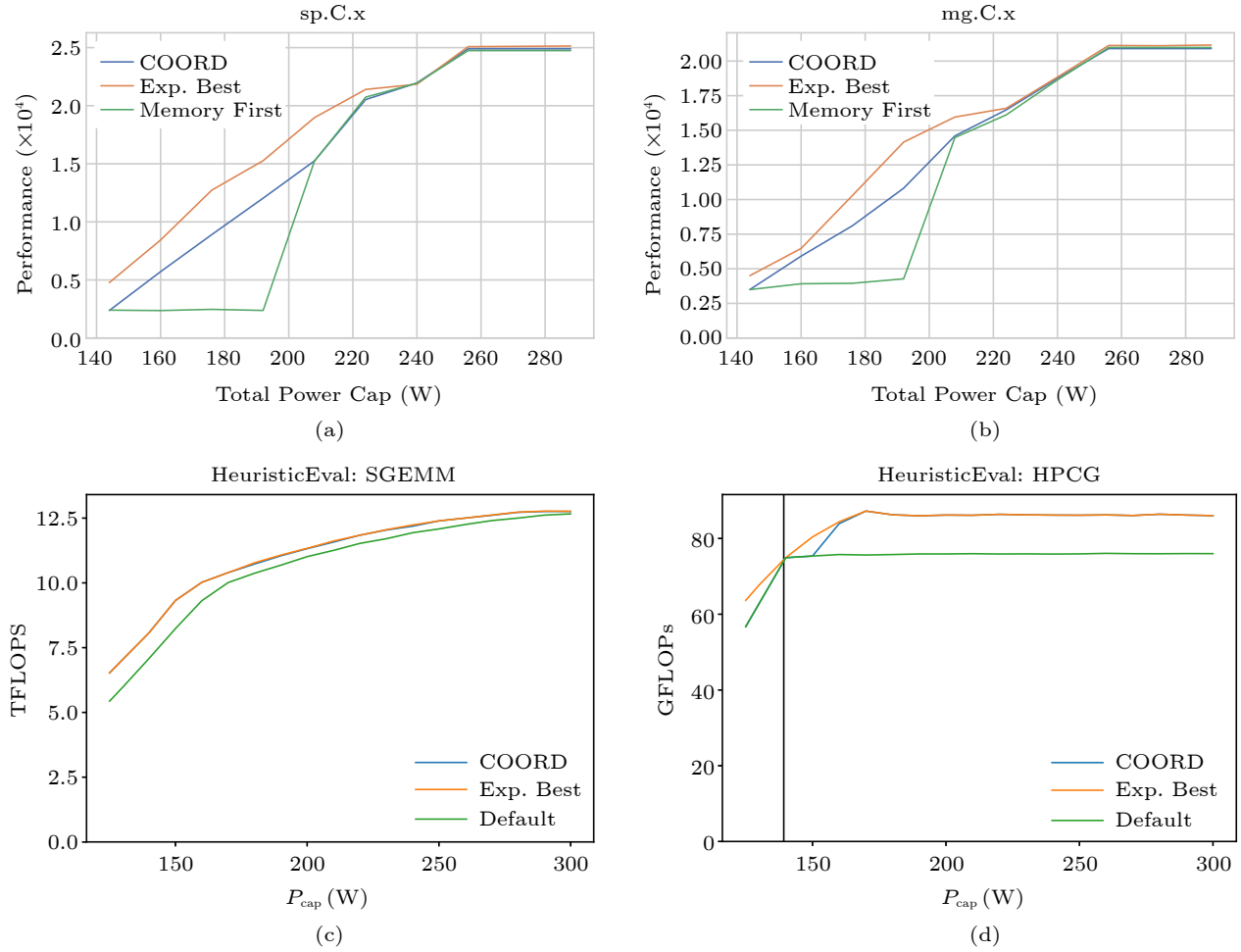


Fig.4. Comparison between COORD and the best identified from experiments on the IvyBridge system and the Titan XP GPU system[3]. The vertical lines in the GPU figures show the value of P_{totref} (the total power cap). (a) NPB SP on IvyBridge. (b) NPB MG on IvyBridge. (c) SGEMM on Titan XP. (d) HPCG on Titan XP. P_{cap}: the experimentally best result.

and memory within nodes. The CLIP framework uses lightweight off-line profiling for application characterization, classifies workloads into three categories based on their scalability, and then applies corresponding power allocations.

4.1 Application-Aware Configuration Selection

There are three types of scalability trends on parallel applications, which we denote as linear, logarithmic, and parabolic. The performance of linear applications increases linearly with concurrency and processor frequency. The performance of logarithmic applications increases linearly until an inflection point, after which the performance growth drops. The performance of parabolic applications increases linearly when concurrency is less than the global maximum. Beyond the global maximum, increasing concurrency causes performance degradation. Both logarithmic and parabolic can be approximated by a piecewise model.

Fig.5 shows how a power budget would impact the three types of applications differently[6]. For a linear application like EP in Fig.5(a), the performance is best at the highest concurrency unless power is lower than the lower bound of the acceptable power. For logarithmic applications, the number of cores activated to achieve the best performance decreases with the power budget, as shown in Fig.5(b). For parabolic applications, the insufficient power budget exacerbates the performance loss of all-core configuration as seen in Fig.5(c). The performance gap between the optimal concurrency and the maximum concurrency also increases when the power budget decreases.

To classify the application scalability trend, we simply compare the performance under two profiling stages: $Perf_{\text{all}}$ and $Perf_{\text{half}}$. $Perf_{\text{all}}$ and $Perf_{\text{half}}$ de-

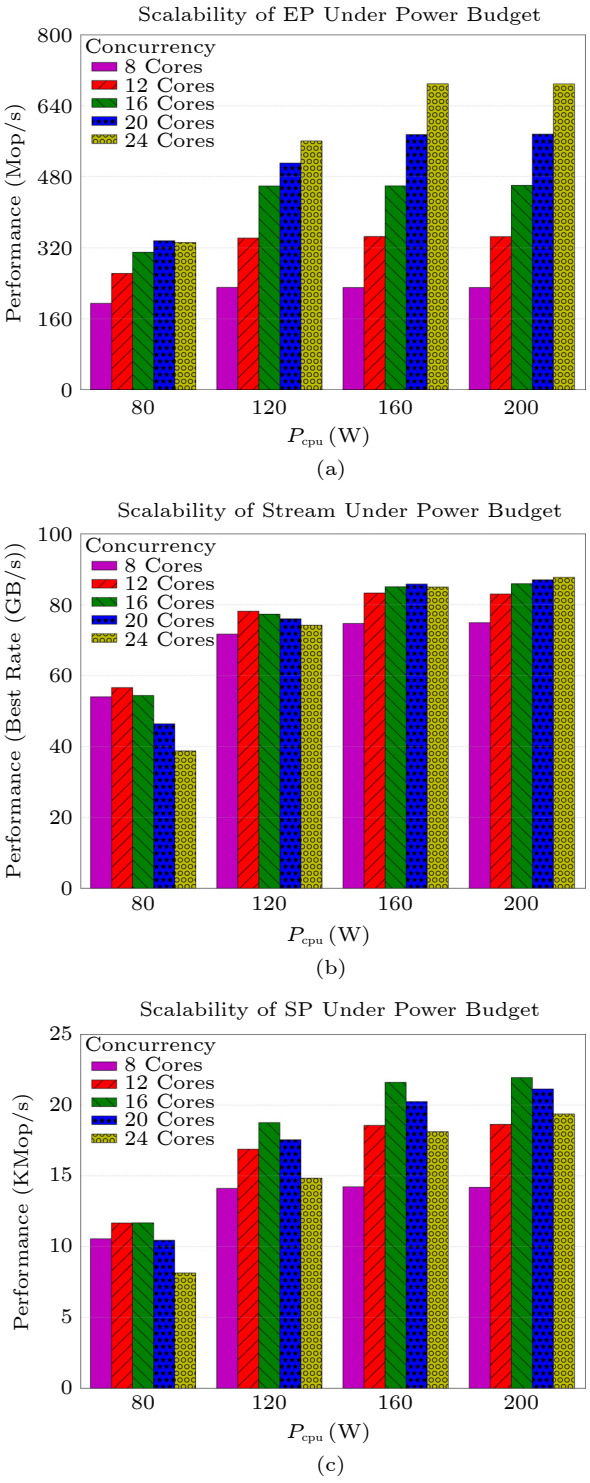


Fig.5. Performance impact of processor power budget for (a) linear, (b) logarithmic, and (c) parabolic applications[6].

note the performance with all and half of the available cores respectively. The applications with $\frac{Perf_{\text{half}}}{Perf_{\text{all}}} < 0.7$ are classified as the linear type; the applications with $0.7 \leqslant \frac{Perf_{\text{half}}}{Perf_{\text{all}}} < 1$ are classified as the logarithmic type; and applications with $\frac{Perf_{\text{half}}}{Perf_{\text{all}}} \geqslant 1$ are classified as the parabolic type. We choose these ratios in our study, but users can adjust them based on their objectives and constraints.

4.2 Application-Oriented Cluster Level Power Allocation

Fig.6 outlines the CLIP framework[6], which includes a profiling module, a data-driven execution configuration recommendation module, an application execution module, and several helper tools to provide a user-friendly power-bounded computing environment.

1) *Profiling Module*. It runs several iterations of the application's kernel function with sufficient power. The system collects performance events and execution time information for future affinity determination and scalability trend classification.

2) *Configuration Recommendation Module*. It takes the profiling data and power budget as inputs and returns a parallel workload execution configuration.

3) *Application Execution Module*. It first checks whether the database contains the profiling data of the workload. If the response is negative, this module requests the profiling module to profile the workloads first and inputs the profiles data to the recommendation module to get the suggested configuration. The application execution module submits the jobs using the suggested configuration to the power-bounded multicore cluster.

4) *System Interface and Helper Tools*. It includes several customized system tools such as a power meter reader, a performance state controller, a power capping controller, and a performance event collector.

As outlined in Algorithm 2, the CLIP power-bounded scheduling algorithm operates in two steps.

Step 1. Searching for the given job in the knowledge database to decide if it is necessary to start smart profiling. Through smart profiling or searching from the knowledge database, CLIP is able to acquire the optimal power range $[P_{\mathbf{cpu}_{L_o}} + P_{\mathbf{mem}_{L_o}}, P_{\mathbf{cpu}_{H_o}} + P_{\mathbf{mem}_{H_o}}]$ for each node. After that, the system inputs the profile data and the given power budget recommendation to decide the number of nodes and the power budget for each node.

Step 2. Inputting the power budget for each node and the profile data for each application to the recommendation module and getting the suggested power budget for the CPU and memory, the number of activated cores, and the optimal core affinity.

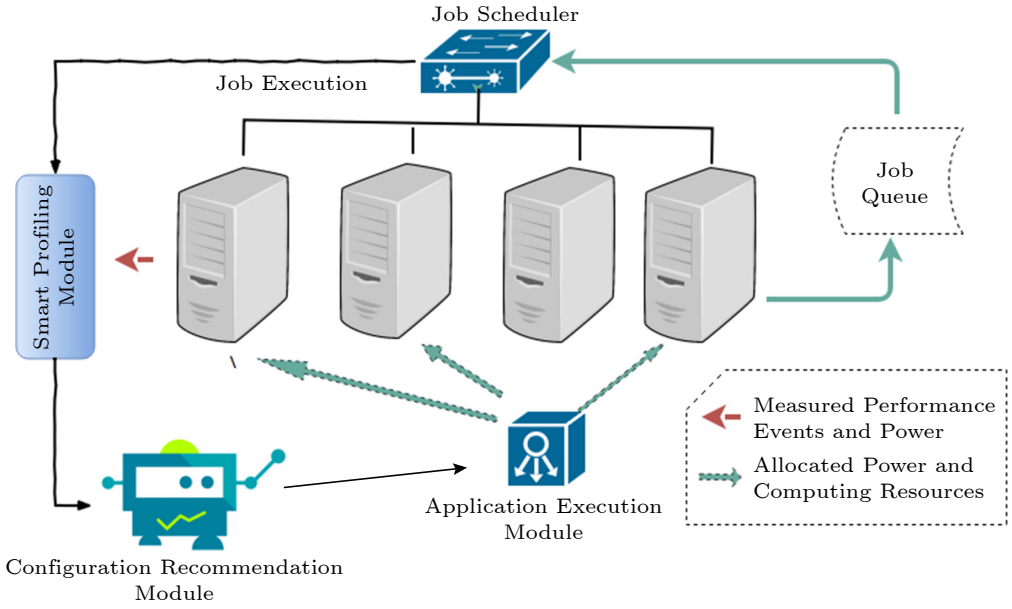


Fig.6. Overview of CLIP[6].

Algorithm 2. CLIP (Cluster Level Intelligent Power Coordination System)[6]

function CLIP(App, C)

Input: P_{ub}: the total power budget for the cluster;
App: the application under study;
C: the cluster with N_{total} nodes;
N_{total}: the total number of nodes in the cluster C

Output: N_{nodes}: suggested number of active compute nodes
$P_{\text{cpu}_{run_i}}$: suggested CPU power for node i;
$P_{\text{mem}_{run_i}}$: suggested memory power for node i;
N_{cores}: suggested number of active cores on each node;
Map: suggested mapping affinity

$[P_{\text{cpu}_{H_o}}, P_{\text{cpu}_{L_o}}, P_{\text{mem}_{H_o}}, P_{\text{mem}_{L_o}}, \text{Profile}] \leftarrow SmartProf(App)$
$[N_{\text{cores}}, Map] \leftarrow$ Recommendation (Profile)
if App has a set of a predefined number of processes $N_{def_1}, ..., N_{def_n}$ **then**
if $N_{def_k} \leqslant P_{\text{ub}}/(P_{\text{cpu}_{L_o}} + P_{\text{mem}_{L_o}}) < N_{def_{k+1}}$ **then**
$N_{\text{nodes}} \leftarrow N_{def_k}$
$P_{\text{node}} \leftarrow P_{\text{ub}}/N_{\text{nodes}}$
for every node i to be activated **do**
$[P_{\text{cpu}_{run_i}}, P_{\text{mem}_{run_i}}] \leftarrow P_{\text{node}}$
end for
end if
else
if $P_{\text{ub}} > N_{\text{total}} \times (P_{\text{cpu}_{H_o}} + P_{\text{mem}_{H_o}})$ **then**
$N_{\text{nodes}} \leftarrow N_{\text{total}}$
else
$N_{\text{nodes}} \leftarrow P_{\text{ub}}/(P_{\text{cpu}_{H_o}} + P_{\text{mem}_{H_o}})$
end if
for every node i to be activated **do**
$P_{\text{cpu}_{run_i}} \leftarrow P_{\text{cpu}_{H_o}} + P_{\text{cpu}_{v_i}}$
$P_{\text{mem}_{run_i}} \leftarrow P_{\text{mem}_{H_o}} + P_{\text{mem}_{v_i}}$
end for
end if
end function

4.3 Evaluation Results

We use the performance of the All-In method which does not enforce a power bound as the baseline performance and then compute the relative performance of CLIP and other power allocation methods. Fig.7 summarizes the comparison results[6], from which we draw several observations.

1) CLIP achieves similar performance as All-In for most of the applications under study, and outperforms All-In by more than 40% for MiniMD and SP-MZ applications of the parabolic type, when there is no specified power bound.

2) CLIP performs best for all the tested benchmarks if the power budget is unlimited or higher enough to support compute and memory components to operate at their maximum capacity.

3) CLIP outperforms All-In, Coordinated, Lower Limit for most cases, specially for logarithmic and parabolic applications.

4) CLIP outperforms Coordinated for parabolic applications (SP-MZ, miniAero and TeaLeaf) by up to 60% overall. When the thread count further increases, textit parabolic applications experience a worsened performance but consume more power. Carefully distributing resources for such applications significantly improves performance.

5) CLIP outperforms Coordinated for logarithmic applications when the power budget is low. Logarithmic applications are common among big data applications that require higher memory bandwidth. This observation confirms the hypothesis that it is beneficial to classify applications and correspondingly set configurations for power-bounded computing.

5 Job Co-Run and Resource Sharing

To improve system efficiency, modern HPC systems allow node sharing among different jobs. While resource sharing has been extensively studied, job scheduling considering both power and hardware resource is a fundamentally new problem. We found that contention is the major key factor that degrades the performance of co-running jobs. Power constraint induces or aggravates resource contention among jobs, particularly in the memory hierarchy. Furthermore, when the total power is limited, balancing power among nodes and components is critical. We study how power limiting affects contention between collocated scientific parallel jobs in multicore-based clusters, and research effective strategies to mitigate contention and maximize system performance under given power budgets.

To estimate the level of contention and mitigate its performance impacts on co-running jobs, we have developed CAPS[7], a Contention-Aware Power-bounded Scheduling approach which uses machine learning models to predict contention using application performance and power profiles. Overall, CAPS embraces two key ideas: 1) infer the contention using applications' performance and power profiles and its variation with power limits, and 2) exploit job collocation and supportive power distribution across nodes and components to mitigate contention caused by power limits.

5.1 Benefits of Resource Sharing Under Power Constraints

Because different applications can have different

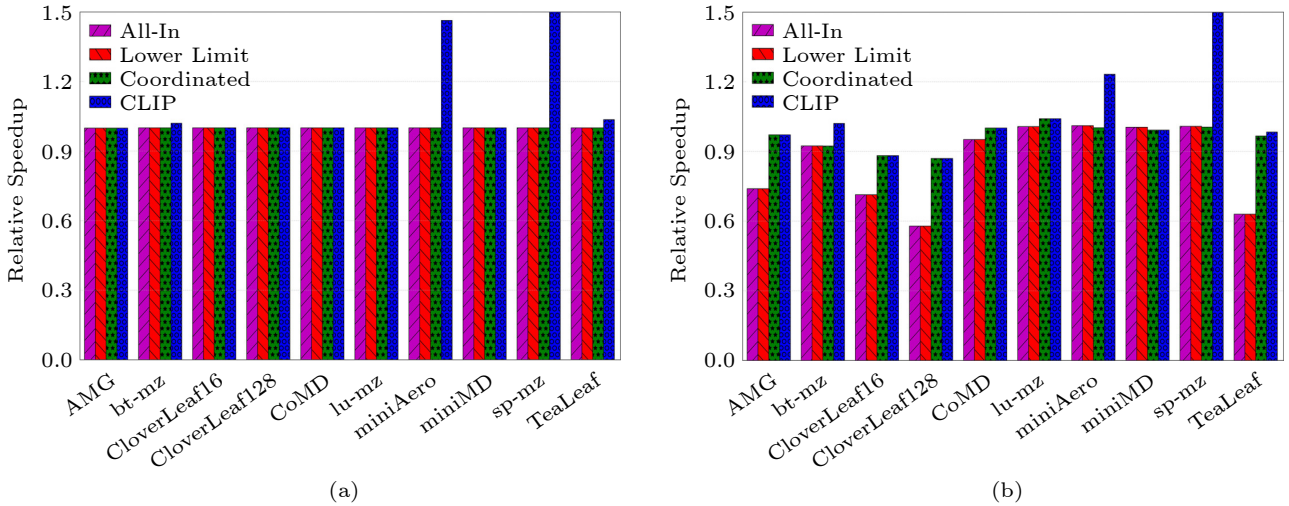


Fig.7. Performance comparison of different power allocation methods under high power budgets[6]. (a) Application performance without power bound. (b) Application performance with $P_b = 1600$ W and 200 W power bound.

resource requirements and power demands, sharing resources among jobs which are complementary to each other can lead to performance gain and efficiency improvement.

Fig.8 shows the throughput difference between two scheduling methods. We run STREAM and EP on an Intel Haswell Dual-processors node. The coarse-grained method runs STREAM with 24 threads and EP with 24 processes serially. The fine-grained method executes STREAM and EP concurrently, with each workload occupying half of the core resources. Correspondingly, the thread/process number of STREAM/EP is reduced to 12 for each application.

As shown in Fig.8, sharing a node between EP and STREAM improves the utilization of both hardware resources and power budgets at the node level[7, 8]. While running STREAM and EP one after another, the power consumption varies from 230 W to 160 W without a power cap. The same power budget (e.g., 220 W) will impact STREAM's performance significantly and is underutilized by EP. Fine-grained scheduling keeps more even power consumption across nodes, and improves the system throughput by more than 20%.

However, to ensure resource sharing given a power bound will benefit system throughput, we must answer several questions. First, how should the system determine complementary workloads that benefit from resource sharing given a power bound? Second, will job collocation still be beneficial under different power budgets? Third, how does the system allocate power to nodes and components for co-scheduling systems?

Our power bounded computing research provides three key insights. First, job collocation is a practical technique to increase system performance under certain power limits. Second, a proper scheduler must be power-aware because power limits can change jobs from non-interfering to interfering. Third, an effective scheduler must dynamically distribute available power to computer components based on the available power and the workload characteristics of the jobs under study.

5.2 CAPS (Contention-Aware Power-Bounded Scheduling)

On a power limited cluster, contentions between collocated jobs are from two sources: 1) shortage of hardware capacity if power is abundant ($P_b = P_\infty$) and 2) hardware capacity reduction due to power limits. These contentions affect system throughput (STP)[9], which is defined as

$$STP(P_b) = \sum_{i,j} \left(\frac{T_i^{|}(P_b)}{T_i^{||}(P_b)} + \frac{T_j^{|}(P_b)}{T_j^{||}(P_b)} \right).$$

Here $T^{|}$ and $T^{||}$ are the execution time of jobs i and j when they sequentially and concurrently run respectively under the same power budget P_b. Both runs use all the CPU cores. STP is a relative metric; $STP > 1$ indicates collocating jobs i and j gains throughput over sequentially executing them.

Contention under abundant power P_∞ has been extensively studied on multicore systems[10, 11]. Prior works commonly use hardware performance monitoring counter (PMC) to infer performance loss of each job and the resulting system throughput. A variety of inference methods have been suggested including statistical modeling and linear regression[12]. Recently, neural networks[13] show promising performance loss prediction on modern multicore systems.

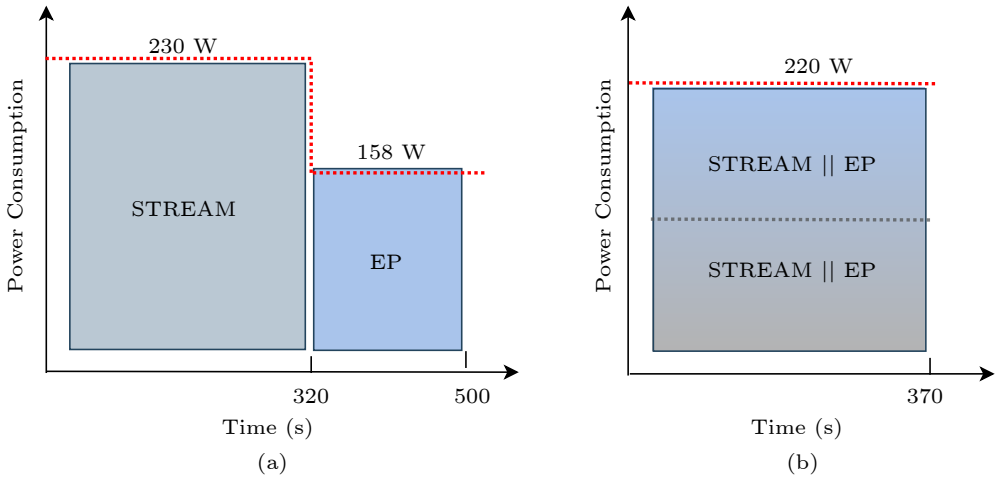


Fig.8. Throughput comparison between (a) coarse-grained and (b) fine-grained (collocation) resource scheduling[7, 8].

In our work, we apply a neural network model which is similar to the one proposed in [13] to infer system throughput when power is abundant ($P_\text{b} = P_\infty$). The model is a 2-layer neural network. The input layer is connected with two hidden layers with 24 × 12 neurons. The model uses ReLU as the activation function and a learning rate of 0.01. The model outputs are the execution times of collocated jobs, which we use to calculate the resulting system throughput. Fig.9 shows the model inputs and outputs.

To increase the model accuracy, our model uses extra inputs like jobs' CPU and memory power consumption, and the performance ratio between using all and half of the cores $\frac{Perf_\text{all}}{Perf_\text{half}}$. Except for the performance ratio, all other inputs are collected when the job runs exclusively on half of the cores distributed across sockets without a power bound. Our model output is the execution time of collocated jobs, which is used to calculate the resulting system throughput. Table 2 lists the details of metrics as input for interference prediction.

5.3 Contention Under Power Limiting

Assuming that the power budget would be optimally distributed between CPU and memory, STP generally decreases as power budget P_b drops from abundant to inadequate: interfering jobs become more contentious, while complementary jobs may start to contend due to reduced capacity of resources and eventually interfere with one another. This trend indicates that power limiting aggravates contentions between collocated jobs.

The scheduler for power bounded job co-run must answer the job compatibility question: for a given power budget P_b and a pair of jobs i and j, should the jobs be collocated for the sake of throughput? To answer this question, we classify job paring into three cases and develop an effective strategy for each case. Denoting the system throughput under abundant power $STP(P_\infty)$, the three cases are listed as follows:

- *Case* 1: $STP(P_\infty) < 1.0$. Never collocate jobs i and j for any given power budget.
- *Case* 2: $1.0 < STP(P_\infty) < 1.2$. Do not collocate jobs i and j for any given power budget. This strategy may lose some opportunities for throughput improvement but is simple.
- *Case* 3: $STP(P_\infty) > 1.2$. Collocate jobs i and j if the given power budget $P_\text{b} > P_\text{th}$, where P_th is a threshold we choose heuristically. This threshold ensures collocating the jobs has a system throughput greater than 1.

Here $STP(P_\infty) = 1.2$ is an empirical value we choose based on our experimental results because it maintains a good tradeoff between individual job's performance loss and the overall system throughput gain. Users may choose a different value for their optimization constraints and objectives.

5.4 The Design and Implementation of CAPS

CAPS is a two-level power coordination scheduler that explicitly models contentions due to job colloca-

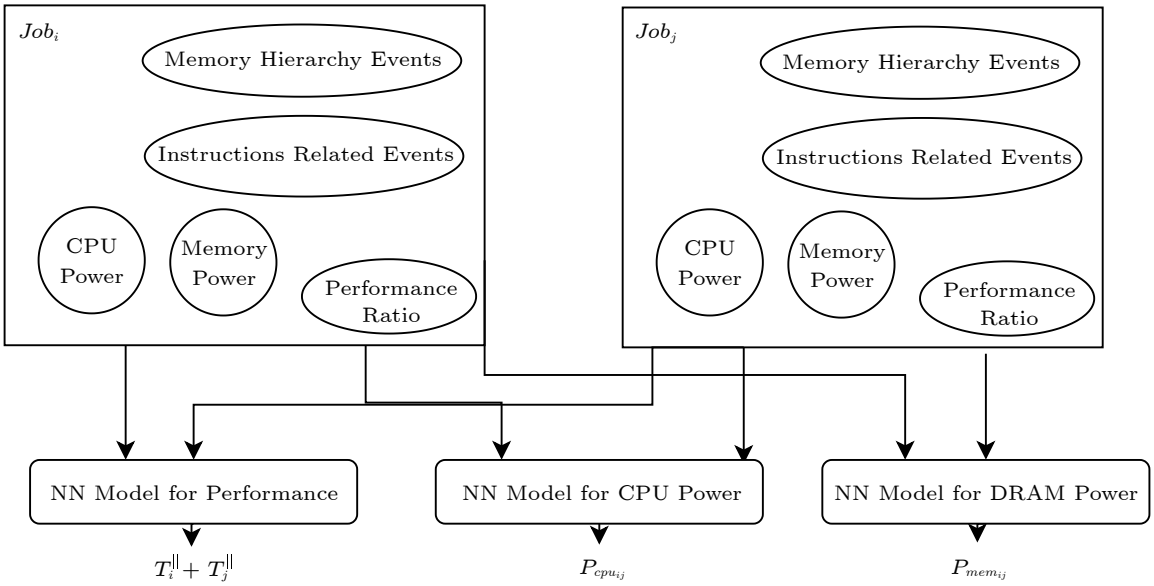


Fig.9. Performance and power estimation models[7, 8].

Table 2. Metrics Used in CAPS for Workload Interference Prediction

Event Type	Description
Memory hierarchy event	Memory read bandwidth
	Memory write bandwidth
	Local L3 cache miss
	Remote L3 cache miss
	L3 miss rate
	L3 request rate
	L2 miss ratio
Instructions related event	Cycles per instruction
	Unhalted time/runtime
Power consumption data	CPU power
	DRAM power
Performance ratio	$Perf_{\text{all}}/Perf_{\text{half}}$

Note: The metrics are platform-specific and may be subject to changes on other platforms.

tion and aims to minimize contentions when searching for optimal power allocations. Specifically, we use a neural network model to estimate $P_{\texttt{cpu}_{H_{ij}}}$ and $P_{\texttt{cpu}_{L_{ij}}}$ from jobs' performance and power profiles when CPUs run at the highest and lowest frequencies respectively. We further use these two values to estimate the threshold power $P_{\texttt{th}_{ij}}$ suitable for their collocation to avoid throughput loss. We always allocate sufficient power to memory to avoid significant performance degradation, and denote this power as $P_{\texttt{mem}_{ij}}$.

Assuming that job j is assigned to run on m nodes and only uses a subset of the cores, and the available power for these nodes is P_r, the scheduler examines the next job in the queue. To efficiently utilize the power budget, the scheduler has different job scheduling and power allocations based on the predicted power consumption of the collocated jobs.

- $P_{\texttt{r}}/m \gg P_{\texttt{cpu}_{H_{ij}}} + P_{\texttt{mem}_{ij}}$: the scheduler may switch to co-run with j to avoid power budget waste or request the system to reclaim extra power.
- $P_{\texttt{r}}/m > P_{\texttt{cpu}_{H_{ij}}} + P_{\texttt{mem}_{ij}}$: the scheduler shifts extra power to other nodes as illustrated in Subsection 5.5.
- $P_{\texttt{r}}/m > P_{\texttt{th}_{ij}} + P_{\texttt{mem}_{ij}}$: the scheduler allocates $P_{\texttt{mem}_{ij}}$ to memory, and the remaining power to processors, where $P_{\texttt{th}}$ is the model's estimated power threshold suitable for job collocation.
- $P_{\texttt{r}}/m < P_{\texttt{th}_{ij}} + P_{\texttt{mem}_{ij}}$: the scheduler may choose to run job j by itself to avoid significantly performance loss by insufficient power.

After managing job scheduling and power allocation within nodes, the scheduler may coordinate power among multiple nodes to achieve global optimal throughput, and ensure all jobs allocated with their acceptable power ranges.

Once a collocated job completes execution, CAPS coordinates power at the node and cluster levels to best fit workloads' power demands. It does not consider the phase changes inside a job to control overhead.

5.5 Evaluation

We evaluate the performance of CAPS on an 8 dual Haswell processors node cluster. We train the models for interference and power prediction using a training dataset collected from the benchmarks and assess the model accuracy using the test dataset. We summarize the key results as follows.

- After having examined multiple neural network models, we find that the model of two hidden layers, with 24 and 12 neurons at the two layers respectively, provides the best accuracy. Thus, we adopt such a network in CAPS for interference prediction.
- The average estimation error of the interference model is about 7%. The model tends to underestimate the interference for some memory intensive workloads.
- The average difference between the prediction power and the actual power is less than 10 W. Thus, these power models satisfy the need to estimate the power consumption of co-scheduling jobs' with high accuracy.

We also find the following rule of thumb with regard to node sharing from our model prediction and experimental results.

- Extreme memory intensive applications like `STREAM`, `CloverLeaf`, `TeaLeaf`, `CloverLeaf3D`, `TeaLeaf3D` significantly interfere with each other. It is not recommended to share a node between these applications.
- Less memory intensive applications cause little interference to extreme memory intensive applications. Conversely, extreme memory intensive applications interfere with others significantly. Nevertheless, the overall throughput increases. Thus co-scheduling extreme memory intensive applications with others applications increases system throughput.
- Compute-intensive applications and compute-intensive applications cause low interference to each other. However, the memory bandwidth could be underutilized when two compute intensive applications are co-scheduled.

CAPS reduces the execution time by 25% when the power is unbounded, in comparison with counter-

parts[7]. When power bounds are enforced, CAPS increases system throughput by 30%. Under the 1 600 W, FCFS allocates 40 W to memory and 160 W to CPU on each node. Because CAPS considers workload contentions and schedules jobs to achieve higher power utilization, it performs consistently better than other methods on power-bounded systems. Experimental details can be found at [7].

6 Conclusions

Power bounded computing is still in its infancy in research and continues to evolve to support ever increasing demand by conventional and emerging workloads such as deep neural network based large ML models. While the power constraint is not strictly enforced, efficiency is on its rise to be paramount to managing energy bills and delivering desired application performance. Power bounded computing aims to translate energy saving into application performance with an upper bound of power and energy consumption.

As HPC computing is increasingly bounded by power consumption, future systems and software must cope with these limits. The key question is how to cope with the power bound and build system management tools to distribute hardware resources and power optimally to achieve maximum performance. We believe power-bounded computing represents a new computing paradigm that distinguishes it from previous low-power computing and power-aware computing because power bounded computing explicitly considers the power bound as a design constraint and emphasizes coordinated power allocations.

In this paper, we described how the paradigm of power bounded computing is applied in the HPC under three contexts: node level power coordination between processors and memory, cluster level between nodes and components, and co-scheduling jobs and power on HPC clusters. Two central principles in our work are balanced power allocations and workload-aware scheduling. This work provides multiple insights in system scale power management to address the power challenge for exascale supercomputing systems: 1) the effectiveness of power bounded computing for maximizing performance and energy efficiency given a total budget, 2) the improvement through job co-running and resource sharing, and 3) the feasibility of dynamic cluster level intelligent power coordination given a total power budget. The power bounded computing paradigm also applies to IoT scenarios where the power constraint is more pervasive.

Our study represents the first published body of work to define and evaluate the paradigm of power bounded high-performance computing. This work has laid the groundwork for power bounded research as we move toward exascale supercomputing. Our long term future directions include developing ideas around power bounded computing for emerging hardware, power stable computing, self-learning resource management, and the trade-off between power, performance, and resilience.

References

[1] Lucas R, Ang J, Bergman K *et al.* Top ten exascale research challenges. DOE Advanced Scientific Computing Advisory Subcommittee (ASCAC) Report, U.S. Department of Energy, Office of Science, 2014. DOI: 10.2172/1222713.

[2] Jeon M, Venkataraman S, Phanishayee A, Qian J J, Xiao W C, Yang F. Analysis of large-scale multi-tenant GPU clusters for DNN training workloads. In *Proc. the 2019 USENIX Annual Technical Conference*, Jul. 2019, pp.947–960.

[3] Ge R, Feng X Z, Allen T, Zou P F. The case for cross-component power coordination on power bounded systems. *IEEE Trans. Parallel and Distributed Systems*, 2021, 32(10): 2464-2476. DOI: 10.1109/TPDS.2021.3068235.

[4] Ge R, Feng X Z, He Y Y, Zou P F. The case for cross-component power coordination on power bounded systems. In *Proc. the 45th International Conference on Parallel Processing (ICPP)*, Aug. 2016, pp.516–525. DOI: 10.1109/ICPP.2016.66.

[5] Ge R, Zou P F, Feng X Z. Application-aware power coordination on power bounded NUMA multicore systems. In *Proc. the 46th International Conference on Parallel Processing (ICPP)*, Aug. 2017, pp.591–600. DOI: 10.1109/ICPP.2017.68.

[6] Zou P F, Allen T, Davis C H, Feng X Z, Ge R. CLIP: Cluster-level intelligent power coordination for power-bounded systems. In *Proc. the 2017 IEEE International Conference on Cluster Computing (CLUSTER)*, Sept. 2017, pp.541–551. DOI: 10.1109/CLUSTER.2017.98.

[7] Zou P F, Feng X Z, Ge R. Contention aware workload and resource co-scheduling on power-bounded systems. In *Proc. the 2019 IEEE International Conference on Networking, Architecture and Storage (NAS)*, Aug. 2019. DOI: 10.1109/NAS.2019.8834721.

[8] Zou P F, Rodriguez D, Ge R. Maximizing throughput on power-bounded HPC systems. In *Proc. the 2018 IEEE International Conference on Cluster Computing (CLUSTER)*, Sept. 2018, pp.156–157. DOI: 10.1109/CLUSTER.2018.00030.

[9] Eyerman S, Eeckhout L. System-level performance met-

rics for multiprogram workloads. *IEEE Micro*, 2008, 28(3): 42–53. DOI: 10.1109/MM.2008.44.

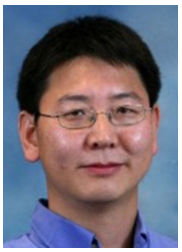
[10] Blagodurov S, Zhuravlev S, Fedorova A. Contention-aware scheduling on multicore systems. *ACM Trans. Computer Systems*, 2010, 28(4): Article No. 8. DOI: 10.1145/1880018.1880019.

[11] Subramanian L, Seshadri V, Ghosh A, Khan S, Mutlu O. The application slowdown model: Quantifying and controlling the impact of inter-application interference at shared caches and main memory. In *Proc. the 48th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec. 2015, pp.62–75. DOI: 10.1145/2830772.2830803.

[12] Kelley J, Stewart C, Tiwari D, Gupta S. Adaptive power profiling for many-core HPC architectures. In *Proc. the 2016 IEEE International Conference on Autonomic Computing (ICAC)*, Jul. 2016, pp.179–188. DOI: 10.1109/ICAC.2016.45.

[13] Mishra N, Lafferty J D, Hoffmann H. ESP: A machine learning approach to predicting application interference. In *Proc. the 2017 IEEE International Conference on Autonomic Computing (ICAC)*, Jul. 2017, pp.125–134. DOI: 10.1109/ICAC.2017.29.

Rong Ge received her B.S. and M.S. degrees in engineering mechanics from Tsinghua University, Beijing, in 1995 and 1998, respectively, and her Ph.D. degree in computer science at Virginia Tech, Blacksburg, Virginia, in 2007. She is the director of the Scalable Computing and Analytics Laboratory in the School of Computing at Clemson University, Clemson. Her research interest includes parallel and distributed systems, machine learning and big data, heterogeneous computing, and performance evaluation and optimization.



Xizhou Feng received his M.S. degree in engineering thermophysics from Tsinghua University, Beijing, his Ph.D. degree in computer science from University of South Carolina, Columbia, and his J.D. degree from Marquette Law School, Wisconsin. He is currently a software engineer at Meta Platform, Inc. His research interests include complex AI models and training efficiency, high-performance computing and cyberinfrastructure, scalable algorithms and systems, computational sciences and informatics, complex system modeling, and the interactions between technology and law.

Pengfei Zou received his B.S. degree in remote sensing science and technology from Wuhan University, Wuhan, in 2012, his M.S. degree in geographical information system (GIS) from University of Chinese Academy of Sciences, Beijing, in 2015, and his Ph.D. degree in computer science (CS) at Clemson University, Clemson, in 2020. His research interest includes parallel and distributed systems, heterogeneous computing, remote sensing, and cross-discipline technologies between GIS and CS.

Tyler Allen is an assistant professor at University of North Carolina at Charlotte. He received his Ph.D. degree in computer science at Clemson University, Clemson, in 2022. His research interests are in high-performance computing, parallel and heterogenous systems, and the system stack across compiler, operating system, and architecture. He has extensive experience in GPGPU computing and manycore systems.