Scientific Research Publishing

Journal of Computer and Communications, 2020, 8, 142-158
https://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

# Distributed Strategy for Power Re-Allocation in High Performance Applications

**Vaibhav Sundriyal, Masha Sosonkina**

Department of Computational Modeling and Simulation Engineering, Old Dominion University, Norfolk, Virginia, USA
Email: vsundriy@odu.edu, msosonki@odu.edu

**How to cite this paper:** Sundriyal, V. and Sosonkina, M. (2020) Distributed Strategy for Power Re-Allocation in High Performance Applications. *Journal of Computer and Communications*, **8**, 142-158.
https://doi.org/10.4236/jcc.2020.812014

**Received:** September 2, 2020
**Accepted:** December 22, 2020
**Published:** December 25, 2020



## Abstract

To improve the power consumption of parallel applications at the runtime, modern processors provide frequency scaling and power limiting capabilities. In this work, a runtime strategy is proposed to distribute a given power allocation among the cluster nodes assigned to the application while balancing their performance change. The strategy operates in a timeslice-based manner to estimate the current application performance and power usage per node followed by power redistribution across the nodes. Experiments, performed on four nodes (112 cores) of a modern computing platform interconnected with Infiniband showed that even a significant power budget reduction of 20% may result in a performance degradation of as low as 1% under the proposed strategy compared with the execution in the unlimited power case.

## Keywords

Multinode Power Allocation, RAPL, UFS, DVFS, Maximizing Performance, Component Power

## 1. Introduction

Power and the subsequent energy consumptions pose major challenges in designing large-scale systems. With the advent of exascale machines, efficient power use, at both software and hardware levels, has become imperative for managing operating costs and failure rates. Maximizing the performance-per-watt value for multinode architectures means that the available power should be appropriately distributed among nodes for optimal performance. All modern exascale computing platforms feature multinode configuration. For example, one of the world fastest supercomputers Summit has more than 4600 compute nodes, each containing several processors (CPUs) and graphics processing units (GPUs) [1]. During computation, CPUs offload highly parallelizable code segments to GPUs

for further acceleration.

In order to operate a system under a given power allocation, it should have components with a capability to regulate their individual power consumption. For example, modern generations of Intel processors enable dynamic voltage and frequency scaling (DVFS) and throttling (idle cycles) to control the power consumption of the processor. Additionally, Intel processors since the Sandy Bridge model have capabilities for both on-board power meters and power clamping through the Intel Running Average Power Limit (RAPL) [2] interface. Beginning from the Intel Haswell microarchitecture, the core and uncore processor subsystems have been decoupled into PP0 and PP1 domains, respectively, so that the uncore frequency can be modified independently of the core frequency in the entire package (PKG) domain. Note that the uncore subsystem contains "non-computational" processor functions, such as quick path interconnect (QPI) controllers, L3 cache, and on-die memory controller.

This work proposes a runtime strategy that redistributes a given power allocation among the compute nodes on which an application runs in a cluster such that the application experiences a minimal performance degradation. The strategy operates in a manner transparent to the application and utilizes a timeslice based approach to periodically monitor application performance and power usage per node. The strategy equalizes power usage among the participating nodes after each timeslice. Note that the network components have not been considered in this work as power re-allocation in the switch for multinode exascale systems is not possible whereas the local network interface card, such as the Infiniband host channel adapter (HCA), has minimal power consumption. In a nutshell, the contributions of this work include

- Proposed a runtime strategy that distributes a given power budget among nodes such that the performance change per node is equalized.
- Developed an algorithm that implements the strategy and chooses appropriate operating frequencies to accommodate the redistributed power budget.
- Shown that the developed algorithm has a low parallel overhead incurred by its communication mechanism proposed here.
- Established a novel usage of the performance and power models to equalize the performance change due to varying workloads per node in distributed computing environments.

The rest of the paper is organized as follows. Section 2 provides the related work. Section 3 discusses the hardware capabilities of the CPU devices used in this work with respect to the frequency scaling and power capping. Section 4 deals with power and performance models utilized in the runtime strategy. Section 5 provides details of the implementation of the runtime strategy. Section 6 shows experimental results and Section 7 concludes the paper.

## 2. Related Work

Much research has been already conducted to measure, model, and budget power on computer components and systems. For budgeting the power in modern

computing systems, the strategies primarily come in two forms: 1) DVFS and/or CPU throttling for processor and memory and 2) hardware-enforced power bounds using RAPL [2]. The work in [3] proposes a multi-input multi-output (MIMO) power control algorithm for distributing a specified power budget between the PKG and DRAM domains to maximize the application performance. Machine learning to determine the sensitivity of application performance with respect to PKG and DRAM power capping was studied in [4] to devise a strategy for power budgeting. A strategy termed *conductor* was proposed in [5] that, given a power budget, utilized configuration space exploration and adaptive power balancing for maximizing application performance. The work [6] proposes a multilevel power distribution framework termed *CLIP* that estimates a per-node power cap by utilizing the workload characteristics and memory accesses to determine processor and memory affinity.

An interpolation method is proposed in [7] that characterizes the effects of strong scaling on an application with varying power distribution between processor and memory. The method is then tested in an overprovisioned system to obtain significant speedups compared to an uncapped system. The work in [8] discusses a hardware-level power capping strategy for limiting DRAM power consumption. Authors in [9] propose a design of a power scheduler capable of enforcing power bounds by using dynamic system-wide power reallocation. Authors in [10] propose Uncore Power Scavenger, a runtime system that dynamically detects phase changes in application and determines the best uncore frequency for every phase to save power without significant impact on the performance. [11] proposes a daemon process DUF, which dynamically adapts the uncore frequency to reduce an application power consumption with a user-given performance degradation. Authors in [12] develop a multi-domain power management framework SysScale to improve the energy efficiency of mobile devices by using DVFS and domain specific parameters. Although [10] [11] [12] consider the uncore domain, they do so by relying on heuristics rather than on specific performance and power models as done in the present work. In addition, the uncore domain is investigated in the multinode setting here.

## 3. CPU and Communication Capabilities for Power Reallocation

The components of an Intel processor can belong either to the "core" or the "uncore" subsystem in a chip. The core subsystem consists of compute cores along with the caches (L1 and L2). The uncore subsystem comprises the rest of the components such as L3 cache, memory controller, and the Intel Quickpath interconnect (QPI). Prior to the Intel Haswell generation, the core and uncore subsystems ran at the same frequency. Since the Haswell generation, the Intel processors have separate frequency domains for the core and uncore subsystems. In general, the Intel Running Average Power Limit (RAPL) interface [2] provides model-specific registers (MSR)'s containing energy consumption estimates for up to four power domains of a machine as follows:

- PKG: for the entire package,
- PP0: for the core subsystem,
- PP1: for the uncore subsystem (available in client-type platforms only, mainly used for general-purpose applications),
- DRAM: main memory (available in server-type machines only).

Intel has also introduced a mechanism, termed uncore frequency scaling (UFS), which allows the user to change the uncore frequency on-the-fly. A procedure of modifying the uncore frequency at the runtime via the MSR 0x620 has been described in authors' previous work [13]. Note that, a UFS procedure has to be used to adjust the uncore power consumption on the Intel server-type platforms since, at present, they do not have a dedicated interface to the PP1 (uncore) power domain as opposed to the PP0 (core) one. For the work in this paper, only the server-type platforms have been considered since they enable power adjustments of DRAM, which is critical for the proposed strategy and efficient application execution, in general as explained in Section 4. In a nutshell, the proposed strategy first—similarly to the work in [14]—aims at shifting power between the PP0 and PP1 domains in a node once the power needs of the DRAM domain are fulfilled. Next, the power re-allocation is performed across the nodes using the platform communication subsystem, which is typically Infiniband [15] in modern computing platforms. Infiniband features high-bandwidth low-latency interconnect across which it provides direct memory-to-memory transfers with no intermediate buffering or copying. The latest Infiniband type, EDR—which is used here—features bandwdith and latency of 100 Gb/s and 0.5 μs, respectively. It also has kernel bypass, which facilitates direct user-level access to the hardware Host Channel Adapter (HCA) with no CPU intervention during data transfers.

## 4. Intranode Performance and Power Modeling

In general, an application cannot be either compute- or memory-intensive in its entirety especially since the workload behavior of an application varies throughout its execution. Hence, for effective distribution of the given power budget, a fine-grained performance and power modeling approach is needed, which considers small segments of execution, termed timeslices. Timeslices of equal duration on the order of frequency scaling overhead, such as 250 ms intervals, have proven to be a good choice in the earlier work (see e.g., [16]) and are considered here for power allocation. Without a timely detection of memory- or compute-intensive workload, the application performance degradation may increase way beyond the 10% of a commonly acceptable performance loss for energy-saving purposes [17]. For example, a more than 600% increase in the execution time was observed for a memory-intensive NAS parallel benchmark CG when the DRAM power was clamped below its maximal usage of 8 W [18]. Due to a possibility of such a disastrous effect on the performance from reducing the DRAM power allocation, the DRAM domain should given the highest priority in

power allocation among all the components/domains within a node once the power budget for that particular node has been determined. Then, the remaining power budget is apportioned within PKG (core and uncore components) in accordance with performance and power models developed by the authors earlier in [14]. A similar power redistribution has been performed recently in the work of Gholkar *et al.* [10]. For the sake of completeness and reference, the rest of Section 4 reproduces nearly verbatim the performance and power models from [14], which are used as a backbone in the runtime strategy described in Section 5.

### 4.1. Performance Model

The model in Equation (1) correlates the application performance, expressed in micro-operations retired, with particular core $f_c(i)$ and uncore $f_u(j)$ frequencies expressed by their corresponding levels, from the highest to lowest, $i, i=1,\cdots,N$ and $j, j=1,\cdots,M$.

$$\mu\tau(i,j)=\frac{f_c(i)}{\mathrm{CPM}_{\mathrm{exe}}+\frac{f_c(i)}{f_c(1)}\left(\mathrm{MAPM}\times\alpha\times\beta_j\right)}, \tag{1}$$

where

$\mu\tau(i,j)$ is the number of micro-operations retired per second at core frequency $f_c(i)$ and uncore frequency $f_u(j)$.

$\mathrm{CPM}_{\mathrm{exe}}$ is the number of cycles per micro-operation retired barring the memory accesses in a second.

$\alpha$ ($0\le\alpha\le1$) is the out-of-core (OOO) overlap factor, which determines the extent of the memory access stalls overlapped with the execution cycles.

MAPM is the number of memory accesses per micro-operation retired in a second.

$\beta_j$ is the number of cycles corresponding to the memory access latency at the uncore frequency $f_u(j)$.

### 4.2. Power Model

The processor power consumption, denoted $P_T(i,j)$, can be expressed as:

$$P_T(i,j)=P_s+k_1\times f_c(i)^3+k_2\times f_u(j)^3+k_3, \tag{2}$$

where $k_1$, $k_2$, and $k_3$ are constants and $f_c(i)$ and $f_u(j)$ are the core and uncore frequencies, respectively. $P_s$ stands for the processor static power consumption, which was measured as 134 W through RAPL. Since uncore (PP1) power limiting is not supported in Intel server processors, the power model in Equation (2) is required to relate the power consumption of core/uncore domains to the corresponding levels of core/uncore frequencies. Parameters $k_1$ and $k_2$ were determined by a regression analysis of the processor power obtained through the RAPL registers at different core and uncore frequencies for several benchmarks. The values $k_1$ and $k_2$ were found to be 0.94 and 0.48,

respectively, indicating that changes in the core frequency affect the processor power consumption more than those in the uncore frequency do so.

Given a power budget for the three domains—PP0, PP1, and DRAM—in a server-type platform, the shifting of power between the core and uncore domains is essentially done by first modifying the uncore frequency and then shifting the corresponding reduction in power to increase the power limit for the core domain to maximize the performance. Equation (3) shows how the power is transferred to the core domain (within the PKG domain) through UFS:

$$P_{\text{PKG}} = \left(P_B - P_{\text{RAPL-MEM}}\right) + \left(P_T\left(1, j_1\right) - P_T\left(1, j_2\right)\right). \quad (3)$$

Specifically, Equation (3) sets the PKG power allocation $P_{\text{PKG}}$ as the sum of the total power budget $P_B$ minus the DRAM power consumption $P_{\text{RAPL-MEM}}$ and the difference in processor power consumption when the uncore frequency is switched from level $j_1$ to $j_2$. In this manner, the reduction in power obtained through UFS is transferred to the PKG power allocation to increase the core frequency. It can be noted here that this work considers the power budgeting for only PP0, PP1, and DRAM domains rather than that for the entire node because it is assumed here that the application execution does not concern the power consumption of the other node components.

## 5. Design and Implementation of the Runtime Strategy

This section first outlines the proposed novel runtime strategy for power allocating among cluster nodes executing a high-performance application. Then, it describes in detail the algorithmic steps implementing the strategy.

The proposed strategy employs a hierarchical approach for maximizing performance of a parallel application on a cluster. First, it divides equally the given power budget among the distributed nodes. Then, within a node, it allocates the power among three power domains, such as core, uncore, and DRAM. In each timeslice, the strategy gathers performance counter information for each participating core. Then, by utilizing the established power and performance models, the strategy determines the maximal core and uncore frequencies for each node. Next, the performance change in each node is determined for the case when the core/uncore frequency may be switched to these maximal frequency admissible by the given power budget. The average of these performance changes is considered in shifting the power allocation across the nodes to the extent that the performance changes become similar. Such a *performance-difference based* power allocation essentially means that application threads within a node will receive a power budget according to their joint workload behavior and that none of the application nodes will fall behind due to a lack of power allocation.

**Communication mechanism.** To implement the proposed strategy, several processes are spawned such that their number is equal to the number of sockets across the nodes involved in executing the application. Note that the strategy processes share the cores with the application threads, which is acceptable due to the low parallel overhead of the strategy implementation (shown in Section 6

experiments). The message passing interface (MPI) was chosen for the implementation and its processes assigned ranks in a standard fashion with rank 0 being a manager process. Figure 1 presents a sample scenario of $K$ application nodes. Each of the $K$ nodes has two sockets (shown as rectangular boxes) and four cores per socket. The cores are numbered contiguously across all the $K$ nodes, from 0 to $8K-1$. The placement of strategy processor ranks is highlighted in color: pink for the worker processes and cyan for the manager. The strategy communications are indicated by arrows. The first step (see Figure 1(a)) is intranode and involves the higher worker rank sending its information to the lower one. In the second step (see Figure 1(b)), all the lower worker ranks in a node communicate their information to the manager rank 0 a single MPI_Gather from all the $K$ nodes.

**Algorithm detailed.** Figure 2 specifies the algorithmic steps of the proposed strategy executed in each rank while a high-level overview of the strategy is given in Figure 3 to emphasize its distributed nature. In Figure 2, Step 1 divides a given power allocation $P_B$ equally across all $K$ nodes and between PKG and DRAM domains in a node, *i.e.*, among $2K$ entities. In Step 2, each rank profiles the application for the first timeslice duration and obtains the relevant parameter values from the performance counters. In Step 3, the operating core frequency $f_c(o_c)$ is determined by using the APERF and MPERF MSRs [19] according to the following relation, which has been first stated in [14]:

$$f_c(o_c) = f_c(1) \times \frac{\Delta\text{APERF}}{\Delta\text{MPERF}}, \tag{4}$$

where $\Delta$APERF and $\Delta$MPERF signify the change in the values of the respective registers over a given time period. Next, from performance counters, Step 4 initializes the micro-operations retired $\mu\tau(o_c, o_u)$, at the operating core frequency $f_c(o_c)$ and the highest uncore frequency $f_u(1)$ (*i.e.*, $o_u = 1$) for the first timeslice of the application execution. The corresponding $\text{CPM}_{\text{exe}}$ is calculated from Equation (1) as follows:

$$\text{CPM}_{\text{exe}} = \frac{f_c(o_c)}{\mu\tau(o_c, 1)} - \frac{f_c(o_c)}{f_c(1)} \times \alpha \times \text{MAPM} \times \beta_1, \tag{5}$$

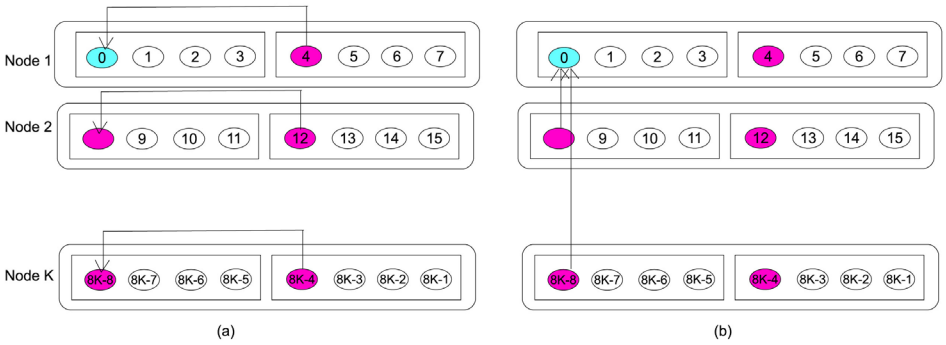


Figure 1. Rank assignment and communications of the strategy implementation in a sample $K$ application node scenario. Worker ranks communicating information: (a) Intranode communications. (b) Internode collective communication with the manager rank 0.

**Input Parameters:**
$K$ : Number of nodes
$V$ : Number of timeslices
$f_c(1), \ldots, f_c(N)$ : Available core frequencies ($N$)
$f_u(1), \ldots, f_u(M)$ : Available uncore frequencies ($M$)
$P_B$ : Total power budget for PKG and DRAM on $K$ nodes
$P_{PKG}$ : Package power limit
$P_{MEM}$ : Memory power limit
$P_{RAPL-PKG}$ : PKG power consumption
$P_{RAPL-MEM}$ : Memory power consumption
CPR, MPR : $\text{CPM}_{\text{exe}}$ and MAPM registers, respectively, of length $L = 3$ each
$\ell = 1$ : counter for filled positions in CPR and MPR

**Algorithm:**
**Step 1.** Set package and memory power limits as $P_{PKG}$=$P_{MEM}$=$P_B/(2K)$
**Step 2.** Sleep while application executes during timeslice $r = 1$
**Step 3.** Calculate operating core frequency $f_c(o_c)$ from APERF and MPERF as in Eq. (4);
Set operating uncore frequency level $o_u = 1$
**Step 4.** Initialize $\mu\tau(o_c, o_u)$, CPR[$\ell$], and MPR[$\ell$] for $r = 1$ from performance counters and Eq. (5)
**For** ($r = 2$, $r \leq V$, $r++$) **do**
**Step 5.** Calculate $\text{CPM}_{\text{exe}}$ and MAPM as
If ($\ell > 0$ and $\ell \leq L - 1$) then
$\text{CPM}_{\text{exe}}$ = CPR[$\ell - 1$]
MAPM = MPR[$\ell - 1$]
else
$\text{CPM}_{\text{exe}}$ = avg(CPR[1], CPR[2],...,CPR[$L$])
MAPM = avg(MPR[1], MPR[2],...,MPR[$L$])
**Step 6.** Calculate $\mu\tau(i, j)$ for all $i = 1, \ldots, N$ and $j = 1, \ldots, M$
**Step 7.** Calculate temporary core $f_c(t_c)$ and uncore $f_u(t_u)$ frequency such that
$\mu\tau(t_c, t_u) = \max_{\substack{i=1,\ldots,N \\ j=1,\ldots,M}} [\mu\tau(i, j)]$ and $P_{RAPL-PKG} \leq \frac{P_B}{K} - P_{RAPL-MEM}$
**Step 8.** Determine the performance change $\theta$ per Eq. (6)
**Step 9.** Gather $\theta$'s at manager rank 0
**Step 10.** Rank 0 determines the average performance change $\theta_{av}$ across all $\theta$'s and broadcasts $\theta_{av}$
**Step 11.** Save operating uncore frequency level $w_u = o_u$;
Determine new operating core–uncore frequency pair ($f_c(o_c), f_u(o_u)$) per Eq. (7) and Eq. (1)
**Step 12.** Set $P_{MEM}$=$P_{RAPL-MEM}$ and $P_{PKG}$ from Eq. (3) with $j_1 = \omega_u$ and $j_2 = o_u$
**Step 13.** If ($\ell == L$) then
Shift CPR and MPR left by one position
$\ell = 0$
**Step 14.** Sleep while application executes duirng current timeslice $r$
**Step 15.** Update $\mu\tau(o_c, o_u)$, CPR[$\ell$], and MPR[$\ell$] from performance counters and Eq. (1);
$\ell = \ell + 1$
**EndFor**

**Figure 2.** Per-rank algorithmic steps of the proposed distributed strategy.

and MAPM is obtained directly from the processor performance counters. Note that Equation (5) has been developed in the authors' prior work [14]. For $r > 1$, Step 5 determines the values of $\text{CPM}_{\text{exe}}$ and MAPM through the history-window prediction mechanism [14] by using a simple averaging function, which calculates the future value as an average of the past values. If the registers CPR and MPR have not been completely filled, then the last values of $\text{CPM}_{\text{exe}}$ and MAPM are used as the next values. In Step 6, $\mu\tau(i, j)$ is determined for all of the available core and uncore frequencies using the values of $\text{CPM}_{\text{exe}}$ and MAPM from Step 5. Next (Step 7), a temporary core-uncore frequency pair $(f_c(t_c), f_u(t_u))$ is determined in each node, such that the predicted $\mu\tau(t_c, t_u)$ is at a maximum with the PKG power consumption not exceeding $\frac{P_B}{K} - P_{\text{RAPL-MEM}}$ in a node. In Step 8, the performance change value, termed $\theta_q$, is calculated in each node $q = 1, \cdots, K$ as

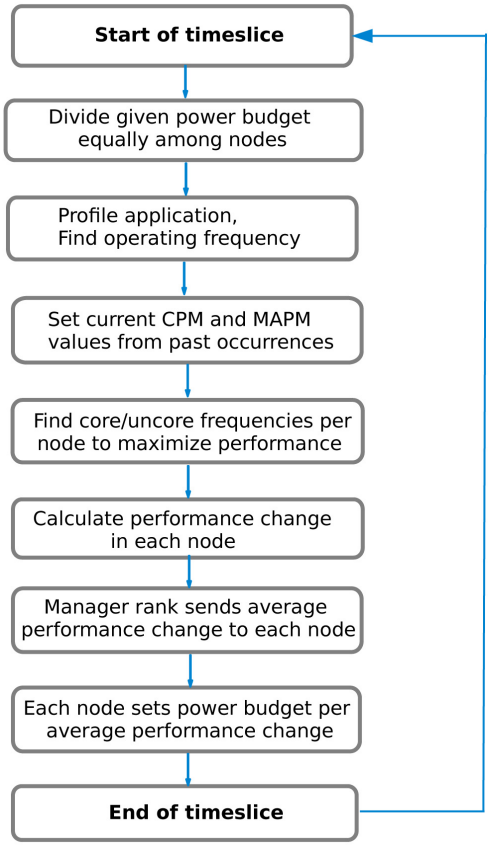


Figure 3. Overview of the proposed distributed strategy that considers application execution time as $V$ consecutive timeslices.

$$\theta_q = \frac{\mu\tau(t_c, t_u) - \mu\tau(o_c, o_u)}{\mu\tau(o_c, o_u)}. \tag{6}$$

In Step 9, worker ranks send their respective performance change values to the manager rank 0. Next, the manager rank calculates their average $\theta_{av}$ across all the $\theta_q$'s received (Step 10). The rationale behind averaging the performance change across all the nodes is to keep all the application threads running in tandem when the power budget changes in a node. This is especially beneficial for that applications the workload behavior of which differ greatly across the nodes. The manager rank then broadcasts the $\theta_{av}$ to all the nodes. In Step 11, by substituting $\theta_{av}$ into $\theta_q$ in Equation (6), each node determines the value of the predicted micro-operations retired at the core and uncore frequencies that balance the performance per node as follows:

$$\widetilde{\mu\tau} = \mu\tau(o_c, o_u)(\theta_{av} + 1). \tag{7}$$

The base performance model (Equation (1)) is then utilized to determine the new operating core-uncore frequency pair $\left(f_c(o_c), f_u(o_u)\right)$ in the following way. First, the left-hand side (LHS) in Equation (1) is set to $\widetilde{\mu\tau}$ from Equation (7). Next, the right-hand side (RHS) is evaluated for all the combinations of the available core and uncore frequency levels. Finally, the RHS value closest to the

LHS is taken to provide the new operating core-uncore frequency level pair $(o_c, o_u)$, from which the corresponding frequencies $f_c(o_c)$ and $f_u(o_u)$ are determined.

In Step 12, the power limit for DRAM is set as the measured DRAM power consumption, while the PKG power limit is set as in Equation (3). In Step 13, if the CPR and MPR registers are completely filled, they are shifted left by one to discard the old values. In Step 14, the application executes the current timeslice $r$ at the chosen PKG and DRAM power limits and the frequencies chosen in Step 7. Step 15 prepares for the next timeslice by updating the values of $\mu\tau(o_c, o_u)$ and MPR[$\ell$] from the performance counters and CPR[$\ell$] is calculated by rearranging Equation (1) with the newly set $(o_c, o_u)$, similarly to the procedures in Step 4.

MPI barrier operations were used after specific steps which require synchronization among the ranks on different nodes. The first barrier operation comes after Step 4 since the ranks need to synchronize after the first timeslice Then, the next barrier operation is needed after Step 10 because all the ranks have to receive their average performance change values before the new core-uncore frequency can be determined. Finally, a barrier operation is needed after Step 12 when all the ranks are synchronized after setting the appropriate power limits and frequencies. The communication overhead of the strategy involves an MPI_Gather (Step 9) and MPI_Broadcast (Step 10) operation per timeslice For a total of $K$ nodes and $S$ sockets per node, the time complexity of the two operations is roughly $\log_2 2K$. A detailed experimental analysis of the overall overhead incurred by the strategy is provided in Section 6.

## 6. Experimental Results

The experiments were performed on four compute nodes of the Turing cluster at Old Dominion University. Each node has two Intel Xeon E5-2695 v3 14-core Haswell-EP processors and 32 GB (4 × 8GB) of DDR4. The core and uncore frequency ranges are 1.2 - 2.3 GHz and 1.1 - 2.9 GHz, respectively. The processor supports 12 levels of core frequency and 19 levels of uncore frequency. Hence 228 level combinations are considered in Step 11 of the algorithm in **Figure 2**.

NAS parallel benchmark suite (NPB) [20] and a package for electronic structure calculations GAMESS [21] were used for evaluating the efficacy of the proposed runtime strategy. Three chosen class D benchmarks, EP.D, BT.D, and FT.D, from NPB provide a good mix of compute- and memory-intensive benchmarks for thorough testing of cluster and node power budgeting, while GAMESS features real-world calculations that are performed by the state-of-the-art algorithms and implementations [22]. A wide range of quantum chemistry computations may be accomplished using GAMESS: Starting from the basic Hartree-Fock Self Consistent Field (SCF) calculations, further improved on by *electron-correlated* methods, such as second-order Mø ller-Plesset perturbation theory (MP2) and Density Functional Theory (DFT) computations, and provid-

ing high-accuracy multi-reference and coupled-cluster (CC) computations. Recent advancements in GAMESS [22] provide LibCChem (C++ CPU/GPU library), fragmentation methods such as the fragment molecular orbital, effective fragment potential and effective fragment molecular orbital methods, hybrid MPI/OpenMP approaches to Hartree-Fock, and resolution of the identity second order perturbation theory. Two GAMESS calculations were chosen. The first is a self consistent field (SCF) computation on an adenosine monophosphate molecule using the cc-pVDZ basis set. The second one is more advanced and resource-intensive. It performs an MP2 calculation on 16 water molecules at the MP2/6-31G(d,p) level of theory. These calculations are referred to as scf-camp and h2o-16, respectively, in the rest of the paper.

To compare the performance of the five chosen inputs under the proposed strategy, a baseline execution mode that does not restrict power consumption, termed all high, was considered here. In the all high mode, the frequency and power limits for all CPU components were raised to their maximum values. A most commonly used power allocation strategy is the one that allocates power among the components based on their respective thermal design power (TDP) [23] limits. For this strategy, termed naïve here, a particular power allocation is observed with RAPL, which mainly reduces the core frequency through DVFS. When the power consumption cannot be reduced by RAPL any further down to the desired limit, a manual lowering of the uncore frequency follows.

Table 1 shows the average power consumption in the all high mode for the five inputs run on the Turing cluster. It can be observed from Table 1 that the compute-intensive inputs, EP and the two GAMESS calculations, as judged by the MAPM value, consume much less power on average compared with the BT.D and FT.D, which are more memory intensive. (This phenomenon has been also observed by the authors [16] where compute- and memory-intensive applications were compared.) In particular, the average power consumption ranges greatly, from 170 to 240 W. Hence, no single power allocation value (set) will accommodate accurate testing of the proposed strategy on all the inputs here. Instead, for each input, the power budgets are chosen as certain percentages of its average power consumption (Table 1). Specifically, the values of 90%, 80%, and 70% were considered here, representing a progressive decrease in the power consumed by the inputs in the all high mode.

If the execution time $T$ is given (as inverse of the application performance), then the percentage $\delta_s$ of performance degradation of a strategy s relative to the all high mode may be calculated as

**Table 1.** Average power consumption and MAPM values of NPB and GAMESS inputs for the allhigh mode.

| | EP.D | FT.D | BT.D | scf-camp | h2o-16 |
|---|---|---|---|---|---|
| Power, (W) | 170 | 240 | 220 | 160 | 150 |
| MAPM, ($\times 10^{-6}$) | 1 | 280 | 248 | 35 | 27 |

$$\delta_s = \frac{T(s) - T(\text{allhigh})}{T(s)}. \quad (8)$$

Figure 4 shows the performance degradation percentage $\delta_{\text{naive}}$ for the five inputs operating under the naïve strategy for the three chosen power budgets with respect to the all high mode. As expected, for all the inputs, the highest power budget of 90% results in the least amount of performance degradation, of 46.8% on average. When the power budget is at 80% level, the average performance degradation for the five inputs increases to 57.3%. This increase came primarily from the reduction in the CPU frequency to its lowest (1.2 GHz) using power limiting through RAPL. Further reduction in the power budget to the 70% level, degrades the performance further, to an average of 60%, which is due to the uncore frequency being reduced to 1.1 GHz to force this power level in addition to the RAPL power limiting. The largest degradation is seen for the memory-intensive FT.D and BT.D inputs. Uncore frequency has no effect on the compute-intensive EP.D benchmark and effects only minimally GAMESS, which was also noted in [14].

Figure 5 shows the performance degradation percentage $\delta_{\text{proposed}}$ for the five inputs operating under the proposed strategy with respect to the all high mode. The average performance degradation is 4.3% for the 90% power budget while a very little degradation of 0.15% is observed for the EP benchmark. This remarkable occurrence is due primarily to ability of the proposed strategy to determine that the uncore power is not being used by the compute-intensive EP.D and make this power available it to the cores. Specifically, for EP.D the uncore frequency is scaled down to 1.1 GHz throughout the execution and, after the initial step, when the strategy algorithm equalizes the budget among the nodes, no other transfer of the power budget among the nodes takes place. This essentially

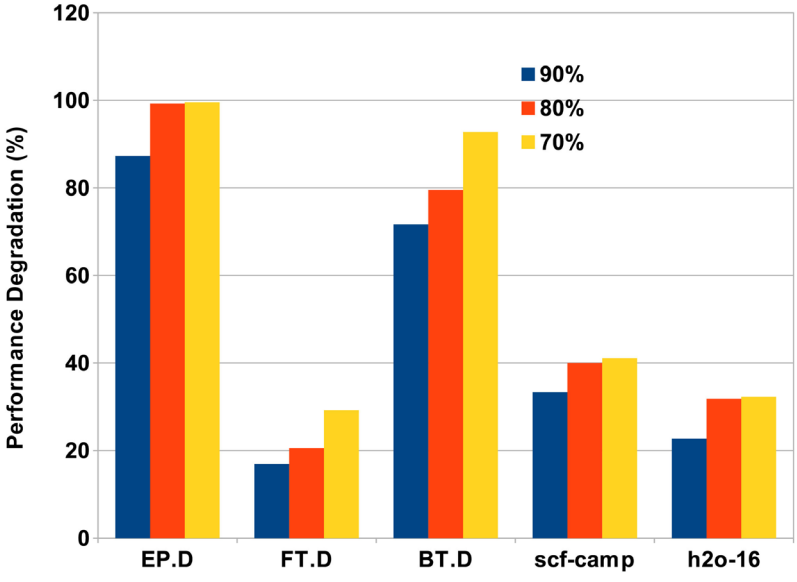


Figure 4. Performance degradation with respect to the allhigh mode for the five inputs operating under the naïve strategy and power budget levels of 90%, 80%, and 70%.

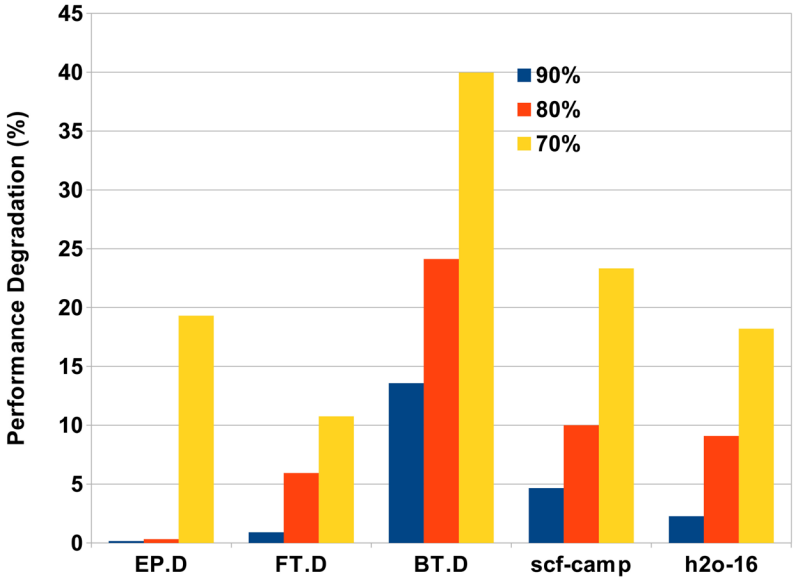


Figure 5. Performance degradation with respect to the allhigh mode for the five inputs operating under the proposed strategy and power budget levels of 90%, 80%, and 70%.

means that while Steps 8 - 12 from Figure 2 execute for EP.D, no difference in values of $\theta_{av}$ are observed after the first iteration the algorithm.

In the case of BT.D and FT.D (Figure 5), their MAPM values vary greatly, each on the order of 20 and 5, respectively. The BT.D benchmark alternates swiftly between compute- and memory-intensive behavior, which does not allow shifting of power from uncore to core while also restricting the ability of the proposed strategy to timely react to the switches between compute- and memory-intensive workloads. This, in turn, results in degrading significantly its performance. For BT.D, the core and uncore frequencies are pulled down to 1.5 GHz and 1.6 GHz, respectively. Conversely, FT.D remains primarily memory-intensive, thereby allowing the proposed strategy to scale the core frequency down to 1.4 GHz for a good portion of its execution. The transfer of power among the nodes takes place several times for BT.D and FT.D inputs because application threads in different nodes tend to behave differently and equalizing performance change across those requires that. Specifically, the strategy algorithm redistributed power among nodes 7 and 10 times for FT.D and BT.D inputs, respectively, during their execution for either of the three power-allocation levels. Given the 90% power budget, the performance degradations are 0.91% and 13.56%, for FT.D and BT.D, respectively.

The scf-camp and h2o-16 inputs remain largely compute-intensive with some heavy I/O activity in-between, which pushes the power consumption to the system idle power, at which point any change in frequency is not reflected in the power consumption. For the most part, these inputs execute at the lowest uncore frequency and incur the performance degradation of 4.5% for scf-camp and 2.3% for h2o-16. The average performance degradation under for the 80% budget is 9.87% as seen in Figure 5, which is still relatively low because the proposed

strategy is still able to find opportunities for uncore and core frequency scaling for the compute- and memory-intensive inputs, respectively, as expected. Note that, similarly to the EP.D benchmark execution, no power allocation was transferred across the nodes running the two GAMESS inputs.

When the power budget is at its lowest (70% here), more reductions in power start to come from the excessive core frequency scaling, which much degrades the performance of the compute-intensive inputs. In particular, EP.D suffers from the degradation of 19.4%. However, even in this case of a very low power allocation, the proposed strategy outperforms the naïve strategy by incurring an average performance degradation of 22.25% for all the inputs, which is about 2.7x less than the penalty incurred by the latter. Moreover, this large performance degradation of 22.25% is still smaller than an average degradation incurred by the naïve for either 90% or 80% levels. Comparing the average $\delta_{\text{proposed}}$ and $\delta_{\text{naive}}$ within in each power level of 90% and 80%, the former is about 10.8x and 5.8x better, respectively, than the latter.

**Power allocation without performance degradation.** The proposed strategy may be also used to optimize the power allocation while maintaining a given performance. In Table 2, such an optimal power budget is provided—as percentage of the power consumed in the all high mode—when now performance degradation incurred. Notice that the EP.D benchmark tolerates the most reduction in the budget (15%) without sacrificing any performance. This is because the reduction in the uncore frequency does not hurt its performance. The BT.D input, on the other hand, admits only 4.7% of the reduction in power because its workload characteristics oscillate much between compute and memory intensity. Recall (Figure 5) that the performance degradation of BT.D was the largest among the five inputs under the proposed strategy across all three power allocation levels. Overall, on average across all the five inputs, the utilization of the proposed strategy saves 9.4% of power without compromising the performance (see Table 2).

**The proposed strategy overhead** is analyzed next. It may be attributed to two sources. The first source is due to the MPI_Gather and MPI_Broadcast operations that collect in the manager process the information from the distributed worker processes followed by the broadcast from the manager to workers. Both collectives (MPI_Gather and MPI_Bcast) were observed to consume very little time compared with the timeslice duration used here (250 ms). In particular on the four nodes of Turing, the combined overhead of these collective communications was found to be 5.7 μs, which is a small fraction of the timeslice. The second source of the overhead is from the intra-process operations as described

Table 2. Minimum power budget as percentage of the power in allhigh mode such that there is no performance degradation.

| | EP.D | FT.D | BT.D | scf-camp | h2o-16 |
|---|---|---|---|---|---|
| Power fraction (%) | 85 | 89.2 | 95.3 | 92 | 92.3 |

in Figure 2, which include reading performance counters and applying the performance and power models. On Turing, these operations were observed to consume about 50 cycles (4 μs), which is negligible compared with the duration of the timeslice. Such an overall low overhead of the proposed strategy admits a seamless oversubscription of the application cores by the strategy processes. Therefore, it can be concluded that the utilization of the proposed strategy does not hinder the underlying application performance.

## 7. Conclusions and Future Work

This paper proposed a runtime strategy that redistributes a given power budget such that the performance differences across the nodes running an HPC application are equalized. This strategy features a manager-worker distributed communication pattern, the lightweight processes of which may oversubscribe application cores. An algorithm has been developed in this paper to implement the strategy that employs timeslice-based power and performance models and applies power limiting to PKG and DRAM power domains along with the UFS in a user-transparent manner.

Experiments with the NAS parallel benchmarks and a real-world quantum chemistry package GAMESS were conducted on four 28-core nodes of a computing platform. They showed that the proposed strategy substantially outperformed a TDP-based power allocation strategy. For instance, the compute-intensive EP.D benchmark incurred, respectively, 1% and 100% performance penalty for the second highest (80%) power allocation considered. Future work will focus on devising performance and power models to compute an optimal number of nodes such that the application performance penalty is minimized under a given power allocation. In addition, the proposed distributed strategy will also be extended to multi-GPU platforms.

## Acknowledgements

The authors are grateful to the reviewers for their helpful and constructive comments This work was supported in part by the U.S. Department of Energy (DOE) Office of Science, Office of Basic Energy Sciences, Computational Chemical Sciences (CCS) Research Program under work proposal number AL-18-380-057 and the Exascale Computing Project (ECP) through the Ames Laboratory, operated by Iowa State University under contract No. DE-AC00-07CH11358, and by the National Science Foundation under grant CNS-1828593.

## Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

## References

[1] Top 500 List. https://www.top500.org/lists/2019/06/

[2] Intel Software Developer Manual. https://software.intel.com/en-us/articles/intel-sdm

[3] Chen, M., Wang, X. and Li, X. (2011) Coordinating Processor and Main Memory for Efficient Server Power Control. In: *Proceedings of the International Conference on Supercomputing*, ACM, New York, 130-140. https://doi.org/10.1145/1995896.1995917

[4] Tiwari, A., Schulz, M. and Carrington, L. (2015) Predicting Optimal Power Allocation for CPU and Dram Domains. 2015 *IEEE International Parallel and Distributed Processing Symposium Workshop*, Hyderabad, 25-29 May 2015, 951-959. https://doi.org/10.1109/IPDPSW.2015.146

[5] Marathe, A., Bailey, P.E., Lowenthal, D.K., Rountree, B., Schulz, M. and de Supinski, B.R. (2015) A Run-Time System for Power-Constrained HPC Applications. In: Kunkel, J. and Ludwig, T., Eds., *High Performance Computing. ISC High Performance* 2015. *Lecture Notes in Computer Science*, Vol. 9137, Springer, Cham, 394-408. https://doi.org/10.1007/978-3-319-20119-1_28

[6] Zou, P., Allen, T., Davis, C.H., Feng, X. and Ge, R. (2017) CLIP: Cluster-Level Intelligent Power Coordination for Power-Bounded Systems. 2017 *IEEE International Conference on Cluster Computing* (*CLUSTER*), Honolulu, 5-8 September 2017, 541-551. https://doi.org/10.1109/CLUSTER.2017.98

[7] Sarood, O., Langer, A., Kalé, L., Rountree, B. and de Supinski, B. (2013) Optimizing Power Allocation to CPU and Memory Subsystems in Overprovisioned HPC Systems. 2013 *IEEE International Conference on Cluster Computing* (*CLUSTER*), Indianapolis, 23-27 September 2013, 1-8. https://doi.org/10.1109/CLUSTER.2013.6702684

[8] David, H., Gorbatov, E., Hanebutte, U.R., Khannal, R. and Le, C. (2010) Rapl: Memory Power Estimation and Capping. In: *Proceedings of the* 16*th ACM/IEEE International Symposium on Low Power Electronics and Design*, ACM, New York, 189-194. https://doi.org/10.1145/1840845.1840883

[9] Ellsworth, D.A., Malony, A.D., Rountree, B. and Schulz, M. (2015) POW: System-Wide Dynamic Reallocation of Limited Power in HPC. In: *Proceedings of the* 24*th International Symposium on High-Performance Parallel and Distributed Computing*, ACM, New York, 145-148. https://doi.org/10.1145/2749246.2749277

[10] Gholkar, N., Mueller, F. and Rountree, B. (2019) Uncore Power Scavenger: A Runtime for Uncore Power Conservation on HPC Systems. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Association for Computing Machinery, New York, Article No. 27. https://doi.org/10.1145/3295500.3356150

[11] André, E., Dulong, R., Guermouche, A. and Trahay, F. (2020) DUF: Dynamic Uncore Frequency Scaling to Reduce Power Consumption. Working Paper or Preprint. https://hal.archives-ouvertes.fr/hal-02401796v2

[12] Haj-Yahya, J., Alser, M., Kim, J., Yaglikci, A.G., Vijaykumar, N., Rotem, E. and Mutlu, O. (2020) SysScale: Exploiting Multi-Domain Dynamic Voltage and Frequency Scaling for Energy Efficient Mobile Processors. 2020 *ACM/IEEE* 47*th Annual International Symposium on Computer Architecture* (*ISCA*), Valencia, 30 May-3 June 2020, 227-240. https://doi.org/10.1109/ISCA45697.2020.00029

[13] Sundriyal, V., Sosonkina, M., Westheimer, B. and Gordon, M. (2018) Comparisons of Core and Uncore Frequency Scaling Modes in Quantum Chemistry Application Games. In: *Proceedings of the High Performance Computing Symposium*, Society for Computer Simulation International, San Diego, 13:1-13:11.

[14] Sundriyal, V., Sosonkina, M., Westheimer, B. and Gordon, M. (2018) Core and Uncore Joint Frequency Scaling Strategy. *Journal of Computer and Communicatio*n, **6**, 184-201. https://doi.org/10.4236/jcc.2018.612018

[15] Shanley, T. and Winkles, J. (2002) InfiniBand Network Architecture. Addison-Wesley Professional, Boston.

[16] Sundriyal, V. and Sosonkina, M. (2016) Joint Frequency Scaling of Processor and DRAM. *The Journal of Supercomputing*, **72**, 1549-1569. https://doi.org/10.1007/s11227-016-1680-4

[17] Ioannou, N., Kauschke, M., Gries, M. and Cintra, M. (2011) Phase-Based Application-Driven Hierarchical Power Management on the Single-Chip Cloud Computer. *International Conference on Parallel Architectures and Compilation Techniques* (*PACT*), Galveston, 10-14 October 2011, 131-142. https://doi.org/10.1109/PACT.2011.19

[18] Sundriyal, V., Sosonkina, M. and Gordon, M.S. (2019) Maximizing Performance under a Power Constraint on Modern Multicore Systems. *Journal of Computer and Communications*, **7**, 252-266. https://doi.org/10.4236/jcc.2019.77021

[19] Intel 64 and IA-32 Architectures Software Developer's Manual Combined Volumes 3A, 3B, and 3C: System Programming Guide. https://software.intel.com/content/www/us/en/develop/articles/intel-sdm.html#combined

[20] Bailey, D.H., Barszcz, E., Barton, J.T., Browning, D.S., Carter, R.L., Dagum, L., *et al.* (1991) The NAS Parallel Benchmarks—Summary and Preliminary Results. In: *Proceedings of the* 1991 *ACM/IEEE Conference on Supercomputing*, ACM, New York, 158-165. https://doi.org/10.1145/125826.125925

[21] Schmidt, M.W., Baldridge, K.K., Boatz, J.A., Elbert, S.T., Gordon, M.S., Jensen, J.H., Koseki, S., Matsunaga, N., Nguyen, K.A., Su, S., Windus, T.L., Dupuis, M. and Montgomery Jr., J.A. (1993) General Atomic and Molecular Electronic Structure System. *Journal of Computational Chemistry*, **14**, 1347-1363. https://doi.org/10.1002/jcc.540141112

[22] Barca, G.M.J., Bertoni, C., Carrington, L., Datta, D., De Silva, N., Deustua, J.E., *et al.* (2020) Recent Developments in the General Atomic and Molecular Electronic Structure System. *Journal of Chemical Physics*, **152**, 154102. https://doi.org/10.1063/5.0005188

[23] Thermal Design Power (TDP) in Intel® Processors, 2019. https://www.intel.com/content/www/us/en/support/articles/000055611/processors.html