Journal of Computer and Communications, 2018, 6, 184-201
http://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

Core and Uncore Joint Frequency Scaling Strategy

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

[1]Old Dominion University, Norfolk, VA, USA
[2]Department of Chemistry, Iowa State University, 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. (2018) Core and Uncore Joint Frequency Scaling Strategy. *Journal of Computer and Communications*, **6**, 184-201.
https://doi.org/10.4236/jcc.2018.612018

Received: August 17, 2018
Accepted: December 26, 2018
Published: December 29, 2018

Abstract

Energy-proportional computing is one of the foremost constraints in the design of next generation exascale systems. These systems must have a very high FLOP-per-watt ratio to be sustainable, which requires tremendous improvements in power efficiency for modern computing systems. This paper focuses on the processor—as still the biggest contributor to the power usage—by considering both its core and uncore power subsystems. The uncore describes those processor functions that are not handled by the core, such as L3 cache and on-chip interconnect, and contributes significantly to the total system power. The uncore frequency scaling (UFS) capability has been available to the user since the Intel Haswell processor generation. In this paper, performance and power models are proposed to use both the UFS and dynamic voltage and frequency scaling (DVFS) to reduce the energy consumption in parallel applications. Then, these models are incorporated into a runtime strategy that performs processor frequency scaling during parallel application execution. The strategy can be implemented at the kernel/firmware level, which makes it suitable for improving the energy efficiency of exascale design. Experiments on a 20-core Haswell-EP machine using the quantum chemistry application GAMESS and NAS benchmark resulted in up to 24% energy savings with as little as 2% performance loss.

Keywords

Uncore Frequency Scaling (UFS), Dynamic Voltage and Frequency Scaling (DVFS), Power, GAMESS, Energy Savings, NAS Benchmarks

1. Introduction

Power is one of the critical constraints for the design of next generation exascale systems and commercial data centers. This constraint is evidenced by the 20

MW power budget set by the US Department of Energy for exascale computing. Such a reality forces the HPC community to transform its goal from maximizing the performance without a power limit to improving performance within similar power budgets. In this scenario, the power consumption growth rate must slow down and deliver more calculations per unit of power.

To be able to operate a system under a given power budget, it is imperative that it is comprised of components which have the necessary capability to limit their power consumption. Previous Intel processor generations 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 microarchitecture, the core and uncore frequency domains have been decoupled, so that the uncore frequency can be modified independently of the core frequency. 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]. 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], the efficacy of UFS (uncore frequency scaling) was explored in terms of its energy-saving potential and its effect on the DRAM and L3 cache access latency. Experiments depicted that larger energy savings can be achieved when UFS and DVFS (dynamic voltage and frequency scaling) are used jointly. In addition, joint and simultaneous DVFS of the processor and DRAM was explored in [3], where novel power and performance models were proposed.

This paper considers the experimental observations explored in [2] and adjusts the performance and power modeling devised in [3] so it can be considered as an extension of both.

In this paper, a runtime strategy is developed that applies frequency scaling to both the core and uncore, for which performance and power models are proposed here as well. In a nutshell, the contributions of this work include

- Discovering opportunities for core and uncore frequency-scaling based on the proposed performance model for out-of-order (OOO) processors.
- Modeling power consumption by using the Intel Running Average Power Limit (RAPL), which determines dynamically the power consumption at various core and uncore frequencies.
- Developing a runtime strategy that uses the proposed power and performance models to determine a pair of core and uncore frequencies yielding the maximum energy savings.

The rest of paper is organized as follows. Section 2 provides the related work. Section 3 describes the proposed performance and power models that accurately determine the effect of core and uncore frequency scaling on the application performance and system power. Section 4 describes the runtime system that uses the proposed models to save energy due to the core and uncore frequency scaling. Section 5 discusses the experiments with the real-world application GAMESS and NAS NPB benchmarks, while Section 6 provides conclusions.

2. Related Work

Power is one of the most important design constraints for the next generation of exascale systems forcing the research community to continuously evaluate and redefine the objectives of HPC power management. There have been two general approaches to obtaining energy savings during parallel application execution. The first approach is to focus on identifying stalls during execution by measuring architectural parameters from performance counters as proposed in [4] [5] [6]. Rountree *et al.* [7], apart from using the performance counters, do a critical path analysis to determine which tasks may be slowed down to minimize the performance loss in the parallel execution. Besides communications, Adagio also monitors the computational parts of the application to determine suitable opportunities to apply DVFS. The second approach determines the communication phases to apply DVFS as, for example, in [8] and [9].

As a 20 MW power limit has been set for the next generation exascale systems, many power limiting strategies have been proposed to operate a computing system under a given power budget. The work in [10] emphasizes conductor, which is a runtime system that dynamically distributes available power to the different compute nodes and cores based on the available slack to maximize performance. The work in [11] explores the coordinated power allocation between different components within a node and based on their observations, an optimal power allocation strategy is proposed. The authors in [12] present results of a study where a power control functionality in PAPI [13] was employed to limit the power consumption in a set of benchmarks on the Intel KNL (Knights Landing) computing platform. The work in [14] studies different strategies for analyzing the data center power and operating system counter based on the utilization logs. A feedback-based hierarchical solution for managing the power of a job on a power-constrained cluster, PShifter, was proposed in [15]. The energy-efficiency of the Intel KNL architecture for Lattice Boltzmann applications was explored in [16]. As the cache sizes and memory density increase, the uncore power consumption becomes quite significant considering the total processor power consumption [17]. The authors in [18] study the uncore power consumption in heterogeneous platforms consisting of both high and low power cores using client device workloads and determine that potential energy savings are very much affected by the uncore component. The work in [19] builds on the READEX (Runtime Exploitation of Application Dynamism for Energy-efficient Exascale computing) tool to enable automatic tuning of both the core and uncore frequency for an application. In addition to the mostly experimental existing work, the present paper focuses on theoretical underpinnings by proposing the performance and power models that link the uncore frequency with the application throughput.

3. Performance and Power Modeling

To effectively map the variation in frequency to the application performance, a model is needed, such that it is valid for modern OOO processors. The extent of the effect of frequency scaling on the application execution time is usually determined through memory accesses and the MIPS rate. The proposed performance model predicts the micro-operations retired at the different core and uncore frequencies depending on several system parameters.

3.1. Performance Variation with Frequency Scaling

The uncore frequency scaling primarily affects the performance of three components in a computing system, namely, L3 cache, DRAM, and Quick Path Interconnect (QPI). Figure 1 depicts the variation in the execution time and the power consumption of four NAS class C benchmarks (EP, CG, MG, and FT). These benchmarks represent a diverse set of real-world computational kernels:

EP: Embarrassingly parallel, which requires virtually no interprocessor communication.

CG: Conjugate gradient method, which estimates the smallest eigenvalue of a large sparse symmetric positive-definite matrix.

MG: Multigrid method, which approximates the solution of a Poisson equation.

FT: Fast Fourier transform used to solve a 3D partial differential equation. The experiments were performed on a 20 core Haswell-EP platform and the power consumption was measured by the authors by using a Wattsup[1] meter. It can be observed from Figure 1 that the execution time for each of the four NAS benchmarks barely changes with the change in the link bandwidth. Therefore, QPI performance monitoring is not considered in the proposed performance model. On the other hand, the power consumption decreases by an average of 2% when the link bandwidth is changed from 9.6 GB/s (maximum) to 6.4 GB/s (minimum).

3.2. Performance Modeling

Assume n levels of the core frequency and m levels of the uncore frequency denoted $f_c(i)$ $i = 1,\cdots,n$ and $f_u(j)$ $j = 1,\cdots,m$, respectively, on a given processor. On the Haswell-EP platform used in this work, the core frequency ranged from 1.2 GHz to 2.4 GHz and the uncore frequency ranged from 0.8 GHz to 2.9 GHz. It was determined in [2] that the variation in uncore frequency primarily affects the L3 cache and DRAM access latency. Therefore, the effect of the core and uncore frequency on the micro-operations retired is identified by employing a modified version of the cycle accounting equation in [3] as

$$f_c(i) = \mu\tau(i,j)\left(\mathrm{CPM}_{\mathrm{exe}} + \frac{f_c(i)}{f_c(1)}\left(\mathrm{L3APM}\times\theta\times\gamma_j + \mathrm{MAPM}\times\alpha\times\beta_j\right)\right), \quad (1)$$

[1]https://www.powermeterstore.com/p1206/watts_up_pro.php.

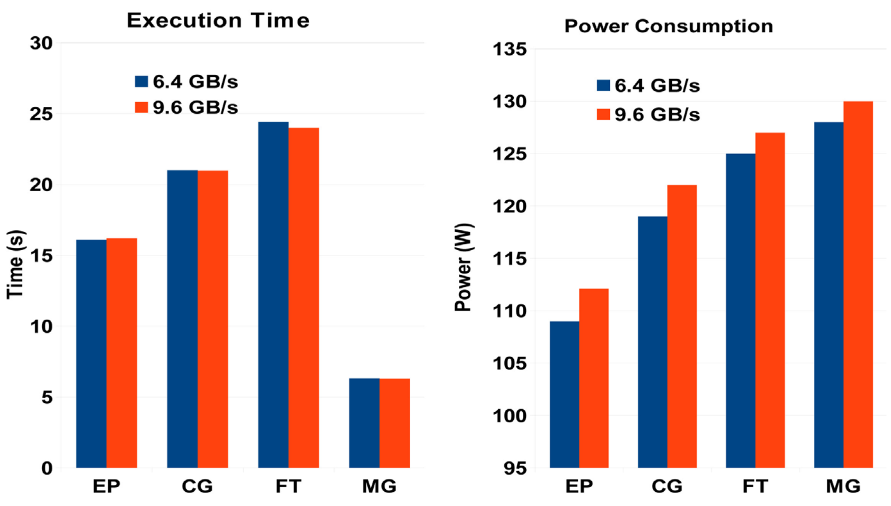

Figure 1. Variation in the execution time and power consumption of four NAS benchmarks EP, CG, FT, and MG with change in the QPI link bandwidth.

where

- $\mu\tau(i,j)$ is the actual number of micro-operations retired per second at core frequency $f_c(i)$ and uncore frequency $f_u(j)$.
- CPM_{exe} is the number of cycles per micro-operation retired barring the memory accesses in a second.
- α and θ $(0 \le \alpha, \theta \le 1)$ are the OOO overlap factors, which determine the extent of the memory and L3 cache access stalls, respectively, overlapped with the execution cycles.
- MAPM and L3APM are the number of memory and L3 cache accesses, respectively, per micro operation retired in a second.
- β_j and γ_j are the number of cycles corresponding to the memory and L3 cache access latency, respectively, at the uncore frequency $f_u(j)$.

The base cycle accounting equation in [3] relates the operating core frequency with the throughput (micro-operations retired), the memory frequency, and the memory intensity of an application. eq:model modifies that base equation, such that it estimates the effect of uncore frequency scaling on both the L3 cache and memory access latencies. By rearranging eq: model, the number $\mu\tau(i,j)$ of micro-operations retired at a core frequency $f_c(i)$ and an uncore frequency $f_u(j)$ can be expressed as

$$\mu\tau(i,j) = \frac{f_c(i)}{\text{CPM}_{\text{exe}} + \frac{f_c(i)}{f_c(1)}\left(\text{L3APM} \times \theta \times \gamma_j + \text{MAPM} \times \alpha \times \beta_j\right)}. \tag{2}$$

Moreover, the performance loss of an application when executed on a core frequency $f_c(i)$ and memory frequency $f_u(j)$ can be estimated as

$$\delta\left(f_c(i), f_u(j)\right) = \frac{\mu\tau(1,1) - \mu\tau(i,j)}{\mu\tau(1,1)}. \tag{3}$$

The values of MAPM and L3APM during runtime are obtained through the processor performance counters and the value of β and γ for varying uncore frequencies are determined through LMbench[2] (the lat_mem_rd API).

Determining Out of Order Overlap Factors

The values of the overlap factors α and θ are critical for the accuracy of the performance model described in Equation (1) Too low a value may overestimate the effect of the memory and L3 cache access latency and vice versa.

Determining the overlap factors through the architectural pipeline method described in [3] can be complicated and inaccurate since it does not consider the relative positioning of the memory accesses in the application execution and for the effect of memory-level parallelism (MLP). Therefore, a regression-based experimental method, described in [3], was used here to determine the values of α and θ.

Class C NAS benchmarks [20] were executed on a single core and their average micro-operations retired, average memory accesses, and $\mathrm{CPM}_{\mathrm{exe}}$ were recorded on different core frequencies for the whole execution, while keeping the uncore frequency constant. Then, Equation (4) was used for the regression analysis in the form:

$$y = a + b \times x_1 + c \times x_2, \tag{4}$$

where for $i = 1, \cdots, 15$ and $j = 1$,

$$x_1 = \mu\tau(i,j) \times \frac{f_c(i)}{f_c(1)} \times \mathrm{MAPM} \times \beta_j,$$

$$x_2 = \mu\tau(i,j) \times \frac{f_c(i)}{f_c(1)} \times \mathrm{L3APM} \times \gamma_j,$$

$$y = f_c(i) - \mu\tau(i,j) \times \mathrm{CPM}_{\mathrm{exe}},$$

$$b = \alpha \text{ and } c = \theta.$$

The analysis resulted in a value of $a \approx 0$, $\alpha = 0.49$, and $\theta \approx 0$ with R^2 of 0.994. The experimentally obtained value of θ indicates that the OOO execution completely masks the L3 cache access delay. Therefore, only the variation in the memory access latency is considered for a change in the uncore frequency for this work. Consequently, Equation (2) is modified as

$$\mu\tau(i,j) = \frac{f_c(i)}{\mathrm{CPM}_{\mathrm{exe}} + \frac{f_c(i)}{f_c(1)} \times \mathrm{MAPM} \times \alpha \times \beta_j}. \tag{5}$$

3.3. Performance Model Validation

To validate the proposed performance model for variation in the core and the uncore frequencies (Equation (1)), several class C benchmarks from the NAS suite (namely, EP, CG, MG, BT, SP and FT) were executed on a single core at the highest and lowest core frequencies with the uncore frequency remaining the same. The average number of micro-operations retired, MAPM, and L3APM were recorded for each benchmark individually for the entire execution. Then,

[2]LMBench web-site: http://www.bitmover.com/lmbench/.

CPM_{exe} was calculated for the execution with the highest core frequency, and eq:model was used to predict the number of micro-operations retired at the lowest core frequency. The predicted number of micro-operations retired was compared to the experimentally determined value with the prediction error ranging from 0.5% to 5.8% and averaging 2.1% for the variation in the core frequency.

To validate the model for variation in the uncore frequency, similar experiments were conducted such that the uncore frequency was varied from 2.9 GHz to 0.8 GHz while the core frequency remained at its maximum value of 2.3 GHz. The prediction error ranged from 0.2% (for EP) to 4.7% (for CG) and averaged at 2.8%. Hence, the proposed performance model appears accurate for variation in both the core and uncore frequencies.

3.4. Power Modeling

To introduce the instantaneous power consumption into the proposed runtime system and select the core and uncore frequencies that minimize the system energy under a performance constraint, the Intel RAPL tool is used, which provides instantaneous processor power consumption. In particular, the processor power consumption can be denoted $P_p(i,j)$ at the core and uncore frequencies $f_c(i)$ and $f_u(j)$, respectively. The instantaneous power consumption, obtained from RAPL, is DC in nature and it may be converted to the corresponding AC power by an appropriate scaling factor *s*, which was determined experimentally for the platform in this work through regression analysis [3].

The power consumption of the processor varies as $f_c(i)^3$ for the core frequency (see, e.g. [3]). To determine whether or not a similar relationship exists between the uncore frequency and the processor power consumption, a set of experiments were performed. **Figure 2** depicts the variation in the RAPL processor (PKG) and DRAM power consumption with variation in the uncore frequency for the class C CG benchmark executing on 16 cores of the Haswell-EP platform. In **Figure 2**, the minor peak observed between the uncore frequencies 2 and 2.5 GHz for DRAM that only the PKG power increases/decreases with a corresponding change in the uncore frequency whereas the DRAM power does not change by much. It can be noted here that the PKG power consumption, which appears to be varying in a linear fashion with the change in the uncore frequency in **Figure 2**, also includes the PKG static power consumption. Let *t* and *z* be the PKG power consumption and $f_u(j)^3$ for different values of uncore frequency, respectively. To confirm that the processor power consumption varies as $f_u(j)^3$, a regression analysis was done on

$$t = e + d \times z, \tag{6}$$

which yielded R^2 of 0.991. Similar values of R^2 were observed when the same regression analysis was performed for the other NAS benchmarks. Consequently, the value $P_p(i,j)$ can be expressed as

$$P_p(i,j) = k_1 \times f_c(i)^3 + k_2 \times f_u(j)^3, \tag{7}$$

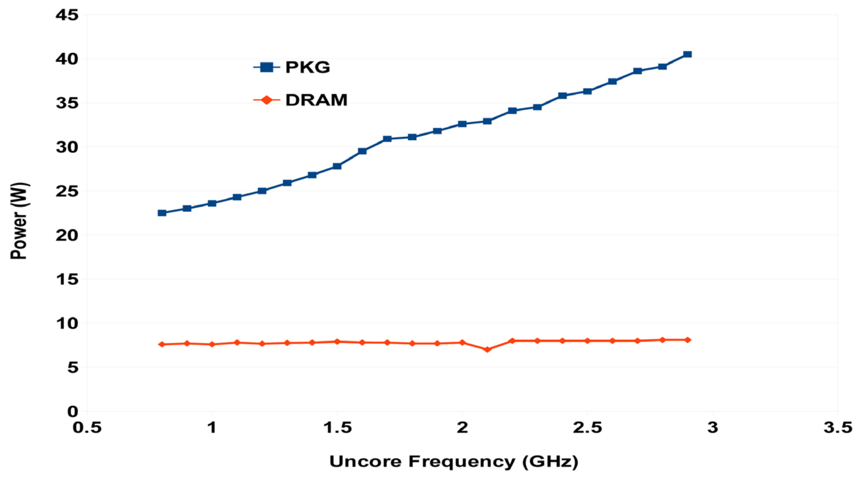

Figure 2. Variation in the PKG and DRAM power consumption with the uncore frequency for the CG benchmark execution on the Haswell-EP platform.

where k_1 and k_2 are constants and $f_c(i)$ and $f_u(j)$ are the core and uncore frequencies, respectively. Parameters k_1 and k_2 are determined through a regression analysis using Equation (6) by getting the processor power through the RAPL registers at different core and uncore frequencies and then performing regressions on those obtained values. Then, the total power consumption $P_T(i,j)$ of a compute node at core frequency $f_c(i)$ and uncore frequency $f_u(j)$ may be expressed as

$$P_T(i,j) = \left(P_p(i,j) + P_m\right) \times s + P_{\text{static}}, \tag{8}$$

where s is the scaling factor as in [3] used to convert DC power values to AC ones, P_m is the memory power consumption, and P_{static} is the static power consumption of the compute node, determined to be 50 Watts through the Wattsup meter.

3.5. Power Model Validation

Since the proposed power model was verified for the variation in the core frequency in [3], the change in the power consumption with respect to the uncore frequency only is being validated here. Figure 3 depicts the measured (by Wattsup) and model-predicted power consumption of the EP benchmark when the uncore frequency was varied from 2.9 GHz to 0.8 GHz. It can be observed from Figure 3 that the proposed power model accurately predicts the system power consumption because the average prediction error is ~2.4% for the experiments in Figure 3. The particular pattern of the predicted power remaining below the measured power at nearly all the uncore frequencies is mainly due the value of the scaling factor s used in these experiments, which primarily depends on the efficiency of the underlying power supply and was determined in [3].

3.6. Fine-Grained Timeslice Approach

The values of CPM_{exe} and MAPM, which can change during the application

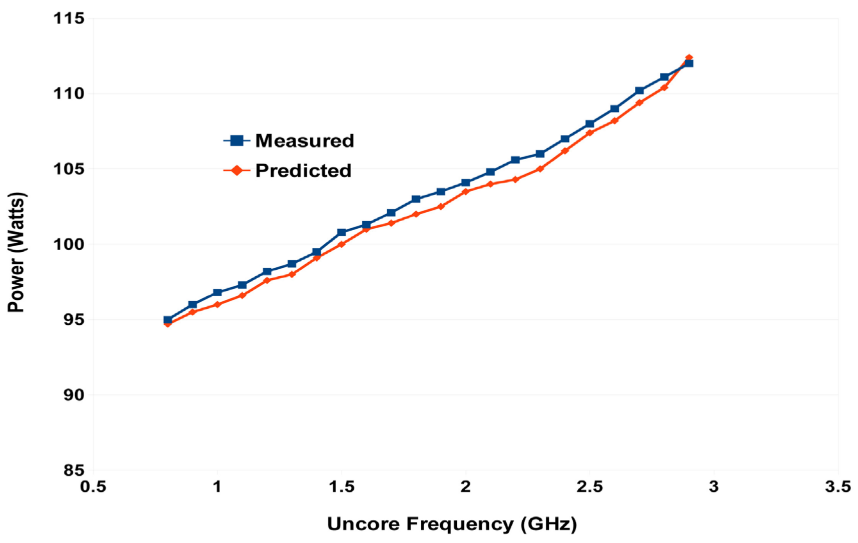

Figure 3. Measured and predicted power consumption of the EP benchmark with variation in the uncore frequency.

execution, are necessary for the prediction of the micro-operations retired at different core and uncore frequencies. Furthermore, an application cannot be entirely considered either CPU or memory intensive especially when frequency scaling is considered since the workload behavior can change in any small timeslice. A coarse-grained approach, such as [21], will not necessarily provide the maximum energy savings, hence the need for a fine-grained timeslice based approach. Each such timeslice needs to be individually assessed, so that its particular compute or memory intensity can be determined and used in the proposed performance and power models. In addition, the frequency of the core/uncore needs to be adjusted accordingly before the next timeslice begins, which requires a workload predicting mechanism.

4. Algorithmic Scheme for Runtime System

The proposed runtime strategy is based on the history-window predictor [3], which employs a window of previous sample L values and predicts the next value as some function g of these L values. To implement this prediction mechanism, two registers—denoted CPR and MPR—of length L are maintained in the runtime system to record the values of $\mathrm{CPM}_{\mathrm{exe}}$ and MAPM, respectively. If the register is not filled, then the corresponding quantity is considered unchanged from the previous prediction.

Runtime Energy-Saving Algorithm

Figure 4 displays the steps of the algorithm underlying the proposed runtime system. Step 1 profiles the application for duration τ and obtains the relevant parameter values from the performance counters. Next, Step 2 initializes the micro-operations retired $\mu\tau(1,1)$ and the total power $P_T(1,1)$ at the highest core and uncore frequencies for the first timeslice of the application execution;

```
Input Parameters:
  $\tau$ : Duration of the 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 ($M$).
  $\gamma$ : User-defined performance loss.
  CPR, MPR : $\mathrm{CPM_{exe}}$ and MAPM registers of length $L = 3$ each.

Algorithm:
Step 1. Execute application during the timeslice $r$ equal to 1.
        Declare counter $\ell = 0$ , $(0 \le \ell \le L-1)$ of the filled positions in CPR and MPR.

Step 2. Initialize $\mu\tau(1,1)$, $P_T(1,1)$, CPR[$\ell$], and MPR[$\ell$] for the first timeslice using the performance
        counters and Eq. (9).
        $\ell = 1$ .

    For ($r = 2,\ r \le V,\ r++$) do
Step 3. Calculate $\mathrm{CPM_{exe}}$ and MAPM as
        If ($\ell > 0$ and $\ell \le L-1$) then
            $\mathrm{CPM_{exe}}$  =  CPR[$\ell - 1$].
            MAPM      =  MPR[$\ell - 1$].
        else
            $\mathrm{CPM_{exe}}$  =  avg(CPR[1], CPR[2],...,CPR[$L$]).
            MAPM      =  avg(MPR[1], MPR[2],...,MPR[$L$]).

Step 4. Calculate $\mu\tau(i,j)$ and $P_T(i,j)$ for all $i = 1,\ldots,N$ and $j = 1,\ldots,M$.

Step 5. Determine $F = \{(a,b)\}$ of core–uncore frequency pairs from Eq. (3), such that $\delta(a,b) \le \gamma$ .

Step 6. Choose the operating core $f_c(o_c)$ and uncore $f_u(o_u)$ frequency in timeslice $r$:

        $P_T(o_c,o_u) \times \tau(1+\delta(o_c,o_u)) = \min_{(f_c(i),f_u(j)) \in F} [P_T(i,j) \times \tau(1+\delta(i,j))]$ .

Step 7. If ($\ell == L$) then
            Shift the CPR and MPR left by one position.
            $\ell = \ell - 1$ .

Step 8. Execute application for the duration $\tau$ at ($f_c(o_c)$, $f_u(o_u)$).

Step 9. Update $P_T(o_c,o_u)$, $\mu\tau(o_c,o_u)$, CPR[$\ell$], and MPR[$\ell$].
        $\ell = \ell + 1$ .
    EndFor
```

Figure 4. Joint core and uncore frequency scaling strategy.

these values are obtained from the performance counters. The corresponding $\mathrm{CPM_{exe}}$ is calculated from Equation (5) as

$$\mathrm{CPM_{exe}} = \frac{f_c(1)}{\mu\tau(1,1)} - \alpha \times \mathrm{MAPM} \times \beta_1, \tag{9}$$

and MAPM is obtained directly from the processor performance counters. Step 3 determines the values of $\mathrm{CPM_{exe}}$ and MAPM through the history-window prediction mechanism by using a simple averaging function, which predicts 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 $\mathrm{CPM_{exe}}$ and MAPM are used as the next values.

In Step 4, $\mu\tau(i,j)$ and $P_T(i,j)$ are determined at all of the available core and uncore frequencies using the values of $\mathrm{CPM_{exe}}$ and MAPM obtained in Step 3. Next (Step 5), a set of all of the core-uncore frequency pairs is determined such that the predicted performance loss does not exceed the performance-loss constraint γ. The threshold value of γ is provided by the user; the resulting

γ is measured using the number micro-operations retired at the end of a timeslice. In Step 6, the energy consumption for the core-uncore frequency combinations found in Step 5 is obtained by multiplying the respective power consumptions and the execution times, and a frequency pair is chosen that minimizes the system energy consumption. Note that the uncore frequency operates at the socket level instead of individual core level. Therefore, if a difference is observed among the chosen uncore frequencies of the threads executing on different cores, their maximum uncore frequency is selected for all the threads on the socket. In Step 7, if the CPR and MPR registers are completely filled, they are shifted left by one to discard the old values. In Step 8, the application executes the current timeslice r at the chosen core-uncore operating frequency pair. In Step 9, the values depending on this frequency pair are updated for the next timeslice.

5. Experimental Evaluation

5.1. GAMESS Overview

GAMESS 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.* [22] developed the Distributed Data Interface (DDI) in 1999, which has been the parallel communication interface for GAMESS ever since. Later [23], DDI has been adapted to symmetric-multiprocessor (SMP) environments featuring shared memory communications within a node, and was generalized in [24] 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 an $2N$-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 $N+i$.

5.2. Experiment Setup

The experiments were performed on a compute node 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 power meter is used with a sampling rate of 1 Hz. The user-defined performance-loss tolerance γ is taken as 10%, which is a typical value to allow for energy savings (see, e.g. [25]).

NAS benchmarks (NPB) and GAMESS were used for evaluating the efficacy of the proposed runtime system and to validate the modeling effort as NPB provides a good mix of compute- and memory-intensive benchmarks to test both core and uncore frequency scaling addressed in this work. The first GAMESS input was constructed to perform the third order Fragmental Molecular Orbital (FMO) [26] calculation—in the conventional mode—for a cluster of 64 water molecules at the 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. The input is referred to as h2o-64 in the rest of the paper. The second input to the GAMESS program was constructed to perform the second order perturbation theory (MP2) energy and gradient calculation—in the direct mode—of substituted silatrane, the 1-trichloromethylsilatrane (TCMS) molecule. MP2 computations were performed on the Haswell-EP node with 6-31G(d) basis set (265 basis functions) which is referred to as silatrane-265 in the rest of the paper.

5.3. Performance and Energy Savings

Figure 5 shows the performance degradation for the four NAS and the two GAMESS inputs when operated under the proposed runtime strategy on the 20

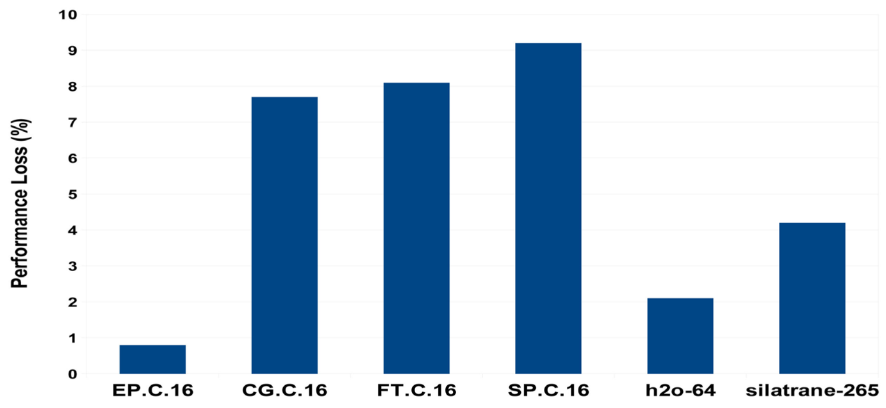

Figure 5. Performance loss for the four NAS and the two GAMESS inputs when operated under the proposed runtime strategy on a 20 core Haswell-EP node.

core Haswell-EP node used in this work. The NAS benchmarks have been depicted as "xx.yy.zz", where "xx", "yy", and "zz" denote benchmark name, class, and number of processes used, respectively. The performance degradation values in **Figure 5** have been normalized to the scenario where both the core and uncore frequency levels remain at their maximum.

The EP benchmark is thoroughly CPU intensive throughout the execution with its performance degrading in a linear manner with the reduction in the core frequency. Therefore, the runtime strategy executes EP at the lowest uncore frequency all the time. While the CG and FT benchmarks are rather memory intensive, the strategy still does not apply any core frequency scaling to them. Instead, CG and FT are operated at 1.8 GHz and 1.4 GHz uncore frequencies, respectively, during their executions. Although the MAPM values for the FT and CG benchmarks are much higher compared with the EP benchmark, the high bandwidth DDR4 is able to significantly overlap computational work with memory accesses, thereby prompting the runtime strategy to keep the highest core frequency for FT and CG. Note also that, in Equation (5), the decrease in the uncore frequency does not degrade the value of micro-operations retired to a large extent. In particular, the uncore frequency scaling targets only the memory controller operation, meaning that the memory-access latency is relatively unaffected, contrary to the power limiting approach in which the memory modules are powered down directly. The SP benchmark is moderately CPU intensive. Therefore, the runtime strategy executes it at 1.9 GHz uncore frequency and the highest core frequency.

For the two GAMESS inputs, the compute processes are kept at the highest core frequency and the lowest uncore frequency since they are extremely CPU intensive. The data servers, on the other hand, primarily facilitate communication functions for the compute processes while not doing much useful work and therefore, they are executed only at the minimum core and uncore frequencies. The difference in the performance degradation observed between the two GAMESS inputs may be explained by the large difference in the execution times of consecutive runs of the same input during the experiments. Specifically, a standard deviation of 4.2% was observed during five different runs of silatrane-265. Overall, the average performance degradation for all of the inputs when operated under the runtime strategy stood at 5.4% which is under the user defined performance loss value of 10% chosen for the experiments.

Figure 6 depicts the energy savings obtained for the four NPB and two GAMESS inputs when operated under the proposed strategy colorblue. The uncore frequency scaling achieved energy savings of 14% and 10.1% for the EP and SP benchmarks, respectively, by itself without the application of core frequency scaling. Both these benchmarks largely remain CPU bound during their execution with minimal L3 cache and memory accesses, depicting that UFS is suitable for such workload behavior. The maximum energy savings (~24%) among all of the inputs are obtained for GAMESS inputs h2o-64 and silatrane-265, when both core and uncore frequency scalings were applied during their execution. This was not the case for the NAS benchmarks. Hence for them, the highest energy savings were obtained for the EP benchmark (14.1%) since it was executed throughout at the lowest uncore frequency (0.8 GHz) with minimal performance degradation. Overall, average energy savings of 15.3% were obtained for the six inputs used in the experiments.

6. Conclusions and Future Work

In this paper, a joint frequency scaling strategy employing both DVFS and UFS was proposed. Detailed power and performance models were devised, which were deployed in a runtime algorithm to dynamically apply UFS and DVFS during application execution in a transparent manner. Experiments on a 20 core Haswell-EP platform with the NPB and GAMESS inputs showed that the strategy provided significant energy savings with minimal performance degradation. Specifically, for a GAMESS input, 24% energy savings were achieved with a

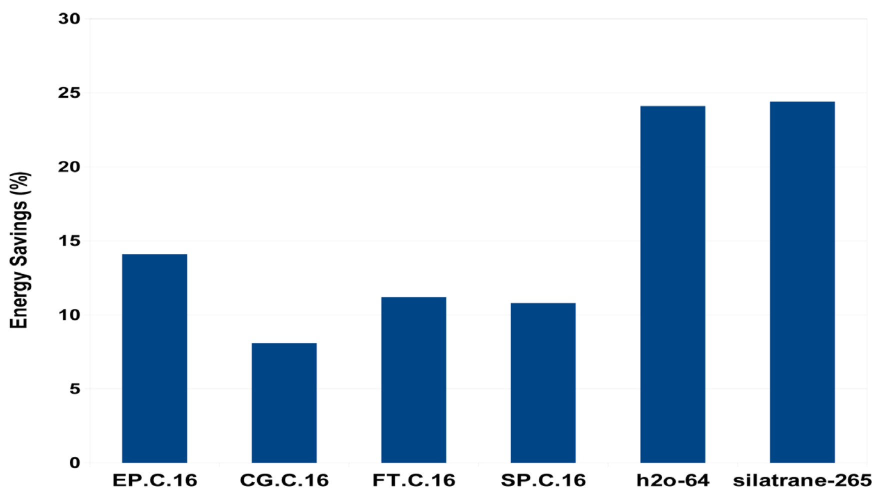

Figure 6. Energy savings for the four NAS and the two GAMESS inputs when operated under the proposed runtime strategy on a 20 core Haswell-EP node.

minuscule 2% performance loss. Overall, the average energy savings and performance loss were found to be 15.3% and 5.1%, respectively, for all of the applications tested. It was also observed that UFS was more potent than DVFS in terms of its potential applicability during the runtime.

Future work will focus on developing runtime power-limiting strategies that will maximize performance under a given power budget by dynamically allocating power to core, uncore, and memory components. The efficacy of DVFS will be investigated for both the DDR3- and DDR4-based platforms as to how DVFS is affected by the memory latency and bandwidth of the underlying memory technology on novel multicore architectures.

While inter-process communications were explicitly targeted in some of the previous works to extract energy savings [27] [28], the future plan also includes adapting and testing the proposed strategy on a distributed system.

Acknowledgements

This work was supported in part by the Air Force Office of Scientific Research under the AFOSR award FA9550-12-1-0476, by the US Department of Energy (DOE) Office of Advanced Scientific Computing Research under the grant DE-SC-0016564 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. The authors would also like to thank the reviewers for providing insightful comments which helped in improving the contributions of this paper.

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.M. and Gordon, M. (2018) Comparisons of Core and Uncore Frequency Scaling Modes in Quantum Chemistry Application GAMESS. *Proceedings of the High Performance Computing Symposium*, *HPC'18*, Baltimore, 1-18, April 2018, 13:1-13:11.

[3] 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

[4] Ge, R., Feng, X., Feng, W. and Cameron, K.W. (2007) CPU MISER: A Performance-Directed, Run-Time System for Power-Aware Clusters. 2007 *International Conference on Parallel Processing* (*ICPP* 2007), Xi'an, 10-14 September 2007, 18. https://doi.org/10.1109/ICPP.2007.29

[5] Hsu, C.H. and Feng, W. (2005) A Power-Aware Run-Time System for High Per-

formance Computing. *Proceedings of the* 2005 *ACM*/*IEEE Conference on Supercomputing*, Seattle, 12-18 November. 2005, 1.

[6] Huang, S. and Feng, W. (2009) Energy-Efficient Cluster Computing via Accurate Workload Characterization. In Cluster Computing and the Grid, 2009. 2009 9*th IEEE*/*ACM International Symposium on Cluster Computing and the Grid*, Shanghai, 18-21 May 2009, 68-75. https://doi.org/10.1109/CCGRID.2009.88

[7] Rountree, B., Lownenthal, D.K., de Supinski, B.R., Schulz, M., Freeh, V.W. and Bletsch, T. (2009) Adagio: Making DVS Practical for Complex HPC Applications. *Proceedings of the* 23*rd international conference on Supercomputing*, *ICS*'09, New York, 8-12 June 2009, 460-469. https://doi.org/10.1145/1542275.1542340

[8] Lim, M.Y., Freeh, V.W. and Lowenthal, D.K. (2006) Adaptive, Transparent Frequency and Voltage Scaling of Communication Phases in MPI Programs. *Proceedings of the* 2006 *ACM*/*IEEE Conference on Supercomputing*, Tampa, 11-17 November 2006, 14. https://doi.org/10.1109/SC.2006.11

[9] Freeh, V.W. and Lowenthal, D.K. (2005) Using Multiple Energy Gears in MPI Programs on a Power-Scalable Cluster. *Proceedings of the Tenth ACM SIGPLAN symposium on Principles and Practice of Parallel Programming*, Chicago, 15-17 June 2005, 164-173. https://doi.org/10.1145/1065944.1065967

[10] 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. Springer International Publishing, Cham, 394-408.

[11] 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* (*ICPP*), Philadelphia, 16-19 August 2016, 516-525.

[12] Azzam, H., Heike, J., Phil, V., Asim, Y., Stanimire, T. and Jack, D. (2018) Investigating Power Capping toward Energy Efficient Scientific Applications. *Concurrency and Computation*: *Practice and Experience*, e4485.

[13] Browne, S., Dongarra, J., Garner, N., Ho, G. and Mucci, P. (2000) A Portable Programming Interface for Performance Evaluation on Modern Processors. *The International Journal of High Performance Computing Applications*, **14**, 189-204. https://doi.org/10.1177/109434200001400303

[14] Khan, K.N., Scepanovic, S., Niemi, T., Nurminen, J.K., Von Alfthan, S. and Lehto, O.-P. (2018) Analyzing the Power Consumption Behavior of a Large Scale Data Center. Computer Science-Research and Development.

[15] Gholkar, N., Mueller, F., Rountree, B. and Marathe, A. (2018) Pshifter: Feedback-Based Dynamic Power Shifting within HPC Jobs for Performance. *Proceedings of the* 27*th International Symposium on High-Performance Parallel and Distributed Computing*, Tempe, 11-15 June 2018, 106-117.

[16] Calore, E., Gabbana, A., Schifano, S.F. and Tripiccione, R. (2018) Software and DVFS Tuning for Performance and Energy-Efficiency on Intel KNL Processors. *Journal of Low Power Electronics and Applications*, **8**, 18.

[17] Loh, G.H. (2008) The Cost of Uncore in Throughput-Oriented Many-Core Processors. *Proceedings of the Workshop on Architectures and Languages for Throughput Applications*, Beijing, June 2008.

[18] Gupta, V., Brett, P., Koufaty, D., Reddy, D., Hahn, S., Schwan, K. and Srinivasa, G. (2012) The Forgotten "Uncore": On the Energy-Efficiency of Heterogeneous Cores. *Proceedings of the* 2012 *USENIX Conference on Annual Technical Conference*, Boston, MA, 13-15 June 2012, 34.

[19] Kumaraswamy, M. and Gerndt, M. (2018) Leveraging Inter-Phase Application Dy-

namism for Energy-Efficiency Auto-Tuning. *International Conference on Parallel and Distributed Processing Techniques and Applications*, Las Vegas, 30 July 2018, 132-138.

[20] 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. *Proceedings of the* 1991 *ACM*/*IEEE conference on Supercomputing*, Albuquerque, New Mexico, 18-22 November 1991, 158-165. https://doi.org/10.1145/125826.125925

[21] Sundriyal, V. and Sosonkina, M. (2018) Modeling of the CPU Frequency to Minimize Energy Consumption in Parallel Applications. *Sustainable Computing*: *Informatics and Systems*, **17**, 1-8. https://doi.org/10.1016/j.suscom.2017.12.002

[22] 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

[23] 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. *Proceedings of the* 2003 *ACM*/*IEEE Conference on Supercomputing*, Phoenix, AZ, 15-21 November 2003, 41.

[24] 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

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

[26] 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

[27] Sundriyal, V. and Sosonkina, M. (2011) Per-Call Energy Saving Strategies in All-to-All Communications. *Proceedings of the* 18*th European MPI Users*' *Group Conference on Recent Advances in the Message Passing Interface*, *EuroMPI*'11, Santorini, 18 September 2011, 188-197.

[28] 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* (*SBAC*-*PAD*), New York, NY, 24-26 October 2012, 155-162. https://doi.org/10.1109/SBAC-PAD.2012.20

Appendices

List of abbreviations used.

AC	Alternating current
CG	Conjugate gradient
CPM_{exe}	Cycles per micro-operations retired excluding the memory accesses in a second
CPU	Central processing unit
DC	Direct current
DDI	Distributed data interface
DRAM	Dynamic random access memory
DVFS	Dynamic voltage and frequency scaling
EP	Embarrassingly parallel
ERI	Electron repulsion integrals
FLOPs	Floating point operations per second
FMO	Fragmental molecular orbital
FT	Fast Fourier transform
GAMESS	General atomic and molecular electronic structure system
L3APM	L3 cache accesses per micro-operation retired
MAPM	Memory accesses per micro-operation retired
MIPS	Million instructions per second
MPI	Message passing interface
MSR	Model specific register
OOO	Out-of-order
PE	Processing element
QPI	Quick path interconnect
RAPL	Running average power limit
SCF-HF	Self consistent field hartree fock
SP	Scalar pentadiagonal
TCP/IP	Transport control protocol/Internet protocol
UFS	Uncore frequency scaling