Received 29 March 2024, accepted 19 April 2024, date of publication 13 May 2024, date of current version 20 May 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3399732

RESEARCH ARTICLE

A Study of Sleep Time Alignment of CPU Cores for Power Saving

IKUO OTANI AND KEI FUJIMOTO
NTT Network Innovation Center, NTT Corporation, Tokyo 180-8585, Japan
Corresponding author: Ikuo Otani (ikuo.otani@ntt.com)

ABSTRACT Existing processing models take an immediate processing model that starts processing tasks as soon as a task arrives. This is wasteful in terms of power consumption, because the power-saving mechanisms of CPUs cannot work effectively. To suppress the increase in server power consumption, we propose a power-saving processing model. In this model, multiple tasks are processed together and the sleep timing of each logical core is aligned so that the C-state of the physical core can be put into a deep state. We implemented the proposed method and evaluated its performance on multiple traffic models. Results showed that it can reduce server power consumption with little effect on the processing performance per second.

INDEX TERMS CPU core, sleep, C-state, power saving.

I. INTRODUCTION

In recent years, network traffic has exploded with the spread of devices such as smartphones and the growth of online video-streaming. Part of this traffic is processed by general-purpose servers located in a data center of a network operator or service provider. More and more servers are needed to cope with the network traffic, and the total power in the data center will increase significantly. According to one estimate [1], the power consumption of the servers is predicted to be about 3,400 times greater in 2050 than in 2018. This will not only affect the management of network operators and service providers but also hinder the low-carbonization of society as a whole.

General-purpose server hardware has been enhanced with power-saving mechanisms to solve the problem of power consumption. Recent CPUs can reduce power consumption considerably by setting appropriate conditions. For example, by using Low Power Idle (LPI, also called C-state), which takes advantage of the idle state of the CPU, power consumption can be reduced by about 4W per CPU core [2]. On the other hand, the application side does not take full advantage of the power-saving mechanism of the server. This is because applications running on general-purpose servers use a processing execution model that is incompatible with power-saving mechanisms. In the processing execution model, the CPU immediately starts processing when a processing request arrives at the application. Even if C-state is used as one of the power-saving mechanisms, the CPU can be woken up from sleep by processing requests in small intervals, so the power-saving effect will be small. In this paper, we call the above processing model the immediate processing model.

We aim to create a technology to further reduce power consumption of servers by maximizing the use of CPU power-saving mechanisms. The technology should be highly versatile and be able to be used in many data center servers to reduce power consumption. Therefore, the technology needs to meet two requirements. Requirement (a) is that it is applicable to a wide range of applications, not limited to specific applications. Requirement (b) is that it does not modify the applications themselves. For this purpose, we devised a power-saving processing model instead of an immediate processing model. In the power-saving processing model, even if a request arrives, the application can stop processing once and force the CPU to sleep. Then, the processing is resumed after the CPU returns from sleep. We implemented the power-saving processing model and evaluated its power-saving effect and performance impact on several traffic models.

The associate editor coordinating the review of this manuscript and approving it for publication was Abdel-Hamid Soliman.

The rest of this paper is organized as follows. In Sect. II, we introduce existing research using CPU power-saving mechanisms. In Sect. III, we explain why the existing processing model increases power consumption. In Sect. IV, we outline our proposed method and explain how the model saves power. In Sect. V, we evaluate the power saving effect and performance impact of the proposed method on applications. In Sect. VI we conclude and discuss future work.

II. RELATED WORK

This section introduces related research on server power saving by utilizing CPU power saving mechanisms.

CARB [3] is a power-saving technology that uses the request arrival rate and response time to calculate the minimum number of cores required to meet service level agreements (SLA) and reduce the number of active CPU cores. However, it violates the requirement (a) because its inability to be applied under conditions where applications are pinned to cores means it cannot be applied to applications where pinning to a specific core is desired.

WASP [4] divides servers into active/sleep server pools and transitions between each server pool in accordance with the amount of processing in the server. In addition, the sleep state of the servers is dropped in stages while also utilizing a delay timer. However, apps must be modified to scale across multiple servers, which does not meet requirement (b).

Peafowl [5] saves power by scaling in the threads that process requests in accordance with the amount of workload, thereby reducing the number of active CPU cores during periods of low load. However, the application must be modified to scale in and out of threads, which does not meet requirement (b).

YAWN [6] uses machine learning to calculate the appropriate C-state residence time on behalf of the C-state governor and performs C-state transitions. It also reduces latency by allocating requests so as to avoid causing C-state transitions and sleeping cores as much as possible. However, apps need to support request load-balancing. This violates requirement (a) because *YAWN* cannot be applied to applications that are not suitable for load-balancing and must be processed in a specific thread.

Dynsleep [7] achieves a deep sleep state by delaying the timing of task processing as much as possible within the latency requirements and processing tasks together. However, apps needs to calculate the allowable delay time to predict the randomness of the request arrival interval and processing time, which violates requirement (b).

μDPM [2] uses a statistical model to predict processing time when a request arrives and delays processing until the appropriate timing to meet the processing deadline. If the processing deadline is exceeded at the current frequency, the frequency will increase. Although not explicitly stated, a queue to hold requests and a function to manage the queue would need to be added to the application like *Dynsleep*.

III. ANALYSIS OF POWER CONSUMPTION IN EXISTING PROCESSING MODEL

In this section, we analyze why existing immediate processing models fail to take advantage of CPU power-saving mechanisms and to reduce power.

A. POWER-SAVING MECHANISM OF CPU

Various vendors produce CPUs, many of which have power-saving mechanisms. For example, most Intel CPUs have LPI (C-state) and Dynamic Voltage and Frequency Scaling (DVFS, also called P-state) as power-saving mechanisms. Since most data centers have Intel CPUs, in this subsection, we provide an overview of C-state and P-state, which significantly affect CPU power.

1) C-STATE

LPI (C-state) puts the CPU to sleep and reduces power by dropping the power consumption elements of the CPU in stages when the CPU is inactive. In C-state, C0 represents the active state, and C1 or higher represents the idle state [8]. The larger the number, the more power consumption factors are eliminated and the deeper the sleep state. The deeper the sleep state, the greater the power-saving effect, but the longer the wake-up time to return to the original active state. C-state is a hierarchical structure. C-state controls logical cores (hardware threads) when Simultaneous Multi-Threading (SMT) is used. Core C-state controls the physical core, and package C-state controls the CPU package (socket) [9].

2) P-STATE

DVFS (P-state) saves power by reducing the frequency of the CPU core and the supply voltage in the active state of the CPU. The lower the CPU frequency, the lower the voltage and power [8]. However, at the same time, the performance that the CPU can deliver is also reduced. Therefore, P-state dynamically reduces frequency and voltage when high performance is not needed.

As mentioned in Sect. II, many existing studies have proposed power saving method by utilizing C-state and P-state. However, existing studies have not attempted to match the control timing of multiple logical cores when using SMT to drop the physical core into a power-saving state. In Sect. IV, we propose a method to reduce power consumption of a physical core by coordinating the control timing of two logical cores associated with one physical core. This idea could be used not only for C-state control but also for P-state control.

B. POWER CONSUMPTION WITH EXISTING PROCESSING MODEL

In this subsection, we explain how power is consumed wastefully in the immediate processing model.

1) CONTINUOUS SHALLOW SLEEPS

When no particular process is being executed, the logical cores of the CPU are in sleep mode. The longer this state lasts,

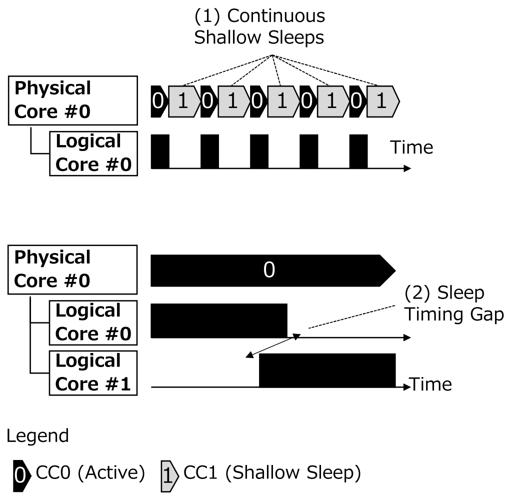

FIGURE 1. Power consumption factors in existing processing model.

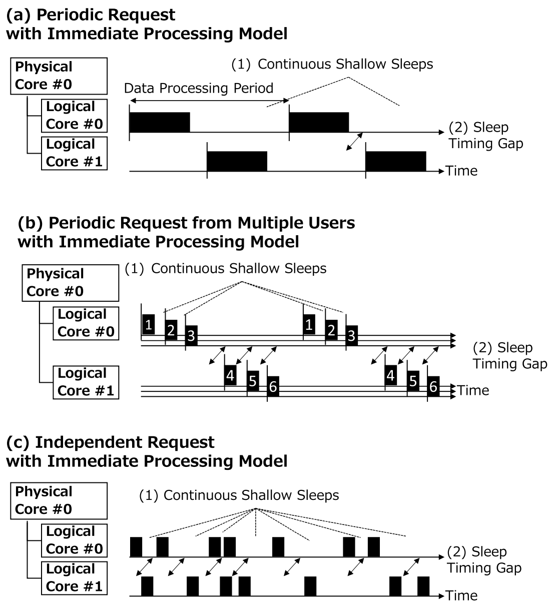
(a) Periodic Request with Immediate Processing Model
Physical Core #0
Logical Core #0
Logical Core #1
(1) Continuous Shallow Sleeps
Data Processing Period
(2) Sleep Timing Gap
Time
(b) Periodic Request from Multiple Users with Immediate Processing Model
Physical Core #0
Logical Core #0
Logical Core #1
(1) Continuous Shallow Sleeps
1 2 3 1 2 3
4 5 6 4 5 6
(2) Sleep Timing Gap
Time
(c) Independent Request with Immediate Processing Model
Physical Core #0
Logical Core #0
Logical Core #1
(1) Continuous Shallow Sleeps
(2) Sleep Timing Gap
Time

FIGURE 2. Immediate processing model impact on traffic models.

the more power is saved. In the immediate processing model, once a processing task arrives, the logical core immediately wakes up and starts processing. After a series of processing tasks is completed, the logical core sleeps again. If processing tasks come frequently, the logical core repeats a short loop of waking and sleeping. The logical core does not enter a deep sleep state and repeats shallow sleep, and the power-saving effect becomes small. We call this "*continuous shallow sleeps*".

2) SLEEP TIMING GAP

A logical core can sleep if there is no processing task on it, but a physical core cannot sleep unless all logical cores on it are sleeping. Moreover, core C-state is capped at the shallower one, so the physical core cannot enter deep sleep. In other words, power cannot be saved unless all the processing timings on the same physical core are synchronized. In the immediate processing model, however, each logical core operates independently when a processing task arrives. Because the processing timings do not align among the logical cores, the physical cores cannot fall in deep sleep. We call this "*sleep timing gap*". These power consumption factors are illustrated in Fig. 1.

C. IMPACT ANALYSIS FOR MULTIPLE TRAFFIC MODELS

The two power consumption factors in the previous subsection can occur in most services. For example, they can occur with applications such as Network Function Virtualization (NFV), Artificial Intelligence (AI) inference, and High-Performance Computing (HPC). The following are examples of workload types. In a periodic workload, tasks are generated in a certain cycle and the tasks are processed at the time they are generated. In a random workload, tasks are generated at random times and the tasks are processed at the time they are generated. In a batch processing workload, multiple tasks are aggregated and processed in batches at specific times. In this paper, we take as an example an NFV-like application that processes network traffic and analyze the impact of each issue on several traffic models. We use three traffic models as examples for our case studies. The first is "Periodic Request". This is traffic that arrives regularly and needs to be processed completely before the next request. The second is "Periodic Request from Multiple Users". This is traffic that arrives regularly from multiple users and needs to be processed completely before the users' next request. The third is "Independent Request". This is traffic that arrives independently and does not need to be processed completely until the next request. In other words, this traffic that can be batched and processed together.

1) IMPACT ON PERIODIC REQUEST

In media processing such as IP telephony, web conferencing, and video distribution, requests arrive periodically in a fixed period and processing begins as soon as a request arrives. Figure 2 (a) shows the periodic request traffic with the immediate processing model. If the data processing period is short, *continuous shallow sleep* can occur. However, if the data processing period is long, such as several 10 ms, the logic core can sleep deeply. Thus, the next *sleep timing gap* is more important. In the figure, the sleep timings of logical cores 0 and 1 are out of sync. This situation can occur when logical cores are processing packets from different sessions. This causes the *sleep timing gap*. If the application is unable to complete processing before the deadline, an error can occur and/or Quality of Service (QoS) can drop. The immediate processing model is certainly suitable from the perspective of meeting such deadlines. However, it generates power waste such as the *sleep timing gap*, and there remains room to take advantage of C-states.

2) IMPACT ON PERIODIC REQUEST FROM MULTIPLE USERS

For example, in web conferencing, the tasks are processed as soon as the requests from multiple users arrive. Figure 2 (b) shows the periodic request traffic from multiple users with immediate processing model. Requests from users 1, 2, and 3 arrive at logical core 0, and those of users 4, 5, and 6 arrive at logical core 1. Because tasks from multiple users are processed on one logical core, the processing interval becomes shorter, and the impact of *continuous shallow sleeps* is more pronounced than in III-C1. In addition, because the amount of processing and number of sleeps increases, the number of times when sleeps are not aligned among logical cores increases. Thus, the impact of *sleep timing gap* is also more pronounced than in III-C1. In this case, there remains room for power consumption reduction.

3) IMPACT ON INDEPENDENT REQUEST

For example, in a typical web server, independent requests arrive from many users. Figure 2 (c) shows the independent request traffic with the immediate processing model. Because the requests come in irregularly, the CPU is woken up occasionally, resulting in *continuous shallow sleeps*. Because the timing of incoming requests for each of the multiple logical cores of the SMT does not align, the *sleep timing gap* also occurs. The immediate processing model is suitable when performance requirements are strict, such as when there is a deadline, but in this case, the requirements are often not that strict. Therefore, it is worth considering ways to save power other than the existing model. We examine whether these issues can be solved in the rest of this paper.

IV. PROPOSED METHOD

We proposed and implemented a power-saving method that takes advantage of the CPU power-saving mechanism. We describe our proposed method and explain how the model saves power.

A. OUTLINE OF PROPOSED METHOD

When adopting the immediate processing model, the power saving potential of longer and deeper sleep cannot be taken advantage of. From a different perspective, this means that there is still room for power saving. In Sect. III-B, we mentioned that the extra power consuming factors in the immediate processing model are (1) *continuous shallow sleeps* and (2) the *sleep timing gap*. The key idea of our proposed method for power saving is to create a forced sleep time by aligning the sleep time of logical cores on the physical cores so that the physical cores have time to enter deep sleep. This can solve the problem of (1) *continuous shallow sleeps*, because it allows a forced sleep with a longer time. In addition, the timing of logical cores for processing and sleeping can be aligned, thus resolving (2) the *sleep timing gap*. Figure 3 shows how the proposed method aligns the sleep timing of logical cores. The difficulty here, however, is that forcing multiple logical cores to sleep can significantly

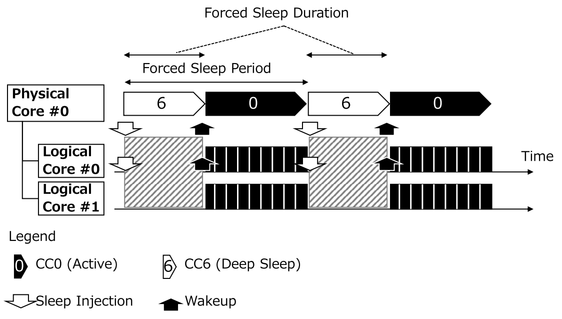

FIGURE 3. Power-saving mechanism of the proposed method.

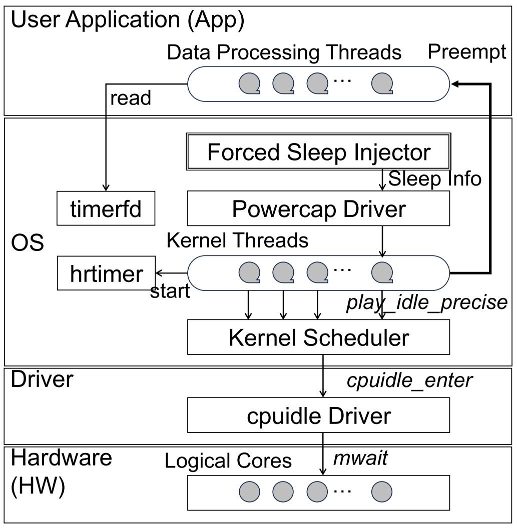

FIGURE 4. Architecture of proposed method.

impact performance. To avoid impacting performance, the proposed method is designed to take a sleep duration and sleep period considering the processing time and period of the application. The details of each of these considerations and implementation examples are described in the rest of this section.

B. DETAILS OF PROPOSED METHOD

1) FORCED SLEEP INJECTION

First, we explain the forced sleep injection function, which is the main component of our proposed method. We introduce a sleep thread in each logical core, and the logical core is put to sleep when the thread issues a sleep instruction. The sleep thread is set to a higher priority than a user application so that the thread can preempt while the application is running. The sleep thread can specify a sleep duration on a timer to sleep each physical core for a specified period of time. Also, the sleep thread can specify a maximum latency that limits the deepest C-state.

The implementaion of the proposed method is shown in Fig. 4. The proposed method is roughly divided into a forced sleep injector (FSI) and a group of Linux kernel drivers. The reason for this is to make it easier to change the logic later.

In addition, the deep kernel part is kept simple to prevent system stability degrading due to bugs and other problems. The reason for utilizing existing kernel drivers is that *mwait* must be issued to drop C-state, but it can only be issued from ring0 (kernel space).

FSI is the brain of the proposed method and determines the logical core to be controlled, sleep duration, sleep period, and so on. FSI should have an interface to give and receive parameters from the user, internal logic to determine sleep time, and an interface to pass sleep information to the Linux kernel. FSI can be implemented as a kernel module to be embedded in the Linux kernel. This allows FSI to use kernel drivers directly without additional user interface. Also, FSI module can be unloaded when forced sleep is not needed. In the hierarchy below FSI, CPU cores are controlled to sleep according to the sleep information determined by FSI. For this purpose, a function to create and manage sleep threads, a scheduler to schedule idle time, and a driver to control the CPU cores during idle time are required. Existing kernel drivers can be used to implement the above. The *powercap* driver creates a Linux kernel thread for each logical core and executes sleep. The Linux kernel scheduler schedules idle class tasks. The C-state (cpuidle) driver issues a *mwait* instruction to the logical core and actually puts the CPU to sleep.

Forced sleep injection is executed as follows. The sleep thread is set to high priority (SCHED_FIFO) to preempt the application thread. The sleep thread preempts the running application when it becomes ready to execute itself. The sleep thread issues a Linux kernel sleep instruction (*play_idle_precise*) and specifies the sleep duration and maximum latency. When the sleep instruction is issued, the sleep thread enters a non-preemptable state to avoid context switch. The C-state driver determines the C-state using the maximum latency as a hint. The sleep thread sets a timer for the sleep period and enters the idle state. When the timer expires, the sleep thread wakes up and enters the preemptable state. If the application was running earlier, Linux kernel switches the context and the application obtains the execution time.

2) DETERMINATION OF SLEEP CONDITIONS

In forcing sleep, the sleep duration and timing should be considered to prevent performance degradation and maximize power savings. For periodic request, the execution time can be estimated on the basis of past results. The sleep duration is determined by subtracting this execution time from the execution period. A certain safety factor can be applied to avoid performance degradation. It is better to use the request execution period as the sleep period. If the sleep period is different from that of the application, the number of times the application sleeps and wakes will increase, which can affect performance. The best timing for the start of sleep is after the estimated execution time if possible, but this is not a requirement.

In independent request processing, the arrival of a request cannot be predicted, so the sleep duration and period are determined to some extent by predetermination. The sleep duration should be long enough to allow the CPU to fall into a deep enough core C-state and should not exceed the upper processing time limit of the application. The CPU utilization can be used to determine the sleep period. For example, the idle ratio (100% - CPU utilization) is used to determine the sleep ratio. To calculate the sleep period, the sleep duration is divided by the sleep ratio.

C. POINTS TO NOTE IN THE PROPOSAL METHOD

To increase the effectiveness of the proposed method, it is better to place threads that handle the same amount of load in logical cores on the same physical core. This is because, even if the sleep timing is aligned by FSI, if the load is different, one of the logical cores is active and the physical core cannot sleep. In addition, to prevent thread placement from being changed, it is recommended to exclude the target logical core from the kernel scheduler by setting *isolcpus*.

If the forced sleep is canceled in the middle of the sleep, the power saving effect becomes small. The sleep thread is processed in SCHED_FIFO, but when the hardware interrupt is raised, the interruption is processed. Therefore, the data processing threads should be kept in a state where hardware interrupts occur as little as possible. For example, two methods can be used. One is to set *irqbalance* to a logical core separate from the data processing thread. The other is to separate a polling thread from the data processing thread if any.

The applicability of the proposed method to each workload is assumed to be as follows. For periodic workloads, tasks arrive one after another in a certain cycle. The proposed method allows the system to sleep steadily until the next task after processing one task, thus achieving a power-saving effect. Furthermore, the power-saving effect can be increased by aligning the sleep timing of the logical cores. For random workloads, tasks arrive at unpredictable and random timing. If the allowable latency for each task is long, the proposed method can be effective by forcing the system to sleep for a certain period of time and processing multiple tasks at once. Similar to periodic workloads, aligning the sleep timing of logical cores is also effective. On the other hand, if the allowable latency for each task is short, it is difficult to process multiple tasks in batch and the proposed method cannot be applied. In batch processing workloads, a large number of tasks are aggregated and processed at once. Because the tasks are already aggregated, it would be inefficient to stop processing and sleep in the middle of the processing. Therefore, the effect of the proposed method is considered to be small.

V. PERFORMANCE EVALUATION

We evaluated how much power the proposed method can save compared to the existing processing model. We also evaluated whether the forced sleep will affect the performance of

TABLE 1. Server specifications.

	Server
Machine	Dell PowerEdge R640
CPU	Intel Xeon Silver 4208 2.1GHz 1 socket 8 physical cores (16 logical cores)
Memory	32 GB
OS/Kernel	Ubuntu 22.04.03 LTS 6.5.0-15-generic
Driver	C-state driver: intel_idle (menu governor) P-state driver: intel_ptate (performance governor)

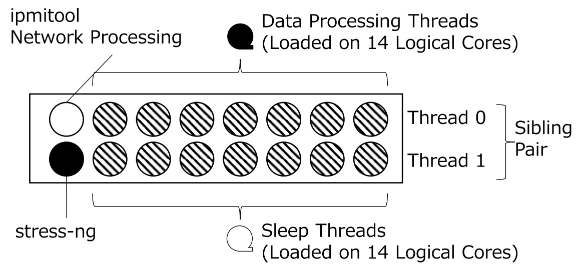

FIGURE 5. Core assignment.

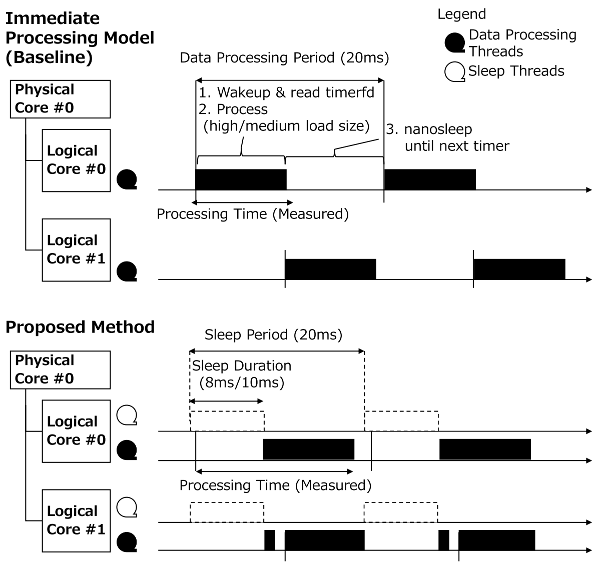
Immediate Processing Model (Baseline)
Data Processing Period (20ms)
Legend
Data Processing Threads
Sleep Threads
Physical Core #0
1. Wakeup & read timerfd
2. Process (high/medium load size)
3. nanosleep until next timer
Logical Core #0
Processing Time (Measured)
Logical Core #1
Proposed Method
Sleep Period (20ms)
Physical Core #0
Sleep Duration (8ms/10ms)
Logical Core #0
Processing Time (Measured)
Logical Core #1

FIGURE 6. Baseline and proposed method in periodic request.

applications. As a performance indicator, we measured throughput because it is thought to be smaller due to dropped requests during the sleep period. Also, we measured request processing time because it is thought to be longer due to injected sleep time and wake-up latency.

A. EVALUATION CONDITIONS

We evaluated power and performance with three typical traffic models analyzed in Sect. III-C. The first is periodic request, which assumes voice data arriving at 20-ms intervals, such as real-time transport protocol (RTP) packets for IP telephony. The second is periodic requests from multiple users, which assumes voice data arriving at 20-ms intervals, like RTP packets for web conferencing. The third is independent request processing, which assumes HyperText Transfer Protocol (HTTP) requests of 500 requests per second (rps). The first and second traffic models correspond to periodic workloads, while the third traffic model corresponds to random workloads. Therefore, as discussed in Sect. IV-C, the proposed method is expected to be effective for these traffic models.

We prepared a data processing application that follows the immediate processing model. The app starts processing on a periodic timer, performs a simple CPU instruction, and sleeps until the next timer. The load size of a request and the arrival interval can be adjusted. It is assumed that hardware interrupts caused by network arrivals are pinned to be accepted by a dedicated core for network processing. Therefore, the target cores should simply process workloads. For baseline data, we acquired the power and the performance data while the data processing threads were working. Each logical core sleeps at its own timing under the control of the existing Linux kernel and driver. On the other hand, for the proposed method, we acquired data while sleep threads were working in addition to the above app. Each logical core sleeps at the same timing under the control of FSI. By comparing the baseline and the proposed method, we can evaluate the power-saving effect of forcing the logical cores to sleep at the same timing.

The sleep period of the proposed method was set to 20 ms, which is the same period of RTP pakcets for IP telephony. Data processing threads consumed about 40% of the CPU time for a medium load. Because about 8 ms of the 20 ms period was consumed to process tasks, we used 10-ms sleep duration as the maximum value and 8-ms sleep duration as a conservative value. For power evaluation, server power increment from the no-load state was collected with *ipmitool*. To suppress temperature and power blurring due to environmental factors, the power was acquired three times over a sufficiently long period of time. For performance evaluation, throughput and request processing time were collected from the application output. We tried five times and averaged the performance values.

The server specifications are listed in Table 1, and core assignment is shown in Fig. 5. One logical core is 100% loaded by *stress-ng*. The reason for this is that when all physical cores are idle, the package C-state becomes PC6 instead of the core C-state.

B. EVALUATION FOR PERIODIC REQUEST

In a periodic request case, each request has to be processed before the next request. We aim to reveal whether the CPU can enter a deep sleep by aligning the sleep timing of each logical core. We tried a request of medium (1-1) or high (1-2) load in a 20 ms period to investigate the proposed method's dependency on the load size for periodic request. Figure 6 shows how the baseline and proposed method work.

Table 2 shows the results of power evaluation. It can be seen that power-saving effects are obtained. In addition,

TABLE 2. **Power evaluation results for periodic request.**

Load Size	Baseline	8-ms sleep	10-ms sleep
(1-1) Medium	13.2W	10.3W (-2.9W)	8.9W (-4.3W)
(1-2) High	15.2W	11.9W (-3.3W)	9.2W (-6.0W)

TABLE 3. **Throughput evaluation results for periodic request.**

Load Size	Baseline	8-ms sleep	10-ms sleep
(1-1) Medium	50.01 rps	50.01 rps	50.01 rps
(1-2) High	50.01 rps	49.63 rps (-0.76%)	41.49 rps (-17%)

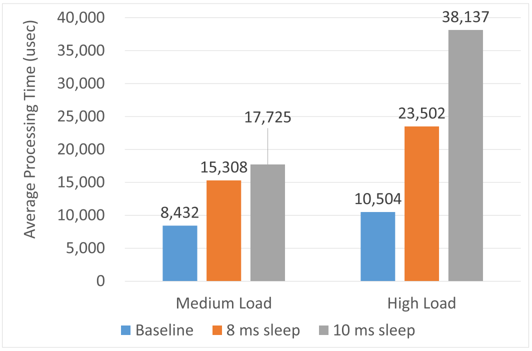

FIGURE 7. **Processing time for periodic request.**

the amount of reduction increases as the sleep period is increased. The combination of medium load (1-1) and 10-ms sleep reduces power by 4.3W (33% of baseline power), and the combination of high load (1-2) and 10-ms sleep reduces power by 6.0W (39% of baseline power). However, because power may reduce due to lower throughput, we should consider the performance impact to determine if this reduction is appropriate.

Table 3 shows the results of the throughput evaluation. We can see that when the sleep period is short, there is no or slight impact. However, if the sleep period becomes longer, we can see that performance is affected. The abovementioned combination of high load (1-2) and 10-ms sleep actually degraded performance. Also, the combination of high load (10.5 ms processing time in the baseline) and 8-ms sleep should ideally have no performance impact, but the performance is slightly degraded. This means a safety factor needs to be applied when using the proposed method in the periodic request case. Because determining the optimal sleep time considering the safety factor is difficult to do manually, it is a future work to automatically calculate the sleep time.

Figure 7 shows the results of the request processing time evaluation. It can be seen that the processing time increases as the sleep period becomes longer. In the medium load (1-1) case, the processing time does not exceed 20 ms up to 10-ms sleep, indicating that power-saving can be achieved without affecting performance. On the other hand, in the high load (1-2) case, the processing time exceeds 20 ms even after 8-ms sleep, which leads to the throughput degradation.

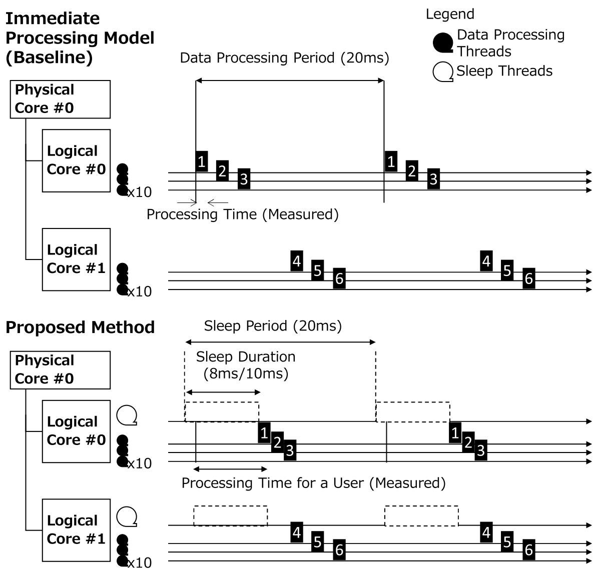

FIGURE 8. **Baseline and proposed method in periodic requests from multiple users.**

TABLE 4. **Power evaluation results for periodic requests from multiple users.**

Num of Users	Baseline	8-ms sleep	10-ms sleep
(2-1) 1 User	1.6W	1.7W (+0.1W)	1.8W (+0.2W)
(2-2) 10 Users	13.5W	10.8W (-2.7W)	9.9W (-3.6W)

C. EVALUATION FOR PERIODIC REQUEST FROM MULTIPLE USERS

In the case of periodic requests from multiple users, each request has to be processed before the users' next request. We aim to reveal whether the CPU can enter a deep sleep by processing a batch of requests from multiple users. We tried (2-2) 10 requests (i.e. 10 users) in a 20-ms period to investigate the proposed method's applicability. The load size of a request is set to one-tenth of (1-1) to host 10 users. For reference, results for a one-user (2-1) case are also obtained. Figure 8 shows how the baseline and proposed method work.

Table 2 shows the results of power evaluation, and Table 5 shows throughput. It can be seen that power-saving effects are obtained with a slight throughput impact in the 10-user case. Figure 9 shows the results of the request processing time evaluation. The average delay is well below 20 ms and seemingly does not affect throughput. On the basis of these findings, we developed the following hypotheses. Because multiple users' requests are processed together, the tail latency may have increased in accordance with the number of requests accumulated in the app's queue. We checked the experimental logs and found that the maximum processing time actually exceeded 20 ms in some cases. It can be said that because of the fluctuation effect compared to single users, the safety factor needs to be considered more carefully in the multiple-user case.

TABLE 5. Throughput evaluation results for periodic requests from multiple users.

Num of Users	Baseline	8-ms sleep	10-ms sleep
(2-1) 1 User	50.01 rps	50.01 rps	50.01 rps
(2-2) 10 Users	500.1 rps	500.0 rps (-0.01%)	499.9 rps (-0.05%)

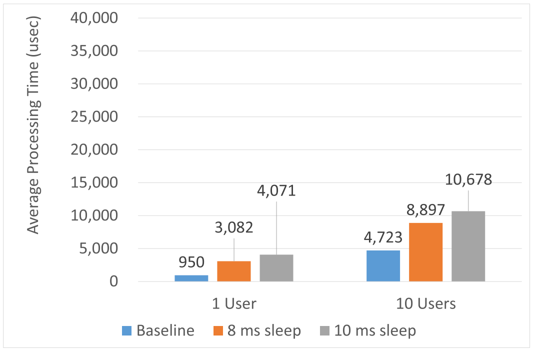

FIGURE 9. Processing time for periodic request from multiple users.

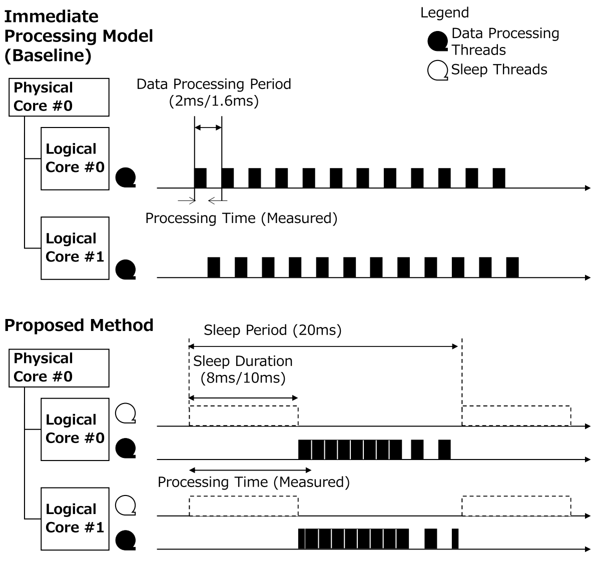

FIGURE 10. Baseline and proposed method in periodic request.

D. EVALUATION FOR INDEPENDENT REQUEST

In the independent request case, unlike the case of periodic requests, each request does not have to be processed before the next request. We aim to reveal whether the CPU can enter a deep sleep by processing a batch of independent requests that come in short periods of time. We prepared two types of requests for this purpose. One (3-1) has a 2-ms interval (500 rps), and the other (3-2) has a 1.6-ms interval (625 rps). Each interval corresponds to CPU utilization of 43% and 54%, respectively. Figure 10 shows how the baseline and proposed method work.

Table 6 shows the results of power evaluation. It can be seen that power-saving effects are obtained at both 2-ms and 1.6-ms intervals. In addition, the amount of reduction increases as the sleep period is increased. The combination of

TABLE 6. Power evaluation results for independent request.

Interval	Baseline	8-ms sleep	10-ms sleep
(3-1) 2 ms	12.9W	9.9W (-3.0W)	9.0W (-3.9W)
(3-2) 1.6 ms	15.0W	11.0W (-4.0W)	9.1W (-5.9W)

TABLE 7. Throughput evaluation results for independent request.

Interval	Baseline	8-ms sleep	10-ms sleep
(3-1) 2 ms	504.8 rps	506.9 rps (+0.41%)	507.7 rps (+0.77%)
(3-2) 1.6 ms	631.1 rps	625.6 rps (-0.87%)	521.1 rps (-17%)

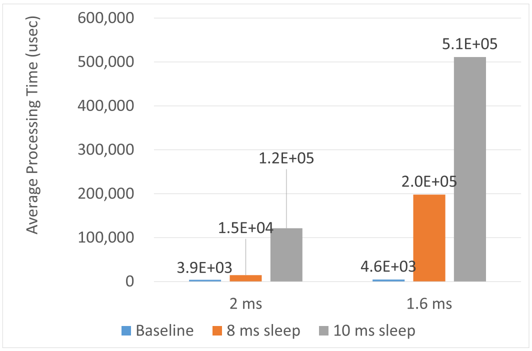

FIGURE 11. Processing time for independent request.

a 2-ms interval and 10-ms sleep reduces power by 3.9W (30% of baseline power), and the combination of a 1.6-ms interval and 10-ms sleep reduces power by 5.9W (39% of baseline power).

Table 7 shows the results of the throughput evaluation. We can see that when the sleep period is short, there is no or slight impact. However, if the sleep period becomes longer, we can see that performance is affected. The abovementioned combination of a 1.6-ms interval (3-2) and 10-ms sleep actually degraded performance. Also, with the 1.6-ms interval, the performance is slightly affected with 8-ms sleep. The combination of a 1.6-ms interval (54% CPU utilization ratio) and 8-ms sleep (40% sleep ratio) should ideally have no performance impact. This means a safety factor needs to be applied when using the proposed method in the independent request case.

Figure 11 shows the results of the request processing time evaluation. It can be seen that the processing time increases drastically as the sleep period is lengthened. The main cause of this is that because multiple requests can be processed at once, as more requests accumulate in the queue, the delay increases rapidly. The 2-ms interval (3-1) and 10-ms sleep appeared to be fine in terms of throughput, but the processing time increased by more than 10 ms. In fact, the system fell into a state where processing could not be completed in time, and this is thought to have affected performance. To avoid such increases in processing time, forced sleep should be skipped when processing time begins to increase.

In summary, the proposed method can take advantage of the power-saving margin that is not utilized in the immediate processing model by aligning the sleep timing between logic cores. The proposed method can also take advantage of the power-saving potential that is not exploited in the immediate processing model by processing multiple requests at once. To avoid performance degradation, the sleep duration needs to be determined with a sufficient safety factor. To avoid processing time increases, forced sleep needs to be skipped appropriately.

VI. CONCLUSION AND FUTURE WORK

To overcome the problem of power wastage with the existing immediate processing model, we proposed, implemented, and evaluated a method to align sleep timing among logical cores. In the proposed method, each logical core is forced to sleep, and the sleep timing can be aligned and multiple requests can be processed together. When the proposed method was applied to the traffic models of periodic requests and independent requests, power was reduced by up to 4.3W (33%) without decreasing the throughput. Determining appropriate sleep duration and skipping forced sleep are needed to avoid performance impact.

The following improvements and evaluations will be considered for future works. The sleep duration should be automatically determined by the processing time characteristics of the application. The sleep start timing should be aligned after the completion of the app process to avoid extra wake-up and sleep in the periodic request case. The arrival timing of independent requests should be modified in accordance with a Poisson process and the performance evaluated again.

REFERENCES

[1] *Current Status and Future Forecast of Data Center Energy Consumption and Technical Issues*, Impact Prog. Inf. Soc. Energy Consumption, Japan Sci. Technol. Agency (JST), Tokyo, Japan, 2021.

[2] C.-H. Chou, L. N. Bhuyan, and D. Wong, "μDPM: Dynamic power management for the microsecond era," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2019, pp. 120–132, doi: 10.1109/HPCA.2019.00032.

[3] X. Zhan, R. Azimi, S. Kanev, D. Brooks, and S. Reda, "CARB: A C-state power management arbiter for latency-critical workloads," *IEEE Comput. Archit. Lett.*, vol. 16, no. 1, pp. 6–9, Jan. 2017, doi: 10.1109/LCA.2016.2537802.

[4] F. Yao, J. Wu, S. Subramaniam, and G. Venkataramani, "WASP: Workload adaptive energy-latency optimization in server farms using server low-power states," in *Proc. IEEE 10th Int. Conf. Cloud Comput. (CLOUD)*, Jun. 2017, pp. 171–178, doi: 10.1109/CLOUD.2017.30.

[5] E. Asyabi, A. Bestavros, E. Sharafzadeh, and T. Zhu, "Peafowl: In-application CPU scheduling to reduce power consumption of in-memory key-value stores," in *Proc. 11th ACM Symp. Cloud Comput.*, Oct. 2020, pp. 150–164, doi: 10.1145/3419111.3421298.

[6] E. Sharafzadeh, S. A. S. Kohroudi, E. Asyabi, and M. Sharifi, "Yawn: A CPU idle-state governor for datacenter applications," in *Proc. 10th ACM SIGOPS Asia–Pacific Workshop Syst.*, Aug. 2019, pp. 91–98, doi: 10.1145/3343737.3343740.

[7] C.-H. Chou, D. Wong, and L. N. Bhuyan, "DynSleep: Fine-grained power management for a latency-critical data center application," in *Proc. Int. Symp. Low Power Electron. Design*, Aug. 2016, pp. 212–217, doi: 10.1145/2934583.2934616.

[8] C. Gough, I. Steiner, and W. A. Saunders, *Energy Efficient Servers: Blueprints for Data Center Optimization*. New York, NY, USA: Apress, 2015, doi: 10.1007/978-1-4302-6638-9.

[9] Intel Corporation. (2011). *Energy-Efficient Platforms—Considerations for Application Software and Services*. [Online]. Available: https://www.intel.com/content/dam/doc/white-paper/energy-efficient-platforms-2011-white-paper.pdf

IKUO OTANI received the B.S. and M.S. degrees in physics from The University of Tokyo, in 2011 and 2013, respectively. Since 2013, he has been with NTT Network Service System Laboratories, where he has engaged in improving the efficiency of network processing in virtualized servers. From 2019 to 2021, he engaged in developing virtualized server infrastructure for the core network at NTT Docomo, Inc. He is currently a Research Engineer at the NTT Network Innovation Center. His research interests include power-aware computing and hardware-assisted task processing. He is a member of IEICE. He received the Network System Research Award and the Young Researcher's Award from the IEICE Technical Committee on Network Systems, in 2018.

KEI FUJIMOTO received the B.E. degree in electrical and electronic engineering and the M.S. degree in informatics from Kyoto University, in 2008 and 2010, respectively. Since 2010, he has been with NTT Network Service System Laboratories, where he has engaged in the development of a transfer system for ISDN services and research of network-system reliability and network API. From 2016 to 2018, he engaged in the creation of new services related to big data at NTT West Corporation. He is currently the Senior Manager at the NTT Network Innovation Center. His current research interests include low-latency networking and power-aware computing. He is a member of IEICE. He was a recipient of the Young Researcher's Award from the IEICE Technical Committee on Network Systems, in 2020, the Highly Commended Paper Award from the IEEE ITNAC, in 2021, and the Best Paper Award from the IEICE, in 2023.