An Experimental Evaluation of Workload Driven DVFS

Ranjan Hebbar
Electrical and Computer Engineering,
The University of Alabama in Huntsville;
rr0062@uah.edu

Aleksandar Milenković
Electrical and Computer Engineering,
The University of Alabama in Huntsville;
milenka@uah.edu

ABSTRACT

Modern processors support *dynamic voltage and frequency scaling* (DVFS) that can be leveraged by BIOS or OS drivers to regulate energy consumed in run-time. In this paper, we describe the results of a study that explores the effectiveness of the existing DVFS governors by measuring performance, energy efficiency, and the product of performance and energy efficiency (*PxEE*), when running both the speed and throughput SPEC CPU2017 benchmark suites. We find that the processor operates at the highest clock frequency even when ~90% of all active CPU cycles are stalled, resulting in poor energy-efficiency, especially in the case of *memory-intensive* benchmarks. To remedy this problem, we introduce two new workload-driven DVFS techniques that utilize hardware events, (i) the percentage of all stalls (*FS-Total Stalls*) and (ii) the percentage of memory-related stalls (*FS-Memory Stalls*), linearly mapping them into available clock frequencies every 10 ms. Our experimental evaluation finds that the proposed techniques considerably improve *PxEE* relative to the case when the processor is running at a fixed, nominal frequency. *FS-Total Stalls* improves *PxEE* by ~26% when all benchmarks are considered and ~67% when only *memory-intensive* benchmarks are considered, whereas *FS-Memory Stalls* improves *PxEE* by ~15% and ~41%, respectively. The proposed techniques thus outperform a prior proposal that utilizes cycles per instruction to control clock frequencies (*FS-CPI*) that improves *PxEE* by 4% and 9%, respectively.

CCS CONCEPTS

General and reference → Measurement; Evaluation; Performance; *Metrics*; • **Hardware →** *Energy metering*

KEYWORDS

DVFS, ACPI, Benchmarks, Energy-efficiency, Measurements.

ACM Reference format:

Ranjan Hebbar and Aleksandar Milenković. 2021. An Experimental Evaluation of Workload Driven DVFS. *In the Companion of the 2021 ACM/SPEC International Conference on Performance Engineering, (ICPE'21 Companion), April 19-23, 2021, Virtual Event, France.* ACM, New York, NY, USA. 8 pages. https://doi.org/10.1145/3447545.3451192

ICPE'21 Companion, April 19–23, 2021, Virtual Event, France

ACM ISBN 978-1-4503-8331-8/21/04…$15.00
https://doi.org/10.1145/3447545.3451192

1 INTRODUCTION

Modern data centers are a vital part of the computing infrastructure. A majority of the servers in data centers currently utilize x86 processors. Modern x86 processors have evolved to be extremely complex hardware structures, integrating multiple physical cores, on-chip interconnect, uncore cache, memory controllers, and a slew of hardware accelerators on a single chip. Each processor core is highly pipelined with a superscalar out-of-order execution engine with speculative execution, hardware prefetching, hyper-threading, advanced vectorization, and various other performance-enhancing structures. Modern x86 processors also include a number of hardware resources dedicated to the monitoring and management of its operating states [4] [22]. Dynamic Voltage and Frequency Scaling (DVFS) is a technique used in modern processors to adjust the clock frequency and supply voltage of individual processor modules, thus enabling significant power and energy savings. Each new generation of processors, starting from Intel's Haswell/Broadwell architecture, adds more sophisticated hardware resources for managing power consumption [5] [14]. Thus, modern processors support several performance states (P-states) that leverage DVFS and power states (C-states) that allow for unused modules to be turned off.

Generally, algorithms for controlling the P- and C-states are carried out by either BIOS firmware or an OS driver, as defined in the Advanced Configuration and Power Interface (ACPI) standard [23]. The control algorithms (in the further text referred to as governors) monitor the utilization of individual processor cores and use it as a primary factor in determining their operating states [17]. The governors send out requests to a dedicated unit to change operating states of individual processor cores and other components at regular time intervals (e.g., ~10 ms). Prior research efforts have proposed a number of analytical models [13] [16] and experimental methods [11] [24] to inform the design and implementation of these governors. However, these proposals have not seen widespread adoption. Many recent proposals leverage performance monitoring units (PMU) that offer a high-fidelity view of processor activity. One of the most promising methods is to use cycles per instruction (CPI) to determine the optimal P-states [25].

In this paper, we first evaluate the effectiveness of the state-of-the-art BIOS/OS governors. Our evaluation was carried out on a workstation with an Intel Core i7-8700K processor, running SPEC CPU2017 benchmark suites. We find that the state-of-the-art governors tend to put processor cores at the highest possible clock frequency, regardless of the properties of benchmarks being executed. While this policy maximizes performance for all types of benchmarks, it results in wasted energy, especially in the case of benchmarks bounded by the memory subsystem. To address this problem, we propose two techniques that consider two perfor-

mance monitoring unit (PMU) events when adjusting the clock frequency of processor cores: the total number of stall cycles (*FS-Total Stalls*) and the total number of memory stalls (*FS-Memory Stalls*). The percentage of the total stall cycles or the percentage of memory stall cycles is linearly mapped to available P-states every 1 ms.

The experimental evaluation considers performance (*P*), energy efficiency (*EE*), and a composite metric that is the product of performance and energy efficiency (*PxEE*). We compare the effectiveness of the proposed techniques relative to the effectiveness of the state-of-the-art governors and the previously proposed *FS-CPI* technique. All three metrics for all techniques considered are normalized to the case when the processor is running at a fixed, nominal frequency (P1-state). We find that the proposed techniques improve energy efficiency at a minimal loss of performance. *FS-Total Stalls* improves *PxEE* by 26% and *FS-Memory Stalls* by 15%, whereas *FS-CPI* improves *PxEE* by 9% when all SPEC CPU benchmarks are considered together. The proposed techniques are especially effective for memory-bound benchmarks. When compared to the state-of-the-art governors, we find that *FS-Total Stalls* improves *PxEE* by 64%, when all benchmarks are considered, and by 134% when only memory-intensive benchmarks are considered, whereas *FS-Memory Stalls* improves *PxEE* by 50% and 97%, respectively.

The rest of the paper is organized as follows, Section 2 gives background information. Section 3 discusses related work. Section 4 describes the proposed techniques for workload-driven DVFS. Section 5 describes the experimental setup used for the evaluation. Section 5.4 describes the results. Finally, Section 7 concludes the paper and discusses possible future work.

2 BACKGROUND & MOTIVATION

Power management in modern CPUs has seen significant improvements over the last two decades. In order to save energy when the processor is idle, the processor can enter a low-power state. Each processor has several power states, called C-states as shown in Figure 1. The C0 state corresponds to the processor active mode, where all processor resources are turned on and the clocks are active. Within this state, multiple performance, or P-states, are available, enabling dynamic changes of the processor clock frequency and power supply voltage. The P0 state corresponds to the processor's highest operating clock frequency in the so-called turbo mode. Higher P-states (P1-Pn) correspond to active states with progressively lower processor clock frequency and power supply. The C1 power state enables gating the internal processor clocks via software while keeping the bus interface and the *Advanced Configuration and Power Interface* (ACPI) active. The higher C-states (C2-Cn) progressively turn off more and more hardware resources, thus saving more energy, albeit at the cost of increased wake-up time.

Each new generation of modern processors introduces a higher number of C- and P-states, faster and more efficient switching between states, and a richer set of functions in processor configuration and power control [5] [15]. The BIOS or an OS driver is generally responsible for sending requests to a dedicated unit (P-Unit in Intel processors) that is responsible for monitoring and controlling thermal and power aspects of any major component on a processor (processor core, memory control, uncore unit, graphical processor unit). This unit then honors or ignores the request based on the thermal and power constraints.

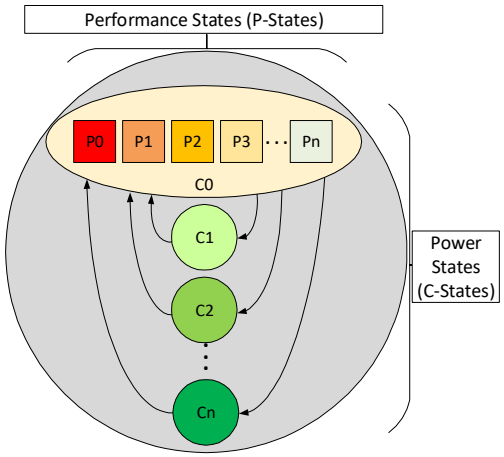

Figure 1: Processor Power States (C-states). The C0 state corresponds to the normal operating mode when the processor is 100% active. Multiple performance states (P0-Pn) allow for frequency and voltage scaling within the active state. Low-power states (C1-Cn) progressively turn off an increasing number of unused resources.

BIOS and/or OS power profiles/governors govern processor thermal and power management. Hardware vendors provide proprietary BIOS governors for P-state and C-state management. If control is transferred to the OS, drivers from processor manufacturers such as the Intel P-state driver (Intel default) or the CPUFreq driver (AMD default) can be utilized. Figure 2 illustrates: (i) the BIOS governors supported by a Dell server (top of the graph), (ii) the OS governors supported by the Intel P-state driver (bottom, left), and (iii) the CPU-Freq generic driver for all Linux-based machines (bottom, right). The *performance* profiles are utilized for latency-sensitive workloads where the processor always runs at the highest possible clock frequency (P0-state) to minimize the response time. However, this policy tends to be wasteful when the system is idle or underutilized. To alleviate this problem, the governors such as *ondemad*, *powersave*, or the *Dell Active Power Control* (DAPC) are utilized. With these, the processor utilization is constantly monitored, and an appropriate P-state is selected in order to minimize energy consumption without sacrificing performance.

The *ondemand* and *powersave* governors employ CPU utilization as the primary metric to determine appropriate P-states. However, the CPU utilization metric used in modern processors can be misleading. The metric widely known as '%CPU' can be accurately described as the percentage of "non-idle time" in a specific time window (e.g., 100 ms). The non-idle time corresponds to the time the processor is running a useful (non-idle) thread. The OS keeps track of the CPU utilization by monitoring context switches. When a non-idle thread begins execution the CPU utilization goes to 100% and remains at this level throughout the

thread execution. It should be noted that CPU utilization does not reflect the actual utilization of the processor pipeline. For example, wasted processor clock cycles due to the processor front-end stalls, structural hazards in the back-end, stalls due to memory reads and writes, and other stalls are not captured by this metric.

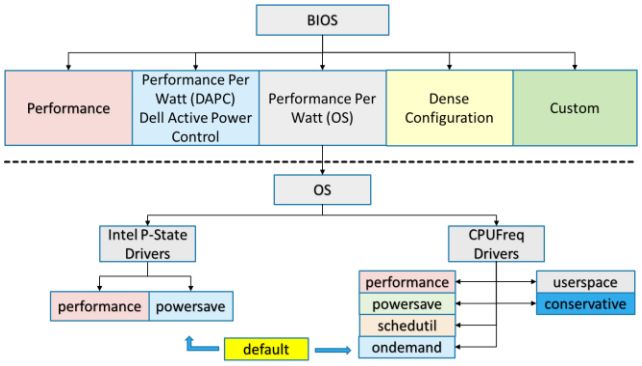

Figure 2: BIOS and OS governors

Figure 3 illustrates a simplified thread execution. In this example, a thread runs on a processor core resulting in 90% CPU utilization (10% CPU executes an idle thread). However, we can further break down the busy processor clock cycles into those doing useful work (30% in this example) and those that are wasted due to stalls (60% in this example). The governors relying on the CPU utilization only, do not take wasted clock cycles into account as they belong to an active thread. In this example, the processor core would typically operate at the highest operating frequency (P0-state). In this paper, we will show that this policy results in wasted energy and that a better policy can be derived to increase the energy-efficiency of benchmarks that spend a lot of time waiting on data from the memory subsystem.

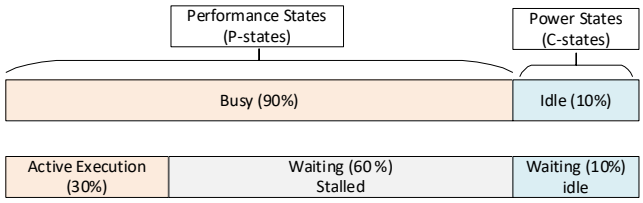

Figure 3: CPU Utilization Metric Breakdown

3 RELATED WORK

The impact of DVFS on energy-efficiency is extensively studied. Thus, researchers have developed an analytical model for power consumed in older generations of processors and have used the model to determine optimal clock frequency [13]. The closed-form mathematical solution to determine optimal frequency [16] saved 7% of the energy used by the NAS benchmarks. Another experimental-based study explored the effects of DVFS and thread counts on the energy-efficiency and performance in several Intel processors using the PARSEC benchmark suite [3]. More recently, a study has shown that parallel benchmarks achieve better energy-efficiency when utilizing higher P-states (lower clock frequency) [8].

However, finding an efficient method to select an optimal operating frequency remains a challenging problem. Past studies have proposed techniques for DFVS that outperform the current power governors. One such method proposes the use of *cycles per instruction* (CPI) when selecting P-states [25]. CPI is a useful tool in assessing the performance of a system running a benchmark. However, in modern superscalar processors, the interpretation of CPI can at times be misleading. As vectorization becomes more prevalent, a significant amount of work can be done with a relatively small number of instructions. This artificially increases CPI relative to when running non-vectorized code. This is especially evident, for example in [9], where the performance of vectorized code (with a relatively high CPI) significantly outperforms the performance of non-vectorized code (with a relatively low CPI). It is better to use fewer SIMD instructions that do more work than to use many scalar instructions that retire faster [1] [26].

Next, CPI-driven DVFS can lead to sub-optimal frequency scaling in benchmarks with a relatively high percentage of stalls originating in the processor front-end. These stalls may result in a high CPI that will in turn lower the processor clock frequency and thus be detrimental to overall performance and energy efficiency. In this paper, we evaluate the CPI-based P-state selection technique by linearly mapping CPI, to the available P-states. Based on an analysis of the results, we propose alternative dynamic voltage and frequency scaling techniques based on the runtime workload parameters obtained from the performance monitoring registers.

4 PROPOSED DVFS TECHNIQUES

Superscalar out-of-order processors consist of the front-end that fetches and decodes machine instructions into micro-operations and the back-end that issues, execute, and retires micro-operations. Multiple micro-operations can be executed and retired in a single clock cycle. The performance of a program directly correlates to the percentage of retired pipeline slots. An unused pipeline slot implies a stall. The Top-down Micro-architectural Analysis Method (TMAM) proposed by A. Yasin provides a practical method to quickly identify true bottlenecks in Intel processors [20]. The TMAM analysis breaks up all pipeline slots into four categories: (i) Pipeline slots containing useful micro-operations that are issued and retired (Retired); (ii) Pipeline slots containing micro-operations that are issued and canceled (Bad Speculation); (iii) Pipeline slots that could not be filled with useful micro-operations due to problems in the front-end (Front-End Bound); and (iv) Pipeline slots that could not be filled with useful micro-operations due to structural and data hazards in the back-end (Back-End Bound).

Based on TMAM, applications can broadly be classified as *compute-intensive*, *balanced*, and *memory-intensive*. *Compute-intensive* refers to applications that are bound by the compute resources available. *Balanced* applications are bound by both the available compute resources and the memory subsystem where performance depends on both compute resources, memory size, and bandwidth. *Memory-intensive* applications are bound by the memory subsystem, where performance is dependent on the available memory size and bandwidth alone.

The processor clock cycles while executing an application can be divided into (i) those that contain at least one pipeline slot that actively dispatches a micro-operation and (ii) those that contain all stalls. The stall cycles can be further divided into (i) on-chip stall cycles and (ii) off-chip stall cycles as shown in Figure 4. The on-chip stall cycles are caused by stalls that originate in the processor front-end and the back-end, excluding the stalls caused by the memory subsystem.

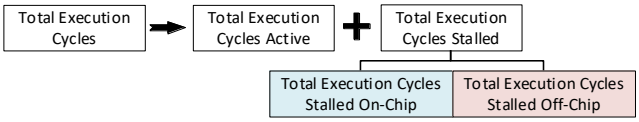

Figure 4: CPU Cycle Breakdown

Prior studies exploring static frequency selection have shown that the performance of *compute-intensive* benchmarks scales linearly with the clock frequency scaling: an increase in the clock frequency results in a proportional increase in the overall performance [8]. These benchmarks have a small percentage of stall cycles, and when they are present, they mainly originate on-chip. The impact of frequency scaling on the performance of *balanced* benchmarks depends on the number and distribution of off-chip requests and the number of stalls associated with them. For *memory-intensive* benchmarks, the impact of frequency scaling on performance is not linear. An increase in the clock frequency typically does not result in the proportional increase in performance, and consequently, a decrease in the clock frequency does not result in the proportional decrease in performance.

We propose two techniques that either use the total stall cycles or the total memory-related stall cycles to determine P-states. The implementation of the techniques is illustrated in Figure 5. The system is initialized to start using the nominal clock frequency (P1-state). We utilize the Performance Monitoring Unit (PMU) to collect the following events of interest: cycles, instructions, the total stall cycles, and the total memory stall cycles. Metrics such as CPI, the total stall ratio, and the total memory stall ratio can be determined every ~1 ms interval. The obtained data are then used to determine P-states as described below. We evaluate three independent techniques based on the collected PMU data. They linearly map a metric of interest to all available P-states on the machine, including P0 (turbo frequencies). The use of P0 ensures that *compute-intensive* benchmarks would not see performance loss.

The first technique, *FS-CPI*, utilizes the CPI as the metric to determine P-state. CPI ranging from 0 to 6 is linearly mapped onto all available P-states. It implements the previously proposed DVFS and it is used here for comparison. The second technique, *FS-Total Stalls*, utilizes the ratio of the total stall cycles to all cycles in the observation period, as the metric to determine the next P-state. This metric accounts for all the stalls associated with the program (both the front-end and back-end stalls). The third technique, *FS-Memory Stalls*, utilizes the ratio of the memory-related stall cycles to all cycles in the observation period to determine the next P-state.

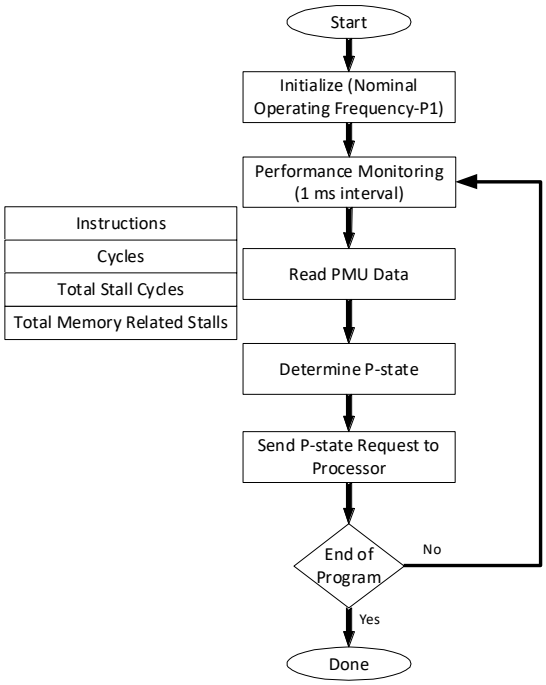

Figure 5: Proposed DVFS Implementation Flowchart

Once the P-state is determined, a request is sent out to the P-unit to switch the P-state. The cycle repeats until the end of the execution of the program. The proposed techniques aim to minimize energy consumption for *memory-intensive* benchmarks that have a significant portion of stalls.

5 EXPERIMENTAL SETUP

This section provides a brief view of the software and hardware setup used in the experimental evaluation. The study utilizes the speed and throughput SPEC CPU2017 benchmark suites for testing the proposed techniques.

5.1 SPEC CPU2017 Overview

SPEC CPU2017 contains 43 benchmarks, organized into four suites [27] [2]. The *fp_speed*/*fp_rate* and *int_speed*/*int_rate* suites (Table 1 and Table 2) include benchmarks with predominantly floating-point data and integer data types, respectively, designed to stress speed (*speed* suites) and throughput (*rate* suites) of modern computer systems. The benchmarks are derived from a wide variety of application domains and are written in C, C++, and Fortran programming languages. The highlighted (green) *speed* benchmarks are parallelizable through OpenMP.

A number of prior studies have characterized the SPEC CPU2017 benchmarks on modern x86 machines [12] [10]. Based on the runtime behavior and resource requirements for each of the benchmarks, they can be classified as *compute-intensive (CI)*, *balanced (B)*, and *memory-intensive (MI)* [7] [6]. Figure 6 shows the classification of each of the CPU2017 benchmarks. The classification helps better understand the impact of frequency scaling on different types of applications.

Table 1: SPEC CPU Floating-point Benchmarks

SPECrate 2017 Floating Point	SPECspeed 2017 Floating Point	Application Area
503.bwaves_r	603.bwaves_s	Explosion modeling
507.cactuBSSN_r	607.cactuBSSN_s	Physics: relativity
508.namd_r	-	Molecular dynamics
510.parest_r	-	Biomedical imaging
511.povray_r	-	Ray tracing
519.lbm_r	619.lbm_s	Fluid dynamics
521.wrf_r	621.wrf_s	Weather forecasting
526.blender_r	-	3D rendering and animation
527.cam4_r	627.cam4_s	Atmosphere modeling
-	628.pop2_s	Wide-scale ocean modeling
538.imagick_r	638.imagick_s	Image manipulation
544.nab_r	644.nab_s	Molecular dynamics
549.fotonik3d_r	649.fotonik3d_s	Computational Electromagnetics
554.roms_r	654.roms_s	Regional ocean modeling

Table 2: SPEC CPU2017 Integer Benchmark

SPECrate 2017 Integer	SPECspeed 2017 Integer	Application Area
500.perlbench_r	600.perlbench_s	Perl interpreter
502.gcc_r	602.gcc_s	GNU C compiler
505.mcf_r	605.mcf_s	Route planning
520.omnetpp_r	620.omnetpp_s	Discrete Event simulation: computer network
523.xalancbmk_r	623.xalancbmk_s	XML to HTML conversion via XSLT
525.x264_r	625.x264_s	Video compression
531.deepsjeng_r	631.deepsjeng_s	AI: alpha-beta tree search (Chess)
541.leela_r	641.leela_s	AI: Monte Carlo tree search (Go)
548.exchange2_r	648.exchange2_s	AI: recursive solution generator (Sudoku)
557.xz_r	657.xz_s	General data compression

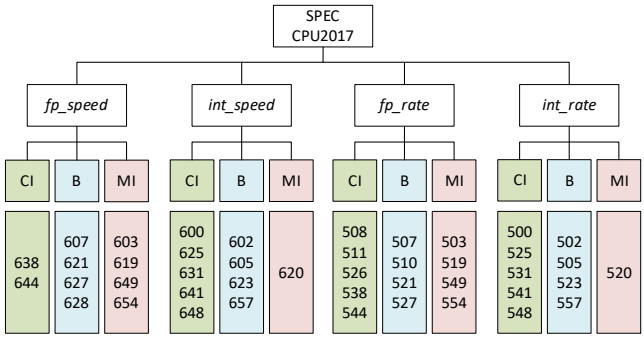

Figure 6: SPEC CPU2017 Classification

5.2 Hardware

The study primarily utilizes a workstation with an Intel x86 processor. The test system is built around an 8th generation Coffee-Lake-based Core i7-8700K with 6 physical processor cores, manufactured using Intel's 14nm++ technology node as described in Table 3. The system has 32 GiB of DRAM. The test system is running Ubuntu 18.04 LTS natively with sufficient power and cooling requirements. The Intel Parallel Studio Cluster XE 2018 is used to compile all the CPU2017 benchmarks with -O3 optimization level [9].

Intel processors, from the Sandy Bridge microarchitecture, have included the *Running Average Power Limit* (RAPL) interface designed to limit on-chip power while ensuring maximum performance [21]. The interface supports fine-grain time measurement of power, energy, and temperature of the socket, individual cores, uncore structures, and on-chip GPUs. Intel has validated the energy estimates provided by the RAPL interface to actual power consumption. Studies have explored the effectiveness of on-chip power meters and explained hardware and soft-ware optimizations as a function of performance and energy efficiency [5]. Various tools make use of the RAPL interface to enable power and energy measurements of different domains [19] [18]. We use the *likwid* tool to read the processor's model-specific registers that measure processor package power consumption.

Table 3: Test System Parameters

Test System	Intel Workstation
Processor	Core i7-8700K
Lithography	14nm
Intel Codename	Coffee-Lake
Core Count	6 (12 Logical Cores)
CPU Max Turbo Freq.	4.70 GHz
All Core Turbo Freq.	4.30 GHz
CPU Nominal. Freq.	3.7 GHz
CPU Min Freq.	0.8 GHz
# P-States	39
RAM	32 GB DDR4
RAM Freq.	2400 MHz
TDP (watts)	95 W

5.3 Metrics

A reference measurement set is established for each of the CPU2017 benchmarks by measuring their execution times, *T(Bi, 3.7)*, and energy consumed, *E(Bi, 3.7)* when the processor clock is set to the nominal frequency of 3.7 GHz. The *speed* benchmarks are run with 6 threads, whereas the *rate* benchmarks are run with 6 copies, thus matching the number of physical processor cores. To compare various techniques, we define performance speedup, *P.S*, calculated as shown in Eq. 1, where $T(Bi, DVFS_{GOV})$ is the execution time of a *speed* benchmark *Bi* when a DVFS governor is in charge of P-states. Similarly, we calculate energy efficiency improvement *EE.I*, of each benchmark as shown in Eq. 2.

$$P.S\ (B_i, DVFS_{GOV}) = \frac{T(B_i, 3.7)}{T(B_i,\ DVFS_{GOV})} \quad (1)$$

$$EE.I\ (Bi, DVFS_{GOV}) = \frac{E(B_i, 3.7)}{E(B_i, DVFS_{GOV})} \quad (2)$$

Finally, a single-number metric that captures both performance and energy efficiency, termed *PxEE*. Consequently, when comparing the effectiveness of a DVFS governor versus the reference run, *PxEE.I* improvement defined in Eq. 3 is used.

$$PxEE.I(B_i, DVFS_{GOV}) = \frac{T(B_i, 3.7) * E(B_i, 3.7)}{T(B_i, DVFS_{GOV}) * E(B_i, DVFS_{GOV})} \quad (3)$$

5.4 Experiments

For a comprehensive evaluation, we use the following DVFS techniques: (a) CPU-utilization-based *OS-ondemand* (also known as *powersave*; it is identical to *performance* for all CPU2017 bench-

marks), (b) *FS-CPI*, (c) *FS-Total Stalls*, (d) *FS-Memory Stalls*, and (e) the manually selected P-states to maximize the *PxEE* metric for each benchmark separately, *Static Selection*. All the techniques are run for the same configuration of the benchmarks and the results are presented as the speedup relative to the reference run with the fixed, nominal frequency (3.7 GHz), *Ref(Nominal)*. The experiments are repeated 3 times and the median value is reported. We find that the margin of error is under 2%.

6 RESULTS

This section discusses the experimental results for performance, energy-efficiency, and *PxEE*.

6.1 Performance

Figure 7 shows the performance speedup, *P.S* as defined in Eq. 1. Expectedly, the *OS-powersave/ondemand* governor provides the best performance. This governor takes advantage of the turbo-modes, yielding a proportional speedup to the *compute-intensive* benchmarks (~15%). In the case of the *balanced* benchmarks, it provides mostly positive results (~7% gain). In the case of *memory-intensive* benchmarks, minimal gains (~1%) are observed.

FS-CPI provides the next best performance overall. It yields ~7% gains for the *compute-intensive* benchmarks, whereas the *balanced* and *memory-intensive* see a performance loss of ~4% and ~3% respectively. *FS-Memory Stalls* degrades the overall performance by ~4%. The *compute-intensive* benchmarks see marginal performance gains. The *balanced* and *memory-intensive* benchmarks see a performance degradation by ~10% and ~5%, respectively. *FS-Total Stalls* has the worst overall performance (loss of ~10%). The degradation is especially high for the *balanced* benchmarks (~19%).

6.2 Energy Efficiency

Figure 8 shows the energy efficiency improvement as defined in Eq. 2. We first observe that the power consumption of *OS-powersave/ondemand* is consistently higher than that of all other techniques, resulting in the lowest energy efficiency improvement metric (a ~26% loss in *EE.I*).

FS-CPI provides the least overall gains in *EE.I* (~4%) of all the techniques. The *compute-intensive* benchmarks see an *EE.I* loss (~12%) because of the use of turbo-mode. The *balanced* applications see the minimal impact (loss of ~1%). The *memory-intensive* application sees a gain of ~31%. Next, *FS-Memory Stalls* provides a bit better energy efficiency overall (~12%). The *compute-intensive* benchmarks see an *EE.I* loss (~13%) because of the use of turbo-mode. The *balanced* applications see marginal gains (~12%). However, the *memory-intensive* application sees a significant gain of ~47%. Finally, *FS-Total Stalls* provides the best overall *EE.I*. Minor degradation is observed for the *compute-intensive* benchmarks. However, it improves energy-efficiency by ~30% for the balanced benchmarks and by ~80% for the *memory-intensive* benchmarks.

6.3 PxEE

Figure 9 shows the *PxEE.I* metric as defined in Eq. 3. The *OS-powersave/ondemand* has the worst *PxEE.I*, especially for the *balanced* and *memory-intensive* benchmarks with a *PxEE.I* loss of ~22% and ~29%, respectively.

FS-CPI has a marginal *PxEE.I* degradation for the *compute-intensive* (~7%) and *balanced* (~4%) benchmarks. However, the *memory-intensive* benchmarks see a gain of ~28%. *FS-Memory Stalls* provides the next best overall *PxEE.I* (~15%). The *compute-intensive* benchmarks see a loss of ~8%. However, the *PxEE.I* improves by ~4% for the *balanced* benchmarks and ~41% for the *memory-intensive* benchmarks. Finally, *FS-Total Stalls* provides the best overall *PxEE.I* of ~26%. A loss of ~6% is seen for the *compute-intensive* benchmarks. However, it improves *PxEE.I* by ~8% for the *balanced benchmarks* and ~67% for the *memory intensive* benchmarks.

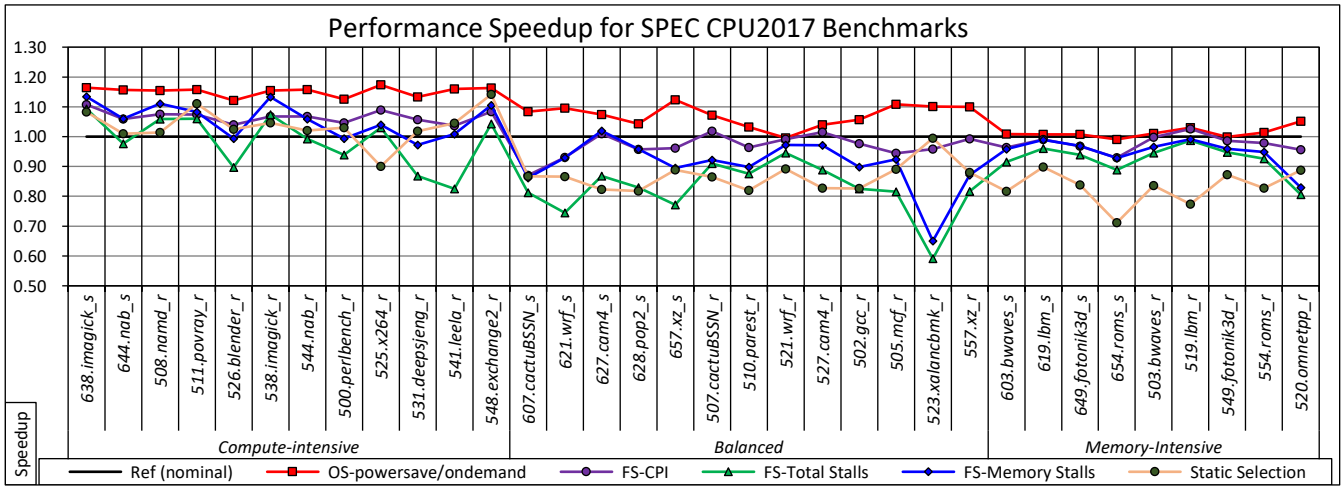

Figure 7: Performance Speedup for individual SPEC CPU2017 Benchmarks

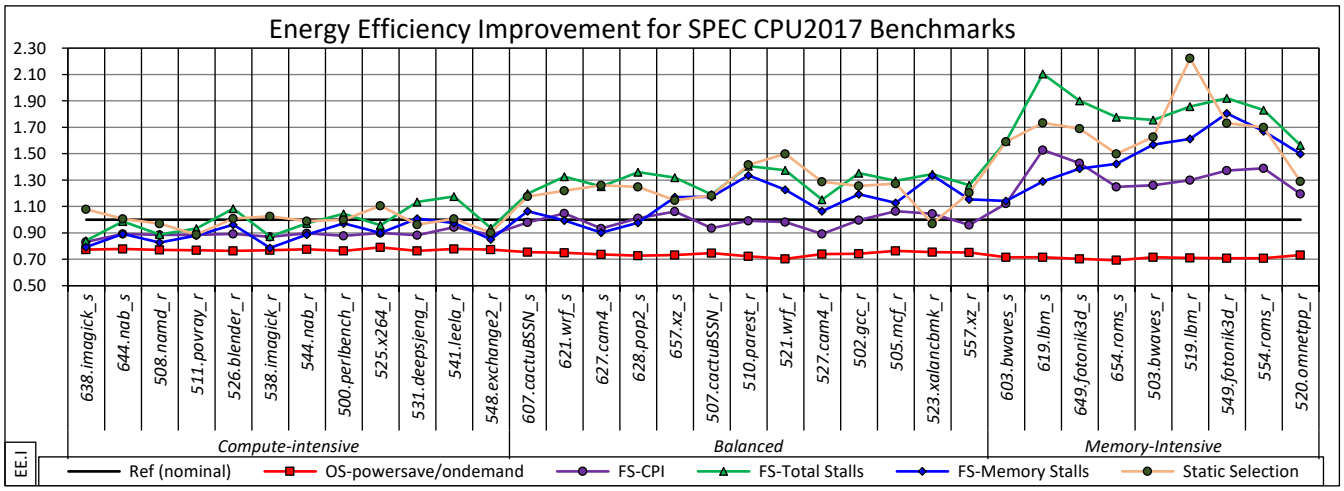

Figure 8: Energy-Efficiency Improvement for individual SPEC CPU2017 Benchmarks

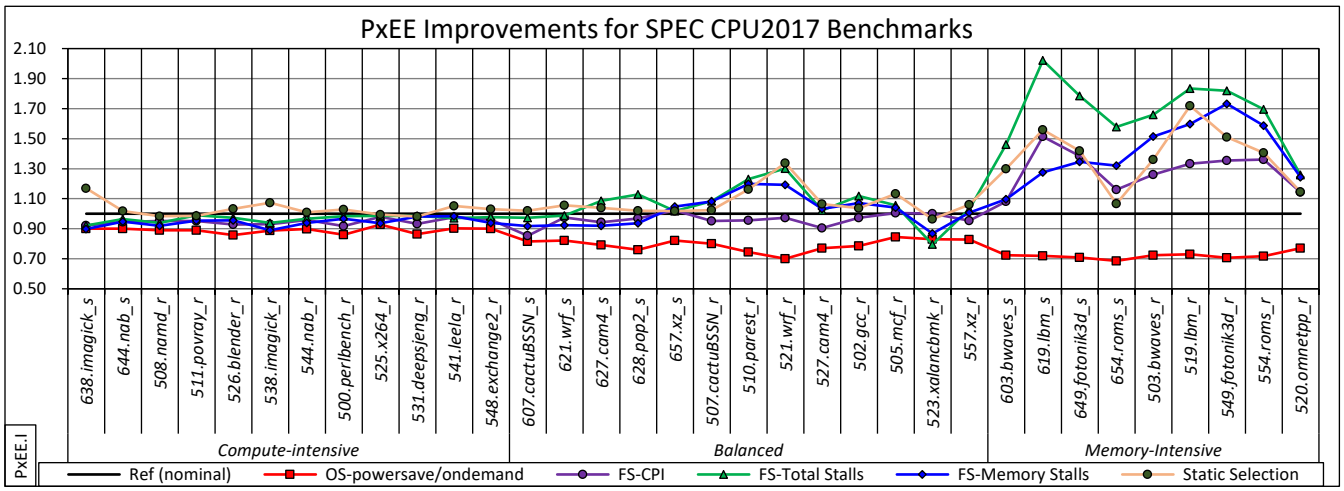

Figure 9: PxEE Improvement for individual SPEC CPU2017 Benchmarks

6.4 Results Summary

Figure 10 shows a summarized view of all three metrics for the evaluated techniques. The speedup metrics are calculated by considering all benchmarks together, i.e., the execution times and energies consumed are summarized across all benchmarks before they are used in equations (1)-(3). Overall, the OS governors (*ondemand/powesave*) provide the best performance, but with a huge penalty in energy efficiency: the total performance gain of 6% *results* in a 26% increase in the energy consumed, resulting in a *PxEE.I* loss of ~23%. While the manual static frequency selection provides an estimate of achievable gains, it is of little practical use, as it would require profiling all benchmarks for all possible P-states in advance. *FS-CPI* offers marginal improvements in energy-efficiency of 4%. *FS-Total Stall* (*FS-TS*) results in performance degradation of 10%, but significant improvements in energy-efficiency (29%) and the composite metric *PxEE.I* (26%). *FS-Memory Stall (FS-MS)* reduces the performance losses but shows modest gains in energy efficiency.

These results underscore the high potential of the proposed techniques. The gains can be further increased if the proposed techniques are only used when the *memory-intensive* benchmarks are run.

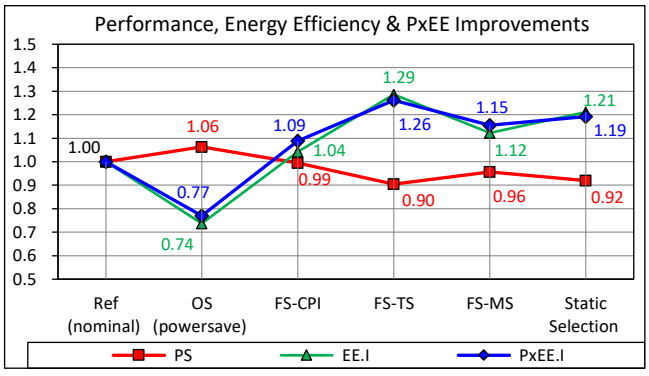

Figure 10: Summary of total performance, energy-efficiency, and PxEE improvements for SPEC CPU2017

7 CONCLUSIONS

Dynamic voltage and frequency scaling is one of the most important tools in regulating processor power consumption. The current implementations of DVFS governors in modern OSes are heavily focused on providing the best possible performance. As the cost of computing increases, more power-oriented DVFS governors need to be implemented. The paper presents the results of the measurement-based analysis of various dynamic voltage and frequency scaling techniques. We observe that the current implementation of DVFS in BIOS/OS is not ideal for *memory-intensive* benchmarks. We investigate the effectiveness of CPI-based frequency selection and propose new techniques that utilize the total stalls and the memory stalls to determine the optimal P-state. Utilizing the *PxEE* metric that incorporates both performance and energy efficiency, we show that our proposed techniques provide *PxEE* improvement of ~29% when using the total stalls and ~15% when using the memory-related stalls as the primary metrics for driving DVFS.

In terms of future work, the proposed techniques *FS-Total Stalls* and *FS-Memory Stalls* implement a linear mapping onto the available operating states (P-states). This assumes that the relationship between the power and frequency is linear. However, in reality, we can observe through measurements that this is not the case. Thus, better mapping of stall parameters to clock frequency is desirable.

REFERENCES

[1] Alaa R Alameldeen and David A Wood. 2006. IPC Considered Harmful for Multiprocessor Workloads. *IEEE Micro* 26, 4 (July 2006), 8–17. DOI:https://doi.org/10.1109/MM.2006.73

[2] James Bucek, Klaus-Dieter Lange, and Jóakim v. Kistowski. 2018. SPEC CPU2017: Next-Generation Compute Benchmark. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering - ICPE '18*, ACM Press, Berlin, Germany, 41–42. DOI:https://doi.org/10.1145/3185768.3185771

[3] Armen Dzhagaryan and Aleksandar Milenković. 2014. Impact of thread and frequency scaling on performance and energy in modern multicores: a measurement-based study. In *Proceedings of the 2014 ACM Southeast Regional Conference* (ACM SE '14), Association for Computing Machinery, New York, NY, USA, 1–6. DOI:https://doi.org/10.1145/2638404.2638473

[4] Lev Finkelstein, Efraim Rotem, Aviad Cohen, Ronny Ronen, and Doron Rajwan. 2013. Power management for multiple processor cores. Retrieved November 18, 2020 from https://patents.google.com/patent/US8402290B2/en

[5] Daniel Hackenberg, Robert Schöne, Thomas Ilsche, Daniel Molka, Joseph Schuchart, and Robin Geyer. 2015. An Energy Efficiency Feature Survey of the Intel Haswell Processor. In *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*, 896–904. DOI:https://doi.org/10.1109/IPDPSW.2015.70

[6] Ranjan Hebbar S R. 2018. Spec CPU2017: Performance, Energy and Event Characterization on Modern Processors. M.S.E. The University of Alabama in Huntsville, United States -- Alabama. Retrieved March 4, 2019 from https://search.proquest.com/docview/2176930551/abstract/2D54E63E98594146PQ/1

[7] Ranjan Hebbar S R and Aleksandar Milenković. 2019. SPEC CPU2017: Performance, Event, and Energy Characterization on the Core i7-8700K. In *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering* (ICPE '19), ACM, New York, NY, USA, 111–118. DOI:https://doi.org/10.1145/3297663.3310314

[8] Ranjan Hebbar S R and Aleksandar Milenković. 2019. Impact of Thread and Frequency Scaling on Performance and Energy Efficiency: An Evaluation of Core i7-8700K Using SPEC CPU2017. In *2019 SoutheastCon*, 1–7. DOI:https://doi.org/10.1109/SoutheastCon42311.2019.9020637

[9] Ranjan Hebbar S R, Mounika Ponugoti, and Aleksandar Milenković. 2019. Battle of Compilers: An Experimental Evaluation Using SPEC CPU2017. In *2019 SoutheastCon*, 1–8. DOI:https://doi.org/10.1109/SoutheastCon42311.2019.9020474

[10] Ankur Limaye and Tosiron Adegbija. 2018. A Workload Characterization of the SPEC CPU2017 Benchmark Suite. In *2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 149–158. DOI:https://doi.org/10.1109/ISPASS.2018.00028

[11] Arindam Mallik, Bin Lin, Gokhan Memik, Peter Dinda, and Robert P Dick. 2006. User-Driven Frequency Scaling. *IEEE Computer Architecture Letters* 5, 2 (February 2006), 16–16. DOI:https://doi.org/10.1109/L-CA.2006.16

[12] Reena Panda, Shuang Song, Joseph Dean, and Lizy K John. 2018. Wait of a Decade: Did SPEC CPU 2017 Broaden the Performance Horizon? In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 271–282. DOI:https://doi.org/10.1109/HPCA.2018.00032

[13] Thomas Rauber, Gudula Rünger, and Matthias Stachowski. 2019. Model-based optimization of the energy efficiency of multi-threaded applications. *Sustainable Computing: Informatics and Systems* 22, (June 2019), 44–61. DOI:https://doi.org/10.1016/j.suscom.2019.01.022

[14] Efraim Rotem, Alon Naveh, Avinash Ananthakrishnan, Eliezer Weissmann, and Doron Rajwan. 2012. Power-Management Architecture of the Intel Microarchitecture Code-Named Sandy Bridge. *IEEE Micro* 32, 2 (March 2012), 20–27. DOI:https://doi.org/10.1109/MM.2012.12

[15] Rober Schöne, Thomas Ilsche, Mario Bielert, Andreas Gocht, and Daniel Hackenberg. 2019. Energy Efficiency Features of the Intel Skylake-SP Processor and Their Impact on Performance. In *2019 International Conference on High Performance Computing Simulation (HPCS)*, 399–406. DOI:https://doi.org/10.1109/HPCS48598.2019.9188239

[16] Vaibhav Sundriyal and Masha Sosonkina. 2018. Modeling of the CPU frequency to minimize energy consumption in parallel applications. *Sustainable Computing: Informatics and Systems* 17, (March 2018), 1–8. DOI:https://doi.org/10.1016/j.suscom.2017.12.002

[17] Guy Therien and Michael Walz. 2006. Power management system that changes processor level if processor utilization crosses threshold over a period that is different for switching up or down. Retrieved November 18, 2020 from https://patents.google.com/patent/US7017060B2/en

[18] Jan Treibig, Georg Hager, and Gerhard Wellein. 2010. LIKWID: A Lightweight Performance-Oriented Tool Suite for x86 Multicore Environments. In *2010 39th International Conference on Parallel Processing Workshops*, 207–216. DOI:https://doi.org/10.1109/ICPPW.2010.38

[19] Vincent M Weaver, Matt Johnson, Kiran Kasichayanula, James Ralph, Piotr Luszczek, Dan Terpstra, and Shirley Moore. 2012. Measuring Energy and Power with PAPI. In *2012 41st International Conference on Parallel Processing Workshops*, 262–268. DOI:https://doi.org/10.1109/ICPPW.2012.39

[20] Ahmad Yasin. 2014. A Top-Down method for performance analysis and counters architecture. In *IEEE International Symposium on Performance Analysis of Systems and Software*, 35–44. DOI:https://doi.org/10.1109/ISPASS.2014.6844459

[21] Huazhe Zhang and Henry Hoffmann. 2015. A Quantitative Evaluation of the RAPL Power Control System. *Feedback Computing 2015* (2015), 6.

[22] Power Management States: P-States, C-States, and Package C-States. Retrieved August 21, 2020 from https://software.intel.com/content/www/us/en/develop/articles/power-management-states-p-states-c-states-and-package-c-states.html

[23] Advanced Configuration and Power Interface - an overview | ScienceDirect Topics. Retrieved January 20, 2021 from https://www.sciencedirect.com/topics/computer-science/advanced-configuration-and-power-interface

[24] US7840825B2 - Method for autonomous dynamic voltage and frequency scaling of microprocessors - Google Patents. Retrieved October 20, 2019 from https://patents.google.com/patent/US7840825B2/en

[25] US8219993B2 - Frequency scaling of processing unit based on aggregate thread CPI metric - Google Patents. Retrieved October 20, 2019 from https://patents.google.com/patent/US8219993B2/en

[26] Recognize and Measure Vectorization Performance. *Intel*. Retrieved January 30, 2021 from https://www.intel.com/content/www/us/en/develop/articles/recognizing-and-measuring-vectorization-performance.html

[27] SPEC CPU® 2017. Retrieved March 19, 2018 from https://www.spec.org/cpu2017/