Co-UP: Comprehensive Core and Uncore Power Management for Latency-Critical Workloads

Ki-Dong Kang*, Gyeongseo Park*, and Daehoon Kim†‡
Future Computing Research Division, *ETRI*, Daejeon, Republic of Korea
†*Department of Systems Semiconductor Engineering*, *Yonsei University*, Seoul, Republic of Korea
{kd_kang, gspark}@etri.re.kr, daehoonkim@yonsei.ac.kr

***Abstract*—Improving energy efficiency to reduce costs in server environments has attracted considerable attention. Considering that processors account for a significant portion of energy consumption in servers, Dynamic Voltage and Frequency Scaling (DVFS) enhances their energy efficiency by adjusting the operational speed and power consumption of processors. Additionally, modern high-end processors extend DVFS functionality not only to core components but also to uncore parts. This is because the increasing complexity and integration of System on Chips (SoCs) have highlighted the substantial energy consumption. However, existing uncore voltage/frequency scaling fails to effectively consider Latency-Critical (LC) applications, leading to sub-optimal energy efficiency or degraded performance. In this paper, we introduce `Co-UP`, power management that simultaneously scales core and uncore frequencies for latency-critical applications, designed to improve energy efficiency without violating Service Level Objectives (SLOs). To this end, `Co-UP` incorporates a prediction model that estimates outcomes of energy consumption and performance as uncore and core frequency changes. Based on the estimated gains, `Co-UP` adjusts to uncore and/or core frequencies to further enhance energy efficiency or performance. This predictive model can rapidly adapt to new and unlearned loads, enabling `Co-UP` to operate online without any prior profiling. Our experiments show that `Co-UP` can reduce energy consumption by up to 28.2% compared to existing Intel's policy and up to 17.6% compared to state-of-the-art power management studies, without SLO violations.**

***Index Terms*—Power Management, Dynamic Voltage and Frequency Scaling, Uncore Frequency Scaling, Latency-Critical Applications.**

I. Introduction

Data center costs exceed $200 billion annually [1], driving research into cost reduction. Recent data center strategies involve deploying more servers than power limits allow, shutting some down during high loads to prevent outages [1]–[4]. In such environments, improving server energy efficiency plays a crucial role in enabling the operation of additional servers within a single data center.

Processors remain major energy consumers in servers, making power management techniques important for enhancing energy efficiency. Dynamic Voltage and Frequency Scaling (DVFS) is a representative technique in processor power management that adjusts the core's operational speed and power consumption to align with processing demands. Modern processors not only support core-level DVFS but also extend the power management technique to uncore components, such as on-chip interconnects and off-chip memory access. As the proportion of power consumption attributed to uncore elements increases, such as through increased cache sizes and more complex on-chip interconnects, the significance of uncore frequency scaling becomes increasingly critical [5].

To dynamically determine efficient uncore frequency, Intel implements a policy that adjusts uncore frequency based on hardware-level metrics. However, since this policy overlooks system- or application-level metrics (*e.g.,* packet response latency), it may lead to unnecessarily high uncore frequency configurations, resulting in increased energy consumption without improving application performance. Furthermore, the lack of consideration can result in excessively low uncore frequency configurations, which may significantly degrade application performance, preventing it from reaching its target performance.

To address the inefficiency of uncore frequency scaling, existing studies have employed methods such as monitoring DRAM power and Instructions Per Second (IPS) [6], or observing average memory access times to guide uncore frequency scaling [7]. However, these studies often overlook the specific characteristics of applications, which can result in performance degradation or suboptimal energy efficiency. Particularly, latency-critical applications have to maintain predefined Service Level Objectives (SLOs) under varying loads, with tail latency (*e.g.*, 99th percentile latency).

Existing studies considering SLOs offer additional opportunities to improve energy efficiency without SLO violations by exploiting DVFS [8]–[10]. `Rubik` [8] dynamically selects the minimum frequency that satisfies the SLOs using a statistical model, while `PEGASUS` [9] adjusts frequency with a feedback controller. `Parties` [10], on the other hand, performs frequency scaling based on the slack, defined as the difference between the SLO and the current tail latency value. However, the limitation of these studies is their exclusion of uncore frequency scaling, which significantly contributes to energy consumption in processors. This results in only suboptimal levels of energy efficiency.

* These authors contributed equally to this work.
‡ Corresponding author.

This research was supported in part by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2018-0-00503, Researches on next generation memory-centric computing system architecture), Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2022-0-00498, Development of high-efficiency AI computing SW core technology for high-speed processing of large learning models), Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2024-00396013, DRAM PIM Hardware Architecture for LLM Inference Processing with Efficient Memory Management and Parallelization Techniques), and National Research Foundation of Korea (NRF) grants funded by the Korean government (MSIT) under Grant (No. 2018R1A5A1060031).

978-3-9826741-0-0/DATE25/© 2025 EDAA

After analyzing the impact of uncore frequency scaling on tail latency and energy consumption in latency-critical applications, we propose Co-UP (comprehensive **Co**re and **U**ncore **P**ower management), which dynamically adjusts core and uncore frequencies to reduce energy consumption without violating SLOs. To this end, Co-UP exploits a prediction model to estimate tail latency and energy consumption outcome for core/uncore frequency configuration candidates on the current application's load. Then, based on a simple greedy algorithm, Co-UP selects and enforces the core and uncore frequency configuration with the lowest energy consumption that do not violate the SLO. The predictive model can rapidly adapt to new and unlearned loads, allowing Co-UP to operate online without requiring any prior profiling. In our evaluation, we demonstrate that Co-UP reduces energy consumption by up to 28.2% compared to the existing Intel policy and by up to 17.6% relative to the state-of-the-art DVFS approach while satisfying SLOs.

To the best of our knowledge, this study is the first to investigate uncore frequency scaling targeted at latency-critical services. The contributions of this research are as follows:

1) We demonstrate the impact of uncore frequency scaling on both performance and energy consumption in latency-critical applications.
2) We propose a power management, 'Co-UP', that utilizes a greedy algorithm based on a prediction model to control both core and uncore frequency scaling.
3) In a real-world experimental setup, we show that Co-UP achieves more energy reduction compared to existing Intel policies and state-of-the-art studies while satisfying SLOs of the latency-critical application.

II. Background

A. Dynamic Voltage and Frequency Scaling

Dynamic Voltage and Frequency Scaling (DVFS) is a representative power management technology for processors, which simultaneously controls the operating performance and power consumption at runtime. The current Linux OS offers an ACPI cpufreq driver to facilitate the use of DVFS technology, incorporating various policies (*i.e.*, governors) [11]. Each governor adjusts the processor's voltage and frequency (V/F) using its own methodology, and these governors include ondemand, performance, userspace, and powersave.

The performance governor ensures the frequency remains at its maximum value, while the powersave governor keeps it at its minimum value. The userspace governor sets and maintains a user-defined frequency value. Finally, the ondemand governor, which is the default governor for the current ACPI cpufreq driver, periodically measures the core's utilization (*e.g.*, every 10 ms). Subsequently, it adjusts the frequency value in proportion to the core utilization. This means that when the core utilization is high, it sets a high frequency value, and vice versa.

B. Uncore Frequency Scaling

Many modern processors offer frequency scaling not only for the core area but also for the uncore area. The core area includes the processor cores and private caches (*e.g.*, L1 and L2 caches in Intel processors), whereas the uncore area encompasses the Last-Level Cache (LLC), Integrated Memory Controller (IMC), on-chip interconnect, UltraPath Interconnect (UPI), and other components. With larger cache sizes, increasingly complex on-chip interconnects, and a greater number of components integrated into the System on Chip (SoC), the share of power consumption attributable to uncore elements grows, emphasizing the importance of uncore frequency scaling [5].

Intel processors offer the hardware-level uncore frequency scaling policy to improve energy efficiency in the uncore area. In addition, Intel processors also provide an interface for manual uncore frequency adjustment by writing to a specific register, instead of relying on the default uncore frequency scaling policy.

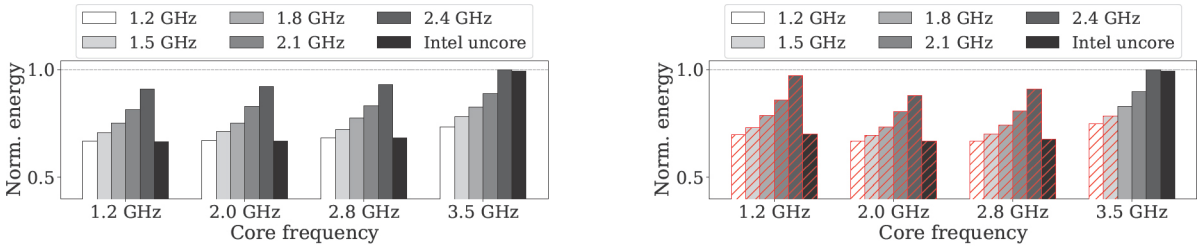
III. Motivation

In this section, we analyze the impact of uncore frequency scaling on the tail latency (*i.e.*, 99th percentile latency) of latency-critical applications and the energy consumption of the processor. To this end, we run representative latency-critical applications, memcached and nginx, on a server machine while generating load from the separate client machine. Similar to prior studies, the SLOs for memcached [12] and nginx [13] are set at 1 ms and 10 ms, respectively, using the 99th percentile latency from the latency-load curve [9]. A more detailed description of the experimental setup is provided in Section V.

A. Impact of uncore frequency scaling in latency-critical applications

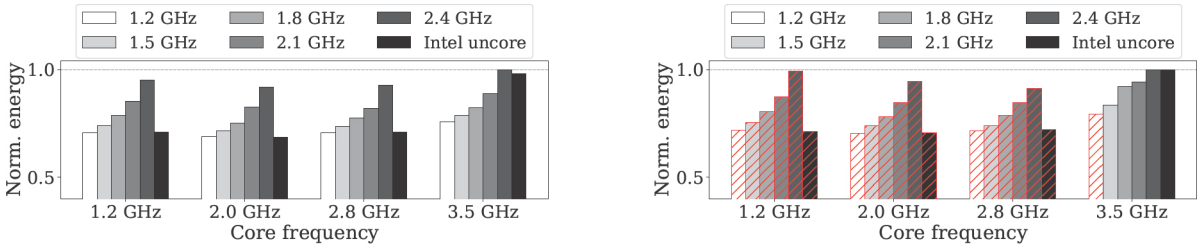
Figure 1 and Figure 2 show the impact of core and uncore frequency adjustments on energy consumption in memcached and nginx, respectively. The y-axis represents the processor's energy consumption, normalized to the value when both the core and uncore frequencies are set to their maximum (*i.e.*, 3.5 GHz for the core frequency and 2.4 GHz for the uncore frequency). Each application is evaluated under two load scenarios, low and high, which are represented by 133 K Queries Per Second (QPS) and 800 KQPS for memcached, and by 6 KQPS and 120 KQPS for nginx. The x-axis depicts core frequency adjustments, with each bar representing the changes in uncore frequency. Furthermore, the uncore frequency settings include Intel's default uncore frequency scaling results (marked as 'Intel uncore' in the legend)." Finally, a bar with a red edge and diagonal stripes represents an SLO violation.

As shown in Figure 1a and Figure 2a, no SLO violations occur in either memcached or nginx under low load, regardless of the core and uncore frequency settings. In contrast, as depicted in Figure 1b and Figure 2b, under high load, SLO violations occur depending on the core and uncore frequency configurations. For instance, as shown in Figure 1b, reducing the core frequency from 3.5 GHz to 2.8 GHz in memcached under high load leads to SLO violations, regardless of the uncore frequency values. Additionally, even with a core frequency set to 3.5 GHz, SLO violations occur when the uncore frequency is set to 1.5 GHz or lower. In terms of energy consumption, it is generally observed that lower core and uncore frequencies lead to reduced energy consumption, with uncore frequency scaling, in particular, contributing significantly to differences in total energy consumption. In summary, not only the core frequency



(a) low load. (b) high load.

Fig. 1. Normalized energy consumption with SLO violation (memcached).



(a) low load. (b) high load.

Fig. 2. Normalized energy consumption with SLO violation (nginx).

but also the uncore frequency has a significant impact on the tail latency of latency-critical applications and on the processor's energy consumption.

B. Limitation of existing uncore power management

As demonstrated in the previous subsection, uncore frequency scaling has a significant impact on energy consumption. However, Intel's default uncore frequency scaling policy currently operates only at the hardware level, without considering software states. For latency-critical applications based on client-server architecture, this is likely to lead to one of two potential issues: (1) unnecessarily high uncore frequency settings, leading to reduced energy efficiency, or (2) excessively low uncore frequency settings, leading to SLO violations.

As depicted in Figure 1 and Figure 2, Intel's default uncore frequency scaling policy, within our experimental setup, almost always sets the uncore frequency to its highest value when the core frequency is at its maximum (*i.e.*, 3.5 GHz). As a result, regardless of the load or application, Intel's uncore frequency scaling results in energy consumption nearly equivalent to the processor's maximum uncore frequency of 2.4 GHz when the core frequency is set at 3.5 GHz. On the other hand, at all other core frequencies, Intel's uncore frequency scaling sets the uncore frequency to nearly the minimum value of 1.2 GHz, resulting in energy consumption outcomes similar to when the uncore frequency is fixed at 1.2 GHz.

Even if Intel's hardware-based uncore frequency scaling policy becomes more sophisticated, it fundamentally provides only suboptimal energy efficiency in latency-critical applications without considering software states like current load or SLOs. Typically, power management for latency-critical applications focuses on enhancing energy efficiency by choosing the lowest frequency that does not violate the SLO, even as the load changes [8]–[10], [14], [15]. Although there have been attempts to improve Intel's uncore frequency scaling policy [6], [16], none of them consider software requirements, leading to suboptimal energy consumption or SLO violations in latency-critical applications. In the evaluation section, we provide a comparison with state-of-the-art uncore frequency scaling [6].

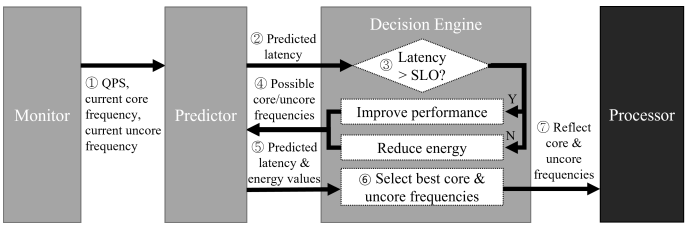


Fig. 3. The overview of Co-UP architecture.

IV. Architecture

In this paper, we propose Co-UP (comprehensive **Co**re and **U**ncore **P**ower management), which scales both core and uncore frequencies for latency-critical applications. Co-UP improves energy efficiency without violating SLOs by monitoring current core and uncore frequency values, along with the load of the latency-critical application. Unlike previous studies that only consider core frequency scaling, Co-UP simultaneously adjusts the two factors that commonly affect tail latency and energy consumption (*i.e.*, core and uncore frequency) using a prediction model that estimates latency and energy consumption.

A. Co-UP Overview

Figure 3 presents an overview of Co-UP architecture. As plotted, Co-UP consists of Monitor that periodically monitors the system state, Predictor that forecasts latency and energy consumption, and Decision Engine that determines the core and uncore frequency combination that minimizes energy consumption without violating SLOs.

Monitor periodically reads the application's load, measured in QPS, along with the current core and uncore frequency values of the processor, and transmits this data to Predictor.

Algorithm 1: Decision Engine

```
1: SLO = Service Level Objective of the application
2: C_STEP = adjustment step for core frequency
3: U_STEP = adjustment step for uncore frequency
4:
5: QPS, cur_core, cur_uncore = Monitor()
6: pred_lat, pred_energy = Predictor(QPS, cur_core, cur_uncore)
7: best_energy = pred_energy
8: best_core = cur_core
9: best_uncore = cur_uncore
10:
11: if pred_latency < SLO then
12:     for core = cur_core - C_STEP to cur_core do
13:         for uncore = cur_uncore - U_STEP to cur_uncore do
14:             pred_lat, pred_energy = Predictor(QPS, core, uncore)
15:             if pred_lat < SLO and pred_energy < best_energy then
16:                 best_energy = pred_energy
17:                 best_core = core
18:                 best_uncore = uncore
19:             end if
20:         end for
21:     end for
22: else
23:     best_energy = maximum energy of the processor
24:     for core = cur_core to cur_core + C_STEP do
25:         for uncore = cur_uncore to cur_uncore + U_STEP do
26:             pred_lat, pred_energy = Predictor(QPS, core, uncore)
27:             if pred_lat < SLO and pred_energy < best_energy then
28:                 best_energy = pred_energy
29:                 best_core = core
30:                 best_uncore = uncore
31:             end if
32:         end for
33:     end for
34:     if best_core == cur_core and best_uncore == cur_uncore then
35:         best_core = best_core + C_STEP
36:         best_uncore = best_uncore + U_STEP
37:     end if
38: end if
39: Reflect best_core and best_uncore to the Processor
```

`Predictor` estimates the tail latency based on the QPS and core/uncore frequency values and sends this to `Decision Engine`, using a prediction model. `Decision Engine` checks whether the predicted latency violates the SLO. If an SLO violation is expected, `Decision Engine` sends options to increase the core or uncore frequency to `Predictor` to enhance performance. On the other hand, if an SLO violation is not expected, `Decision Engine` provides options for decreasing the core or uncore frequency to `Predictor` to reduce the processor's energy consumption. `Predictor` sends the predicted latency and energy consumption values for the provided core and uncore frequency combinations back to `Decision Engine`. Finally, `Decision Engine` utilizes the received latency and energy consumption values to select the core and uncore frequency combination that is predicted to have the lowest energy consumption without violating the SLO, based on a greedy algorithm, and applies it to the processor.

B. Co-UP Architecture

Algorithm 1 describes how Co-UP's `Decision Engine` determines the core and uncore frequencies that improve energy efficiency without violating the SLO of latency-critical applications. Algorithm 1 is called by a periodic timer, and it begins by reading the QPS and the current core and uncore frequency values from `Monitor` (line 5). `Decision Engine` passes the values received from `Monitor` to `Predictor` to estimate latency and energy consumption (line 6). Then, the currently predicted energy consumption and the current core and uncore frequencies are temporarily set as the best values (lines 7-9).

If the predicted latency is smaller than the SLO value, `Decision Engine` explores the possibility of reducing the core and uncore frequencies to lower energy consumption (lines 11-21). The core and uncore frequencies are adjusted within the range of C_STEP and U_STEP, respectively (lines 12-13). If the predicted values meet the SLO and further reduce energy consumption, the optimal configuration is updated with those values (lines 15-19). In contrast, if the initial predicted latency violates the SLO, `Decision Engine` begins an exploration of increasing the core and uncore frequencies (lines 23-33). Accordingly, the core and uncore frequencies are adjusted to increasing levels within the range of C_STEP and U_STEP, respectively (lines 24-25). As previously, if there is a configuration within the search range that reduces energy consumption while maintaining SLO compliance, those values are updated as the optimal configuration (lines 27-31). If no core and uncore frequency configuration solves the SLO violation, `Decision Engine` increases both frequencies by the STEP value for the next iteration (lines 34-37). Finally, `Decision Engine` reduces energy consumption without violating the SLO by applying the searched core and uncore frequencies to the processor (line 39).

The proposed `Co-UP` requires an appropriate prediction model for `Predictor`, and to achieve this, we use a simple linear regression model. This model is trainable online, and during training, `Co-UP` uses the actual energy measurements from the processor instead of predicted energy, and receives the 99th percentile latency from the client instead of predicted latency. Additionally, if previously untrained QPS, core frequency, or uncore frequency inputs are encountered, `Decision Engine` can randomly choose the next core and uncore frequency. As the collected information accumulates, the accuracy of the prediction model improves, and after sufficient training, `Co-UP` can use the predicted values instead of the actual latency and energy consumption measurements. The real-time operation and convergence process of online learning are demonstrated in Section V.

V. Evaluation

A. Methodology

We utilize an Intel Xeon Gold 6144 processor with eight cores to evaluate `Co-UP`. The core frequency of this processor can be set from 1.2 GHz to 3.5 GHz, while the uncore frequency can be adjusted from 1.2 GHz to 2.4 GHz. Both STEP values of `Co-UP` are set to one. The applications running on the server comprise an in-memory key-value store, memcached, and a web server, nginx. Both are configured to operate with eight threads aligned with an eight-core architecture. The server employs an Intel 82599 NIC, while the client is connected to the server via a D-Link DXS-1210-12SC 10GbE switch. The client uses 20 threads to generate varying levels of load for each application, with low, medium, and high loads. The loads correspond to 133 KQPS, 466 KQPS, and 800 KQPS for memcached, and 6 KQPS, 66 KQPS, and 120 KQPS for nginx.

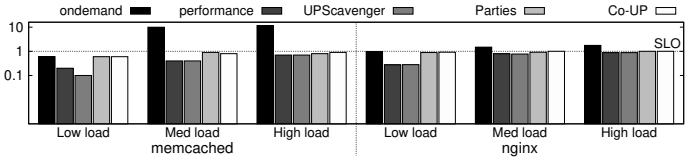


Fig. 4. Tail latency (normalized to SLO of each application).

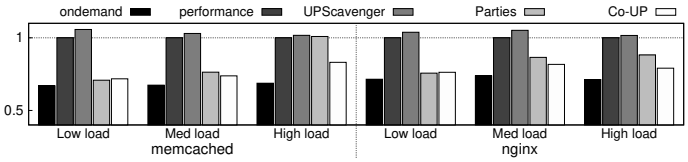


Fig. 5. Energy consumption (normalized to performance of each application).

To measure energy consumption, we utilize the processor's RAPL interface, and Hyper-threading is disabled.

In order to evaluate `Co-UP` that adjusts both core and uncore frequencies simultaneously, we quantitatively compare it against a total of four existing approaches. First, to compare against Intel uncore frequency scaling, we utilize the two core frequency governors provided by Linux, namely ondemand and performance. Next, we compare with `UPScavenger` [6], a state-of-the-art uncore frequency scaling approach. `UPScavenger`, similar to Intel uncore frequency scaling policy, adjusts the uncore frequency based on the monitoring of hardware components. To this end, `UPScavenger` periodically measures DRAM access and IPS. It reduces the uncore frequency when DRAM access decreases, while it increases the uncore frequency when IPS decreases. Lastly, for state-of-the-art core frequency scaling, we use `Parties` [10]. `Parties`, a core frequency scaling approach designed to improve the energy efficiency of latency-critical applications, controls the core frequency using a feedback controller. `Parties` periodically reads the tail latency from the client and adjusts the core frequency based on the slack, which is the difference between the tail latency and the SLO value. In the case of `UPScavenger`, the core frequency is controlled by the performance governor, while in the case of `Parties`, Intel's default uncore frequency scaling is used for uncore frequency control.

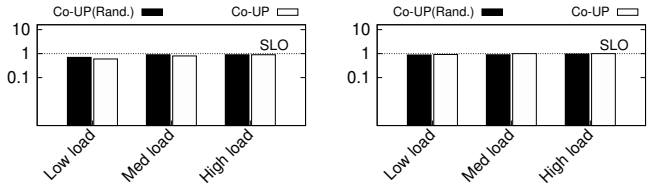
B. Experimental Results

Figure 4 presents the results for tail latency, while Figure 5 illustrates the results for energy consumption. The results in Figure 4 are normalized to the SLO, while the results in Figure 5 are normalized to the energy consumption when both the core and uncore frequencies are set to their maximum values. Note that `UPScavenger`, `Parties`, and `Co-UP` all require time to adapt to each load, and we measure the tail latency and energy consumption results after sufficient convergence. Moreover, since `UPScavenger`, `Parties`, and `Co-UP` all function on a periodic basis, setting an appropriate time interval is essential. To maintain consistency, we set this interval uniformly to 500 ms, following the configuration used in `Parties`.

First, as shown in Figure 5, the ondemand governor consistently demonstrates the lowest energy consumption across all cases. This is because latency-critical applications generally have low average CPU utilization, leading the ondemand governor to set a low core frequency, and consequently, Intel's uncore frequency scaling is also typically set to its minimum value. However, such low core and uncore frequency settings are likely to result in SLO violations. As a result, as shown in Figure 4, the ondemand governor leads to SLO violations for all loads except low load in both memcached and nginx. In contrast, the performance governor consistently shows the best tail latency results across all cases, but this comes at the cost of reduced energy efficiency. Specifically, even under low load, by setting the core frequency to its maximum, the uncore frequency also remains nearly at its maximum, resulting in increased energy consumption due to unnecessary performance provisioning.

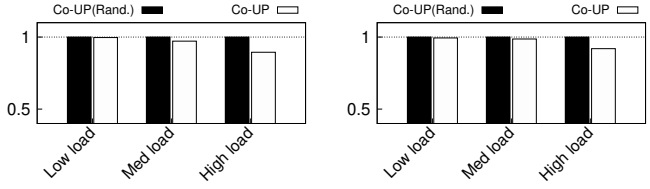
In the case of `UPScavenger`, while it demonstrates tail latency results nearly similar to those of the performance governor, it actually shows worse results in terms of energy consumption. This is because, like Intel's uncore frequency scaling, `UPScavenger` does not account for software state, and therefore, it misses the opportunity to appropriately lower the uncore frequency for latency-critical applications. As a result, `UPScavenger` generally sets the uncore frequency to its maximum, and due to its periodic operation, it also slightly increases energy consumption, leading to the worst energy consumption results in all cases. `Parties`, a core frequency scaling technique designed for latency-critical applications, reduces energy consumption by setting appropriately low core frequencies based on a feedback controller in response to application load. However, since `Parties` does not consider uncore frequency scaling, it provides only suboptimal energy consumption under medium and high load, following Intel's uncore frequency scaling policy. Notably, under high load, `Parties` fails to operate efficiently in terms of energy consumption because Intel's uncore frequency scaling mostly uses either the minimum or maximum values in our experimental environment. In other words, to satisfy the SLO with the minimum uncore frequency value, `Parties` has to set a high core frequency, which results in setting the maximum core frequency under high load. Additionally, when the maximum core frequency is set, Intel's uncore frequency scaling often sets the maximum uncore frequency, leading to the worst-case energy consumption.

`Co-UP`, by jointly considering and adjusting both core and uncore frequencies, consistently demonstrates the lowest energy consumption results without SLO violations across medium and high loads, regardless of the application. Under low load, `Co-UP` incurs slightly higher energy consumption compared to the ondemand governor, due to periodic monitoring, and a similar level of increased energy consumption is observed with `Parties`. In summary, `Co-UP` reduces energy consumption by 28.2%, 26.1%, and 16.8% for low, medium, and high loads, respectively, in memcached, and by 23.7%, 18.3%, and 21% for low, medium, and high loads, respectively, in nginx, compared to the combination of the performance governor and Intel uncore frequency scaling, without any SLO violations. Addi-



(a) memcached. (b) nginx.

Fig. 6. Comparison of policies on tail latency.



(a) memcached. (b) nginx.

Fig. 7. Comparison of policies on energy consumption.

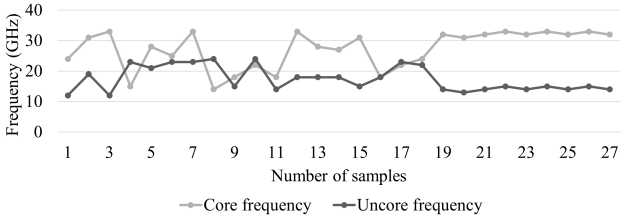
tionally, compared to the state-of-the-art core frequency scaling technique, Parties, Co-UP reduces energy consumption by -1.6%, 3.3%, and 17.6% for low, medium, and high loads, respectively, in memcached, and by -0.7%, 5.6%, and 10.3% for low, medium, and high loads, respectively, in nginx.

Comparison to alternative. Co-UP adjusts the core and uncore frequencies through selections based on a prediction model and a greedy algorithm. To validate the efficiency of this policy, we compare Co-UP with a policy that controls both the core and uncore frequencies based on randomness. Co-UP based on the Random policy operates similarly to the proposed Co-UP, but randomly selects either the core or uncore frequency for adjustment, increasing or decreasing the frequency accordingly. As in the previous experiment, the random version of Co-UP also requires time to converge. Therefore, we compare the tail latency and energy consumption after the random-based Co-UP has sufficiently converged, with an operational interval set to 500 ms.

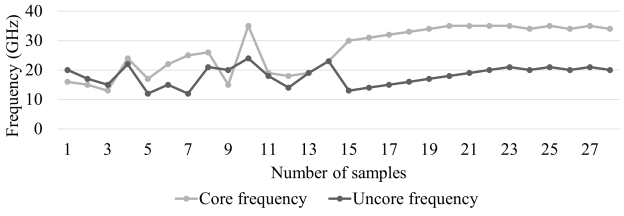
Figure 6 and Figure 7 show the results for tail latency and energy consumption of the applications, respectively. The results in Figure 6 are normalized to the SLO, while the results in Figure 7 are normalized to the energy consumption of the random version of Co-UP. As shown in Figure 6, since we measure after the core and uncore frequency decisions have sufficiently converged, neither the random version of Co-UP nor Co-UP results in SLO violations. However, as shown in Figure 7, Co-UP saves 0.3%, 2.7%, and 10.5% energy compared to the random version for low, medium, and high loads, respectively, in memcached, and reduces energy consumption by 0.6%, 1.3%, and 8% for low, medium, and high loads, respectively, in nginx. This difference occurs because the random-based Co-UP converges to the suboptimal core and uncore frequencies compared to Co-UP.

Although methods like Deep Neural Networks (DNNs) may achieve higher accuracy, they involve trade-offs due to computational overhead. We leave the exploration of optimal approaches as future work.

Adaptiveness of novel loads. To demonstrate that Co-UP can rapidly respond to new loads, we experimentally show how quickly it adapts and converges to untrained workloads.



(a) Medium load.



(b) High load.

Fig. 8. Convergence result of Co-UP (memcached).

Figure 8 shows the core and uncore frequencies set by Co-UP, which has not yet trained on these loads, in real time, while the client generates medium and high loads for the memcached application. The operation interval of Co-UP is set to 500 ms, as in the previous experiments.

As shown in Figure 8, initially, Co-UP explores different values since it has not yet trained on these QPS. During this period, it collects and learns from the resulting tail latency and energy consumption. Subsequently, for both medium and high loads, Co-UP converges around the 20th sample to a value predicted to minimize energy consumption without violating the SLO. Thus, while dependent on the operation interval, Co-UP can converge to appropriate core and uncore frequencies within approximately 10 seconds, even when encountering a new load.

VI. Conclusion

This paper examines the significance of uncore frequency scaling in latency-critical applications and its effects on tail latency and energy consumption. We highlight the crucial impact of uncore frequency scaling on both performance and energy consumption in latency-critical applications. By introducing 'Co-UP,' power management designed to simultaneously control both core and uncore frequency scaling, we demonstrate the effectiveness of using a prediction model and a simple greedy algorithm to dynamically adjust frequency scaling in both core and uncore regions, achieving energy efficiency while keeping SLOs. Experiments show that Co-UP provides superior energy efficiency compared to the existing Intel policy and state-of-the-art DVFS approach, achieving reductions of up to 28.2% and 17.6%, respectively, without violating SLOs. Co-UP provides a robust solution for enhancing energy efficiency in modern data centers, where energy costs and performance constraints are critical concerns. Our research offers valuable insights into power management strategies for latency-critical applications, emphasizing the importance of both core and uncore frequency scaling. We expect this work to contribute to understanding and enhancing energy-efficient frequency scaling in various computing environments.

REFERENCES

[1] S. Li, X. Wang, F. Kalim, X. Zhang, S. A. Jyothi, K. Grover, V. Kontorinis, N. Narodytska, O. Legunsen, S. Kodakara *et al.*, "Thunderbolt:{Throughput-Optimized},{Quality-of-Service-Aware} power capping at scale," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 1241–1255.

[2] Q. Wu, Q. Deng, L. Ganesh, C.-H. Hsu, Y. Jin, S. Kumar, B. Li, J. Meza, and Y. J. Song, "Dynamo: Facebook's data center-wide power management system," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 469–480, 2016.

[3] Y. Li, C. R. Lefurgy, K. Rajamani, M. S. Allen-Ware, G. J. Silva, D. D. Heimsoth, S. Ghose, and O. Mutlu, "A scalable priority-aware approach to managing data center server power," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2019, pp. 701–714.

[4] A. G. Kumbhare, R. Azimi, I. Manousakis, A. Bonde, F. Frujeri, N. Mahalingam, P. A. Misra, S. A. Javadi, B. Schroeder, M. Fontoura *et al.*, "{Prediction-Based} power oversubscription in cloud platforms," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, 2021, pp. 473–487.

[5] 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 *2012 USENIX Annual Technical Conference (USENIX ATC 12)*, 2012, pp. 367–372.

[6] N. Gholkar, F. Mueller, and B. Rountree, "Uncore power scavenger: A runtime for uncore power conservation on hpc systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019, pp. 1–23.

[7] X. Chen, Z. Xu, H. Kim, P. Gratz, J. Hu, M. Kishinevsky, and U. Ogras, "In-network monitoring and control policy for dvfs of cmp networks-on-chip and last level caches," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 18, no. 4, pp. 1–21, 2013.

[8] H. Kasture, D. B. Bartolini, N. Beckmann, and D. Sanchez, "Rubik: Fast analytical power management for latency-critical systems," in *Proceedings of the 48th International Symposium on Microarchitecture*, 2015, pp. 598–610.

[9] D. Lo, L. Cheng, R. Govindaraju, L. A. Barroso, and C. Kozyrakis, "Towards energy proportionality for large-scale latency-critical workloads," in *ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2014, pp. 301–312.

[10] S. Chen, C. Delimitrou, and J. F. Martínez, "Parties: Qos-aware resource partitioning for multiple interactive services," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2019, pp. 107–120.

[11] D. Brodowski and N. Golde, "Cpu frequency and voltage scaling code in the linux kernel," *URL https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt*, 2002.

[12] B. Fitzpatrick, "Distributed caching with memcached," *Linux journal*, vol. 124, 2004.

[13] W. Reese, "Nginx: the high-performance web server and reverse proxy," *Linux Journal*, p. 2, 2008.

[14] M. Alian, A. H. Abulila, L. Jindal, D. Kim, and N. S. Kim, "Ncap: Network-driven, packet context-aware power management for client-server architecture," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017, pp. 25–36.

[15] K.-D. Kang, G. Park, H. Kim, M. Alian, N. S. Kim, and D. Kim, "Nmap: Power management based on network packet processing mode transition for latency-critical workloads," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 143–154.

[16] M. Chadha and M. Gerndt, "Modelling dvfs and ufs for region-based energy aware tuning of hpc applications," in *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2019, pp. 805–814.