Scientific Research Publishing

Journal of Computer and Communications, 2019, 7, 252-266
http://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

Maximizing Performance under a Power Constraint on Modern Multicore Systems

Vaibhav Sundriyal[1], Masha Sosonkina[1], Bryce Westheimer[2], Mark S. Gordon[2]

[1]Old Dominion University, Norfolk, Virginia, USA
[2]Iowa State University and Ames Laboratory, Ames, IA, USA
Email: vsundriy@odu.edu, msosonki@odu.edu, westheb@iastate.edu, mark@si.msg.chem.iastate.edu

How to cite this paper: Sundriyal, V., Sosonkina, M., Westheimer, B. 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

Received: August 26, 2018
Accepted: July 26, 2019
Published: July 29, 2019

Abstract

Energy efficiency and energy-proportional computing have become a central focus in modern supercomputers. These supercomputers should provide high throughput per unit of power to be sustainable in terms of operating cost and failure rates. In this paper, a power-bounded strategy is proposed that maximizes parallel application performance under a given power constraint. The strategy dynamically allocates power to core, uncore, and memory power domains within a node to maximize performance under a given power budget. Experiments on a 20-core Haswell-EP platform for a real-world parallel application GAMESS demonstrate that the proposed strategy delivers performance within 4% of the best possible performance for as much as 25% reduction in the minimum power budget required for maximum performance.

Keywords

Uncore Frequency Scaling (UFS), Dynamic Voltage and Frequency Scaling (DVFS), Power Budget, GAMESS

1. Introduction

Power consumption has become a major concern for modern and future supercomputers. For the current topmost petascale computing platforms in the world, it is typical to consume power on the order of several megawatts as depicted in the biannual TOP 500 list[1], which may cost on the order of several million dollars. In the quest for exascale performance, the power consumption growth rate must slow down and deliver more calculations per unit of power, giving rise to power-bounded computing in which components of a computing system operate under a fixed power budget such that performance is maximized.

[1]http://top500.org/.

Previous generations of Intel processors used either a fixed uncore frequency or a common frequency for the core and uncore. The uncore describes the functions of a processor that are not handled by the core, such as the L3 cache and on-chip interconnect. Starting from the Intel Haswell micro-architecture, the core and uncore frequency domains have been decoupled, so that the uncore frequency can be modified independently of the core frequency, typically done by dynamic voltage and frequency scaling (DVFS). The uncore frequency has a significant impact on the on-die cache-line transfer rates as well as on the memory bandwidth. By default, the uncore frequency is set by the hardware and can be specified via the model-specific register (MSR) *UNCORE_RATIO_LIMIT* [1]. This technique is denoted uncore frequency scaling (UFS). The latest Intel CPUs work with at least two clock speed domains: one for the core (or even individual cores) and one for the uncore, which includes the L3 cache and the memory controllers.

In the authors' previous work [2] [3], the efficacy of UFS was explored in terms of its energy-saving potential and a strategy was proposed, which employed both DVFS and UFS to maximize energy savings for parallel application execution under a performance constraint. Experiments showed that larger energy savings can be achieved when UFS and DVFS are used jointly. In addition, joint and simultaneous DVFS of the processor and DRAM was explored in [4], where novel power and performance models were proposed.

The Intel Running Average Power Limit (RAPL) interface [5] provides MSRs containing energy consumption estimates for up to four power planes or domains of a machine as follows:

- PKG: for the entire package,
- PP0: for the cores,
- 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).

  The authors' previous research [6] considered primarily PP0 and DRAM domains for budgeting power to solve the parallel application performance optimization problem in the quantum chemistry software GAMESS [7] [8]. The present paper adds the PP1 (uncore) domain, similarly to the work described in [2], to solve this problem and proposes a power-bounded runtime strategy, which maximizes the parallel application performance under a given power budget. In essence, the work presented here may be considered as a combination of [2] and [6] because it determines optimal values for both uncore and core frequencies with the goal to distribute a given power budget to hardware components such that the application performance is maximized. Note that, because the server platform used in this work does not provide a separate PP1 interface to limit uncore power, UFS is used to achieve uncore power shifting within a given power budget. In a nutshell, the contributions of this work include:

- Determining the priority of the power budget allocation to the three domains, namely, core PP0, uncore PP1 and memory DRAM.
- Devising novel performance and power models to correlate changes in uncore frequency to PKG power consumption.
- Proposing a runtime power-bounded strategy to maximize parallel application performance under a given power budget by carefully allocating power to PKG, DRAM and uncore domains.
- Maximizing performance of a quantum chemistry application GAMESS under power constraints.

The rest of the paper is organized as follows. Section 2 provides the related work. Section 3 studies power allocation priorities among power domains. Section 4 proposes performance and power models. Section 5 develops the runtime strategy to maximize performance under a given power budget for any parallel application. Section 6 shows experimental results while Section 7 provides conclusions.

2. Related Work

Power is one of the most prominent HPC challenges, forcing the objectives and approaches of HPC power management to continuously evolve. Therefore, extensive research has been conducted to measure, model, and budget power on computer components and systems. In this section, a brief discussion of previous work in power capping and closely related work in system-level power and energy savings is studied.

The two most commonly used techniques to limit the power consumption of a node come in the form of 1) DVFS/Throttling for processor and memory [9] [10] and 2) Hardware enforced power bounds from RAPL [5]. The authors in [11] propose a runtime system termed *conductor* that dynamically distributes available power to different compute nodes and cores based on the available slack to improve performance. It also performs either upscaling or downscaling of processor frequency to decrease execution time and to save energy in an indirect manner through power clamping. Reference [12] explores the coordinated power allocation among different components within a node, observing which optimal power allocation strategy is proposed. The authors in [10] propose models that predict the performance of HPC computations under varying caps for different components in a node. A cluster level power allocation framework termed *CLIP* was proposed in [13], which performs application characterization along with performance modeling to allocate power budget to nodes and their components to maximize performance in a cluster.

The work in [14] discusses a hardware level power capping strategy for limiting DRAM power consumption. A multi-level hierarchical variation-aware approach of power management is proposed in [15], which at the macro level partitions the system power budget across jobs, and at the micro level, evaluates the power allocation based on application performance metrics. The idea of hardware overprovisioning has been used in [16] by proposing a scheme for determining the optimal number of nodes while distributing power between the CPU and memory. The design of a power scheduler capable of enforcing power bounds by employing dynamic system-wide power reallocation was discussed in [17].

Most of the work discussed in this section primarily focused on redistributing power between the processor cores (PP0) and memory (DRAM) domains, whereas the uncore (PP1) one has largely been ignored. This paper considers the uncore domain and proposes a strategy that resolves the power allocation problem to maximize system throughput at the runtime.

3. Power Allocation Priority

For appropriately allocating a given power budget among different RAPL domains, it is imperative to determine the order in which power should be distributed among them because insufficient allocation to a power domain may have drastic negative effects on the application performance.

Figure 1(a) & Figure 1(b) show the change in the power consumed (left *y*-axis) and execution time (right *y*-axis) for the NAS parallel benchmarks EP (embarassingly parallel) and CG (conjugate gradient), respectively, with the varying power allocation between PKG and DRAM power domains under a fixed power budget of 100 watts (W). Note that the power limits change along the *x*-axis in a

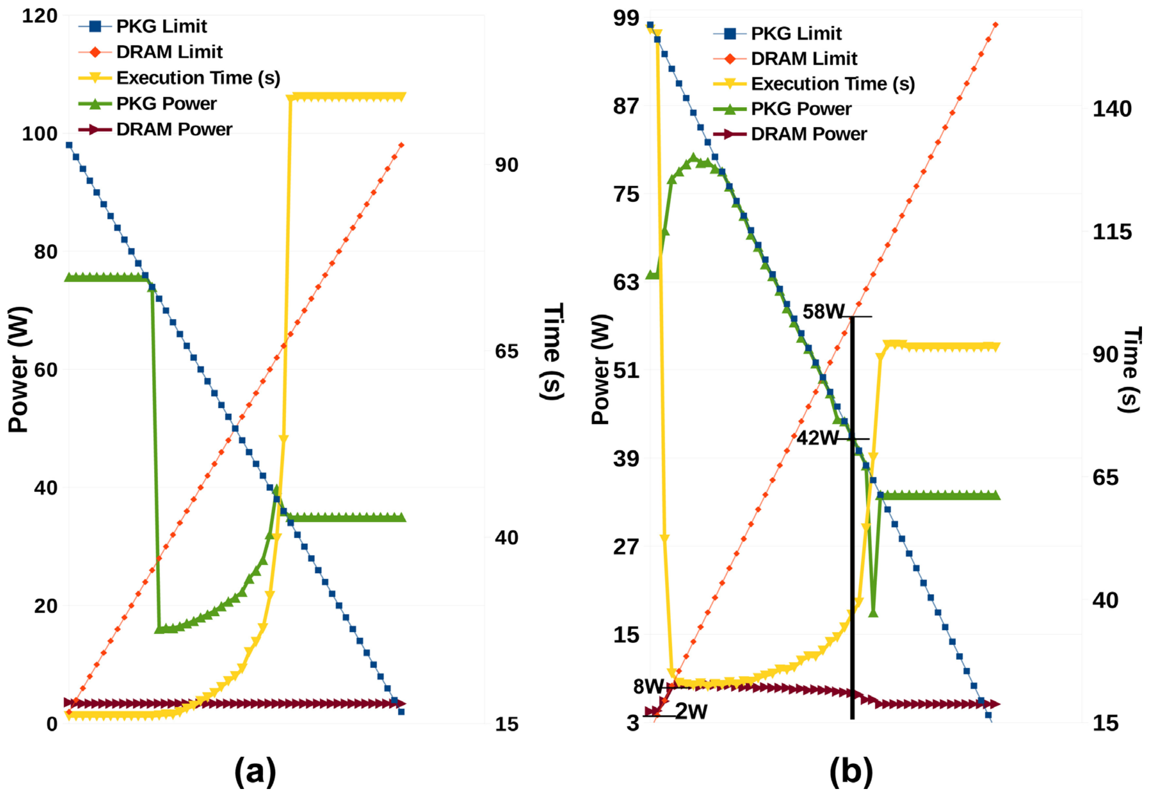

Figure 1. PKG and DRAM power budget allocation out of 100 W of total power budget and the corresponding change in the power consumption and execution time for (a) EP and (b) CG NAS parallel benchmarks. The sequence of (PKG, DRAM) power-budget pairs: $(98,2),(96,4),\cdots,(2,98)$ is along the *x*-axis.

sequence of pairs (PKG, DRAM) obtained from changing both PKG and DRAM power values by 2 W at a time. The power limits essentially set an upper limit for the maximum allocated power consumption of the respective component. For example, in **Figure 1(b)**, the black vertical bar is drawn to indicate that, for the (PKG, DRAM) allocation pair of (42, 58) W, marked with horisonal dashes where the bar crosses the corrsponding power limit lines, the PKG and DRAM power consumptions are observed as 42 W and 7 W, respectively, and the execution time is 36.9 seconds.

It can be observed from **Figure 1(a)**, that for the compute intensive EP benchmark, the performance is sensitive to the PKG power allocation and remains unaffected by the changes to the DRAM power allocation. Specifically, the execution time remains stable until the PKG limit is decreased from 100 to 70 W. For the PKG allocation less than 70 W, the performance degrades and the execution time keeps increasing until the PKG allocation is 34 W. Any further decrement in the PKG allocation from this point, neither decreases the measured PKG power consumption nor degrades the performance any further. On the other hand, for the memory intensive CG benchmark, the execution time is sensitive to both PKG and DRAM power allocations, see **Figure 1(b)**. In particular, the performance degrades rapidly by over 600% when the DRAM power allocation is changed from 8 to 2 W, showing an increase in the execution time from 25 to 155 seconds. Note that the DRAM power consumption does not change when its allocation is increased beyond 8 W. The performance does not change while the PKG power allocation is above 76 W, after which the execution time increases despite the increase in the DRAM power limit. For CG, the effect of reducing the DRAM power allocation is much more severe on performance compared to that of reducing the PKG power allocation. This may be explained by the fact that, when RAPL limits the DRAM power consumption, it essentially cripples DRAM bandwidth according to the power-performance model in [14], whereas reducing the PKG power allocation essentially modifies the operating frequency of the processor cores. Therefore, given a specific power budget, the DRAM domain must have the highest priority of all three power domains when it comes to allocating the power budget. As for the PKG and uncore power domains, the power allocation between them may be decided by using a performance model proposed in [2].

4. Performance and Power Modeling

To effectively distribute the power budget to the application performance, a fine-grained performance model is needed. A power model is also required to correlate the variation in core and uncore frequency with resultant power consumption to effectively apply the power limits. In this section, the two models are discussed.

4.1. Performance Model

A performance model proposed in a previous work [2] is used here. This model (in Equation (1)) correlates 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.

α $(0 \le \alpha \le 1)$ 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.

β_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 [2]:

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

where k_1 and k_2 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 12 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 0.97 and 0.46, 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) depicts how the power is transferred to the core domain (within the PKG domain) through UFS:

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

Specifically, Equation (3) sets the PKG power limit P_{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 limit to increase the core frequency and subsequently to improve performance.

5. Runtime Power-Bounded Strategy

The proposed runtime strategy is based on the *history-window* predictor [4], which employs a window of the previous L values of a measured parameter and predicts its next value as some function g of these past L values. To implement this prediction mechanism, two registers—denoted CPR and MPR—of length L are maintained to record the values of CPM_{exe} and MAPM, respectively. If the register is not filled, then the corresponding quantity is considered unchanged from the previous prediction.

Figure 2 displays the algorithmic steps of the proposed runtime strategy

```
Input Parameters:
 $\tau$ : Duration of timeslice $r$.
 $V$ : Number of timeslices.
 $f_c(1),\ldots,f_c(N)$ : Available core frequencies ($N$).
 $f_u(1),\ldots,f_u(M)$ : Available uncore frequencies ($M$).
 $P_B$ : Combined power budget for PKG and DRAM.
 $P_{PKG}$ : 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 of length $L = 3$ each.
 $\ell = 0$ : counter for filled positions in CPR and MPR ($0 \le \ell \le L-1$).


Algorithm:
Step 1.  Set the package and memory power limits: $P_{PKG}{=}P_{MEM}{=}P_B/2$.
Step 2.  Execute application during timeslice $r = 1$.
Step 3.  Calculate operating core frequency $f_c(\omega_c)$ from APERF and MPERF as in Eq. (4).
Step 4.  Initialize $\mu\mathcal{T}(\omega_c,1)$, CPR[$\ell$], and MPR[$\ell$] for $r = 1$ using the performance counters and Eq. (5).
         $\ell = 1$.
         For ($r = 2,\ r \le V,\ r++$) do
Step 5.  Calculate $\text{CPM}_{\text{exe}}$ and MAPM as
         If ($\ell > 0$ and $\ell \le L-1$) 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 $\mu\mathcal{T}(i,j)$ for all $i = 1,\ldots,N$ and $j = 1,\ldots,M$.
Step 7.  Set the optimal core $f_c(o_c)$ and uncore $f_u(o_u)$ frequency in timeslice $r$ such that
         $\mu\mathcal{T}(o_c,o_u) = \max_{\substack{i=1,\ldots,N\\ j=1,\ldots,M}} [\mu\mathcal{T}(i,j)]$ .
Step 8.  Determine $P_T(o_c,o_u)$ from Eq. (2) for $f_c(o_c)$ and $f_u(o_u)$.
Step 9.  Set $P_{MEM}{=}P_{RAPL-MEM}$, $P_{PKG}$ from Eq. (3) with $j_1 = 1$ and $j_2 = o_u$.
Step 10. If ($\ell == L$) then
            Shift the CPR and MPR left by one position.
            $\ell = \ell - 1$.
Step 11. Execute application for the duration $\tau$.
Step 12. Update $\mu\mathcal{T}(\omega_c,\omega_u)$, CPR[$\ell$], and MPR[$\ell$].
         $\ell = \ell + 1$.
     EndFor
```

Figure 2. Algorithmic steps of the runtime power-bounded strategy.

maximizing parallel application performance under a given power budget. Step 1 divides the user-defined power budget P_B equally between PKG and DRAM power domains. Step 2 profiles the application for the first timeslice of the duration τ and obtains the relevant parameter values from the performance counters. In Step 3, the operating core frequency $f_c(\omega_c)$ is determined by using the APERF and MPERF MSRs [18] according to the relation:

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

where ΔAPERF and ΔMPERF 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, $\mu\tau(\omega_c, 1)$, at the operating core frequency $f_c(\omega_c)$ and the highest uncore frequency $f_u(1)$ for the first timeslice of the application execution. The corresponding CPM_{exe} is calculated from Equation (1) as:

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

and MAPM is obtained directly from the processor performance counters. For $r > 1$, Step 5 determines the values of CPM_{exe} and MAPM through the history-window prediction mechanism 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 CPM_{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 CPM_{exe} and MAPM from Step 5. Next (Step 7), the optimal core-uncore frequency pair is determined, such that the predicted number of micro-operations retired is at its maximum. In Step 8, the total power consumed at the chosen frequency pair is determined using Equation (2) in Step 9, 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 10, if the CPR and MPR registers are completely filled, they are shifted left by one to discard the old values. In Step 11, the application executes the current timeslice r at the chosen PKG and DRAM power limits and the frequencies chosen in Step 7. In Step 12, the values of $\mu\tau(\omega_c, \omega_u)$, CPR[$\ell$], and MPR[$\ell$] are updated with the corresponding operating frequency pair (ω_c, ω_u) to be used in the next timeslice.

6. Experimental Results

The experiments were performed on a compute node, termed Gwent having two Intel Xeon E5-2630 v3 10 core Haswell-EP processors with 32 GB (4 × 8 GB) of DDR4. The core and uncore frequency ranges are 1.2 - 2.3 GHz and 0.8 - 2.9 GHz, respectively. To measure the node power and energy consumption, a Wattsup[2] power meter is used with a sampling rate of 1 Hz.

[2]https://www.wattsupmeters.com/.

6.1. Overview of GAMESS

GAMESS [7] [19] is one of the most representative freely available quantum chemistry applications used worldwide to do *ab initio* electronic structure calculations. A wide range of quantum chemistry computations may be accomplished using GAMESS, ranging from basic Hartree-Fock and Density Functional Theory computations to high-accuracy multi-reference and coupled-cluster computations.

The central task of quantum chemistry is to find an (approximate) solution of the Schrödinger equation for a given molecular system. An approximate (*uncorrelated*) solution is initially found using the Hartree-Fock (HF) method via an iterative *self-consistent field* (SCF) approach, and then improved by various *electron-correlated* methods, such as second-order Møller-Plesset perturbation theory (MP2). The SCF-HF and MP2 methods are implemented in two forms, namely *direct* and *conventional*, which differ in the handling of electron repulsion integrals (ERI, also known as 2*-electron integrals*). Specifically, in the conventional mode all ERIs are calculated once at the beginning of the interactions and stored on disk for subsequent reuse whereas in the direct mode ERIs are recalculated for each iteration as necessary. The SCF-HF iterations and the subsequent MP2 correction find the energy of the molecular system, followed by evaluation of energy gradients.

Data Server Communication Model: The parallel model used in GAMESS was initially based on replicated-data message passing and later moved to MPI-1. Fletcher *et al.* [20] developed the Distributed Data Interface (DDI) in 1999, which has been the parallel communication interface for GAMESS ever since. Later [21], DDI has been adapted to symmetric-multiprocessor (SMP) environments featuring shared memory communications within a node, and was generalized in [22] to form groups out of the available nodes and schedule tasks to these groups. In essence, DDI implements a PGAS programming model by employing a data-server concept.

Specifically, two processes are usually created in each PE (processing element) to which GAMESS is mapped, such that one process does the computational tasks while the other, called the *data server*, just stores and services requests for the data associated with the distributed arrays. Depending on the configuration, the communications between the compute and data server processes occur either via TCP/IP or MPI. A data server responds to the data requests initiated by the corresponding compute process, for which it constantly waits. If this waiting is implemented with MPI, then the PE is polled continuously for the incoming message, thereby being always busy. Hence, it is preferred that a compute process and data server do not share a PE to avoid significant performance degradation. When executing on a 2N-processor machine, the compute C and data server D process ranks are assigned as follows: $C_i \in [0, N-1]$ and $D_i \in [N, 2N-1]$, where $(i = 0, \cdots, N-1)$. Thus, the data server D_i associated with the ith compute process C_i is $N+i$.

6.2. Experiment Setup

NAS benchmarks (NPB) [18] and GAMESS were used for evaluating the efficacy of the proposed runtime strategy and to validate the modeling effort as NPB provides a good mix of compute- and memory-intensive benchmarks to test the core, uncore and DRAM power limiting addressed in this work. The first GAMESS calculation was set-up to perform the third order Fragment Molecular Orbital (FMO3) [23] calculation—in the conventional mode—for a cluster of 64 water molecules at the Restricted Hartree-Fock RHF/6-31G level of theory. As such, it involves calculations of fragment monomers, dimers, and trimers. The system is partitioned into 64 fragments such that each fragment is a unique water monomer and is referred to as h2o-64 in the rest of the paper. The second GAMESS calculation also performs an FMO3 calculation on 20 water molecules at the MP2/6-31G(d, p) level of theory. As such, each fragment *N*-mer (monomer, dimer, and trimer) is calculated sequentially using all compute elements allocated to the GAMESS executable. Three-body calculations at the RHF/6-31G(d, p) level of theory are also performed and are critical in order to capture the significant exchange and charge-transfer effects present in a cluster of water molecules. This calculation is referred to as wat-20 in the rest of the paper.

Table 1 depicts the PKG and DRAM power consumptions, with a 100 W power budget, for the three NAS benchmarks EP, CG, and LU and two GAMESS calculations executing at the highest core and uncore frequencies on Gwent. It can be observed from Table 1 that the compute-intensive benchmark EP tends to have lower DRAM power consumption due to its low memory utilization as compared with the rest of the test cases, which are more memory intensive [4]. For all the inputs, the total power consumption ranges from 80.2 to 88 W. Therefore, to stress-test the proposed runtime strategy. three power budgets of 70, 60, and 50 W were chosen because they are substantially lower than the power consumption needed to maintain maximum performance for these input benchmarks.

6.3. Strategy-Guided Performance under a Power Budget

Figure 3 shows the performance degradation for each input when the proposed runtime strategy is used to distribute the chosen power budgets of 70, 60, and 50 W.

EP.C.20: For the highest power budget of 70 W, the strategy selects the highest core and a low uncore frequency of 1.1 GHz, which results in a performance

Table 1. PKG and DRAM power consumption (W) of NAS NPB benchmarks and GAMESS inputs to achieve the maximum performance with a 100 W power budget. In the NAS benchmark column names, the two-letter prefix denotes benchmark name, "C" stands for class C, and the two-digit suffix states the number of processes used.

Benchmark	EP.C.20	CG.C.16	LU.C.16	h2o-64	wat-20
PKG	77	80	80	80	82
DRAM	3.2	8	7	5	5

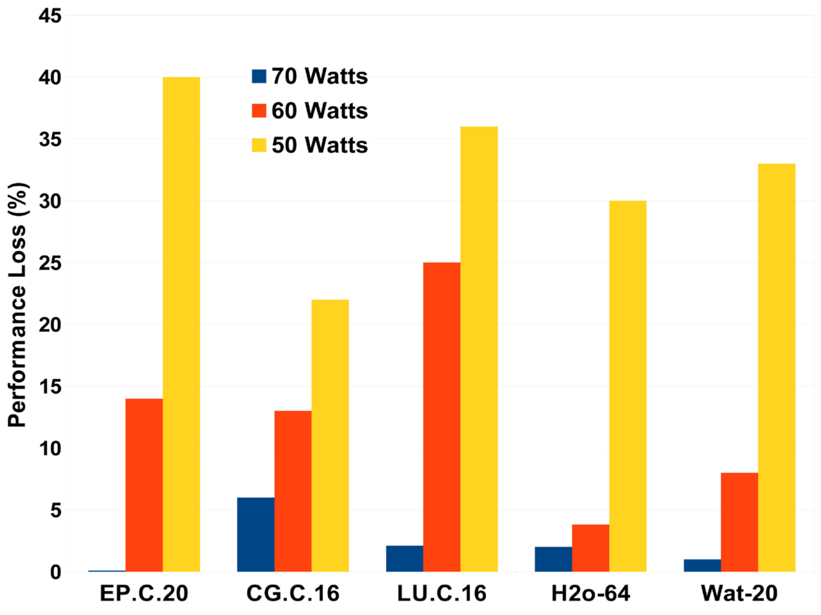

Figure 3. Performance degradation for the NAS parallel benchmarks and GAMESS inputs under three power budgets of 70, 60, and 50 W when the proposed power-bounded strategy.

degradation of 1%. These frequencies were chosen by the strategy because the EP benchmark is substantially compute-intensive and any decrease in the core frequency may substantially degrade performance. Therefore, when only the uncore frequency is reduced its equivalent additional available power is added to the PKG power budget bringing it close to the 77 W needed for the maximum performance. When the total power budget is reduced to 60 W, the uncore frequency is reduced to its lowest value. This reduces the PKG power consumption by ~10 W and subsequently provides an opportunity to increase the allocated PKG power to 67 W, as obtained from eq:pkg and measured PKG and DRAM power consumptions of ~57 W and ~3 W, respectively. However, this extra power allocation due to the uncore frequency downscaling is not enough to enforce the given power budget of 60 W without also reducing the core frequency from its highest value. Therefore, a performance degradation of 13% was observed for the reduced core frequency of 2.1 GHz. Similarly, the power budget of 50 W resulted in performance degradation of 40% since the core frequency had to be reduced even further to accommodate the tight power constraints.

CG.C.16: When the power budget is 70 W, the uncore frequency is set to 2.1 GHz by the strategy, and the resultant performance degradation is 8%. Even though CG is memory-intensive benchmark, as was determined from eq:uops and [24], scaling the uncore frequency results in a smaller performance loss compared to reducing the PKG power limit and, thus, reducing the core frequency. The 60 W and 50 W power budgets result in 13% and 21% performance losses, respectively.

LU.C.16: Its memory intensity lies between that of EP and CG. Therefore, the performance degradation under the three power budgets appears to be in between the corresponding performance-loss values of the EP and CG benchmarks.

GAMESS calculations: The two GAMESS inputs h2o-64 and wat-20 are mostly compute-intensive (see tab:pow) throughout their execution; and their compute processes are somewhat memory-intensive at certain execution phases as compared with the data servers. At the 70 W power budget, per runtime strategy, the data servers are operated at the minimum uncore frequency and the compute processes operate at 1.1 GHz uncore frequency throughout the execution. Subsequently, the performance losses for h2o-64 and wat-20 budget are 1% and 2%, respectively, the majority of which is due to the overhead of the strategy itself. When the power budget is reduced further to 60 W, the uncore frequency for both the data servers and compute processes is scaled to its minimum value and the additional power availability allows the PKG limit to be set to 66 W, leading to a core frequency of 2 GHz. This results in 4% and 8% performance loss for h2o-64 and wat-20, respectively. The 50 W power budget pulls the PKG power allocation down to 56 W, requiring to reduce the core frequency even more and resulting in an average performance loss of 32% for these GAMESS calculations.

6.4. Minimum Power Budget for GAMESS Calculations

The proposed strategy does not take into account the specifics and knowledge of the given application. Hence, its decisions may not result in the maximum optimizations, which is a trade-off between using the strategy as "black-box" and maintaining good performance under power-budget constraints for a variety of applications.

In order to find a minimum power budget to keep the GAMESS performance at its maximum, a knowledge of the relative performances of data servers and compute processes is needed. As explored in a previous work, data server performance is not affected at all by DVFS [25]. With this knowledge, the minimum power budget required for the GAMESS calculations considered here without any performance degradation on Gwent is 59 W (and without using the proposed strategy). Under this power budget, the core frequency of the data servers is reduced to its lowest value of 1.2 GHz, and the PKG portions for compute processes, data servers, and the DRAM are allocated 36, 18, and 5 W, respectively.

7. Conclusions

In this paper, a runtime strategy that employs UFS to redistribute the power budget was proposed. The strategy may be used as a "black box" to maximize parallel application performance under a given power budget. Power and performance models were devised, which were deployed in a runtime strategy to dynamically apply power limiting to PKG and DRAM power domains along with the UFS in a user-transparent manner. Experiments on a 20-core Haswell-EP platform with the NAS parallel benchmarks and a real-world test case of two GAMESS calculations showed that the strategy provided near maximum performance even with substantially limited power budgets. Specifically, for a GAMESS calculation, a 25% reduction in the power consumption resulted in only a 4% performance loss. It was also observed that even for memory-intensive applications, the strategy chose the uncore frequency to be reduced first under a power budget instead of reducing the PKG power limit (*i.e.*, the frequency of the cores).

Future work will focus on testing the efficacy of the PKG power limiting on the platforms with the DDR3- and DDR4-based memory architectures and on accelerators, such as GPUs. Taking into account the application-architecture behavior, and thereby developing a "gray-box" strategy for runtime power allocations power will also be studied. While inter-process communications are explicitly targeted in the authors' previous works [26] [27] to obtain energy savings, the future plan also includes adapting and testing the proposed strategy on a distributed system.

Acknowledgments

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, by the U.S. Department of Defense High Performance Computing Modernization Program, through a HASI grant.

Conflicts of Interest

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

References

[1] Lento, G. (2014) Optimizing Performance with Intel Advanced Vector Extensions. https://computing.llnl.gov/tutorials/linux_clusters/intelAVXperformanceWhitePaper.pdf

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

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

[4] 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

[5] (2016) Intel Software Developer's Manual.

[6] Sundriyal, V., Sosonkina, M. and Gordon, M. (2017) Runtime Power Limiting in

GAMESS on Dual-Socket Nodes. 2017 *International Conference on Computational Science and Computational Intelligence*, Las Vegas, NV, 14-16 December 2017, 1594-1599. https://doi.org/10.1109/CSCI.2017.277

[7] 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., Montgomery Jr., J.A., *et al.* (1993) General Atomic and Molecular Electronic Structure System. *Journal of Computational Chemistry*, **14**, 1347-1363. https://doi.org/10.1002/jcc.540141112

[8] Sundriyal, V., Gaenko, A., Sosonkina, M. and Zhang, Z. (2013) Energy Saving Strategies for Parallel Applications with Point-to-Point Communication Phases. *Journal of Parallel and Distributed Computing*, **73**, 1157-1169. https://doi.org/10.1016/j.jpdc.2013.03.006

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

[10] 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, India, 25-29 May 2015, 951-959. https://doi.org/10.1007/978-3-319-20119-1_28

[11] 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*, Springer International Publishing, Cham, 394-408. https://doi.org/10.1109/IPDPSW.2015.146

[12] Ge, R., Feng, X., He, Y. and Zou, P. (2016) The Case for Cross-Component Power Coordination on Power Bounded Systems. 2016 45*th International Conference on Parallel Processing*, Philadelphia, PA, 16-19 August 2016, 516-525. https://doi.org/10.1109/ICPP.2016.66

[13] 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*, Honolulu, HI, 5-8 September 2017, 541-551. https://doi.org/10.1109/CLUSTER.2017.98

[14] 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

[15] Gholkar, N., Mueller, F. and Rountree, B. (2016) Power Tuning HPC Jobs on Power-Constrained Systems. In: *Proceedings of the* 2016 *International Conference on Parallel Architecture and Compilation*, ACM, New York, 179-191. https://doi.org/10.1145/2967938.2967961

[16] 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*, Indianapolis, IN, 23-27 September 2013, 1-8. https://doi.org/10.1109/CLUSTER.2013.6702684

[17] 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

[18] Bailey, D.H., Barszcz, E., Barton, J.T., Browning, D.S., Carter, R.L., Dagum, L., Fatoohi, R.A., Frederickson, P.O., Lasinski, T.A., Schreiber, R.S., Simon, H.D., Venkatakrishnan, V. and Weeratunga, S.K. (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

[19] Gordon, M.S. and Schmidt, M.W. (2005) Advances in Electronic Structure Theory: GAMESS a Decade Later. In: Dykstra, C.E., Frenking, G., Kim, K.S. and Scuseria, G.E., Eds., *Theory and Applications of Computational Chemistry*: *The First Forty Years*, Elsevier Science, Amsterdam, Netherlands, 1167-1189.

[20] Fletcher, G.D., Schmidt, M.W., Bode, B.M. and Gordon, M.S. (2000) The Distributed Data Interface in GAMESS. *Computer Physics Communications*, **128**, 190-200. https://doi.org/10.1016/S0010-4655(00)00073-4

[21] Olson, R.M., Schmidt, M.W., Gordon, M.S. and Rendell, A.P. (2003) Enabling the Efficient Use of SMP Clusters: The GAMESS/DDI Model. In: *Proceedings of the* 2003 *ACM*/*IEEE Conference on Supercomputing*, ACM, New York, 41. https://doi.org/10.1145/1048935.1050191

[22] Fedorov, D.G., Olson, R.M., Kitaura, K., Gordon, M.S. and Koseki, S. (2004) A New Hierarchical Parallelization Scheme: Generalized Distributed Data Interface (GDDI), and an Application to the Fragment Molecular Orbital Method (FMO). *Journal of Computational Chemistry*, **25**, 872-880. https://doi.org/10.1002/jcc.20018

[23] Fedorov, D.G. and Kitaura, K. (2004) The Importance of Three-Body Terms in the Fragment Molecular Orbital Method. *The Journal of Chemical Physics*, **120**, 6832-6840. https://doi.org/10.1063/1.1687334

[24] Sundriyal, V. and Sosonkina, M. (2016) Joint Frequency Scaling of Processor and DRAM. *The Journal of Supercomputing*, **72**, 1549-1569.

[25] Sundriyal, V., Gaenko, A., Sosonkina, M. and Zhang, Z. (2013) Energy Saving Strategies for Parallel Applications with Point-to-Point Communication Phases. *Journal of Parallel and Distributed Computing*, **73**, 1157-1169.

[26] Sundriyal, V. and Sosonkina, M. (2011) Per-Call Energy Saving Strategies in All-to-All Communications. In: Cotronis, Y., Danalis, A., Nikolopoulos, D.S. and Dongarra, J., Eds., *Recent Advances in the Message Passing Interface*. *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, Heidelberg, 188-197. https://doi.org/10.1007/978-3-642-24449-0_22

[27] Sundriyal, V., Sosonkina, M. and Gaenko, A. (2012) Runtime Procedure for Energy Savings in Applications with Point-to-Point Communications. 2012 *IEEE* 24*th International Symposium on Computer Architecture and High Performance Computing*, New York, 24-26 October 2012, 155-162. https://doi.org/10.1109/SBAC-PAD.2012.20