The Case for Cross-Component Power Coordination on Power Bounded Systems

Rong Ge, *Member, IEEE,* Xizhou Feng, *Member, IEEE,* Tyler Allen, *Member, IEEE,* and Pengfei Zou, *Member, IEEE,*

Abstract—Modern computer systems are increasingly bounded by the available or permissible power at multiple layers from components to systems. To cope with this reality, it is necessary to understand how power bounds impact the design and performance of emergent computer systems. Prior work mainly focuses on power capping and budgeting on individual components without coordinating them to achieve the best possible performance. In this work, we study the problem of power bounded computing and power allocation across computer components on CPU and GPU-accelerated systems. We investigate the dynamics between cross-component power allocation and generalize the performance impacts, and propose lightweight heuristics to maximize performance. We draw multiple insights: (1) for a given application and power bound, there exists a maximum achievable performance which requires coordinated power allocation among components for balanced computation and memory access; (2) the max performance increases with the total power bound but only in a definite range specific to applications; (3) the dynamics of power allocations has categorical patterns with regard to performance trends and actual power use; (4) the categorical patterns can be leveraged to design coordinated power allocations. These findings suggest the promises of cross-component coordination in forthcoming power bounded high performance computing.

Index Terms—Power-bounded computing, cross-component power coordination, power capping, performance analysis, power management.

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 centers [1]. For example, computer components must operate within their thermal design power, and the first exascale systems are imposed with a power budget of 20 – 30 megawatts [29]. Such power bounds are set due to physical, technical, and economical reasons. First, the cooling devices and facilities for components, machines and server rooms have certain capacities that set the ceiling of permissible power density. Second, the available power capacities along the path from power source to buildings and servers have practical upper bounds. Third, the energy cost for powering up computer systems can be significant or even prohibitive. At the rate of ¢10/joule in U.S., the annual energy bill for an exascale computer is \$20 – 30 million. Nevertheless, systems are expected to sustain the performance growth. The upcoming exascale computers must deliver more than doubled performance with the same power budget, and future generations face the same expectations. Power has been constraining HPC scalability, availability and affordability and it is urgent to develop technologies that optimize performance under power bounds.

Today's systems and building blocks make the problem more challenging. First, they have a significant larger power envelop. For example, a compute node of the Summit supercomputer integrates 44 CPU cores, 6 high end GPUs, 512 GB DDR4 memory, and 96 GB HBM2 memory. The peak power consumption exceeds 2,800 Watts. Inefficient use of such a big power envelop would lead to significant waste. Second, unlike previous generations where CPUs are the only dominating power consumer, today's systems have multiple major consumers, i.e., CPUs, GPUs, and memory. Memory may consume more power than processors on big-memory systems [2], and high end GPUs consume comparable or more power than CPUs. These components must be simultaneously managed to maximize performance under limited power. Third, hardware resources are generally overprovisioned. Most applications are unable to fully utilize all the hardware all the time, and CPU algorithms don't use GPUs while GPU algorithms typically only use CPUs for kernel offloading and data management. Activating components without disparity not only wastes the limited available power but also suffers suboptimal performance. Hence, maximizing performance under limited power is complicated at an unprecedented level.

Prior work has predominately focused on individual components, resulting in suboptimal performance and power efficiency. Power aware computing exploits DVFS capable components and lowers their performance states when the full capacity is not needed [15, 25]. Power capping ensures component power within the specified cap through software managed hardware technologies on CPUs, memory, and GPUs [14, 23]. Little work has studied multiple components. Hanson et al [10, 20] has studied power shifting between processors and memory and proposed feedback based control to adjust their distribution, and Coscale [16] simultaneously adjusts CPU and memory fre-

• *R. Ge, X. Feng, T. Allen, P. Zou are with the School of Computing, Clemson University, Clemson, SC 29634.*

Manuscript received September 1, 2020.

quency to reduce power consumption while meeting the target performance. They neither pinpoint the best power coordination across the components for a given available total power nor locate the performance and power inefficiency caused by poor power distributions.

Increasing performance within a power budget requires new methodologies on today's systems. To meet this need, we propose power-bounded computing that considers power as a scarce resource and promotes cross-component power coordination. The goal is to utilize every watt of available power to increase application performance and system throughput. Power-bounded computing is advantageous for several reasons. First, by viewing power instead of hardware as a scarce resource, it centers scheduling around power and activates components judiciously, particularly suitable for hardware overprovisioned systems. Second, it coordinates power across components to balance their operations, and thus gains higher performance than approaches focusing on individual components.

In this work we focus on power allocation on compute nodes which are the building blocks of HPC systems. At the node level, either compute or memory access can be the performance bottleneck, depending on the workloads and power allocation. We thus study the balance between processing units and memory modules, i.e., CPUs and main memory, and GPUs and GPU global memory. Their balance is necessary to maximize performance for a given power and requires coordinated power allocations. A challenge is that a small power shift may lead to significant performance loss. We investigate the dynamics across processor memory in power allocation and application performance, identify the patterns of power allocation scenarios, and develop optimal power coordination strategies. Overall, the major contributions of this work include:

1) We promote cross-component power coordination and demonstrate it can improve performance significantly, e.g., 35% for GPU computing and more for CPU computing. This study is the first-of-its-kind and especially timely as HPC systems are increasingly constrained by power. It provides a direction on how to sustain performance growth under limited available power.
2) We unveil the distinct patterns of performance and power dynamics and generalize them with categorization, and further validate them on multiple generations of CPU and GPU architectures. Such generalization enables the design of optimal cross-component power allocations on modern HPC building blocks.
3) We show different workloads considerably vary in characteristics while sharing common patterns, suggesting the need of integrating application awareness into power scheduling and management.
4) We present a heuristic algorithm to quickly pinpoint near-optimal cross-component power allocations with lightweight application profiling.

2 Power Bounded Computing

High performance, 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 is capable of reconfiguring itself according to its current workload to achieve better performance under the same power budget.

As per-node power is simply the sum of the power consumed by all components on the node, enforcing a per-node power limit requires allocating an appropriate amount of power to each component in a coordinated manner. Cross-component coordination is crucial for a compute node to deliver maximum application performance corresponding to a specific power budget, particularly when power becomes such a scarce resource that can not concurrently meet the maximum demands of all components.

To gain insight on the dynamics between power allocation and application performance without being distracted by the complexity of coordinating multiple closely interacting components, we focus on the problem of power coordination between processors and DRAM modules on hosts and discrete GPU accelerators, i.e., CPUs and memory, and GPU Streaming Multiprocessing cores (SMs) and global memory. We make this simplification based on two observations. First, processor cores and memory modules dominate node power consumption on current and emergent HPC systems. Second, shifting power across processors and memory affects both compute speed and data access rate, the two primary factors that jointly determine the performance of HPC applications.

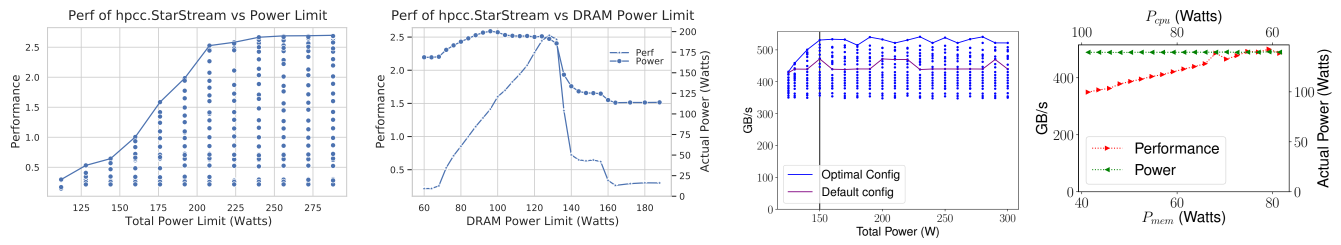
2.1 Motivating Examples

To examine the performance impact of power limit and cross-component power coordination, we run a series of experiments on an Intel Xeon 20-core IvyBridge machine and a Nvidia Titan XP GPU accelerated machine respectively. In these experiments, we specify a total power budget and then distribute it among processors and memory modules by capping the power on each individual component. We use the Intel's "Running Average Power Limit" RAPL technology to cap the power for the CPU based machine, and `nvidia-settings` for the discrete GPU accelerator.

Figure 1 summarizes the experimental results of enforcing power bounds on processors and DRAMs for the Stream benchmark. The CPU version is from the HPC Challenge suite and the GPU-version is from GPU-Stream respectively. Note the reported performance is for per-core for CPU computing and for the entire card for GPU computing. While CPU computing and GPU computing have different performance and power dynamics in quantities, they both show common important points.

First, the total power budget has a significant impact on performance. Specifically, for a given budget, there exists an upper bound for the achievable performance. As the budget increases, the upper performance bound increases — in a nonlinear manner — and then flattens once the budget reaches a certain value. This observation holds on both CPU and GPU computing, though the upper performance bound reaches the maximum sooner on GPU computing.

Second, power allocation between processors and memory has a significant impact on application performance for the same total power budget. Given a total power budget 208 Watts for CPU computing, an optimally coordinated power allocation leads to up to 30× better performance compared to a poorly coordinated power allocation. While



(a) CPU computing

(b) GPU computing

Fig. 1. Performance of Stream (in GB/s) with (a) CPU computing and (b) GPU computing. Performance vs. total power budgets (left), and performance vs. cross-component power allocations for a certain total budget (right). The given budget is 208 W for CPU computing and 140 W for GPU computing. Bandwidth per core is reported on the CPU system, and total on the GPU system.

the total power budget and shifting range are smaller on GPU, the best performance is still over 30% higher than the poorest given 140 Watts.

Third, power capping on individual components is effective to limit system power under budgets. As shown in the right figures, the total power consumed by processors and memory is always under the given budget 208 Watts on CPU computing, and 140 Watts on GPU computing.

Fourth, in both CPU and GPU computing, the provisioned power budget could be fully consumed even if the delivered performance is very poor, suggesting significant power waste with certain power allocations.

These examples motivate us to delve into *the following research questions* and seek answers to them:

1) For a given application W, what is the upper performance bound $\texttt{Perf}_{max}$ for a given total power budget P_b, and what is the $\texttt{Perf}_{max} \sim P_b$ relationship? The answers will provide system designers and programmers with performance optimization targets and proper power budgeting policies required for sustained performance growth with limited power.
2) For a given application W and an allowable total power budget P_b, what is the optimal distribution of P_b among processors and memory that ensures $\texttt{Perf}_{max}$? The answer will enable optimal workload and power scheduling to meet optimization objectives and constraints.
3) Why do poor cross-component power allocations cause performance and power inefficiency? The answers will enable us to locate inefficiency bottlenecks, and identify solutions for improvement.
4) What ranges of P_b are acceptable regarding achievable performance and power efficiency? The answers will guide power schedulers to set appropriate power budgets and reclaim unused power for global efficiency.

2.2 The Problem of Power Bounded Computing

To answer the above questions, we formulate the problem of **power bounded computing** at the node level as follows:

Given a parallel workload W, a machine M comprising a set of K power-boundable components $C_1, C_2, \ldots, C_K$, and a total power bound P_b, 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) $\texttt{perf}_{max} = max_{\alpha \in A}\, \texttt{perf}(\alpha, W, M)$,
(2) $\alpha^* = \text{argmax}_{\alpha \in \mathcal{A}}\, \texttt{perf}(\alpha, W, M)$, and
(3) $\sum_{i=1}^{K} P_i^* \leq P_b$.

Here, α is a power allocation tuple and $\mathcal{A}$ is the space that comprises all possible values of α. A component is power-boundable if it can and will always operate under the specified power cap. The performance metric, $\texttt{perf}$, can have different definitions depending on both the application and the user's demand. Example measures include compute rate, performance-to-power ratio, and system throughput.

In this paper, we discuss a simplified power bounded computing problem and the cross-component power coordination under the following assumptions:

(a) The machine M consists of two types of major components — processors and memory modules. A homogeneous machine concerns about multiple CPU cores and memory modules, and a discrete heterogenous GPU concerns about GPU cores and the global memory.
(b) All processor cores are grouped into an aggregated component (**CPU** or **GPU**) with its power budget P_{cpu} or P_{SM} evenly distributed to all cores.
(c) All memory modules are aggregated with the power budget P_{mem} evenly distributed among memory devices for CPU or GPU computing.

These simplifications are valid for many MPI, OpenMP, or CUDA parallel applications that have balanced workload across the processing units and memory devices. Here we focus on coordinating power between processors and memory for such systems and exploring fundamental scheduling guidelines. We leave the investigation of unbalanced workloads and hybrid computing in our future work.

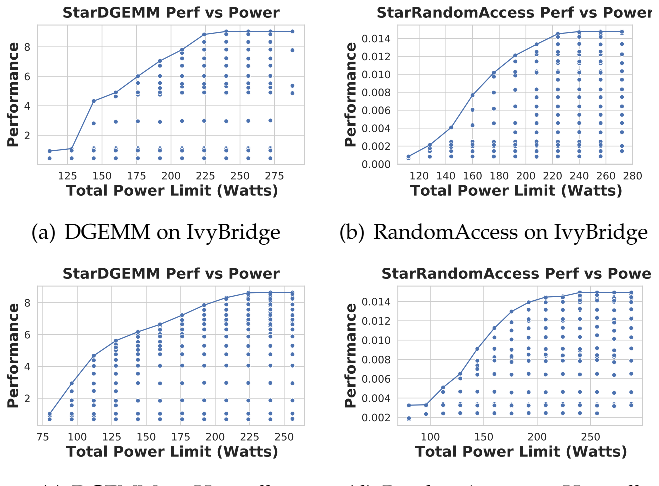
3 Dynamics of Processor-Memory Power Allocations

The examples in Figure 1 not only highlight that power limits have a decisive impact on achievable performance, but also reveal the importance of cross-component power allocation and patterns of the dynamics. Understanding the impacts of power limits and identifying these patterns help us better understand the power problems and identify the optimal power allocations in HPC.

3.1 The Effects of Power Limits

To illustrate the impacts of power limits, we conduct a set of experiments on two generations of CPU-based multicore systems. One consists of two Intel Xeon IvyBridge 10-core processors capable of per-processor DVFS from 1.2 to 2.5 GHz and 256 GB 1600 MHz DDR3 memory. The other consists of two Intel Xeon Haswell 12-core processors capable of per-core DVFS from 1.2 to 2.3 GHZ and 256 GB DDR4 memory, which has a minimum of 2133 MHz and no defined maximum clock speed. DDR4 consumes less power, partly due to less frequent refreshing of its content and technology evolution. We distribute a given total power budget P_b to processors and memory in various pairings, and repeat for a range of total power budgets.

Figure 2 shows the overall results for DGEMM and RandomAccess benchmarks from the HPC challenge suite. The curve shows the variations of $\texttt{perf}_{max}$ with P_b, and the data points show performance from different processor-memory power allocations. For both applications, as the total power budget increases, the achievable max performance increases monotonically at varying rates and then flattens, though at different inflection points. The $\texttt{perf}_{max} \sim P_b$ curve consists of several segments. Take DGEMM on the IvyBridge system as an example. Performance grows very slowly when P_b is very small, i.e., less than 125 Watts, fast when P_b continues to increase, then slowly again when P_b is more than 145 Watts, and finally stops growing once P_b is greater than 240 Watts. The gradually flattened performance suggests diminishing return from large power provision. While both DGEMM and RandomAccess present similar patterns, DGEMM gains performance more quickly and has a larger max power demand than STREAM. Between the two systems, the Haswell-based delivers better performances at small total power budgets for a given application, mainly due to the higher power efficiency and higher bandwidth on DDR4 than DDR3. Nevertheless, the two systems consume similar power when performance reaches the maximum.



(a) DGEMM on IvyBridge (b) RandomAccess on IvyBridge

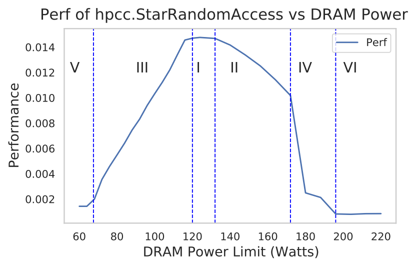
(c) DGEMM on Haswell (d) RandomAccess on Haswell

Fig. 2. Upper performance bound $\texttt{perf}_{max}$ varies with P_b.

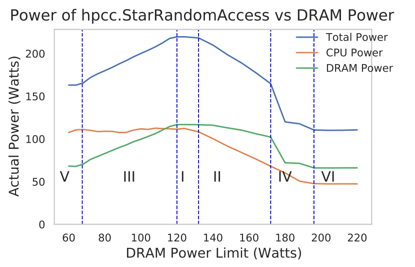
From these experiments we draw some insights for designing power budgeting algorithms and policies. First, as a low total power budget results in low performance *and* power efficiency, it should not be allocated to run new jobs. Instead, it can be added to boost the execution of current jobs or reclaimed by higher level schedules. Second, power overbudgeting wastes power without increasing performance. Therefore, schedulers should avoid budgeting excessively larger power than what applications can consume. Third, optimal schedulers should differentiate between applications and their demands.

3.2 Categorizing Power Allocation Scenarios

Under a given power budget, we observe that performance differs significantly depending on the power distribution across processors and memory. The impacts of power allocations fall into six categories. To be consistent with the aforementioned simplified cross-component power coordination problem, we explicitly denote P_{cpu}, P_{mem}, $P_b = P_{cpu+mem}$, and $\alpha = (P_{cpu}, P_{mem})$.



(a) Perf. on IvyBridge



(b) Power on IvyBridge

Fig. 3. Categorization of power allocation scenarios. The plots of (a) application performance and (b) actual power consumptions for different power allocations between processors and memory modules visually reveal six categories of power allocation scenarios.

Take the RandomAccess benchmark on the IvyBridge system as an example, there exist six categories of power allocation scenarios for $P_b = 240$ Watts as follows:

I. *Adequate power allocation for both CPUs and memory.* The power allocation $P_{mem} \in [120, 132]$ (Watts) in Figure 3 falls into this scenario. The power caps of both CPU and memory exceed their maximum power demands respectively. Because both components are able to operate at their highest performance state at the same time, an application can achieve its maximum performance determined by its workload characteristics. Meanwhile, since power is not a constraint and component power reflects the workload characteristics, the actual power consumption of each component stays constant. For the case shown in Figure 3, it is 112 and 116 Watts on processors and memory respectively.

The existence of Scenario I highlights several points:

(a) There exists a maximum power demand for a given workload.
(b) Supplying more power than the maximum demand does not lead to a higher application performance;
(c) Increasing the power budget beyond the maximum demand will increase the span of scenario I.
(d) When a power budget is much greater than the application's maximum power demand, the unused power should be reclaimed by the system for other uses.

II *Adequate memory power, lightly constrained CPU power.* The power allocation $P_{cpu} \in [68, 118]$ (Watts) or $P_{mem} \in [132, 172]$ (watts) in Figure 3 falls into this scenario, where the power allocated to CPUs is lightly constrained 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.
Scenario II is not an ideal allocation in this example with $P_b = 240$ (Watts) because memory wastes its power budget while CPU is constrained by available power. However, scenario II provides important heuristics for making power scheduling decisions: warrant memory power to deliver best possible performance when the total power budget is insufficient to maximally power up both CPUs and DRAM.

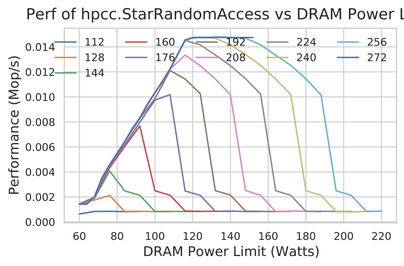
III *Adequate CPU power, constrained memory power.* The power allocation $P_{cpu} \in [160, 212]$ (Watts) or $P_{mem} \in [68, 120]$ (watts) in Figure 3 falls into this scenario. CPUs receive more power budget than needed. Their actual power stays relatively constant and is slightly smaller than the maximum demand, regardless if more budget is allocated. Meanwhile, DRAM receives inadequate allocations, and its actual consumption is close to the budget. In scenario III, application performance is bounded by memory performance; increasing power allocation to memory dramatically improves application performance. In Figure 3, we observe that the spans of power allocation regions for scenario II and III are both large. However, power allocations in scenario II are more favorable than allocations in scenario III because application performance is much more sensitive to the constraint on memory power allocation.

IV *Adequate memory power, seriously constrained CPU power.* The power allocation $P_{cpu} \in [40, 66]$ (Watts) in Figure 3 falls into this scenario. CPU power is significantly under-budgeted while DRAM is over-budgeted. The application performance drops sharply from those in Scenarios II and III. In scenario IV, memory consumes much less power that its allocation, mainly due to the fact that CPUs make less frequent memory request.

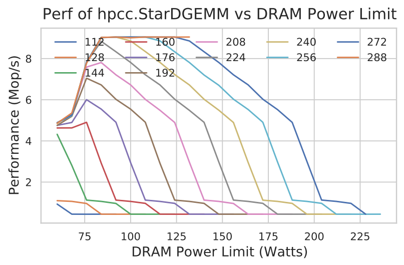
V *Adequate CPU power, minimum memory power.* The power allocation $P_{mem} \in [-, 68]$ (Watts) in Figure 3 falls into this scenario. The actual CPU power is close to the maximum CPU power corresponding to a given workload which is about 108 Watts.

VI *Adequate memory power, minimum CPU power.* The power allocation $P_{mem} \in [200, -]$ (Watts) in Figure 3 falls into this scenario. CPU receives its minimal or close to minimal power allocation. It consumes a minimum hardware determined power of 48 Watts if a lower budget is allocated. Meanwhile, memory receives excessive power budget. This scenario cannot ensure the system power bound and often delivers the worst performance.

The total power budget affects the appearance of power allocation categorization scenarios. As shown in Figure 4(a), while the general patterns look similar, the number of categories and the span of each scenario vary with the total power budget. For example, if the total power budget is less than the sum of maximum cpu power and memory power demands, scenario I does not appear. By further reducing the power budget, the spans of scenario II and III will reduce or disappear correspondingly. The disappearing scenarios are those delivering relatively high performance and utilizing the available power.



(a) Star Random Access



(b) Star DGEMM

Fig. 4. The patterns of performance impact of cross-component power allocations for (a) HPCC star random access and (b) EP-DGEMM. Experiments are conducted on the IvyBridge based system.

3.3 Under the Hood

The component power capping technologies can explain the above categorization patterns of cross-component power allocations as detailed in [19]. The Intel's RAPL interface [22] allows users to specify the power limits for the processor package, DRAM, and other RAPL domains and then transitions components to an appropriate power state to meet the power limit. The leveraged technology and corresponding power states include processor and memory sleep states (C-state), clock throttling state (T-state), and processor DVFS (P-state) [21].

When the received power budget is higher than the maximum power demand, the processor runs at the highest stable P-state (typically the nominal frequency) as in scenario I and consumes the maximum power[1]. When the power budget is lower but still higher than the demand at the lowest P-state, RAPL applies DVFS to adjust the processor's P-state to meet the power limit as in scenario II, as experimentally confirmed in [19]. To meet further reduced processor's power budget, RAPL starts to use the clock throttling mechanism to set a T-state or even a sleep state (C-state). Consequently, reduced performance is observed as in scenario IV. The operating system demands a minimum power for the processor to operate as in scenario VI.

1. We don't consider the turbo boost state as it is dynamically controlled by hardware.

Similarly, RAPL uses memory power limiting states and bandwidth throttling to limit DRAM's power consumption [14]. While DRAM bandwidth throttling reduces memory power proportionally, it decreases memory access rate, resulting in a proportional decrease of application performance shown in scenario III. Since the operating system requires a minimum memory power for the system to operate, a power budget lower than OS's required minimum memory power is disregarded, as reflected scenario VI.

3.4 Optimal Power Allocations

3.4.1 The Balance for the Upper Performance Bound

For a given power budget, the optimal cross-component power allocation delivers the best performance among all possible allocations. In essence, it provides a balanced interaction between compute and memory access, while the other allocations are bounded by either of them.

Figure 5 shows the allocated capacity and utilization of compute and memory access for DGEMM and STREAM on the IvyBridge system. The capacity R_{max} of a component K for the power budget P_K is approximated with its highest rate, i.e., when the other component is excessively powered. For example, the compute capacity under 80 Watts is the measured compute rate when CPU is allocated 80 Watts and the memory is overly allocated with 100 (greater than the known maximum consumption). With the optimal power allocation, the utilization — the ratio of the actual rate R and the capacity R_{max} — 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 compute. Similarly, when memory is under powered, application is bounded by memory access.

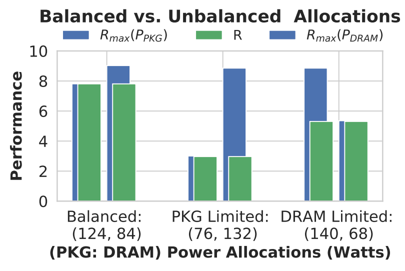
Different applications have different demands for compute and memory access, and have different compute intensities — the ratio of computation rate to memory bandwidth on the same system. Resultantly, their optimal power allocations differ. DGEMM is compute intensive and has a high power demand for CPUs,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

3.4.2 Locating the Optimal Power Allocation

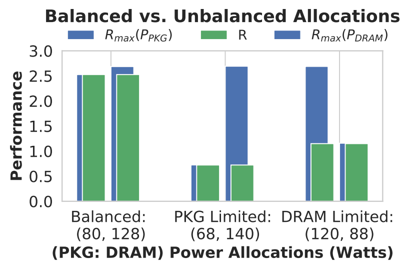
The optimal cross-component power allocation is specific to the given power budget. It is located at Scenario I given sufficient power, and usually at the intersection of two neighboring scenarios given smaller power budgets, as shown in Figure 4(a). As power budget decreases, the optimal allocation is at the intersection of Scenarios **II** and **III**, and further moves to the intersection of Scenarios **III** and **IV**. 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. For example, from the optimal power allocation

2. Scenario V may not respect the cap set on DRAM if the cap is lower than the minimum.



(a) DGEMM on IvyBridge



(b) STREAM on IvyBridge

Fig. 5. Balanced compute and memory access for a given total power budget of 208 Watts.

TABLE 1
Optimal Allocation and Critical Component vs. Power Budget.

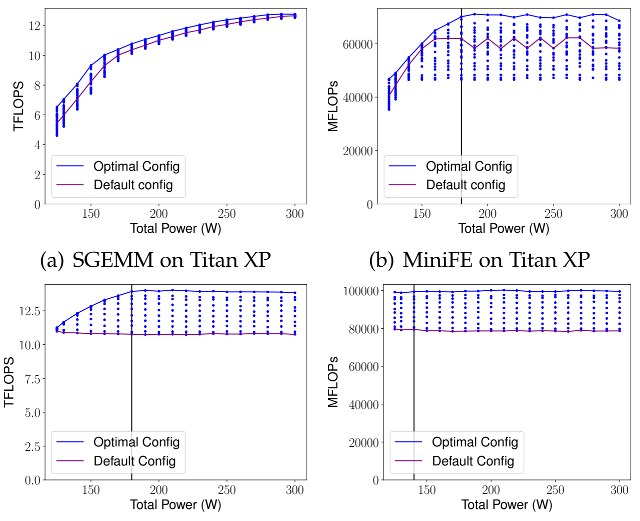
P_{ub}	Valid Alloc. Scenarios	Optimal Allocation		
		Intersection		Critical Comp.
large	I, II, III, IV, V, VI	I		none
↓	II, III, IV, V, VI	$\underline{\text{II}}$	III	DRAM
↓	III, IV, V, VI	$\underline{\text{III}}$	IV	CPU
↓	IV, V, VI	$\underline{\text{IV}}$	VI	DRAM
small	V, VI	$\underline{\text{V}}$[2]	VI	CPU

(P_{cpu} = 108, P_{mem} = 116) for the star random access benchmark and a budget of 224 Watts, shifting 24 Watts from DRAM to processors reduces performance by 50%, but shifting 24 Watts from processors to DRAM reduces performance by 10%.

We mark the *critical component* as the one that, if under powered, drastically degrades the application performance. For example, for RandomAccess on the IvyBridge system, the critical component is DRAM for $P_b = 224$ Watts and CPUs for $P_b = 176$ Watts. The existence of a critical component suggests that a power allocation strategy ensures the power budget for the critical 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.

4 Power Allocation Scenarios on GPUs

Figure 1 highlights that GPU computing is similar to CPU computing. Figure 6 confirms these observations with SGEMM and MiniFE applications on a Nvidia Titan XP and Titan V card respectively. In the experiments we adjust SM or DRAM frequency offsets respectively through `nvidia-settings`. Each data point in Figure 6 represents a SM or DRAM frequency offset setting. On the Titan XP card, SGEMM's upper performance bound continues to increase without flattening for the range of power supported by the GPU, indicating it demands more than 300 Watts. MiniFE's upper performance bound increases until the power cap is greater than 180 Watts. For a given power budget, MiniFE has a greater performance variation than SGEMM, i.e., 35% on average vs up to 25%. On the Titan V card, SGEMM's upper performance bound continues to increase until the power cap reaches 180 Watts, while MiniFE's upper performance bound does not change in the power range under study. We also observe on both cards that the default power capping mechanism for Nvidia GPUs fails to reach the maximum performance.



(a) SGEMM on Titan XP (b) MiniFE on Titan XP

(c) SGEMM on Titan V (d) MiniFE on Titan V

Fig. 6. Upper performance bound vs. power cap for a Titan XP and a Titan V GPU card respectively.

The dynamics of GPU cross-component power allocation and categories has some unique features, due to the smaller range of power management and the underlying power capping mechanism shown in Figure 7. First, fewer categories appear in the application profiles, e.g., categories I, III and II on Titan XP and category III on Titan V. GPU hardware excludes categories (IV & V & VI) that would deliver an unacceptable low performance, by disallowing low power caps on SMs and memory. In addition, the largest performance difference is only about 30% among allocations of the same power budget. Second, unlike independent management of processors and DRAM on the host, where unused power budget on one component is simply wasted, the GPU power capping automatically reclaims unused power budget 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' demand. Third, with the new SM and HBM2 technologies, Titan V has a smaller total and DRAM power range than Titan XP.

On Titan V, application performance is generally memory bounded, and increases with memory power allocation. On Titan XP, applications present three performance patterns, depending on their compute intensity.

1) Compute intensive applications like SGEMM. The highest performance under a given power budget is achieved by allocating the minimum power to DRAM. The performance curves show Categories I & II. Performance is largely constant (Category I) for large P_b values or decreases (Category II) for small P_b values as memory power allocation increases. Meanwhile, the performance curves are dispersed and diverge. The dynamics indicates application performance is constrained by SM power.
2) Memory intensive applications like Stream and miniFE. The highest performance is achieved by allocating maximum power to DRAM for a given large power budget, and by balancing between SM and DRAM otherwise. The performance curves show Categories III & II. Performance increases monotonically with memory power allocation at the same rate if P_b is large (Category III), and the performance curves with different P_b's overlap. Performance may decrease if P_b is small (Category II). The dynamics indicates application performance is mainly constrained by memory power but shift to be constrained by SM power if the total power cap is small.
3) Applications in between, such as Cloverleaf. The highest performance is achieved by allocating maximum power to DRAM for a given large power budget, and by balancing between SM and DRAM otherwise. The performance curves show Categories III & II. Performance increases monotonically but at a small rate if P_b is large, increases then decreases if P_b is small. The performance curves with different power caps do not overlap, but diverge as memory power allocation increases. The dynamics indicates application performance requires balanced allocation across SM and memory.

5 A Heuristic Power Allocation Method

Leveraging the categorization of scenarios and analysis of optimal allocations, we develop a heuristic power allocation method to the problem of cross-component coordination. This method eliminates the need of exhaustive or fine-grain profiling to locate the optimal power allocation for any given power budget.

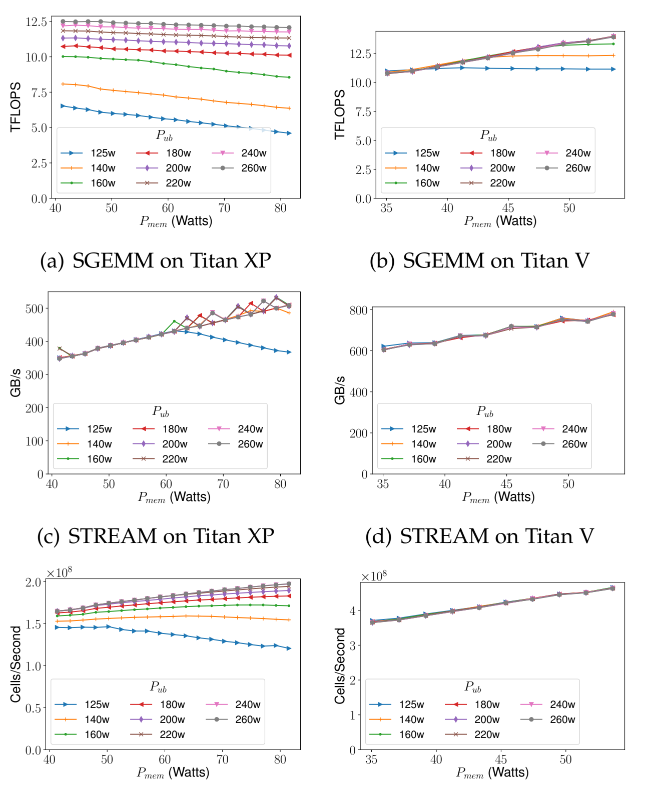
5.1 Category-Based Heuristic for CPU Computing

In Figures 3 and 4, we observe that each power allocation category is limited by a component and thus investigate the power values at which performance inflects. For the CPU experimental platforms, there are four critical processor power values (P_{cpu,L_i} $for\ i = 1..4$) and three critical memory power values (P_{mem,L_i} $for\ i = 1..3$). These application-specific values define boundaries of power allocation scenarios and correspond to the transition points at which RAPL switches from one power-saving mechanism to another, e.g., from P-state to T-state to sleep-state. Specifically, the seven critical power values are described as follows.

P_{cpu,L_1}: the maximum processor power consumption. Processor runs at its highest performance state (P-state).

P_{cpu,L_2}: the power when the processor operates at the lowest performance state. $[P_{cpu,L_2}, P_{cpu,L_1}]$ forms the power range of processor P-states.

P_{cpu,L_3}: the power when the processor is at the lowest percentage of clock throttling on the system.



(a) SGEMM on Titan XP (b) SGEMM on Titan V

(c) STREAM on Titan XP (d) STREAM on Titan V

(e) Cloverleaf on Titan XP (f) Cloverleaf on Titan V

Fig. 7. Performance trends as memory power allocation increases under various total power caps on Titan XP and Titan V respectively. The memory power is estimated using memory frequency setting and empirical power models built from experiment data on the card.

P_{cpu,L_4}: the minimum power when the processor actively executes applications. If imposed with a lower budget, the processor still consumes P_{cpu,L_4}. P_{cpu,L_4} is the same across all applications and hardware controlled.

P_{mem,L_1}: the highest DRAM power when both CPUs and DRAM run at the highest performance state to execute the application.

P_{mem,L_2}: the corresponding DRAM power when the processor power is at P_{cpu,L_3}.

P_{mem,L_3}: the minimum DRAM power set by the hardware for a running system. If DRAMs are imposed with a lower budget, they still consume P_{mem,L_3} Watts. This minimum power is the same across all applications.

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 $P_{cpu,L_2} + P_{mem,L_2}$ to fall into category I, II or III 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.

Based on these two heuristics and previous discussions on optimal allocation for each scenario category, we develop a category-based power coordination **COORD** method as shown in Algorithm 1. We assume dedicated execution environments where only one job runs on the system simultaneously, which holds true on traditional high performance computing systems. We consider fixed total power budgets and distributions across components prior to a job execution. Provided offline application profiling, this method does not incur runtime overhead, and can be integrated into batch systems such as Slurm. In the future, we will investigate how to adapt this algorithm to support online dynamic power budgeting and distribution, and multi-task and multi-tenant systems.

Algorithm 1 Category-Based Heuristic Power Coordination

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

Essentially, **COORD** breaks the set of possible power budgets into four subsets: (A) adequate budgets for both components to operate at the highest performance state, (B) adequate budgets only for one component to operate at the highest performance state, and in this case we prioritize memory power allocation as it has a greater impact on performance, (C) neither component has adequate budget to run at its highest performance state, and in this case we proportionally allocate power between processors and memory, and (D) 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 under a given power budget. As discussed in Section 6, the propositions are confirmed by our experimental results.

5.2 Algorithm Adjustments for GPU Computing

Because GPU computing has a smaller range for power allocation and the hardware settings already exclude unacceptable low power budgets, **COORD** can be simplified and uses fewer parameters. Particularly, only two parameters are needed for each application:

P_{totmax}: total power when no cap is imposed. This value can also be used to determine if the application is compute intensive, e.g., a value close to hardware maximum (300 Watts on the Titan XP GPU) indicates compute intensive.

P_{totref}: total power when memory runs at the nominal frequency, and SM runs at the min paring frequency.

Two other parameters which are pre-obtained for the GPU card and applicable for all applications are: P_{memmin} and P_{memmax}.

The general GPU version of **COORD**, shown in Algorithm 2, first checks excessive power budgeting and signals the higher level scheduler. It then considers three cases based on the power budget and the application's compute intensity: (A) for compute intensive applications, assign minimum memory power and the remaining to SMs, (B) for other applications, assign maximum memory power and the remaining to SMs if $P_b \geq P_{ref}^{tot}$, and (C) otherwise, assign power to memory to run it in a balanced fashion with SMs and the remaining power to SMs. The balance is specified with the parameter γ, which is empirically set 0.5 in our experiments. To support cards such as Titan V, this algorithm can be further reduced to just support the memory intensive case, and allocate maximum power to memory and remaining to SMs.

Algorithm 2 Category-Based Heuristic for GPU Computing

```
procedure COORD($P_b$)
    $status \leftarrow Success$
    if $P_b \geq P_{totmax}$ then
        $status \leftarrow Hint: power\ surplus\ (P_b - P_{totmax})!$
    if comp. intensive app then                    ▷ Compute intensive
        $P_{mem} \leftarrow P_{memmin}$
        $P_{SM} \leftarrow (P_b - P_{mem})$
    else
        if $P_b \geq P_{totref}$ then              ▷ Memory intensive
            $P_{mem} \leftarrow P_{memmax}$
            $P_{SM} \leftarrow (P_b - P_{mem})$
        else                                       ▷ in between
            $P_{SM} \leftarrow (P_b - P_{mem})$
            $P_{mem} \leftarrow P_{memmin} + \gamma(P_b - P_{totmin})$
            $P_{SM} \leftarrow (P_b - P_{mem})$
    return $(P_{SM}, P_{mem}, status)$
```

6 Experimental Results and Discussions

6.1 Experimental Methodology

We experimentally evaluate the problem of cross-component power coordination on two CPU-based server nodes and two GPU cards described in Table 2. We disable the processor's turbo boost performance state and hyperthreading. The GPU cards have a thermal and power specification of 250 Watts, which is the default power cap. Users can set a higher power cap up to 300 Watts through `nvidia-smi`.

TABLE 2
CPU and GPU platforms used in experiments

Platform	Processor	Memory
CPU Platform I	2 Xeon 10-core IvyBridge procs	256 GB DDR3
CPU Platform II	2 Xeon 12-core Haswell procs	256 GB DDR4
GPU Platform I	Nvidia Titan XP	12 GB GDDR5X
GPU Platform II	Nvidia Titan V	12 GB HBM2

For CPU computing, we use 11 parallel benchmarks listed in the top half of Table 3 including those from the HPC Challenge Benchmark (HPCC) [26], NAS Parallel Benchmarks (NPB) [4], and UVA STREAM. In each experiment, we use all physical CPU cores to run one benchmark and assign each core a MPI process or OpenMP thread. For GPU computing, we use 6 CUDA example programs or exascale computing proxies listed in the bottom half of Table 3.

6.2 The Patterns of Power Allocation Scenarios

We first verify whether the patterns and categorizations are universal among a wide range of parallel applications. We apply the same profiling process to all benchmarks listed in

TABLE 3
List of Benchmarks Used in This Study

Benchmark	Description and Workload Pattern
SRA	Embarrassingly parallel, random memory access
STREAM	Synthetic, measuring memory bandwidth
DGEMM	Matrix multiplication, compute intensive
BT	Block Tri-diagonal solver, compute intensive
SP	Scalar Penta-diagonal solver, compute/memory
LU	Lower-Upper Gauss-Seidel solver, compute/memory
EP	Embarrassingly Parallel, compute intensive
IS	Integer Sort, random memory access
CG	Conjugate Gradient, irregular memory access
FT	Discrete 3D fast Fourier Transform, compute/memory
MG	Multi-Grid operation, compute/memory
SGEMM	Compute intensive, CUBLAS implementation
STREAM	Memory intensive, CUDA version of STREAM
CUFFT	memory intensive, CUDA example
MiniFE	Memory intensive, ECP proxy
Cloverleaf	compute/memory, ECP proxy
HPCG	Memory intensive, HPL benchmark

Table 3 with various combinations of power budgeting and allocations on the CPU and GPU systems. The experimental results in Figure 8 show there exist both universal patterns and workload-specific features with regard to how application performance, actual power, and energy efficiency vary with different power allocations under a total budget.

Patterns Common to All Benchmarks

All benchmarks share the following common patterns.

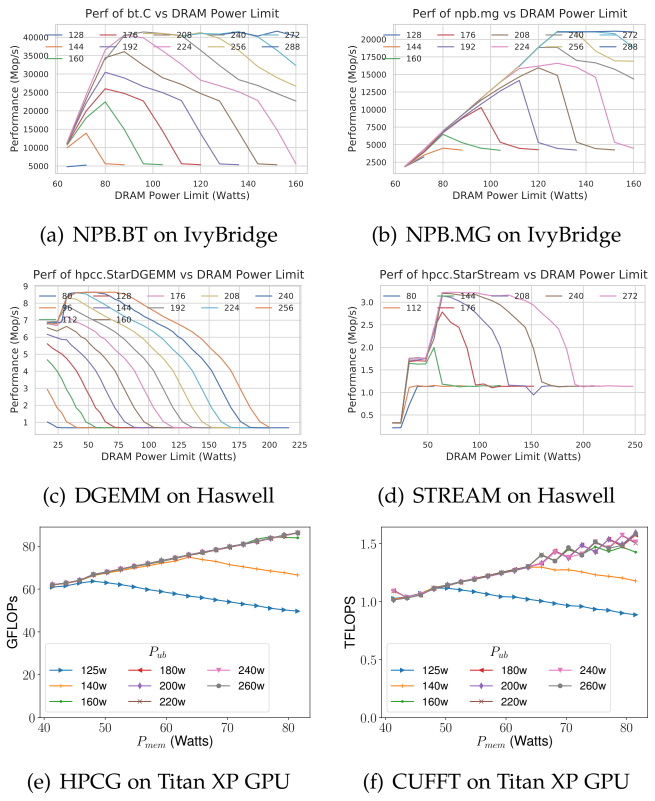
1) There exist six categories in CPU computing, and three categories in GPU computing, each characterizing a unique pattern of how cross-component allocations impact the achieved performance and actual power.
2) The actual system and component power consumptions fall between an upper bound and a lower bound. An ideal power budget would be slightly above the upper bound to ensure a robust power coordination.
3) Coordinated cross-component power coordinations are crucial for maximizing application performance and effective utilization of the available power budget.
4) Small power budgets should not be accepted due to low performance and power efficiencies.

Workload Dependent Variations

The experimental results also reveal several workload-dependent and platform-dependent pattern variations. The major variations include the sensitivity of application performance to power allocation (e.g., the slope of the performance plot), the range of each scenario category, the magnitude of maximum power consumptions, and the optimal power allocation points. We highlight three key findings.

First, computation and memory access patterns determine the shape of the performance-power curves. On the IvyBridge system, memory intensive workloads like MG demand more power budget for memory, while computing intensive workloads like BT require more budget for processors. This finding highlights the importance of application awareness when power is a scarce resource.

Second, the execution phases of workloads impact the regularity of the performance-power curves in each scenario category. Kernel benchmarks like EP-dgemm consist of a single phase, while pseudo-applications like BT and MG may comprise multiple memory access patterns. The less



(a) NPB.BT on IvyBridge (b) NPB.MG on IvyBridge

(c) DGEMM on Haswell (d) STREAM on Haswell

(e) HPCG on Titan XP GPU (f) CUFFT on Titan XP GPU

Fig. 8. Performance profiles of various applications on the three platforms. While all benchmarks share similar patterns of allocation scenarios, each has application-specific patterns.

regular curves of BT and MG suggest the need of adaptive scheduling inside the application for best performance.

Third, the newer memory architecture in the Haswell system does not reduce the power demand for the maximum performance. Meanwhile, the GPU power capping is intelligent by excluding low power capping settings.

6.3 The Accuracy of Heuristic Power Coordination

We now evaluate the category-based heuristic power coordination method **COORD** for CPU and GPU computing respectively. We compare **COORD** against two counterparts: the best found in the experimental dataset and the memory-first strategy allocating maximum power to power. The results are shown in Figure 9. Overall, **COORD** differs from the best by less than 5% for large power caps (preferred) and by 9.6% on average for all power caps for all CPU benchmarks, and less than 2% for GPU benchmarks. On the CPU system, **COORD** generally outperforms the memory-first strategy proposed in [19] for small power budgets. **COORD** carefully distributes the budgets to balance CPUs and memory, while the memory-first strategy conservatively allocates power to memory and distributes the remaining to the CPUs, aiming to avoid a larger performance degradation if memory is under-budgeted.

Given a power budget greater than applications' max power demand, **COORD** delivers the same performance or close to the best allocation for most of the cases. In addition, it only allocates to components adequate powers that are lower than those set in sweeping experiments. Algorithm **COORD** could further hint the system to redirect the budget surplus for other purposes. These results indicate that **COORD** is accurate under a practical power budget. NPB LU on the IvyBridge machine shows a data point where the heuristic method outperforms the best among experiments. This is due to two reasons. First, the sweeping uses a certain power stepping and thus does not necessarily include the distributed selected by the heuristic algorithm. Second, there is a small performance variation, i.e., $< 5\%$ among multiple runs for the same program.

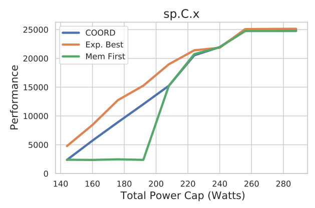
One noteworthy observation is that **COORD** outperforms the default Nvidia GPU power capping method by up to 33% for the applications under study. This is because **COORD** is aware of applications and available power budgets, while the default uses the same strategy to distribute power between GPU SMs and global memory. Specifically, it always runs memory at the nominal (the highest stable) speed, no matter what is the imposed total power cap or what application is running. Such obliviousness results in inferior performance and power budget wasted. **COORD** instead adjusts the memory power and speed according to applications' demands.

7 Related Work

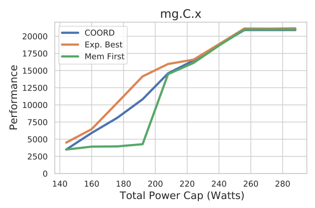
This work extends previous power-aware high performance computing to systems with limited power budgets. Power-aware computing adapts a component's performance-power state to meet the demand of workloads and reduce power with little performance impact. A majority of existing power-aware HPC research has been focused on effectively using power-aware components [15, 32] or controlling software execution by adjusting the degree of concurrency and the number of participating cores [24]. Although power-aware computing and power bounded computing employ common power management technologies, the former does not enforce a power bound.

Power budgeting and power capping have been extensively exploited in commercial data center power provisioning [36]. Effective power budgeting technologies use power aware components [15, 32], server consolidation [12], virtual machine placement/migration [36], and workload scheduling [5]. Recently, researchers have expanded the power budget idea into HPC and explored various methods that could enforce a power cap on HPC systems. These methods fall into three categories: (1) software control of the components' power states, for example, applying DVFS and clock-gating to control CPU power [6, 8, 17], or managing both processors and memory power [10, 20, 34]; (2) software control of workload execution like throttling or increasing the number of concurrent threads [7, 11, 27, 31]; and (3) hardware-based power-capping capability on CPUs [9, 13, 22] and GPUs [3, 28, 37]. In this work, we focus on the coordination of power capping on competing components for executing single jobs within one system.

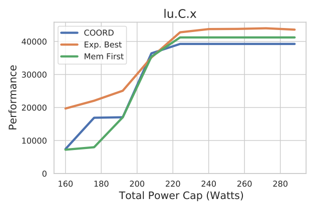
Several studies investigated the problem of optimal power allocation between cores and uncores [35], between CPU and memory [16, 18, 30, 33, 34], among CPU and accelerator [3, 7, 37]. Sarood et al. [30] use an interpolation method, which samples a moderate subset of power allocation points and identifies the optimal among all possible allocation points. They show capping both CPU and memory power could lead to a higher performance than



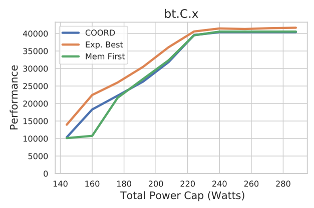
(a) NPB SP on IvyBridge



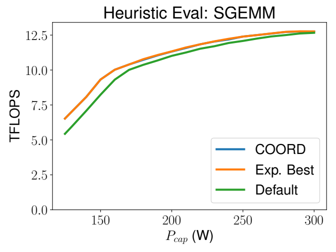
(b) NPB MG on IvyBridge



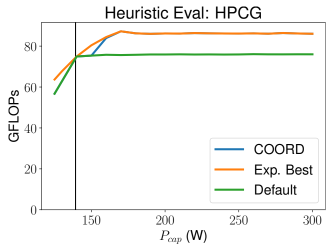
(c) NPB LU on IvyBridge



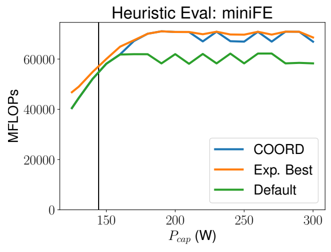
(d) NPB BT on IvyBridge



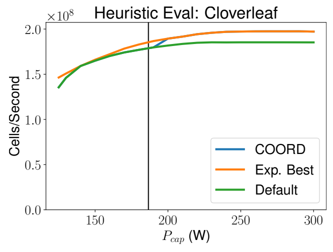
(e) SGEMM on Titan XP



(f) HPCG on Titan XP



(g) MiniFE on Titan XP



(h) Cloverleaf on Titan XP

Fig. 9. Comparison between **COORD** and the best identified from experiments on IvyBridge system and the Titan XP GPU system. The vertical lines in the GPU figures show the value of P_{totref}.

just capping CPU power. Motivated by the fact that different applications have different performance behaviors in a power-capped environment, Tiwari et al. [34] model the performance degradation caused by reduced CPU and DRAM power caps for various workloads represented with a set of static and dynamic program characteristics, which is obtained with binary code instrumentation and code analysis. More recently, researchers study job co-running on integrated CPU-GPU systems under power caps [3, 37], and performance and power modeling of CPU, GPU, and memory frequency scaling [7].

This work differs from the state-of-the-art that considers power as hard constraints and cross-component power allocations in HPC in several main aspects. First, it reveals and generalizes the categorical patterns. Such generalization is the first of its type capturing the dynamics of cross-component allocation and the performance impact. Second, it reveals the imposed power cap decides the maximum achievable performance, providing insights about the waste of excessive power budgeting. Third, it extends existing work in cross-component power coordination from CPU computing [19, 30, 33, 34] to GPU computing and details the similarity and differences. Fourth, it demonstrates the validity of lightweight power coordinations on multiple processor and memory technologies, advancing the state-of-the-art that requires extensive profiling [30] or instrumented detailed profiling [7, 34].

8 Conclusion

As HPC systems are increasingly bounded by power, they must cope with these power bounds. In this work, we study coordinated power allocation between processors and memory modules and its implications to power-bounded HPC systems. Our research reveals there exist categorical patterns in the dynamics between power allocation and application performance. These patterns can be used to design power coordination methods that optimally coordinate power across components and maximize application performance under given power budgets.

This study leads to several insights to power-bounded computing. First, the designated node-level power budget must be above certain threshold to deliver desirable application performance, and such threshold can be derived from components' critical power values corresponding to hardware power limiting mechanisms. Second, power allocation between components must be coordinated to maximize application performance and power efficiency. Third, node-level power coordination is key to higher level power-bounded scheduling by requesting and enforcing an appropriate power budget and returning the excessive budget to an upper lever scheduler.

This work focuses on dedicated computing environments for single jobs. We plan to extend this study to other heterogeneous systems such as big.LITTLE architectures and multi-task computing environments.

Acknowledgements

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

References

[1] Arrhenius Equation for Reliability. http://www.jedec.org/standards-documents/dictionary/terms/arrhenius-equation-reliability.

[2] HPE Has Constructed The Largest Single-Memory Computer System Ever Built. https://www.forbes.com/sites/aarontilley/2017/05/16/hpe-160-terabytes-memory/?sh=265e05a3383f.

[3] Powercoord: Power capping coordination for multi-cpu/gpu servers using reinforcement learning. *Sustainable Computing: Informatics and Systems*, (28):1–11, 2020.

[4] D. H. Bailey, E. Barszcz, J. T. Barton, et al. The NAS parallel benchmarks. *International Journal of High Performance Computing Applications*, 5(3):63–73, 1991.

[5] A. Banerjee, T. Mukherjee, G. Varsamopoulos, and S. Gupta. Cooling-Aware and Thermal-Aware Workload Placement for Green HPC Data Centers. In *Green Computing Conference*, pages 245–256, 2010.

[6] K. J. Barker, D. J. Kerbyson, and E. Anger. On the Feasibility of Dynamic Power Steering. In *Proceedings of the 2nd International Workshop on Energy Efficient Supercomputing*, 2014.

[7] T. Baruah. Energy efficient execution of heterogeneous applications. Master's thesis, Northeastern University, Boston, Massachusetts, 12 2017.

[8] J. M. Cebrián, J. L. Aragon, and S. Kaxiras. Power Token Balancing: Adapting CMPs to Power Constraints for Parallel Multithreaded Workloads. In *2011 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 431–442. IEEE, 2011.

[9] J. Chen, X. Qi, F. Wu, J. Fang, Y. Dong, Y. Yuan, Z. Wang, and K. Li. More bang for your buck: Boosting performance with capped power consumption. *Tsinghua Science and Technology*, 26(3):370–383, 2021.

[10] M. Chen, X. Wang, and X. Li. Coordinating Processor and Main Memory for Efficientserver Power Control. In *Proceedings of the international conference on Supercomputing*. ACM, 2011.

[11] R. Cochran, C. Hankendi, A. K. Coskun, and S. Reda. Pack & Cap: Adaptive DVFS and Thread Packing under Power Caps. In *Proceedings of the 44th annual IEEE/ACM International Symposium on Microarchitecture*, pages 175–185. ACM, 2011.

[12] R. Das, J. Kephart, C. Lefurgy, et al. Autonomic Multi-Agent Management of Power and Performance in Data Centers. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, pages 107–114. AAMAS, 2008.

[13] H. David, C. Fallin, et al. Memory Power Management via Dynamic Voltage/Frequency Scaling. In *Proceedings of the 8th ACM International Conference on Autonomic computing*. ACM, 2011.

[14] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le. RAPL: Memory Power Estimation and Capping. In *Proceedings of the 16th ACM/IEEE International Symposium on Low Power Electronics and Design*, ISLPED '10, pages 189–194, New York, NY, USA, 2010. ACM.

[15] Q. Deng, D. Meisner, A. Bhattacharjee, T. Wenisch, and R. Bianchini. CoScale: Coordinating CPU and Memory System DVFS in Server Systems. In *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 143–154, Dec 2012.

[16] Q. Deng, D. Meisner, A. Bhattacharjee, T. F. Wenisch, and R. Bianchini. Coscale: Coordinating cpu and memory system dvfs in server systems. In *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 143–154, 2012.

[17] M. Etinski, J. Corbalan, J. Labarta, and M. Valero. Parallel job scheduling for power constrained HPC systems . *Parallel Computing*, 38(12):615 – 630, 2012.

[18] K. Fukazawa, M. Ueda, Y. Inadomi, M. Aoyagi, T. Umeda, and K. Inoue. Performance analysis of cpu and dram power constrained systems with magnetohydrodynamic simulation code. In *2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 626–631, 2018.

[19] R. Ge, X. Feng, Y. He, and P. Zou. The case for cross-component power coordination on power bounded systems. In *2016 45th International Conference on Parallel Processing (ICPP)*, page 516–525, 2016.

[20] H. Hanson, W. Felter, W. Huang, C. Lefurgy, K. Rajamani, F. Rawson, and G. Silva. Processor-Memory Power Shifting for Multi-Core Systems. In *The 4th Workshop on Energy Efficient Design*, 2012.

[21] E. J. Hogbin. Acpi: Advanced configuration and power interface, 2015.

[22] Intel. Volume 3B: System Programming Guide, Part 2. *Intel 64 and IA-32 Architectures Software Developer's Manual*, June 2013.

[23] C. Lefurgy, X. Wang, and M. Ware. Power Capping: a Prelude to Power Shifting. *Cluster Computing*, 11(2):183–195, 2008.

[24] D. Li, B. R. de Supinski, M. Schulz, D. S. Nikolopoulos, and K. W. Cameron. Strategies for Energy Efficient Resource Management of Hybrid Programming Models. *IEEE Transactions on Parallel and Distributed Systems*, 24(1):144–157, 2013.

[25] X. Li, R. Gupta, S. V. Adve, and Y. Zhou. Cross-Component Energy Management: Joint Adaptation of Processor and Memory. *ACM Transactions on Architecture and Code Optimization (TACO)*, 4(3), Sept. 2007.

[26] 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 *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 213, 2006.

[27] A. Marathe, P. E. Bailey, D. K. Lowenthal, B. Rountree, M. Schulz, and B. R. de Supinski. A Run-time System for Power-constrained HPC Applications. In *High Performance Computing*, pages 394–408. Springer, 2015.

[28] T. Patki, Z. Frye, H. Bhatia, F. Di Natale, J. Glosli, H. Ingolfsson, and B. Rountree. Comparing gpu power and frequency capping: A case study with the mummi workflow. In *2019 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)*, pages 31–39. IEEE, 2019.

[29] V. Sarkar, S. Amarasinghe, D. Campbell, et al. Exascale Software Study: Software Challenges in Extreme Scale Systems. *DARPA Information Processing Techniques Office, Washington DC*, 14:159, 2009.

[30] O. Sarood, A. Langer, L. Kalé, B. Rountree, and B. De Supinski. Optimizing Power Allocation to CPU and Memory Subsystems in Overprovisioned HPC Systems. In *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER)*, pages 1–8. IEEE, 2013.

[31] A. Sharifi, A. K. Mishra, S. Srikantaiah, et al. PEPON: Performance-Aware Hierarchical Power Budgeting for NoC Based Multicores. In *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*. ACM, 2012.

[32] V. Sundriyal and M. Sosonkina. Joint Frequency Scaling of Processor and DRAM. *The Journal of Supercomputing*, pages 1–21, 2016.

[33] V. Sundriyal, M. Sosonkina, B. M. Westheimer, and M. Gordon. Maximizing performance under a power constraint on modern multicore systems. *Journal of Computational Chemistry*, 7:252–266, 2019.

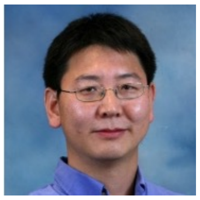
[34] A. Tiwari, M. Schulz, and L. Carrington. Predicting Optimal Power Allocation for CPU and DRAM Domains. In *Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International*, pages 951–959, May 2015.

[35] B. Wang, J. Miller, C. Terboven, and M. Müller. Operation-aware power capping. In *European Conference on Parallel Processing*, pages 68–82. Springer, 2020.

[36] X. Wang and Y. Wang. Coordinating Power Control and Performance Management for Virtualized Server Clusters. *Parallel and Distributed Systems, IEEE Transactions on*, 22(2):245–259, 2011.

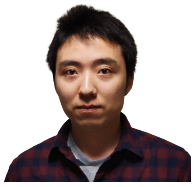
[37] Q. Zhu, B. Wu, X. Shen, L. Shen, and Z. Wang. Co-run scheduling with power cap on integrated cpu-gpu systems. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 967–977, 2017.

Rong Ge received the BS and MS degrees in Engineering Mechanics from Tsinghua University in 1995 and 1998 respectively, and the PhD degree in Computer Science at Virginia Tech in 2007. She is the director of the Scalable Computing and Analytics Laboratory in the School of Computing at Clemson University. Her research interest includes parallel and distributed systems, machine learning and big data, heterogeneous computing, and performance evaluation and optimization.



Xizhou Feng received his PhD degree in Computer Science from University of South Carolina and his JD degree from Marquette Law School. He is an HPC system scientist and research associate professor at Clemson University. His research interests include high performance computing and cyberinfrastructure, scalable algorithms, computational sciences and informatics, complex system modeling, and the interactions between technology and law.

Tyler Allen is a Ph.D. candidate in the Computer Science division of the School of Computing at Clemson University. 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.



Pengfei Zou received the BS degree in Remote Sensing Science and Technology from Wuhan University in 2012, the MS degree in Geographical Information System from University of Chinese Academy of Sciences in 2015, and the PhD degree in Computer Science at Clemson University in 2020. His research interest includes parallel and distributed systems, heterogeneous computing, remote Sensing and cross-discipline technologies between GIS and CS.