Uncore Power Scavenger: A Runtime for Uncore Power Conservation on HPC Systems

Neha Gholkar[1], Frank Mueller[1], Barry Rountree[2]
[1]North Carolina State University, USA, ngholka@ncsu.edu, mueller@cs.ncsu.edu
[2]Lawrence Livermore National Laboratory, USA, rountree1@llnl.gov

ABSTRACT

The US Department of Energy (DOE) has set a power target of 20-30MW on the first exascale machines. To achieve one exaflop under this power constraint, it is necessary to minimize wasteful consumption of power while striving to improve performance.

Toward this end, we investigate uncore frequency scaling (UFS) as a potential knob for reducing the power footprint of HPC jobs. We propose Uncore Power Scavenger (UPSCavenger), a runtime system that dynamically detects phase changes and automatically sets the best uncore frequency for every phase to save power without significant impact on performance. Our experimental evaluations on a cluster show that UPSCavenger achieves up to 10% energy savings with under 1% slowdown. It achieves 14% energy savings with the worst case slowdown of 5.5%. We also show that UPSCavenger achieves up to 20% speedup and proportional energy savings compared to Intel's RAPL with equivalent power usage making it a viable solution even for power-constrained computing.

CCS CONCEPTS

• **Software and its engineering → Power management**;

KEYWORDS

high-performance computing; power awareness

ACM Reference Format:
Neha Gholkar, Frank Mueller, Barry Rountree. 2019. Uncore Power Scavenger: A Runtime for Uncore Power Conservation on HPC Systems. In SC '19: The International Conference for High Performance Computing, Networking, Storage, and Analysis, November 17–22, 2019, Denver, CO. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3295500.3356150

1 INTRODUCTION

The Department of Energy (DOE) has set a power budget of 20-30MW on each of the exascale machine sites to maintain a feasible power demand that can be met by operational budget constraints and power plants. Today's fastest supercomputer, Sunway TaihuLight, consumes 15.3MW to deliver 93PFlop/s [5]. Exascale supercomputers, which are slated to arrive in 2020, are expected to achieve an exaflop under 20-30MW of power. To achieve this goal, we need to achieve at least an order of magnitude improvement in performance with respect to the state of art systems without exceeding the DOE's power constraint.

Chip manufactures have provided various knobs such as Dynamic Frequency and Voltage Scaling (DVFS), Intel's Running Average Power Limit (RAPL) [11, 28], and software controlled clock modulation [28] that can be used by system software to improve power efficiency of systems. Various solutions [6, 7, 16, 18, 26, 33, 41, 42] have been proposed that use these knobs to achieve power conservation without impacting performance. While these solutions focused on the power efficiency of cores, they were oblivious of the uncore, which is expected to be a growing component in the future generations of processors [34].

Fig. 1 depicts the architecture of a typical Intel server processor. A chip or a package consists of two main components, core and uncore. A core typically consists of the computation units (e.g., ALU, FPU) and the upper levels of caches (L1 and L2) while the uncore contains the last level cache (LLC), the quick path interconnect (QPI) controllers and the integrated memory controllers. With increasing core count and size of LLC and more intelligent integrated memory controllers on newer generations of processors, the uncore occupies as much as 30% of the die area [25], significantly contributing to the processor's power consumption [8, 23, 51]. The uncore's power consumption is a function of its utilization, which varies not only across applications but it also varies dynamically within a single application with multiple phases. We observed that Intel's firmware sets the uncore frequency to its maximum on detecting even the slightest of uncore activity resulting in high power consumption. We can save power replacing such a naive scheme with a more intelligent uncore frequency modulation algorithm.

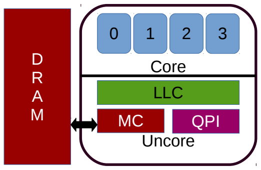


Figure 1: A processor is a chip consisting of core and uncore. The uncore consists of memory controllers (MC), Quick Path Interconnect (QPI) and the last level cache (LLC).

SC '19, November 17–22, 2019, Denver, CO, USA

ACM ISBN 978-1-4503-6229-0/19/11…$15.00
https://doi.org/10.1145/3295500.3356150

Toward this end, we propose Uncore Power Scavenger (UPSCavenger), a runtime system that automatically modulates the uncore frequency to conserve power without significant performance degradation. For applications with multiple phases, it automatically detects phase changes and dynamically resets the uncore frequency for each phase. To the best of our knowledge, UPSCavenger is the first runtime system that focuses on the uncore to improve power efficiency of the system.

To this end, the paper makes the following contributions:

- We explore uncore frequency scaling and its impact on power consumption and performance of various applications.
- We propose Uncore Power Scavenger (UPSCavenger), a runtime system that reduces the power consumption and the energy footprint of applications by automatically modulating the uncore frequency.
- We propose a new light-weight approach for dynamic phase detection that leverages real-time measurements and does not require any prior information about the application.
- We evaluate UPSCavenger with multiple HPC benchmarks and applications on a cluster over 286 cores to show that it conserves power and achieves significant energy savings compared to the default configuration. Our evaluations also show that UPSCavenger achieves performance improvement over Intel's RAPL under equivalent power constraints.

The paper is organized as follows. Sections 2 introduces uncore frequency scaling and analyze its impact on performance. Section 3 discusses the design of UPSCavenger, the proposed runtime system. Section 4 describes the implementation and the experimental setup. Section 5 presents the evaluation of UPSCavenger with multiple applications and benchmarks. To assess energy savings we compare UPSCavenger with the default configuration, i.e. without explicitly setting the uncore frequency, while for performance we compare UPSCavenger with Intel's RAPL mechanism. Section 7 summarizes the contributions.

2 OVERVIEW

From Haswell processors onward, Intel has introduced uncore frequency scaling (UFS) that can be used to modulate the frequency of the uncore independent of the core's operating state. UFS is exposed to software via a model-specific register (MSR) addressed at 0x620. Bits 15 through 8 of this MSR encode the minimum uncore frequency multiplier while bits 7 to 0 encode the maximum uncore frequency multiplier. The product (frequency multiplier x 100MHz) gives us the frequency. For our architecture the minimum and the maximum uncore frequency multipliers are 12 and 27, respectively. We use the msr-safe [48] kernel module to read from and write to the UFS MSR.

2.1 Single Socket Performance Analysis of UFS

We conducted our experiments on Broadwell nodes. Each node has two sockets. Each socket hosts one Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz processor. Each processor has 8 cores. Each node has 128GB memory. While we conducted performance analysis of a wide range of workloads, in this section we present datasets for three representative workloads that have distinct performance characteristics, viz., Embarrassingly Parallel (EP), the Block Tridiagonal solver (BT) and the Multi-grid solver (MG) from the NAS parallel benchmark [4] suite. EP is a compute-intensive benchmark, BT is a last level cache-bound benchmark and MG is a memory-bound benchmark. We used the MPI version of these codes in our experiments. The uncore frequency was modulated from 2.7GHz to 1.2GHz in steps of 0.1GHz. We report performance in terms of job completion time in seconds, average package power and DRAM power in Watts, memory bandwidth and LLC misses per second. Each experiment was repeated five times. We report averages across five runs.

Figures 2(a), 2(b) and 2(c) depict the effects of uncore frequency scaling on EP, BT and MG, respectively. The x-axis represents uncore frequency (from high to low). The default uncore frequency is 2.7GHz. Each figure presents five datasets, viz., completion time in seconds, package power and DRAM power reported by Intel's Running Average Power Limit (RAPL) [28] registers, memory bandwidth and LLC misses per second. The y-axis represents data normalized with respect to the default configuration (UCF=2.7GHz). Table 1 shows the raw data for EP, BT and MG at 2.7GHz.

Table 1: Metrics for EP, BT and MG at 2.7GHz

Metric	EP	BT	MG
PKG power	40W	46W	50W
DRAM power	19W	23W	54W
Memory Bandwidth [MB/s]	8	3983	37327
LLC misses per second	0.003M	3M	33M

Observation 1: UFS leads to (package) power savings without significant performance degradation for CPU-bound applications.

This is illustrated by Figure 2(a) showing data for EP, a representative CPU-bound benchmark. EP uses L1 and L2 caches aggressively but otherwise has a negligible memory footprint. For EP, package power decreases linearly with the uncore frequency. However, completion time and DRAM power remain constant. This is because EP neither uses the LLC nor the memory controllers heavily. The reduction in package power can be attributed to the uncore's idle power consumption.

Observation 2: UFS reduces power for cache-bound applications but aggressive frequency reduction may lead to performance degradation.

This is illustrated by Figure 2(b) showing data for BT, a cache-bound application. It uses LLC aggressively. With the default configuration, the LLC cache miss rate is only 3M per second as most of its LLC accesses are hits and memory bandwidth consumption is 3.9GB/s. Package power decreases linearly with the uncore frequency. However, completion times increase slightly at lower uncore frequencies. Reducing the uncore frequency leads to fewer LLC misses per second and lower memory bandwidth which leads to nominal performance degradation (slowdown) only beyond 1.5GHz. It is important to note that the DRAM power remains constant.

Observation 3: UFS leads to significant performance degradation for memory-bound applications.

This is illustrated by Figure 2(c) showing data for MG, a memory-bound application. With the default configuration, its LLC miss rate is 33M per second while it consumes 37GB/s (≈10X that of BT) of memory bandwidth. Consistent with previous observations,

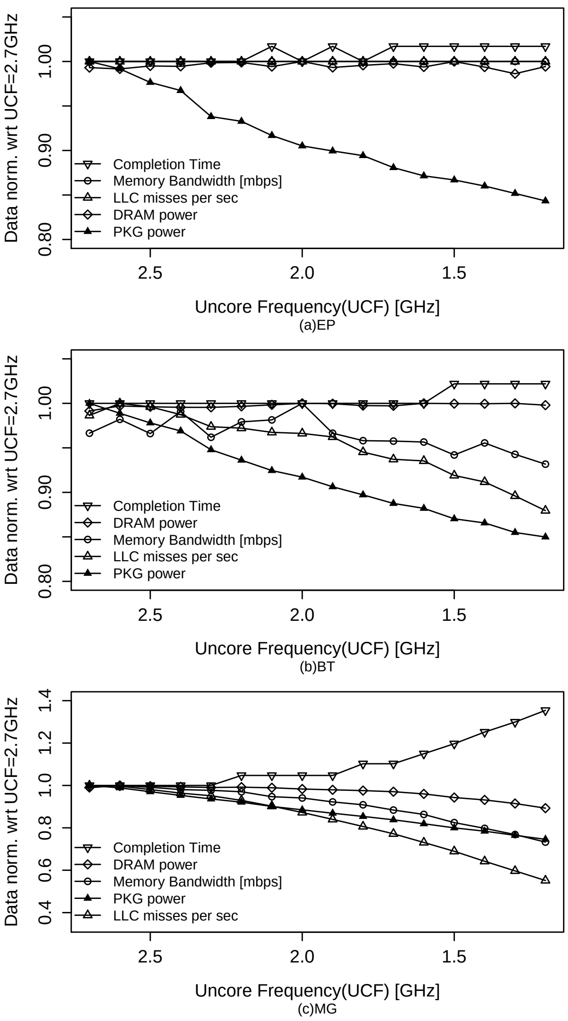


Figure 2: Effects of UFS on EP, BT and MG.

even for MG package power decreases linearly with the uncore frequency. The completion time increases linearly with the uncore frequency under 2.1GHz. Reducing the uncore frequency leads to slower LLC and slower memory controllers. Slower memory controllers send memory requests at a lower rate to DRAM leading to lower DRAM power consumption as shown in the figure. Unlike EP and BT, DRAM power reduces with the uncore frequency beyond 2.1GHz for MG. This decline in DRAM power is also coupled with significant performance degradation beyond 2.1GHz reaffirming our previously stated inference.

From observations 1-3, we conclude the following:

- *The decline in DRAM power as a result of lowering the uncore frequency is an indication of potentially significant performance degradation.*
- *The uncore frequency can be reduced safely without a significant impact on performance only up to the point until which DRAM power remains unchanged.*

Observation 4: Applications consist of multiple distinct phases spanning across the spectrum from compute-intensive to memory-intensive.

This is illustrated by Figure 3 depicting the DRAM power profile and the IPC profile of MiniAMR [44]. The x-axis represents the timeline while the y-axis represents data normalized to each metric at maximum frequency. Five distinctly identifiable phases, P1-P5, are annotated in the plot. P1-P5 are short compute-intensive phases characterized by low DRAM power and high IPC occurring between relatively long memory-intensive phases characterized by high DRAM power and low IPC. Hence, the phase transitions can be detected as follows:

- The transition from a compute-intensive phase to a memory-intensive phase can be identified by a rise in DRAM power.
- The transition from a memory-intensive phase to a compute-intensive phase can be identified by a decline in DRAM power.

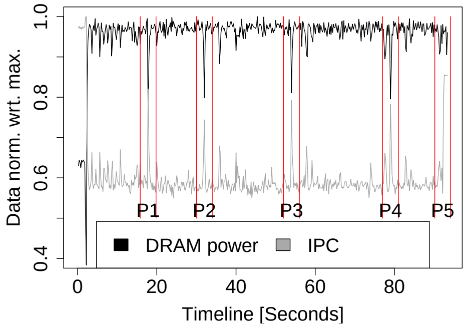


Figure 3: Phases in MiniAMR

PHASE CHANGE DETECTION

Let ΔIPC be the change in IPC and $\Delta DRAM\_power$ be the change in DRAM power during a fixed time interval. Algorithm 1 depicts the pseudo code for phase change detection.

Hypothesis: Static UFS is sub-optimal for applications with distinct phases.

By combining observations 1-4, we hypothesize that for an application consisting of multiple phases with distinct uncore utilization, DRAM access rates, and CPU intensity, statically setting a single uncore frequency for the entire execution is sub-optimal. Hence, we need a dynamic approach that automatically detects phase changes and determines the best uncore frequency for each phase.

While techniques like power gating can be leveraged to save power for phases with zero uncore utilization [19, 35, 37, 43], i.e., while all processes are suspended, uncore frequency scaling achieves power conservation during phases with non-zero but low uncore utilization, which is often the case for HPC applications. Using techniques like sleep states requires prediction of the length of the zero utilization phases to evaluate the trade-offs of the overheads of entering and exiting from a sleep state. Uncore frequency scaling is free of such overheads. Hence, we use uncore frequency scaling in our proposed approach.

Algorithm 1. *Phase Change Detection*

```
procedure DETECT_PHASE_CHANGE
    if $\Delta DRAM\_power == 0$ then
        no phase change detected
    else if $\Delta DRAM\_power > 0$ then
        phase change detected
        compute-intensive to memory-intensive
    else                                        ▷ $\Delta DRAM\_power < 0$
        phase change detected
        memory-intensive to compute-intensive
    end if
end procedure
```

end

3 UNCORE POWER SCAVENGER (UPSCAVENGER)

We propose UPSCavenger, a power-aware runtime system that aims at conserving power by dialing down the uncore's operating frequency opportunistically without significant performance degradation. Figure 4 depicts the high-level architecture of UPSCavenger and the mapping of its components to the components of an HPC system. Figure 4(a) shows the architecture of a typical HPC cluster. It consists of multiple compute server nodes. Each server node hosts two sockets with a single 12-core processor. Multiple parallel jobs spanning across one or more nodes can run on a cluster, e.g., job1 runs on the top 4 nodes while job2 runs on bottom 4 ones. An instance of UPSCavenger runs alongside each job.

The high-level design of the UPSCavenger runtime is depicted in Figure 4(b). The UPSCavenger runtime system allocates one UPSCavenger agent to each socket in a job. Each UPSCavenger agent monitors its socket's performance and DRAM power. It dynamically and automatically manages a socket's uncore frequency to conserve power. UPSCavenger agents make asynchronous decisions based only on the local information pertaining to their respective sockets. Hence, UPSCavenger does not require any global synchronization across the individual UPSCavenger agents. The UPSCavenger agent is a closed-loop feedback controller [49]. The general idea of a closed-loop feedback controller is depicted in Figure 5.

A feedback-control loop consists of three stages that are repeated periodically, viz., sensor, control signal calculator, and actuator.

System is the component whose characteristics are to be monitored and actuated by the controller. The current state of the system is defined in terms of the *process variable* (PV). The desired state of the system, PV=K, where K is a constant, is called the setpoint (SP). It is input to the feedback controller. A feedback-control consists of three main modules, viz., sensor, control signal calculator, and actuator.

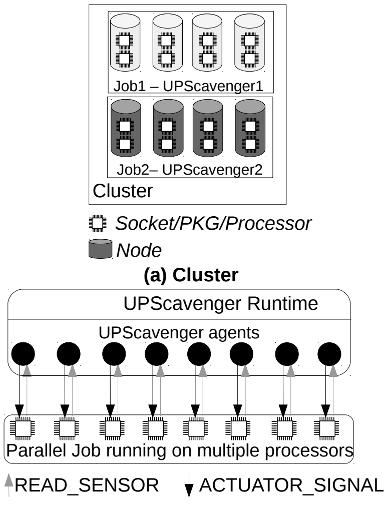
Sensor

The sensor periodically monitors the state of the system defined in terms of process variable. This is indicated by READ_SENSOR.

Control Signal Calculator

The error (err) in the system state is calculated as the difference between the setpoint and the measured process variable. The feedback controller determines the new value for the actuator as a function of this error to drive the system closer to the setpoint.

Actuator



(a) Cluster

(b) UPScavenger Controller for a job

Figure 4: UPSCavenger Overview

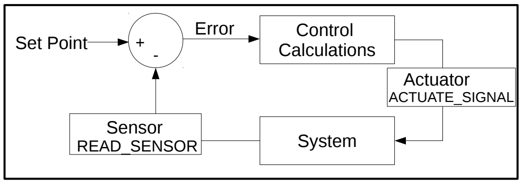


Figure 5: Closed-loop Feedback Controller

The actuator applies the calculated signal to the system. This is indicated by ACTUATE_SIGNAL.

3.1 UPSCavenger Agent

Each UPSCavenger agent is a closed-loop feedback controller that performs three tasks. First, it periodically monitors the socket by measuring its performance in terms of instructions per cycle (IPC) and its DRAM power consumption. Second, depending on the measured power and IPC, it determines the uncore frequency that would drive the system closer to the setpoint and actuates. Third, it detects phase changes at runtime and resets the setpoint for every new phase. The architecture of a UPSCavenger agent is depicted in Figure 6.

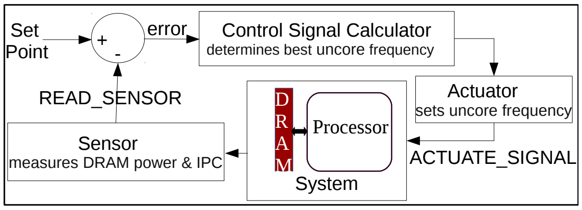


Figure 6: UPSCavenger Agent

System, Process Variable and Setpoint

The system of a UPSCavenger is a socket consisting of a processor and its local DRAM. The process variable of a UPSCavenger agent is its socket’s DRAM power consumption, i.e., PV = DRAM power, measure of socket’s DRAM utilization, which again, correlates with the uncore utilization. Prior work [19, 37, 56] has used models that rely on multiple performance counters to determine application characteristics. Our approach has lower overhead than this approach. UPSCavenger relies on a single register (DRAM RAPL MSR) and hence, issues just one MSR request to quantify/determine the state of the system. UPSCavenger automatically determines the setpoint, which is the maximum DRAM power consumption for every phase of the application, i.e., For an application with multiple phases, UPSCavenger agent automatically resets the setpoint on detecting a phase change.

Sensor

The sensor periodically monitors DRAM power and IPC of the socket. This is indicated by READ_SENSOR.

Control Signal Calculator

The control signal calculator takes as inputs the socket’s DRAM power consumption and IPC measured at runtime. It then determines the current state of the system to decide the next course of action. Table 2 describes the variables used by the control signal calculator. Figure 7 presents the control logic.

During its first invocation (INITIALIZATION), the UPSCavenger agent sets the setpoint, SP_DRAM_power, to the observed DRAM power, PV_DRAM_power. After initialization, the system enters into a periodic control loop. IPC and DRAM power are measured for each sampling interval. ΔDRAM_power is calculated as $PV\_DRAM\_power - SP\_DRAM\_power$.

State 1: Candidate for UCF reduction

If ΔDRAM_power is zero, it implies that the DRAM power has not declined (from the setpoint for the current phase) as a result of the previous actuation. This puts the system in state 1, i.e., it is a candidate for lowering uncore frequency in the current iteration of the control loop. Therefore, the actuator decrements the uncore frequency and control returns to the monitoring block.

Table 2: Variables in Control Signal Calculation

Variable	Description
ucf	uncore frequency
PV_DRAM_power	DRAM power consumption measured during current sampling interval
SP_DRAM_power	setpoint
err = Δ DRAM power	PV_DRAM_power - SP_DRAM_power
previous_IPC	IPC measured during previous sampling interval
current_IPC	IPC measured during current sampling interval
Δ IPC	current_IPC - previous_IPC
MAX_UCF	maximum uncore frequency for the architecture

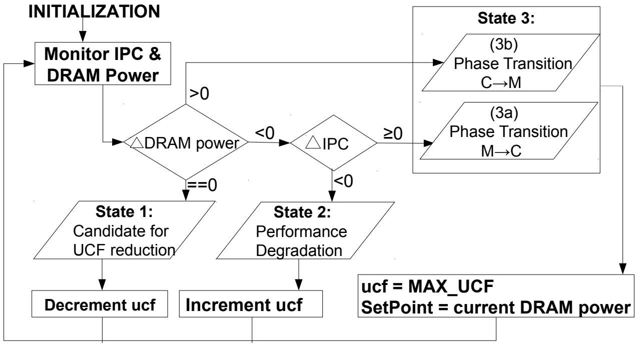

INITIALIZATION
Monitor IPC & DRAM Power
>0
△DRAM power
<0
△IPC
≥0
==0
<0
State 1: Candidate for UCF reduction
State 2: Performance Degradation
Decrement ucf
Increment ucf
State 3:
(3b) Phase Transition C→M
(3a) Phase Transition M→C
ucf = MAX_UCF
SetPoint = current DRAM power

Figure 7: Control Logic: A State Machine

State 2: Increment UCF

If ΔDRAM_power is less than zero, it is an indication of a drop in DRAM power from the setpoint for the current phase. This drop could either be an indication of a phase transition from a memory-intensive to a compute-intensive phase or a detrimental effect of excessive uncore frequency lowering in the previous iteration of the control loop. To identify the cause of the observed drop, we compare IPC. ΔIPC is the difference between IPC during the current and the previous sampling intervals. If ΔIPC is less than zero, it is an indication of performance degradation. Hence, this observation puts the system in state 2. In response to this state, the actuator increments the uncore frequency and control returns to the monitoring block.

State 3: Phase Transitions

If ΔDRAM_power is less than zero and ΔIPC is greater than or equal to zero, it is an indication of a rise in CPU activity with a decline in DRAM power. This implies that a phase transition from memory-intensive to compute-intensive phase occurred. Hence, this observation puts the system in state 3 (3a).

If ΔDRAM_power greater than zero, it implies that the measured DRAM power is higher than the setpoint (maximum) for the current phase. This implies that there has been a phase transition from compute-intensive to memory-intensive phase. Hence, this observation puts the system in state 3 (3b).

Actuator. The actuator is responsible for taking action in response to the determined state of the system. This is indicated by ACTUATE_SIGNAL. The UPSCavenger agent sets the socket’s uncore frequency by writing to the MSR (at 0x620).

State 1: The actuator decrements the uncore frequency to conserve power.

State 2: The actuator increments the uncore frequency to revert the detrimental effects of the previous actuation.

State 3: On detecting a transition to a different phase, the actuator sets the uncore frequency to the maximum value (MAX_UCF) and sets the setpoint (say SP_DRAM_power) to the measured DRAM power (PV_DRAM_power).

UPSCavenger does not need prior information about any phases that occur during application execution. UPSCavenger’s IPC and DRAM power-driven phase change detection suffices to automatically detect these phases with distinct behavior (e.g., checkpointing to detect that is more memory-intensive than a blocked matrix multiplication kernel).

Resetting the Setpoint to maximum DRAM power for a new phase: It is important to note here that in response to both the phase transitions (even in case of M->C) we set the uncore frequency to MAX_UCF. As a result of this, the socket is guaranteed to have the opportunity to draw maximum DRAM power (at MAX_UCF) during the subsequent sampling interval of the new phase. This ensures that UPSCavenger doesn't constrain the performance of the newly detected phase based on the setpoint of the previous phase. If this maximum DRAM power (measured as PV_DRAM_power during the subsequent sampling interval) is greater than SP_DRAM_power', the system enters state 3 again. In this case, SP_DRAM_power is now intentionally set to the maximum DRAM power for the new phase.

After actuation, the control returns to the monitoring block of the control logic.

Sample Interval. We invoke the UPSCavenger agents periodically every 200ms in our experiments. Sampling under 200ms lead to high overheads. 200ms was the shortest sample interval that had negligible overheads of obtaining measurements, setting the uncore frequency while letting the change of frequency reflect in the next sampling interval. We conducted a sensitivity study by varying the sample interval from 200ms to 1s. We observed that our runtime system has negligible overheads at 200ms while a longer sampling interval makes it sluggish because of two reasons. First, it has fewer opportunities of detecting phase changes and modulating the uncore frequency. Second, it leads to longer reaction times. A sampling interval of 200ms is sufficiently long to actuate the uncore frequency modulation and observe its effect and it is sufficiently frequent to have a fast reacting runtime system. This threshold is driven by the overhead of UPSCavenger, not by any assumption about the length of phases in an application. Applications with even shorter phases would not benefit from UPSCavenger as they would not allow enough time to modulate the frequency.

4 IMPLEMENTATION AND EXPERIMENTAL FRAMEWORK

We conducted our experiments on a 108 node cluster of heterogeneous processors with Infiniband interconnect using 16 Broadwell nodes. Each node has two Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz processors and each processor has 8 cores. Typical HPC systems, like ours, provide exclusive node access. Each MPI job runs on uniquely allocated nodes. No two jobs (or applications) share nodes. We use all the cores of every socket in a job. We pin one MPI rank to a unique core on every socket. Typical MPI applications are NUMA-aware, i.e., all the cores access memory on the socket-local DRAM via the local memory channel consuming the local memory bandwidth. We observed that the remote memory bandwidth consumption is negligible and constant over time compared to the local memory bandwidth for all applications. Hence, we can use each socket's DRAM power as an approximation of the socket's uncore utilization.

UPSCavenger is a light-weight library that is with the application at the time of compilation. It is implemented leveraging the MPI standard profiling interface [38] (PMPI). The UPSCavenger agents are initiated in the wrapper function of the MPI_Init MPI call, which is the first MPI function call invoked by an MPI application. We use signals to set timers for periodic invocation of UPSCavenger agents on every socket in a job. The system is restored to the default state (maximum uncore frequency multiplier is set to 27) in the wrapper function of the MPI_Finalize MPI call, which is the final MPI call invoked by an MPI application indicating its completion. As UPSCavenger is a shared library that can be linked to the application and it leverages the PMPI interface for initialization and termination, no modifications to the application's source code are required to use UPSCavenger. Applications only need to be linked to our library by modifying their Makefile at the time of compilation.

At each invocation, a UPSCavenger agent periodically monitors the state of the system by measuring DRAM power consumption and IPC. For the DRAM power measurement, we leverage Intel's Running Average Power Limit (RAPL) [28]. From Sandy Bridge processors onward, Intel supports this interface that allows the programmer to measure (or constrain) the power consumption of a processor/socket/package (PKG) as well as the DRAM by reading from (or writing to) the RAPL MSR. We also use RAPL in some of our experiments to constrain the power consumption of the processors. RAPL has a power capping mechanism implemented in hardware that guarantees that the power consumption does not exceed the power limit specified by the user. The UPSCavenger agent measures IPC by reading the fixed counters [28].

Depending on the control calculations, the UPSCavenger agent modulates the uncore frequency by writing to the bits 0 to 7 of the UFS MSR addressed at 0X620 that encode the maximum uncore frequency multiplier. The minimum uncore frequency multiple is set to 12, as 1.2GHz is the minimum uncore frequency on our platform. The msr-safe kernel module installed on this cluster enabled us to read from and to write to Intel's RAPL and UFS MSRs and fixed counters indirectly via the libmsr library [48].

In Figure 7, we compare ΔDRAM_power and ΔIPC to absolute zero. However, in the implementation we use a realistic error margin of ±5% to avoid triggering the UPSCavenger agent for just minor fluctuations in the measurements.

UPSCavenger agents in a job do not need any global synchronization or communication with other agents. They monitor their respective sockets and make local decisions based on local knowledge about the system. Hence, UPSCavenger is a very light weight runtime system with only nominal overheads of reading and writing to MSRs that are several orders of magnitude lower than the total completion time of a typical HPC job.

5 EXPERIMENTAL EVALUATION ON A MULTI-NODE CLUSTER

For evaluation we used Embarrassingly Parallel (EP), Scalar Pentadiagonal solver (SP), Block Tri-diagonal solver (BT), Multi-Grid (MG) and Fourier Transform (FT) from the NAS parallel benchmark suite [4] and CoMD and MiniAMR from the ECP proxy applications suite [1]. We used input sizes C and D, 200x200x200 and 100X100x100 for NPB benchmarks, CoMD and miniAMR, respectively. We used the MPI version of these codes in our experiments. Our selection of applications ensures that we evaluate UPSCavenger

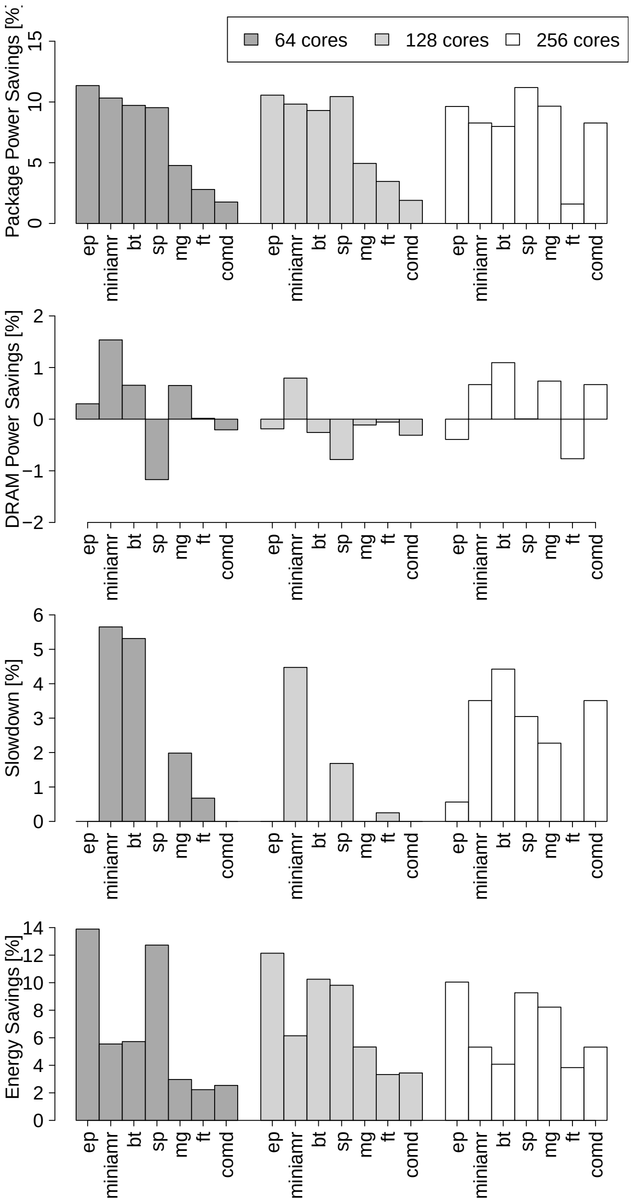


Figure 8: Package and DRAM power savings achieved with UPSCavenger and the resulting slowdowns and energy savings with respect to the baseline

with applications spanning across the spectrum from compute-intensive to memory-intensive. For example, EP is a CPU-bound application that uses uncore minimally, SP and BT use uncore (LLC) intensively with moderate memory (DRAM) access rates while other applications have high memory (DRAM) access rates. We report average package power savings over the duration of the job and slowdowns caused by the UPSCavenger runtime system with respect to the baseline. We chose the default configuration, where Intel's firmware modulates the uncore frequency, as the baseline. We also compare an application's performance with UPSCavenger to its performance with Intel's Running Average Power Limit (RAPL) with equal power consumption. Each experiment was repeated five times. We report averages across all repetitions with standard deviation of 1-5%.

Figure 8 depicts power savings achieved with UPSCavenger, resulting slowdowns and energy savings with respect to the baseline. We show results for 64, 128 and 256 core experiments as shown in the legend. X-axes represent the benchmarks and applications. Y-axes represent (top to bottom) package and DRAM power savings, slowdown and energy saving with respect to the baseline.

Observation 1: UPSCavenger achieves up to 11% package power savings.
For applications like EP, MiniAMR, BT and SP that have longer CPU-bound phases than others (such as MG, FT, CoMD), UPSCavenger achieves higher package power savings. These codes keep the cores busy and incur fewer LLC misses per second. As a result, they are not as dependent on long latency memory accesses and the uncore as the rest. Hence, the uncore frequency can be lowered for the CPU-intensive phases to save power.

Observation 2: UPSCavenger has a marginal impact on DRAM power.
This is a desired effect. UPSCavenger is designed to reduce uncore frequency so long as it does not impact DRAM utilization. We measure DRAM utilization in terms of DRAM power. Hence, a substantial decline in DRAM power is an indication of decline in DRAM utilization (unless a phase change occurs) which can lead to performance degradation.

Observation 3: UPSCavenger causes worst-case slowdown of 5.5%.
In some cases, UPSCavenger achieves power savings at the cost of marginal slowdown for a few applications. This includes the overhead of the runtime system, i.e., measuring DRAM and PKG power, IPC, calculating the feedback and setting appropriate uncore frequency. The observed performance degradation is under 5.5% across all codes with some codes experiencing no slowdown. The reported overhead includes the effect of throttling uncore frequency momentarily below the phase's optimal frequency before UPSCavenger detects performance degradation and recovers by incrementing it.

Observation 4: UPSCavenger achieves up to 14% energy savings.
UPSCavenger achieves energy savings of at least 5% across all applications and core counts, despite the slowdown incurred in some cases. UPSCavenger conserves energy by simply reducing the uncore's energy usage when it is underutilized, thus reducing energy wastage and shifting towards more energy efficient execution.

Observation 5: Intel's default scheme consists of a naive algorithm that sets the uncore frequency to its maximum.
Intel's default algorithm sets uncore frequency to 2.7GHz on detecting even the slightest uncore activity. Figure 9 depicts the uncore frequency profile for runs of EP with default configuration (top) and UPSCavenger (bottom). X-axes represent the timeline while y-axes represent the uncore frequency. EP is a CPU-intensive embarrassingly parallel benchmark that does not use the uncore at all. It operates on core-local cache and incurs nominal LLC misses per second. Even in case of such a workload, the default scheme sets the uncore frequency to 2.7GHz, which is the maximum for our specific architecture.

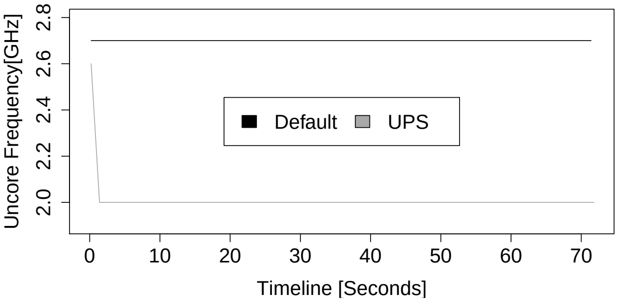


Figure 9: Uncore frequency profiles for EP with UPSCavenger and the default configuration.

Unlike this scheme, UPSCavenger intelligently modulates uncore frequency by making measurement-driven decisions at runtime based on DRAM power and IPC achieved by the socket. Our automatic and dynamic algorithm leads to a constant uncore frequency of 2GHz for the entire duration of EP. For other applications like CoMD and MiniAMR, the uncore frequency continuously changes.

Figure 10 and 11 depict job profiles of MiniAMR and CoMD, respectively, for 4 node (8 socket) runs. We show two sets of plots (left and right). Plots on the left are the job profiles with the default configuration (baseline) while those on the right are with UPSCavenger. X-axes represent the timeline while y-axes represent the uncore frequency, DRAM power, package power (top to bottom) for each of the sockets of the job. Each socket is represented by a unique color as depicted in the legend. With the default configuration, all sockets operate at the maximum uncore frequency (2.7GHz) constantly for both MiniAMR and CoMD. Hence, the uncore frequency profiles of all the sockets coincide. However, with UPSCavenger, the UPSCavenger agents dynamically modulate the uncore frequency of sockets.

Observation 6: UPSCavenger detects phase change and increases the uncore frequency to UCF_MAX leading to spikes in package power.
UPSCavenger detects phase changes based on deflection in DRAM power and IPC. Figure 10 shows the DRAM power profile for MiniAMR. At least five phase changes indicated by dips in DRAM power can be observed. When each of these phase changes occurs, UPSCavenger agents automatically detect it and reset the uncore frequency of their corresponding sockets to UCF_MAX. This leads to a spike in the package power consumption as the uncore consumes higher power when operating at a higher uncore frequency. Even with the default configuration, the power consumption varies across packages in a job. This is a result of manufacturing variation across processors [20, 27].

Observation 7: With UPSCavenger, packages spend more time at lower power than with the default configuration.
After the initial spike in package power for each newly detected phase, UPSCavenger gradually lowers the uncore frequency to the lowest level before it starts affecting DRAM power. At this low uncore frequency, the packages consume less power than at UCF_MAX (depicted in PKG power plots in Figure 10 and Figure 11), thus leading to power savings.

Observation 8: Even in case of codes with frequent phase changes, UPSCavenger saves power.
For applications such as CoMD depicted in Figure 11 that incur frequent phase changes with short-lived phases, UPSCavenger does not stabilize at a constant lower uncore frequency for an extended duration. Instead, it intelligently and automatically modulates the uncore frequency between several (higher and lower) levels. As a result, packages spend some time at frequencies lower than that selected by the default scheme. This leads to overall power savings over baseline as shown in Figure 8.

Observation 9: UPSCavenger outperforms Intel's RAPL.
Intel's RAPL is a power capping mechanism implemented in hardware that restricts the power consumption of a processor to a value specified in the RAPL MSR. Figure 12 depicts the comparison of UPSCavenger with RAPL. We first recorded the completion time and the resulting power consumption (say P Watts) of an application with UPSCavenger. We then measured the application's performance without UPSCavenger under a P Watt power bound enforced by restricting power consumption of the processors in the job using RAPL. Figure 12 depicts the total (package and DRAM) power savings obtained by UPSCavenger, resulting performance improvements and energy savings with respect to RAPL. We show results for 64, 128 and 256 core experiments (see legend). X-axes represent the benchmarks and applications. Y-axes represent (top to bottom) total power savings, speedup and energy saving with respect to RAPL. The power consumption with UPSCavenger is within ±2% of RAPL. Interestingly, UPSCavenger achieves up to 20% speedup over RAPL leading to proportional energy savings. This indicates that even for a power-constrained computing paradigm that aims at maximizing performance under a strict power budget, UPSCavenger outperforms RAPL.

Observation 10: In a power-constrained environment, UPSCavenger deducts power from the uncore without slowing the cores when possible while Intel's RAPL always deducts power from the cores.
This is illustrated in Figure 13, which depicts the effective per core frequency achieved with RAPL and with UPSCavenger for equivalent power consumption of BT. We first recorded the effective core frequency measured via Intel's APERF and MPERF MSRs [28] and the resulting power consumption (say P Watts) of an application with UPSCavenger. The effective frequency is calculated as $\frac{\Delta APERF*BASE\_FREQUENCY}{\Delta MPERF}$, where the base frequency on our architecture is 2.1GHz. We then measured the effective core frequency without UPSCavenger under a P Watt power bound enforced by restricting power consumption of the processors in the job using RAPL. The x-axis represents the timeline while the y-axis represents the effective core frequency.

RAPL lowers the core frequency from 2.3GHz down to 1.8GHz to save power, which leads to performance degradation. UPSCavenger achieves equivalent power savings by opportunistically reducing the uncore frequencies without affecting the core frequencies allowing the cores to operate at the maximum frequency of 2.3GHz.

6 RELATED WORK

Chip manufactures have provided various knobs to modulate power directly, e.g., power capping, or indirectly, e.g., dynamic voltage

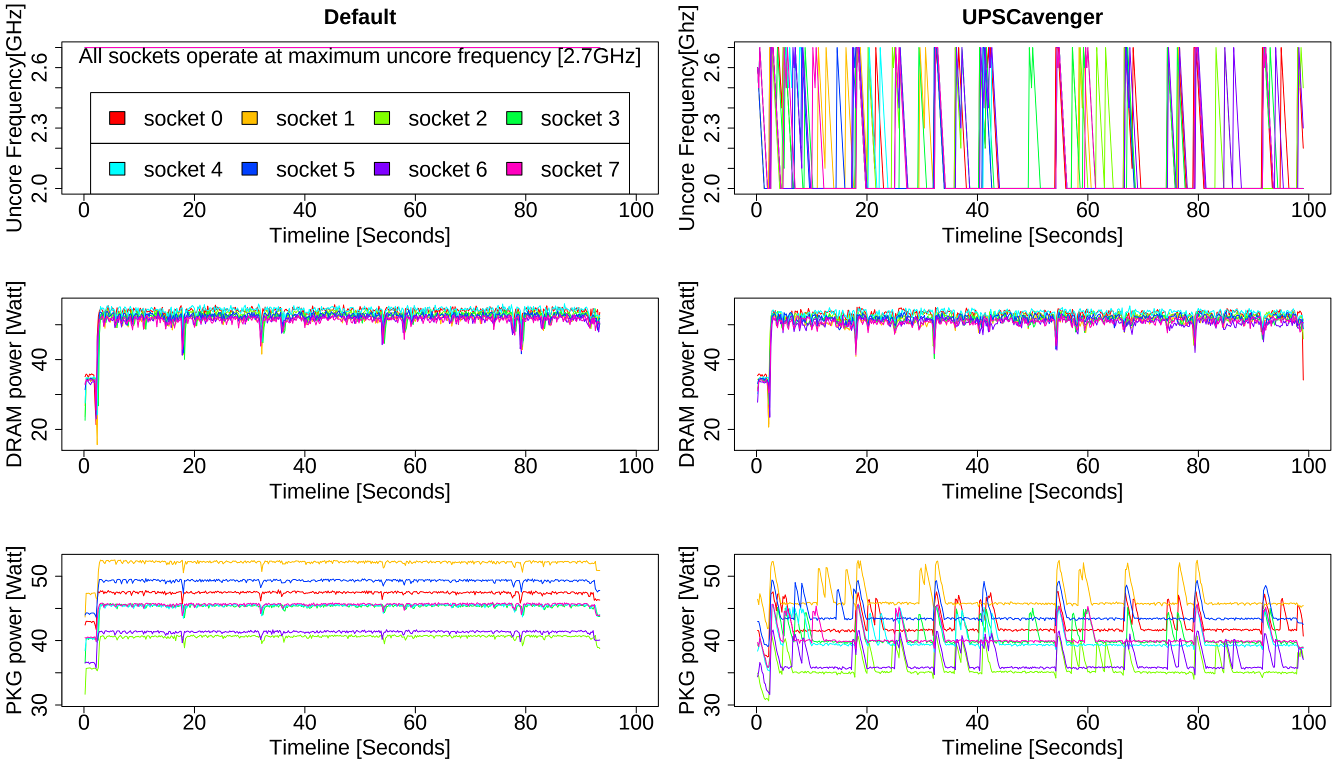


Figure 10: Power and uncore frequency profiles for 8 socket runs of MiniAMR with default configuration (left) and UPSCavenger (right).

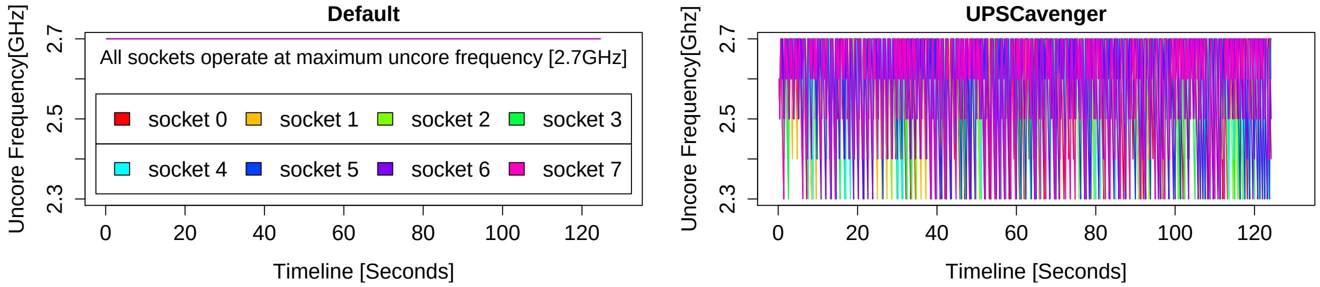


Figure 11: Uncore frequency profiles for 8 socket runs of CoMD with default configuration (left) and UPSCavenger (right).

frequency scaling (DVFS), of various components of a server. Conventionally, DVFS has been used to modulate the frequency of cores [28]. Intel processors prior to Haswell maintained all cores at a common operating state, i.e., voltage and frequency. Starting with Haswell, Intel introduced per core DVFS. From Sandy Bridge processors onward, Intel introduced running average power limit (RAPL) [11, 28], a power capping mechanism implemented in hardware that constrains the average collective power consumption of the cores and the uncore of a processor. A plethora of solutions [6, 16–18, 20, 21, 26, 33, 41, 42, 50] have been proposed that leverage these knobs to improve the power efficiency of a system. DVFS-based solutions were oblivious of the power consumption of the uncore, which is expected to be a growing component in the future generations of processors [34]. RAPL-based solutions off-loaded the management of the uncore’s power to Intel’s RAPL implementation, which we show is sub-optimal.

Early work exploited DVFS to reduce CPU frequencies during idle time, e.g., due to early arrival at MPI barriers and collectives [30]. An ILP-based approach to model energy [41] was demonstrated to be effective during the runtime of MPI codes to determine optimal power levels for the cores [42]. These works were trying to conserve energy without sacrificing performance by much when utilizing just one core of a node resulting in underutilization of the system. Memory DVFS [10] was also proposed as a technique to conserve energy for workloads that are not constrained in memory bandwidth. UPSCavenger differs in that its foremost objective is

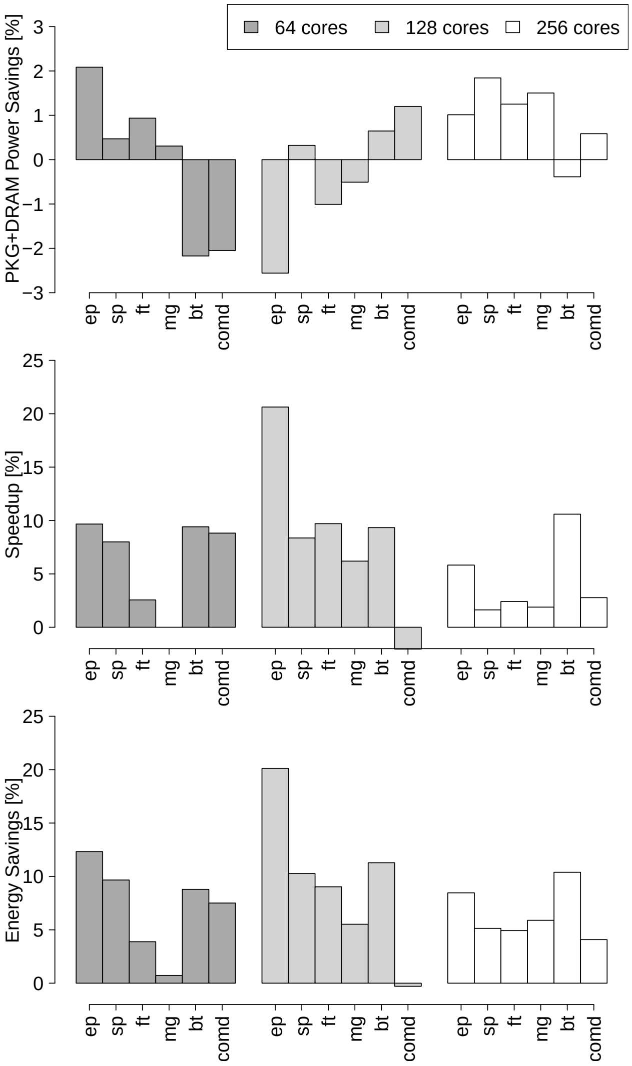


Figure 12: Package and DRAM power savings, speedups and energy savings achieved by UPSCavenger with respect to RAPL

to reduce the inefficient power consumption of the uncore of each processor of an HPC job without reducing system utilization, i.e., using all cores of a processor. It conserves power and achieves energy savings by dynamically modulating the frequency of the uncore based on real-time feedback from the system.

Prior work [9, 32] leveraged the effect of concurrency throttling and thread locality to save power and increase performance. An ILP-based runtime approach has been shown to determine how many cores an application should be run on to stay within a given power budget [31, 55]. Marathe et al. [36] proposed Conductor, a runtime system that speeds up an application's critical path through an adaptive socket power-allocation algorithm that periodically performs dynamic voltage frequency scaling (DVFS) and dynamic concurrency throttling (DCT) based on application behavior and power usage. Our work differs in terms of granularity and adaptivity. First, we reduce uncore power across resources at processor chip level, exploiting dynamically adaptive feedback methods. Second, UPSCavenger does not modify the concurrency or the thread locality, which can be tightly coupled with the application's input.

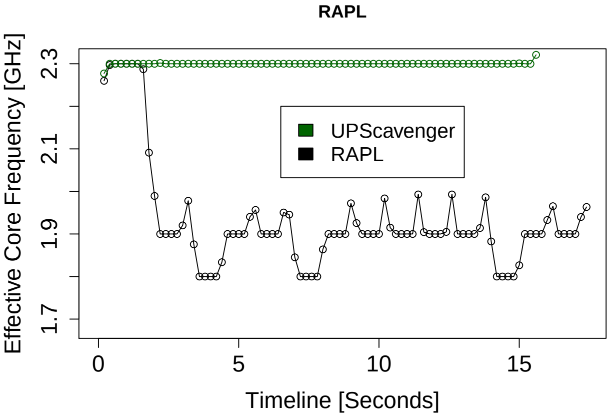


Figure 13: Effective core frequency profiles for BT with RAPL and UPSCavenger for equal power consumption.

In recent work, Gholkar et al. [20] presented PTune, a process variation-aware power tuner that uses performance characterization data for all sockets on a cluster and application execution characteristics to minimize the runtime of a job under its power budget. It assumes that the jobs are moldable, i.e., the number of MPI ranks can be varied at the time of job scheduling. UPSCavenger does not require jobs to be moldable. It also does not require any prior characterization data to make uncore frequency decisions at runtime. In [21], they proposed PShifter, a feedback-based mechanism that shifts power from processors idling at barriers and collectives to the processors on the critical path to improve performance under a power constraint. PShifter does not strive to save power. Its control calculations depend on the utilization of the cores while being oblivious of the uncore's utilization. Our approach is a dynamic closed-loop feedback controller that specifically monitors the uncore and depending on its utilization reduces any wasteful power consumption by modulating the uncore frequency at runtime. Ellsworth et al. proposed Pow, a system-wide power manager that re-distributed the power budget across a system via scheduling while adhering to a global machine-level power cap [13]. This approach salvaged the unused power but did not address inefficient power usage. Our approach goes one step further by detecting and reducing the wasteful power consumption within a job.

Capacity-improving schemes that increase job throughput have been developed under power limitations by exploiting "hardware overprovisioning", i.e., by deploying more nodes that will be powered at a time [14, 15, 39, 45, 46]. In such a system, the characteristics of a code under strong scaling were used to calculate the optimal number of processors considering core and memory power [47]. Modifications to the batch scheduler in how small jobs are backfilled depending on their power profile can further increase job capacity [40]. While these solutions aimed at achieving performance or throughput improvement while utilizing as much of the power budget as possible, our approach aims at conserving power by reducing wasteful power consumption with marginal impact on performance. We do not explicitly aim at *maximizing* the machine's throughput, which is beyond the scope of this paper, but our runtime reduces the overall power and energy footprint of the jobs making more power available for more new jobs to be scheduled on an overprovisioned system.

Hackenberg et al. [24] observed that the uncore frequency depends on the core frequency even when there is no uncore activity. Unlike UPSCavenger, which dynamically modulates the uncore frequency using the measurement-driven closed-loop feedback controller, Sundriya et al. [52, 53] propose a runtime approach that relies on a static model for estimating the frequency. More recent work [54] proposed a dynamic core and uncore frequency scaling strategy that partially relied on regression models using prior data to make dynamic decisions. UPSCavenger is independent of such dependencies and overheads of prior characterization. Von et al. [57] proposed a neural network-based uncore frequency scaling approach that was simulated for chip multi-core (CMP) power management. Da-Cheng et al. [29] simulated a semi-supervised reinforcement learning based approach for dynamic frequency scaling to maximize performance under a power budget. Unlike this work, we implement and evaluate our runtime system on real hardware, i.e., a Broadwell cluster. Our runtime is suitable for power management of a multi-node and a multi-core system.

Our UPSCavenger work is unique in that it is the first of its kind that targets the otherwise neglected uncore component of a chip. It dynamically modulates the frequency of the uncore depending on its utilization measured at runtime. It automatically detects new phases within an application and resets the uncore frequency for each phase. Proposed frameworks like Redfish, the PowerAPI, and Intel's GEOPM [2, 3, 12, 22] can integrate UPSCavenger as a unique closed-loop feedback-based policy for job power management on a cluster. UPSCavenger will also relieve the application developers of the burden to explicitly indicate phase changes as required by the APIs (like GEOPM) as UPSCavenger automatically detects phases without any explicit information from the developer.

7 SUMMARY

We explored uncore frequency scaling and its impact on performance, power and energy consumption of various HPC applications. To the best of our knowledge, this is the first study of uncore frequency scaling conducted on Broadwell processors. We proposed UPSCavenger, a runtime system that automatically modulates the uncore frequency to conserve power without significant performance degradation. As a part of UPSCavenger, we also introduced an algorithm that automatically detects phase changes at runtime. Our evaluations indicate that UPSCavenger achieves up to 10% energy reduction with less than 1% slowdown. It achieves up to 14% energy savings with a worst case slowdown of 5.5% compared to the default configuration. We also show that UPSCavenger achieves up to 20% speedup with proportional energy savings compared to Intel's RAPL with equivalent power usage. While UPSCavenger can be easily deployed as a runtime on any HPC system, we intend to collaborate with semiconductor manufacturers to incorporate this into their firmware.

8 ACKNOWLEDGEMENTS

We extend our thanks to Livermore Computing for facilitating Dedicated Access Time on the LLNL cluster. This work was supported in part by NSF grants 1525609, 1058779, 1217748, 0958311, and by Consortium for Advanced Simulation of Light Water Reactors (CASL). This material is also based upon work supported by the U.S. Department of Energy's Lawrence Livermore National Laboratory. Office of Science, under Award number DE-AC52-07NA27344 and supported by Office of Science, Office of Advanced Scientific Computing Research (LLNL-CONF-656877).

REFERENCES

[1] Ecp proxy apps suite. https://proxyapps.exascaleproject.org/ecp-proxy-apps-suite/.

[2] Panel on power api/redfish/geopm. https://eehpcwg.llnl.gov/documents/conference/sc16/SC16_Laros_Eastep_Benson_API.pdf.

[3] Redfish api. https://www.dmtf.org/standards/redfish/.

[4] NASA Advanced Supercomputing Division, NAS Parallel Benchmark Suite v3.3. 2006. http://www.nas.nasa.gov/Resources/Software/npb.html.

[5] Top500 Supercomputer Sites. 2017. https://www.top500.org/lists/2017/11/.

[6] P. E. Bailey, A. Marathe, D. K. Lowenthal, B. Rountree, and M. Schulz. Finding the limits of power-constrained application performance. In *SC*, 2015.

[7] S. Bhalachandra, A. Porterfield, S. L. Olivier, and J. F. Prins. An adaptive core-specific runtime for energy efficiency. In *IPDPS*, pages 947–956, May 2017.

[8] H.-Y. Cheng, J. Zhan, J. Zhao, Y. Xie, J. Sampson, and M. J. Irwin. Core vs. uncore: The heart of darkness. In *Proceedings of the 52Nd Annual Design Automation Conference*, DAC '15, pages 121:1–121:6, New York, NY, USA, 2015. ACM.

[9] M. Curtis-Maury, J. Dzierwa, C. D. Antonopoulos, and D. S. Nikolopoulos. Online power-performance adaptation of multithreaded programs using hardware event-based prediction. In *Supercomputing*, pages 157–166, 2006.

[10] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu. Memory power management via dynamic voltage/frequency scaling. In *Proceedings of the 8th ACM International Conference on Autonomic Computing*, ICAC '11, pages 31–40, New York, NY, USA, 2011. ACM.

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

[12] J. Eastep, S. Sylvester, C. Cantalupo, B. Geltz, F. Ardanaz, A. Al-Rawi, K. Livingston, F. Keceli, M. Maiterth, and S. Jana. Global extensible open power manager: A vehicle for hpc community collaboration on co-designed energy management solutions. In J. M. Kunkel, R. Yokota, P. Balaji, and D. Keyes, editors, *High Performance Computing*, pages 394–412, Cham, 2017. Springer International Publishing.

[13] D. A. Ellsworth, A. D. Malony, B. Rountree, and M. Schulz. Pow: System-wide dynamic reallocation of limited power in hpc. In *High-Performance Parallel and Distributed Computing*, pages 145–148, 2015.

[14] M. E. Femal and V. Freeh. Boosting data center performance through non-uniform power allocation. In *International Conference on Autonomic Computing*, 2005.

[15] M. E. Femal and V. W. Freeh. Safe overprovisioning: using power limits to increase aggregate throughput. In *In International Conference on Power-Aware Computer Systems*, December 2005.

[16] V. Freeh, F. Pan, N. Kappiah, and D. K. Lowenthal. Using multiple energy gears in mpi programs on a power-scalable cluster. In *PPoPP*, pages 164–173, June 2005.

[17] V. Freeh, F. Pan, N. Kappiah, D. K. Lowenthal, and R. Springer. Exploring the energy-time tradeoff in mpi programs on a power-scalable cluster. In *IPDPS*, May 2005.

[18] R. Ge, X. Feng, W. c. Feng, and K. W. Cameron. Cpu miser: A performance-directed, run-time system for power-aware clusters. In *2007 International Conference on Parallel Processing (ICPP 2007)*, pages 18–18, Sept 2007.

[19] P. Gepner and M. F. Kowalik. Multi-core processors: New way to achieve high system performance. In *International Symposium on Parallel Computing in Electrical Engineering (PARELEC'06)*, pages 9–13, Sept 2006.

[20] N. Gholkar, F. Mueller, and B. Rountree. Power tuning hpc jobs on power-constrained systems. In *PACT*. ACM, 2016.

[21] N. Gholkar, F. Mueller, B. Rountree, and A. Marathe. Pshifter: Feedback-based dynamic power shifting within hpc jobs for performance. In *HPDC*. ACM, 2018.

[22] R. E. Grant, M. Levenhagen, S. L. Olivier, D. DeBonis, K. T. Pedretti, and J. H. L. III. Standardizing power monitoring and control at exascale. *Computer*, 49(10):38–46, Oct 2016.

[23] V. Gupta, P. Brett, D. Koufaty, D. Reddy, S. Hahn, K. Schwan, and G. Srinivasa. The forgotten 'uncore': On the energy-efficiency of heterogeneous cores. In *Proceedings of the 2012 USENIX conference on Annual Technical Conference*, June 2012.

[24] D. Hackenberg, R. SchÃűne, T. Ilsche, D. Molka, J. Schuchart, and R. Geyer. An energy efficiency feature survey of the intel haswell processor. In *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*, pages 896–904, May 2015.

[25] D. L. Hill, D. Bachand, S. Bilgin, R. Greiner, P. Hammarlund, T. Huff, S. Kulick, and R. Safraneki. The uncore: A modular approach to feeding the high-performance cores. In *Intel Technology Journal*, volume 14, 2010.

[26] C.-h. Hsu and W.-c. Feng. A power-aware run-time system for high-performance computing. In *Supercomputing*, page 1, 2005.

[27] Y. Inadomi, T. Patki, K. Inoue, M. Aoyagi, B. Rountree, M. Schulz, D. Lowenthal, Y. Wada, K. Fukazawa, M. Ueda, M. Kondo, and I. Miyoshi. Analyzing and mitigating the impact of manufacturing variability in power-constrained supercomputing. In *SC*, pages 78:1–78:12, 2015.

[28] Intel. Intel-64 and IA-32 Architectures Software Developer's Manual, Volumes 3A and 3B: System Programming Guide. 2011.

[29] D.-C. Juan and D. Marculescu. Power-aware performance increase via core/uncore reinforcement control for chip-multiprocessors. In *Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design*, pages 97–102, 2012.

[30] N. Kappiah, V. Freeh, and D. K. Lowenthal. Just in time dynamic voltage scaling: Exploiting inter-node slack to save energy in mpi programs. In *SC*, Nov 2005.

[31] A. Langer, E. Totoni, U. S. Palekar, and L. V. Kalé. Energy-efficient computing for hpc workloads on heterogeneous manycore chips. In *Proceedings of Programming Models and Applications on Multicores and Manycores*. ACM, 2015.

[32] D. Li, B. R. de Supinski, M. Schulz, K. W. Cameron, and D. S. Nikolopoulos. Hybrid mpi/openmp power-aware computing. In *IPDPS*, volume 10, pages 1–12, 2010.

[33] M. Y. Lim, V. Freeh, and D. K. Lowenthal. Adaptive, transparent frequency and voltage scaling of communication phases in mpi programs. In *SC*, 2006.

[34] G. H. Loh. The cost of uncore in throughput-oriented many-core processors. In *In Proc. of Workshop on Architectures and Languages for Troughput Applications (ALTA)*.

[35] K. Ma and X. Wang. Pgcapping: Exploiting power gating for power capping and core lifetime balancing in cmps. In *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*, PACT '12, pages 13–22, New York, NY, USA, 2012. ACM.

[36] A. Marathe, P. E. Bailey, D. K. Lowenthal, B. Rountree, M. Schulz, and B. R. de Supinski. A run-time system for power-constrained hpc applications. In J. M. Kunkel and T. Ludwig, editors, *High Performance Computing*, pages 394–408, Cham, 2015. Springer International Publishing.

[37] D. Meisner, C. M. Sadler, L. A. Barroso, W.-D. Weber, and T. F. Wenisch. Power management of online data-intensive services. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, pages 319–330, New York, NY, USA, 2011. ACM.

[38] S. Mintchev and V. Getov. Pmpi: High-level message passing in fortran77 and c. In B. Hertzberger and P. Sloot, editors, *High-Performance Computing and Networking*, pages 601–614, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[39] T. Patki, D. K. Lowenthal, B. Rountree, M. Schulz, and B. R. de Supinski. Exploring Hardware Overprovisioning in Power-constrained, High Performance Computing. In *International Conference on Supercomputing*, pages 173–182, 2013.

[40] T. Patki, D. K. Lowenthal, A. Sasidharan, M. Maiterth, B. Rountree, M. Schulz, and B. R. de Supinski. Practical Resource Management in Power-Constrained, High Performance Computing. In *HPDC*, 2015.

[41] B. Rountree, D. K. Lowenthal, S. Funk, V. Freeh, B. R. de Supinski, and M. Schulz. Bounding energy consumption in large-scale mpi programs. In *SC*, Nov 2007.

[42] B. Rountree, D. K. Lowenthal, M. Schulz, V. Freeh, and T. Bletsch. Adagio: Making dvs practical for complex hpc applications. In *ICS*, Nov 2009.

[43] A. Samih, R. Wang, A. Krishna, C. Maciocco, C. Tai, and Y. Solihin. Energy-efficient interconnect via router parking. In *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, pages 508–519, Feb 2013.

[44] Sandia National Laboratory. Mantevo project home page. https://software.sandia.gov/mantevo, June 2011.

[45] O. Sarood. *Optimizing Performance Under Thermal and Power Constraints for HPC Data Centers*. PhD thesis, University of Illinois, Urbana-Champaign, Dec 2013.

[46] O. Sarood, A. Langer, A. Gupta, and L. V. Kale. Maximizing throughput of overprovisioned hpc data centers under a strict power budget. In *SC*, 2014.

[47] O. Sarood, A. Langer, L. V. Kale, B. Rountree, and B. de Supinski. Optimizing Power Allocation to CPU and Memory Subsystems in Overprovisioned HPC Systems. In *Proceedings of IEEE Cluster 2013*, Sept 2013.

[48] K. Shoga, B. Rountree, M. Schulz, and J. Shafer. Whitelisting msrs with msr-safe. In *3rd Workshop on Extreme-Scale Programming Tools at SC*, Nov. 2014.

[49] C. A. Smith and A. Corripio. Chapter 6: Control systems and their basic components. In *Principles and Practice of Automatic Process Control*, pages 154–192.

[50] R. Springer, D. K. Lowenthal, B. Rountree, , and V. Freeh. Minimizing execution time in MPI programs on an energy-constrained, power-scalable cluster. In *PPoPP*, May 2006.

[51] B. Subramaniam and W.-c. Feng. Towards energy-proportional computing for enterprise-class server workloads. In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, ICPE '13, pages 14–26, NY, USA, 2013. ACM.

[52] V. Sundriyal, M. Sosonkina, B. M. Westheimer, and M. Gordon. Uncore frequency scaling vs dynamic voltage and frequency scaling: A quantitative comparison. In *Society for Modeling & Simulation International*, SpringSim-HPC, 2018.

[53] V. Sundriyal, M. Sosonkina, B. M. Westheimer, and M. S. Gordon. Comparisons of core and uncore frequency scaling modes in quantum chemistry application GAMESS. In *Proceedings of the High Performance Computing Symposium, SpringSim (HPC) 2018, Baltimore, MD, USA, April 15-18, 2018*, pages 13:1–13:11, 2018.

[54] V. Sundriyal, M. Sosonkina, B. M. Westheimer, and M. S. Gordon. Core and uncore joint frequency scaling strategy. In *Journal of Computer and Communications, 6*, pages 184–201, 2018.

[55] E. Totoni, A. Langer, J. Torrellas, and L. Kale. Scheduling for hpc systems with process variation heterogeneity. In *Technical Report YCS-2009-443, Department of Computer Science, University of York*, 2014.

[56] V. M. Weaver and S. A. McKee. Can hardware performance counters be trusted? In *2008 IEEE International Symposium on Workload Characterization*, pages 141–150, Sept 2008.

[57] J. Y. Won, X. Chen, P. Gratz, J. Hu, and V. Soteriou. Up by their bootstraps: Online learning in artificial neural networks for cmp uncore power management. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, pages 308–319, Feb 2014.