

# A Comparative Study of the Effectiveness of CPU Consolidation versus Dynamic Voltage and Frequency Scaling in a Virtualized Multi-Core Server

Inkwon Hwang, Student Member and Massoud Pedram, Fellow, IEEE

**Abstract**—Companies operating large data centers are focusing on how to reduce the electrical energy costs of operating data centers. A common way of cost reduction is to perform dynamic voltage and frequency scaling (DVFS), thereby matching the CPU's performance and power level to incoming workloads. Another power saving technique is CPU consolidation, which uses the minimum number of CPUs necessary to meet the service request demands and turns off the remaining unused CPUs. DVFS has been already extensively studied and verified its effectiveness. On the other hand, it is necessary to study more about effectiveness of CPU consolidation. Key questions that must be answered are how effectively the CPU consolidation improves the energy efficiency and how to maximize the improvement. These questions are addressed in this paper. After understanding modern power management techniques and developing an appropriate power model, this paper provides an extensive set of hardware-based experimental results and makes suggestions about how to maximize energy efficiency improvement through CPU consolidation. In addition, the paper also presents new online CPU consolidation algorithms, which reduce the energy delay product up to 13% compared to the Linux default DVFS algorithm.

**Index Terms**—Algorithm, consolidation, energy efficiency, and virtualization.

## I. INTRODUCTION

DATA centers consist of a very large number of server machines that can be leased to provide cloud services to a whole slew of clients running many different applications. The number of servers employed in data centers has been rapidly increasing, confirmed by continuous increase in the BLADE server shipments in US and worldwide. Although the energy efficiency of server machines has been improving, this efficiency advances have not kept pace with the increase in cloud computing services and the concomitant increase in the number and size of data centers. As a result, an ever increasing amount of electrical energy is being consumed in today's data

centers, giving rise to concerns about the carbon emission footprint of data centers and the costs of operating them. The latter is especially important concern from the viewpoint of datacenter owners and operators (as well as their customers/clients who must eventually pay the bill).

Two widely employed techniques for reducing the costs of operating data centers are server consolidation and DVFS. The former aims at minimizing the number of ON servers in a datacenter by consolidating all the incoming jobs into as few servers as possible whereas the latter attempts to match the performance of each ON server to the current workloads so that energy can be saved at the workload level of each server. Server consolidation is needed and complements DVFS technique because of the energy non-proportional behavior of modern servers [1], and an unfortunate effect by which a server machine operating at a low performance level tends to consume power close to the power it consumes at its peak performance level. This is somewhat natural and expected because an electronic circuit (with server being a special case) consumes static power (leakage in CMOS digital circuits) regardless of whether it provides any computational services. The issue is, however, worse than simple leakage and has to do with the fact that many components within a modern server system (e.g., "uncore" logic within the processor chip, DRAM modules on the board, many of the I/O controllers, and even the network interface) cannot be scaled/modulated to exhibit a linear relationship between their power consumption and delivered performance levels.

A data center is typically under-utilized; it has been designed to provide the required performance and satisfy its service level agreements (SLAs) with clients even during peak workload hours, and hence, at other times its resources are vastly under-utilized. For example, the minimum and the maximum utilization of the statically provisioned capacity of Facebook's data center are 40% and 90%, respectively [2]. Hence, in light of the energy non-proportionality of today's server base, a greater amount of energy costs can be reduced by consolidating jobs into as few server machines as possible and turning off the unused machines. The server consolidation has been studied very well, and many studies have suggested the use of virtual machine migration (VMM) as a means of doing server consolidation [3-7].

Manuscript received April 15, 2015; revised August 31, 2015; accepted October 14, 2015. This work was supported in part by a grant from the Semiconductor Research Corporation (No. 2012-HJ-2292).

The authors are with the Department of Electrical Engineering, University of Southern California, Los Angeles, CA 90089 USA (e-mail: inkwonhw@usc.edu; pedram@usc.edu).

Although server consolidations can greatly lower a data center's total energy consumption, there is still room for further energy savings due to the limitations and overheads associated with the server consolidation. For one, it is difficult to conduct server consolidation very frequently because the migration of tasks or virtual machines causes high overheads; e.g., heavy network traffic, high network latency and large system boot time, plus large energy consumption to move virtual machines and their local contexts around. Because of these overheads, there is a relatively long period between server consolidation decision times. To avoid the SLA violations during each timing period when virtual machine to server assignments are fixed, virtual machines (or tasks) are not too tightly consolidated into the active server set in order to provide a safe margin of operation. Too aggressive a server consolidation strategy will result in violation of client SLAs. The longer the period between migrations is, the larger the aforesaid margin should be (i.e., more server machines should be activated, each at a lower average utilization rate). Hence, server machines are still under-utilized after datacenter-level server consolidation, which implies that there is the potential of further energy savings through additional resource management techniques as explained below.

There are a number of resources in a computer system such as computing, storage, and I/O bandwidth. This study focuses on the computing resource, i.e., the CPU, which is a major energy consumer. A well-known and popular energy-aware CPU management technique is a *Dynamic Voltage and Frequency Scaling* (DVFS) [3, 8, 9]. The DVFS was introduced decades ago, and it has been one of the most effective power saving techniques for CPUs. The amount of the energy savings by DVFS, however, is decreasing due to the following reasons. First, the supply voltages have already become quite low; hence, the remaining headroom for further supply voltage reductions is small and shrinking. Second, many modern servers have two or more processor chips, each chip containing multiple CPUs (cores<sup>1</sup>) but a single on-chip power distribution network shared by all the CPUs. Because of this sharing, the CPUs on the same package must operate at the same supply voltage level and hence the same clock frequencies<sup>2</sup>. Unless we do 'perfect' load balancing among CPUs sharing the same power bus, the voltage level that is set for the most highly loaded CPU will result in energy waste because all the other under-utilized CPUs run at higher frequency than what is actually needed. Third, in a virtualized server system, it is difficult to gather sufficient information about the running applications, which is necessary to choose the optimal clock frequency and voltage level for the CPUs. This is because the *virtual machine manager* (hypervisor), which conducts DVFS, resides in a privileged domain whereas the applications are running in a different domain (virtual machine domain) [4].

Another well-known CPU energy management technique is *Dynamic Power State Switching*. Many modern processors support multiple power states (known as C, Core C, and

Package C-states). Each C-state specifies the circuits that are turned ON or OFF. Based on the history of recent workloads of CPUs, the *operating system* (OS) suggests C-state of each CPU. Based on this suggestion, appropriate CC and PC-states are chosen by a *Power Control Unit* (PCU). This PCU is programmed using fine-tuned algorithms. These algorithms are well designed but still not perfect, so we believe that further energy savings can be achieved if there is software-level assistance. In this study, we present a CPU consolidation technique, which helps the PCU to reduce more energy. This technique explicitly defines sets of active and inactive (sleep) CPUs. It guarantees to meet performance target and reduce energy consumption by minimizing power state switches.

There are research studies that investigate the effectiveness of the CPU consolidation. In [10] the authors show that consolidation across CPUs in a single processor and two processor systems offers a very small amount of energy savings. They used their own benchmark which is not the standard and may not create realistic workloads. In [11] Jacob et al. compare *core-level power gating* (CPG) with DVFS and show that CPG saves more energy by 30% than DVFS. This result implies the energy savings by the CPU consolidation may be larger if the processor supports the CPG. However, the reported results are calculated from a combination of real measurements and their leakage estimator (the adopted leakage power model is somewhat simple). In [2] the authors present a technique called *core count management* (CCM), which is a variant of the CPU consolidation technique, and report 35% energy savings. However, all results are obtained from a simulator, and the power and performance models used in the simulator are again fairly simplistic.

This paper is differentiated from the prior work because of the following reasons. First, all results are obtained not from a simulation but from actual hardware measurements. Second, well designed benchmarks (PARSEC and SPECWeb2009) are used for the experiment, and we believe the workloads from these benchmarks are realistic. Third, it is intensively analyzed and discussed how CPU consolidation improves energy efficiency rather than just showing final results. Fourth, this paper compares energy efficiency improvement by CPU consolidation, DVFS, and combined techniques (i.e., both consolidation and DVFS are combined).

A preliminary version of this work has been published in [12]. This archival paper is a substantially extended version, which includes a completely new power model, using a new benchmark (PARSEC), vastly more detailed experimental results and discussions, and a more efficient online CPU consolidation algorithm.

The strength of this paper is that all results are collected not based on any simulations but rather from experiments on a server system comprising two Intel® Xeon® Westmere E5620 processors. It is true that our conclusions are therefore applicable to the server under consideration. However, as also explained below, this server is an archetypical Intel server that can be used in a wide range of scenarios from small cluster of servers to larger data center computing. Hence, we believe there is value and benefit in understanding and quantifying the

<sup>1</sup> The terms 'CPU' and 'core' are used interchangeably in this paper.

<sup>2</sup> Some processors are capable of independent DVFS among cores while Intel® processors are not. Intel® Xeon® processors are used for this study.

energy efficiency gains that can be achieved by using the right combination of DVFS and CPU consolidation in such a server. Clearly, the basic power management strategy must be enhanced when we go from the level of an individual server or a small server clusters to a very large (potentially heterogeneous) number of servers in a data center (see for example, references [13-15]); However, further study on the server level will be essential and important in deriving global power management strategies at higher levels. Moreover, to the best of our knowledge, this is the only published result that provides such a detailed discussion of the core and package C-states in a modern server, develops and validates power and latency models for applications running on these servers as a function of the deployed DVFS and/or consolidation policy, and suggests effective yet simple, solutions for applying a combination of DVFS and CPU consolidation to these servers (esp. for I/O bound applications).

The remainder of the paper is organized as follows. Existing CPU power management technologies are reviewed in Section II. In Section III we present the power and delay models. Based on these models we discuss how CPU consolidation can improve energy efficiency. The metrics for quantifying energy efficiency are presented in section IV. The experimental system setup is explained in Section V. Section VI presents our extensive experimental results and discussions. Finally, we summarize the results and provide final conclusions and insights in Section VII.

## II. BACKGROUND – POWER MANAGEMENT TECHNOLOGIES

The purpose of this study is to understand how effectively CPU consolidation improves the energy efficiency of server systems so as to maximize the improvement. The consolidation interacts with existing power management technologies, so it is helpful to understand these technologies. In this section, the processor power and performance states are briefly reviewed. In addition, we review Intel® Quickpath Interface technology which may affect the energy savings by consolidation. Before starting a discussion, a few confusing terms is clearly defined below:

- *CPU* – all circuits used to perform arithmetic/logic operations and L1 and L2 cache memories. A term ‘*core*’ is used interchangeably with a word ‘*CPU*’ in this paper
- *uncore* – all components in a processor except cores
- *package* – a physical unit which has core and uncore. A word ‘*package*’ is used interchangeably with a word ‘*processor*’
- *CPU consolidation* – it is simply called ‘consolidation’ unless it is confusing
- *total utilization* – sum of percentage of times when CPUs are running codes. For example, when two CPUs are fully utilized, the total utilization is 200%
- *average utilization* – average utilization per core. It is calculated by dividing total utilization by the number of active CPUs, so it should be equal to or less than 100%. This term is simply called ‘*utilization*’ or ‘*util*’
- *throughput* – the number of tasks (jobs) processed in a second
- *delay* – total amount of time spent for executing a task. This includes the time when a task is suspended and that waiting in a queue of the CPU scheduler.

### A. Processor power states (*C*, *CC*, *PC*-States)

The Advanced Configuration and Power Interface (ACPI) specification was developed as an open standard for OS-directed power management. Many modern operating systems (OS) meet this specification. First, the specification defines C-States as processor power states; when a processor is in a higher-numbered C-State, which is also called a ‘deeper’ sleep state, a larger area of internal circuitry is turned off or inactive, which reduces power dissipation. On the other hand, it also takes longer time to go back to the operating state (i.e., C0 state). The number of supported C-States is processor dependent. For example, the Intel® Core™ i7 processor (code-named Nehalem) supports the following core states: C0 (normal operating mode—cores in this state are either executing code or are in standby), C1 (autoHALT—a low power state entered when all threads within the core execute a HLT or MWAIT instruction), C1E (autoHALT with lowest frequency and voltage operating point), C3 (deep sleep—cores in this state flush their L1 instruction cache, L1 data cache, and L2 cache to the L3 shared cache; Clocks are shut off to each core), and C6 (deep power down—cores in this state save their architectural state before removing core voltage). See Intel® Core™ i7 datasheets for more detailed information.

The C-state is also known as a logical C-State. An OS ‘requests’ a change in C-State of logical cores<sup>3</sup>, but the request may be denied (e.g., auto demotion). The decision about demotion is made based on each core’s immediate residency history (think of this as the breakeven time for the proposed power state transition); if an amount of future residency in idle, which is estimated from the residency history, is insufficient, then a request to transition a core into a deeper power state will be ignored. In general, the entry/exit costs (latency and energy overheads) increase when the processor/core escapes from a deeper sleep state; hence, the auto demotion prevents unnecessary excursions into deeper power states, and thereby, it reduces both latency and energy overheads of power state switches.

In addition to the logical C-states there are two more types of hardware C-States: *core* ( $CC_n$ ) and *package* C-state ( $PC_n$ ). Based on logical C-state switch request from an OS, a *Power Control Unit* (PCU) decides about the CC and PC-states. Each core and/or package can switch its power state independently; that is, CC-state of a core may be different from that of other cores. Likewise, PC-state of one package could be different from that of another package. The  $CC_0$  state is a special state that deserves more discussion; a core is in this state when it is executing tasks or when it is standing by for the next task to arrive (but is still in the normal operating mode). Note that a core does not immediately switch from  $CC_0$  state to a deeper sleep state when it becomes idle.

The PC-state of a package depends on the CC-states of cores in the package. In particular, a package state can be  $PC_k$  only when all of its cores are in  $CC_k$  state or deeper CC-states. It is because some resources shared by cores cannot be turned off

<sup>3</sup> A logical core is identical to a physical core unless Intel® hyper-threading is enabled. In this study hyper-threading was disabled.

unless all cores are inactive. For example, the Intel® i7 processor's L3 cache is shared by multiple cores, so a package should stay in active state if any core is still active. Otherwise, the active cores cannot function properly.

### B. Processor performance state (P-States)

While a processor executes codes, it can be in one of several performance states (P-State), which specify the clock frequency and corresponding voltage level. At the higher frequency, the performance is higher, but its power dissipation is also higher. Similar to the C-States, the number of supported P-States is processor- dependent, and a frequency is higher at lower numbered P-States, that is, P0 is the highest performance state.

An OS chooses the P-State based on the historical workload information. The OS may not choose the same state for all cores, but all cores in Intel® processors will run at the same clock frequency. Therefore, even if the OS sets different P-States for the cores, only one state is selected and applied. In general, the highest performance state is chosen, but another decision policy may be used. Because of this hardware constraint of the current Intel® processors, it is recommended to distribute the workload evenly among all active cores. Otherwise, the selected P-State will be appropriate only for some cores, but not for the others.

### C. Core-level power gating

Recent state-of-the-art Intel® processors are capable of core-level power gating; processors can completely shut down some of the cores (the OFF cores consume nearly zero power). Processors with the power gating feature have additional C-State at which power dissipation is nearly zero, but with the largest entry/exit costs. Note that the processor used in this study supports core-level power gating.

### D. Intel® Quickpath Interconnect (QPI)

In the past there was only one memory controller (MC) in a system because one MC was enough for single core/processor systems. However, modern server systems, which have multiple packages (processors), need multiple MCs to achieve a target performance level. With a single MC, performance of all packages reduces as soon as a CPU in one package uses memory controller heavily. In order to address such a memory bottleneck issue, Intel® invented QPI; each package owns an integrated memory controller (IMC) and communicates with other packages through the QPI. An example of a dual processor system is shown in Fig. 1. . Half of DDR3 slots are directly connected to a first package while the other half is connected to a second package. Therefore, if a core in package #1 needs data located in a remote DDR3, which is connected to package #2, the data is requested and received through QPI. Hence, the PC-state of some package may not be the deepest sleep state even when all CPUs in that package are inactive. This is because any CPU in other packages may be requesting data from the DDR3 slots that are directly connected to the package in question. Due to this phenomenon, it is expected that power savings achievable by package-level consolidation are negligible. This will be discussed in more details in a later section.

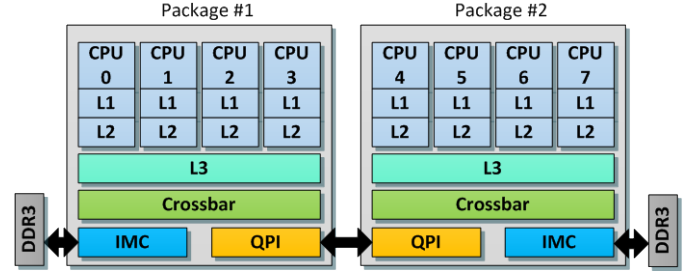


Fig. 1. Intel® QPI block diagram

## III. POWER, DELAY AND CONSOLIDATION

In this section we present power and delay models for Chip Multiprocessor (CMP) server systems. Using these models, we will discuss how the consolidation affects the energy efficiency and delay. This discussion is abstraction level, so it may be too simplified to cover everything of real situations. However, we believe it is sufficient for deriving insights. The discussion about the power/latency tradeoffs will be verified by empirical results in Section VI. Note that thermal issues (e.g., leakage power variation as a function of chip temperature) are not considered. This is because we can do consolidation only when the system is under-utilized, which also implies that the temperature of processor chips is not very high.

### A. Power model

This section presents a full platform-level power dissipation model, accounting for the power consumed by all components within a server system. This model estimates the system power dissipation by using statistical data reported by the system; i.e., the percentage of time spent in specific CC and PC-states.

The processor power dissipation consists of *core* and *uncore* power dissipations. Some notations used for the model and their definitions are shown below:

- $P_{active}^{core}$  – Power dissipation by a core when the core is active and executing code. This active power is a function of P-State of active cores. This is also dependent on the type of workload, but we do not consider this factor because consolidation does not change characteristics of the current workload.
- $P_{CC_n}^{core}$  – Power dissipation by a core at  $CC_n$  state. Note that  $P_{CC_0}^{core}$  is different from  $P_{active}^{core}$ : a core may be simply standing by while it is in  $CC_0$  (i.e., it is not executing any tasks although it is fully on)
- $P_{PC_n}^{uncore}$  – Power dissipation by the uncore when a package is in the  $PC_n$  state
- $T_{active}^{core_i}$  – Percentage of time when a core is active and executing tasks, which is also called *core utilization* ( $util_i$ )
- $T_{CC_n}^{core_i}$  – Percentage of time when a core is in  $CC_n$  state
- $T_{PC_n}^{uncore}$  – Percentage of time spent when a package is in  $PC_n$  state

Total (server platform) power dissipation is the sum of the processor power dissipation ( $P^{proc}$ ) and the power consumed by other system components ( $P^{other}$ ), e.g., I/O, memory, and hard disc drive.  $P^{other}$  is fixed and independent of the DVFS or CPU consolidation, so it acts as a fixed offset on top of the  $P^{proc}$ .

$$P^{total} = P^{proc} + P^{other} = \left( \sum_i P^{core_i} + \sum_j P^{uncore_j} \right) + P^{other} \quad (1)$$

The core power dissipation can be estimated using  $P_{active}^{core}$ ,  $P_{CC_n}^{core}$ ,  $T_{active}^{core_i}$  and  $T_{CC_n}^{core_i}$  as shown below.  $CC_0$  is a special state; a core is in the  $CC_0$  state when the core is executing codes. However, the core may be in the  $CC_0$  state even when it is idle. For example, a CPU stays in the  $CC_0$  state for a certain amount of time (i.e., a timeout period) before switching to deeper power state.

$$\begin{aligned} P^{core_i} &= P_{active}^{core} \cdot T_{active}^{core_i} + P_{CC_0}^{core} \cdot (T_{CC_0}^{core_i} - T_{active}^{core_i}) \\ &+ \sum_{n \geq 1} P_{CC_n}^{core} \cdot T_{CC_n}^{core_i} \\ &= (P_{active}^{core} - P_{CC_0}^{core}) \cdot T_{active}^{core_i} + \sum_{n \geq 0} P_{CC_n}^{core} \cdot T_{CC_n}^{core_i} \end{aligned} \quad (2)$$

Similar to the core power dissipation, the uncore power dissipation is:

$$P^{uncore_j} = \sum_n P_{PC_n}^{uncore} \cdot T_{PC_n}^{uncore_j} \quad (3)$$

Note that  $\forall i: \sum_{n \geq 0} T_{CC_n}^{core_i} = 1$  and  $\forall j: \sum_{n \geq 0} T_{PC_n}^{uncore_j} = 1$

### B. CPU consolidation and power dissipation

In this section we discuss whether consolidation reduces power dissipation or not. We do not cover its impact on delay, which is discussed in the following section. The discussion here is based on an assumption that consolidation is performed in a correct way so that throughput remains the same; in other words, a sufficiently large number of CPUs are always active. Lower power dissipation at the same throughput means less energy is consumed for the same workloads, which also means more energy-efficient operation. The discussion in this section focuses on how power dissipation changes with consolidation. Therefore, we can see if energy efficiency is improved or not.

Consolidation reduces the number of active CPUs, that is, the type and level of workloads do not change. Therefore, it is expected that  $P^{other}$  is not affected by consolidation. Hence, we focus on changes in the core and uncore power dissipations:

$$\Delta P^{total} = \Delta \left( \sum_i P^{core_i} \right) + \Delta \left( \sum_j P^{uncore_j} \right) \quad (4)$$

We start with the power impact on cores:

$$\begin{aligned} \Delta \left( \sum_i P^{core_i} \right) &= (P_{active}^{core} - P_{CC_0}^{core}) \Delta \left( \sum_i T_{active}^{core_i} \right) \\ &+ \sum_{n \geq 0} P_{CC_n}^{core} \Delta \left( \sum_i T_{CC_n}^{core_i} \right) \end{aligned} \quad (5)$$

In the above equation, term  $\sum_i T_{active}^{core_i}$  is not affected by consolidation because the workload level does not change (i.e.,  $\Delta \sum_i T_{active}^{core_i} = 0$ ). Therefore,

$$\Delta \left( \sum_i P^{core_i} \right) = \sum_{n \geq 0} P_{CC_n}^{core} \Delta \left( \sum_i T_{CC_n}^{core_i} \right) \quad (6)$$

As shown in the above equation, the power savings of consolidation is a function of changes in the sum of  $T_{CC_n}^{core_i}$ .

Let us assume that power state transition is ideal: 1. CC-state immediately switches to the deepest sleep state ( $CC_6$ ) without any delay when a core becomes idle (i.e.,  $T_{active}^{core_i} = T_{CC_0}^{core_i}$ ) 2. There is no power state switch cost: additional delay and power consumption when power state switches. Based on these assumptions, there is negligible change in core power dissipation by consolidation because all cores are in the  $CC_6$  state when they are idle:

$$\begin{aligned} \Delta \left( \sum_i P^{core_i} \right) &= P_{CC_6}^{core} \Delta \left( \sum_i T_{CC_6}^{core_i} \right) = P_{CC_6}^{core} \Delta \left( \sum_i (1 - T_{active}^{core_i}) \right) \\ &= P_{CC_6}^{core} \left( -\Delta \sum_i T_{active}^{core_i} \right) = 0 \end{aligned} \quad (7)$$

However, this assumption is not realistic. Because of non-negligible switch costs, a core may not switch promptly its power state when it becomes idle; if the core is in low power state for the very short time, then switch costs could be greater than power savings by the switch. Consolidation can decrease power consumption by reducing the number of switch (hence, reducing switch costs). Fig. 2. depicts an example which shows how consolidation reduces the costs; there are two CPUs and two CC-states available:  $CC_0$  and  $CC_6$ . When a task is given to a CPU, the CPU executes the task ( $CC_{0-active}$ ). When the execution is done, the CPU stays in the  $CC_0$  state ( $CC_{0-idle}$ ) for certain amount of time before switching to  $CC_6$ . For the rest of period, the CPU is in the  $CC_6$  state. From the upper case in the figure (Fig. 2. (a)), we can see one  $CC_6$ -to- $CC_0$  switch and two  $CC_0$ -to- $CC_6$  switches. On the other hand, from a consolidation case (Fig. 2. (b)), there is only one switch:  $CC_0$ -to- $CC_6$ . In addition, CPUs reside in the  $CC_{0-idle}$  state for shorter amount of time. Therefore, in this example, consolidation reduces power dissipation, which also means it improves energy efficiency. However, consolidation may increase execution time of a task. In this example, the 2<sup>nd</sup> task cannot be executed promptly because a CPU is running the previous task (task 1) and this CPU is the only one active CPU. Therefore, we have to consider performance degradation and decide whether performs consolidation or not.

Second, we discuss about power dissipation impact on uncore power by consolidation. As discussed in Section II.A, a package can switch its power state to deeper one only when all cores in the package are idle. A uncore can stay longer at deeper power state when both CPUs are active (Fig. 2. (a)); there is overlap where both CPUs are in active state, so both CPUs are in the  $CC_6$  state for longer period than the 2<sup>nd</sup> case where only one CPU is active. In other words, consolidation may increase uncore power. However, consolidation can reduce the percentage of time spent in the  $CC_{0-idle}$  state, so if the reduction is greater than the overlap, uncore power dissipation may be reduced by consolidation.

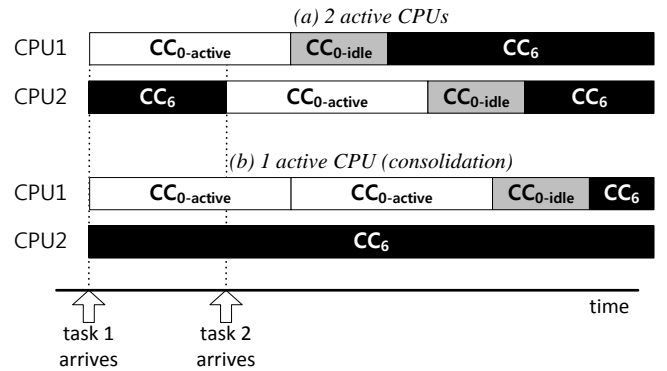


Fig. 2. Example of CC-state switch by consolidation

We have discussed about impacts on power dissipation using the power model, but real system is too complicated for the

model to consider all factors which affect power dissipation. Therefore, we run experiment on a real system and quantify the power savings by consolidation.

### C. Delay model

As discussed in the previous section, consolidation may increase delay of tasks. In this section, we discuss the impact of consolidation on delay. The proposed delay model is a function of core utilization ( $T_{active}^{core_i}$ ). In general, the delay increases rapidly when a CPU approaches full utilization [5]:

$$D_i = \frac{e}{1 - T_{active}^{core_i}} + f \quad (8)$$

$D_i$  is the delay of the  $i^{\text{th}}$  CPU ( $core_i$ ). Coefficient  $e$  represents how sensitive the delay is to the core utilization; with larger  $e$ , delay increases more rapidly as the core utilization approaches 1. Another coefficient  $f$  represents a lower bound on the delay, that is, the delay may not reduce below certain value even when the core utilization is very low ( $D_i \geq e + f$ ). These coefficients are task-dependent, that is, coefficients for one task might be different from those for another task. They are also hardware-dependent. This delay is affected by consolidation because  $T_{active}^{core_i}$  is a function of the active CPU count. When  $K$  tasks are assigned to the system every second, the tasks are evenly distributed to the  $m$  active CPUs by a scheduler; therefore, each CPU is assigned  $K/m$  tasks every second. The core utilization  $T_{active}^{core_i}$  is linearly proportional to the workload ( $K/m$ ):

$$T_{active}^{core_i} = d(K/m) \quad (9)$$

Coefficient  $d$  represents the amount of CPU resource (i.e. the number of CPU cycles) needed for executing a task. A task with higher  $d$  needs more CPU cycles compared to another task with smaller  $d$ . Now we can model the delay as a function of the active CPU count ( $m$ ) and the total number of tasks ( $K$ ):

$$D = \frac{e}{1 - d(K/m)} + f \quad (10)$$

Delay increases as the core utilization  $T_{active}^{core_i}$  increases. The increase rate at high utilization is greater than that at low utilization. Consolidation increases the core utilization, but if we keep the core utilization lower than certain level (threshold), then delay increase by consolidation will be insignificant. Hence, it is important to find this threshold and keep core utilization lower than the threshold.

Coefficients in the delay model may be application dependent, so a threshold for one application may be different from that for another application. Therefore, we will find thresholds for various kinds of benchmark tests. From experiments, we recommend that the average core utilization is no more than 70% for CPU-bound applications. Note that for memory-bound applications where the execution time limit is tight, contention can occur on other shared resources (including bus, second level cache, and main memory) and hence a limit on average CPU utilization will not be sufficient. However as we will show later for such applications CPU consolidation is not an effective technique anyways. Details will be presented in Section VI.

## IV. ENERGY EFFICIENCY METRICS

In the previous discussion, the term ‘energy efficiency’ has been used without defining it. In order to determine if consolidation improves the energy efficiency or not, we have to precisely define what ‘energy efficiency’ is. Depending on how it is defined, consolidation may or may not enhance the energy efficiency. In this study we use two metrics for energy efficiency: *energy per task* (E/task) and *energy-delay product per task* (ED/task).

### A. Energy per task (E/task)

This metric is often used for comparing energy efficiency among different platforms. A term ‘task’ denotes an instance of executing a specified benchmark. This metric is simply calculated using average power consumption ( $P_{avg}$ ) and throughput (i.e., the number of tasks processed in a second):

$$E / task = \frac{E_{gross}}{\#of\ tasks} = \frac{P_{avg} \times Time}{\#of\ tasks} = \frac{P_{avg}}{throughput} \quad (11)$$

The consolidation may decrease this metric, but it can also reduce performance. If ‘throughput’ is selected as a performance indicator, then this metric also includes performance information in it. If consolidation reduces E/tasks, we can say energy savings dominate performance degradation (i.e., throughput reduction). However, for another performance definition, this metric may be insufficient; if we have to care of execution time as well as throughput, this metric does not include performance information. For example, if the execution time increases due to consolidation but throughput does not change, this metric shows that the energy efficiency is improved without any performance degradation. This may mislead into a wrong decision. Hence, we introduce another metric at the following section.

### B. Energy-delay product per task (ED/task)

A ‘delay’ in this metric is the average execution time of tasks. This also includes period when a task is suspended by the CPU scheduler and CPUs execute other tasks. This metric is calculated using the average power dissipation, throughput, and execution time:

$$ED / task = (E / task) \times delay = \frac{P_{avg} \times delay}{throughput} \quad (12)$$

Depending on a metric a different power management technique can be determined as the best one. Hence, we will report energy efficiency improvement of both metrics.

## V. EXPERIMENTAL SETUP

A goal of this study is to quantify energy efficiency improvement of consolidation and to find a way to maximize the improvement. In addition, we will compare consolidation with DVFS which is the most popular technique. Because a real system is too complicated to be well simulated, all data shown in the following sections are measured from experiments (not simulations).

### A. Hardware test-bed and XEN

The server system under test has two Intel<sup>®</sup> Xeon<sup>®</sup> Westmere E5620 processor packages, and each package in turn includes

four CPUs in it (Fig. 1. ). As mentioned in Section II.B, all CPUs in the same package run at the same clock frequency and voltage. However, the power state of a CPU can be different from that of the other CPUs in the same package. Each 64-bit CPU has its own dedicated 256KB L1 and 1MB L2 caches but shares a 12 MB L3 cache with the other CPUs. The total size of the system memory is 6GBytes. This processor supports seven clock frequency levels, from 1.6GHz to 2.4GHz.

This considered server system may appear too small to represent typical servers in data centers. A common myth is that data centers always consist of large-size servers which have many processors. In fact this is not true for all data centers; the Google data center consists of clusters of inexpensive desktop-class machines [16, 17]. As another example, the Facebook data center is comprised of dual processor servers [18, 19]. There are a few reasons why data centers consist of many small servers rather than fewer large-size servers [20]: first, resource management in many processor servers is a complex and challenging task, so actual performance may not high enough. Second, the license cost of resource management software for large servers is high. Third, it is tricky to properly handle a failure of individual component, that is, failure of one processor in a large server may cause the whole server system to fail, taking out a big chunk of computing resources within a data center. Hence, our setup is realistic and representative of typical server systems found in some data centers.

A power analyzer tool measures the total platform (system) power dissipation, which includes total power consumed by all components; e.g., processor, HDD, DRAM, fan, and so on. None of the components other than CPU are optimized to achieve any power savings. For example, cooling fans are running at highest speed all the time and high performance HDDs are used all the time in order to avoid any risk of performance degradation. Hence, the system power dissipation is very high even when the system is idle (we call this quantity the *standby power* from now on). In order to compensate potential power inefficiency of other system components, we calculate and report ‘power dissipation’ as the difference between the total system power and the standby power:

$$power = power_{measured} - power_{standby} \quad (13)$$

The reported power value thus accounts for dynamic power consumption of all system components. The standby power of our system is 98.1W. When the system is fully loaded, the system power is about 160W; that is, we report 61.9W as the power consumption. Consolidation is needed only when the system is under-utilized. If the average core utilization is 50%, calculated power consumption is about 30W. If consolidation reduces power dissipation by 15W, then we report 50% power savings. On the other hand, the power savings would have to be reported as only 12% if we had used the total system power for the calculation. We believe reporting 50% total dynamic power saving is more indicative of the actual effect of consolidation that reporting 12% saving in the total platform power. All power dissipation numbers reported in the following sections are calculated using the above equation unless there is specific description. A photo of the system under test is shown in Fig. 3.



Fig. 3. The server system along with a power analyzer

We have built the virtualized system using XEN (version 4.0.1), which is an open source hypervisor-based virtualization product and provides the APIs for changing VM configurations: the number of virtual CPUs (vCPU), clock frequencies, and the set of active CPUs. We change these configurations by calling the XEN built-in functions.

### B. Benchmarks- PARSEC and SPECWeb2009

For this study two different benchmark suits are used: 1. the Princeton Application Repository for Shared-Memory Computers (PARSEC) [21] and 2. SPECWeb2009. PARSEC consists of 13 multithreaded and shared-memory programs, which represent next-generation programs for CMP. All these programs are designed and developed for real applications. Characteristics of these programs are very different from one another, and they represent wide range of applications. Therefore, we can make strong conclusion using PARSEC benchmark. Note that there are total of 13 programs are provided, but we use 11 programs. It is because ‘*facesim*’ and ‘*ferret*’ programs are very unstable and often crashed in our setup. The PARSEC benchmark does not provide an I/O-bound program, so SPECWeb2009 was used as an I/O-bound one.

For PARSEC, we present improvement of both metrics: E/task and ED/task. On the other hand, for SPECWeb2009, we only present ED/task. Delay, which is defined as turn-around time for SPECWeb2009, is very important for web service, so E/task is not an appropriate metric for SPECWeb2009.

## VI. EXPERIMENTAL RESULTS AND DISCUSSIONS

In this section, experimental results of PARSEC and SPECWeb2009 running on the system under test are presented and discussed below. We start from presenting a detailed power model as a function of CC and PC-states. We also investigate the consolidation overhead<sup>4</sup> and suggest that the number of virtual CPUs (vCPU) has to be dynamically changed to reduce the overhead. It is also important to find out which set of CPUs should be active in order to maximize energy efficiency. Next we report the E/task and ED/task improvement of PARSEC using three techniques: 1. DVFS, 2. Consolidation, and 3. Combined. Finally, we present a highly effective, yet simple, online consolidation algorithms for SPECWeb2009 and report energy efficiency improvement.

<sup>4</sup> The DVFS overhead has been extensively studied in reference [22].

### A. Power model derivation and verification

This section presents a full platform-level power dissipation model, accounting for the power consumed by the core and uncore components within the target server system. As will be seen, this model is more detailed than the generic one that was described in Section III.A.

Our system allows limiting the deepest C-state, and we can set the limit to C1, C2, or C3 by using the *xenpm* [23] of the XEN hypervisor. The hardware-reported information for each C-state limit is shown in TABLE I. As shown in this table, not all information is available; percentage of times spent in the  $CC_0$ ,  $CC_1$ ,  $PC_0$ , and  $PC_1$  states are not reported, hence these unreported times will be estimated. Our goal is to estimate the power dissipation when all C-states are available, i.e., the C-state limit is C3, but this is a difficult undertaking. Therefore, we start from the simplest case when the C-state limit is C1. Subsequently, we go over the second case when the C-state limit is C2. Finally, we will derive the power equation when the C-state limit is C3.

TABLE I  
C-STATE LIMIT AND HARDWARE-REPORTED INFORMATION

C-state limit	Core C-state				Processor C-state			
	$T_{CC_0}$	$T_{CC_1}$	$T_{CC_3}$	$T_{CC_6}$	$T_{PC_0}$	$T_{PC_1}$	$T_{PC_3}$	$T_{PC_6}$
C1	available but		n/a	n/a	available but		n/a	n/a
C2	not reported		OK	n/a	not reported		OK	n/a
C3			OK	OK			OK	OK

Power dissipation is dependent on the C-state limit as shown in Fig. 4. ; for the higher C-state limit, the power dissipation is lower. Note that utilization and system power reported in this figure are all measurements; In particular, power is measured using the power analyzer tool whereas the utilization is reported by *xentop*. The power difference among different C-state limits is greater when the utilization is lower. This is because cores stay in the C0 state most of the time when utilization is high.

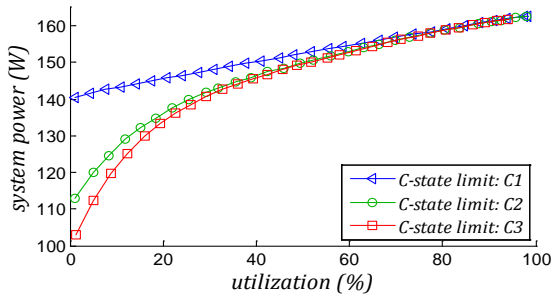


Fig. 4. Power dissipation vs. utilization for C-state limits

We do not provide details about how we derive the power dissipation equations for the three C-state limits since the derivations are involved and do not fit in the page limit that we have. Full derivations can be found in the Appendix of a USC CENG technical report [24]. The key idea behind the derivation is to start with equations (2) and (3), and then use a combination of analytical expansion of terms, lookups from hardware-reported information (TABLE I), and regression analysis to derive the appropriate power macro-models as shown in TABLE II. Note that time spent in power states of a core is almost identical to one another because a CPU scheduler evenly

distributes tasks. Hence, these times in TABLE II are core-independent terms.

TABLE II  
POWER MACRO-MODELS FOR THE SERVER SYSTEM UNDER TEST (COMPRISING TWO INTEL® XEON® E5620 PROCESSOR PACKAGES) [24]

C-state limit	Power equation
C1	$p_{est.}^{total} = 21.88T_{active} + 141.12$
C2	$p_{est.}^{total} = 22.48T_{active} - 5.76T_{CC_3} - 31.16T_{PC_3} + 140.7$
C3	$p_{est.}^{total} = 22.48T_{active} - 5.76T_{CC_3} - 8.56T_{CC_6} - 31.16T_{PC_3} - 42.55T_{PC_6} + 140.7$

The power models presented in the above table are highly accurate; Fig. 4. shows a comparison between measurements and model predictions for the case that C-state limit is set to C3; estimation is very close to measurements.

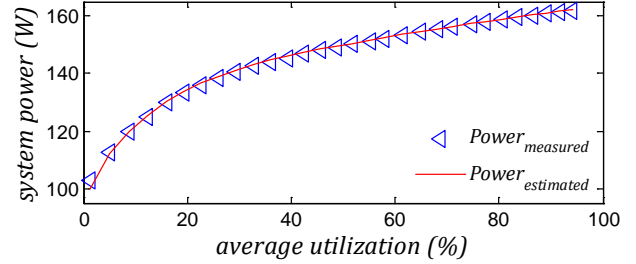


Fig. 5. Power estimation vs. measurements when the C-state limit is C3

The first coefficient (for active state) in TABLE II is application dependent. The main point here is not finding very accurate parameters of the power model but showing that power dissipation can be well estimated using CC/PC-state stats. Therefore, we will see how those states are changed by consolidation in order to understand how consolidation improves energy efficiency.

### B. Package-level consolidation

As shown in equation (3), uncore power is a function of PC-states. If we have more than one package in a system, further power savings may be achieved by package-level consolidation: select CPUs from the minimum number of packages and put other packages in the deepest power state.

Package consolidation can reduce the total time spent in the active state ( $PC_0$ ), which is obtained by summing over all packages the time that each package spends in its  $PC_0$  state; thus, the uncore power dissipation decreases. In particular package consolidation utilizes as few packages in a server as possible, so the amount of time when multiple CPUs in the same package are in the  $CC_0$  state at the same time increases; i.e., the  $CC_0$  state overlap time increases and therefore, the total time spent in the  $PC_0$  state (which is equal to  $\sum_i T_{PC_0}^{core_i}$ ) decreases. When we consider an extreme case, this point becomes more obvious. Let us say one CPU is chosen from each package to remain active. Then the total time spent in  $PC_0$  state will be greater than or equal to times spent in the  $CC_0$  state of each CPU because there is no possibility for  $CC_0$  state overlap. In comparison if the two CPUs are chosen from the same package, then only one of the packages will be active and even then the time spent in  $PC_0$  for that package is less than or equal to times spent in the  $CC_0$  state of each CPU because there



is  $CC_0$  state overlap.

The above discussion is based on a key assumption, i.e., the PC-state of a package is independent of that of other packages. In practice, this assumption is far from the truth. Fig. 6. depicts  $PC_0$  state of two packages, which reveals the opposite. The data presented in the figure is for a case when all active CPUs are chosen from exactly one package, and therefore, all CPUs in the other package are idle. The figure shows that  $PC_0$  states of both packages are nearly identical. Indeed, the same behavior is observed for other PC-states ( $PC_3$  and  $PC_6$ ), although not shown here. This implies that all packages should stay in the active state ( $PC_0$ ) when any CPU, which may in fact reside in another package, is active. This is because an active CPU may need data from a remote DRAM, so not only the package where the active CPU is located but also all the other packages should remain in active states to provide the requested data (details of Intel® QPI architecture has already been discussed in Section II.D) in order to avoid a significant additional latency.

As discussed above, PC-states of all packages are nearly the same due to Intel® QPI, hence little or no energy savings are expected from package-level consolidation. On the other hand, if a task accesses memory infrequently (such as in the case of CPU bound tasks), package consolidation may save further energy. In other words, package-level consolidation may or may not improve energy efficiency depending on the characteristics of tasks. We discuss later whether or not package-level consolidation results in any energy savings.

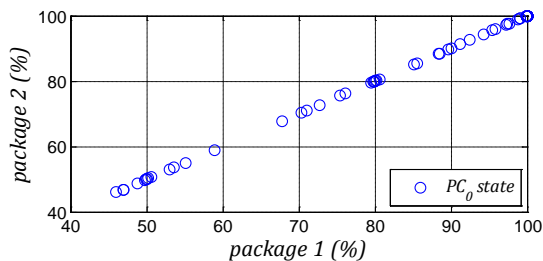


Fig. 6. Relationship between  $PC_0$  states of two packages of the target server when there are 4 active CPUs in exactly one of the packages

There are 8 CPUs in the system, so only when all these CPUs are inactive, both packages can be switched to inactive states ( $PC_3$  or  $PC_6$ ). Hence, we expect that the two packages are active most of the time even when the total utilization is low. As depicted in Fig. 7. ,  $PC_0$  state is 100% when total utilization is greater than 150% out of 800%. This implies that there is a very small room for uncore power reduction.

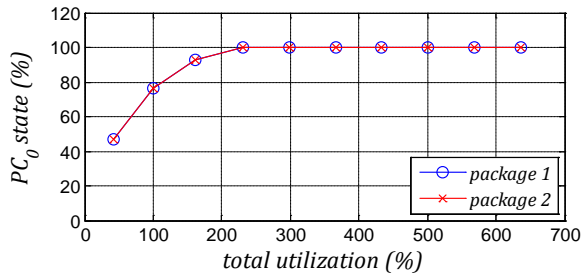


Fig. 7. Percentage of the time that each package in the target server is in the  $PC_0$  state as a function of the total utilization

### C. Consolidation overhead – vCPU count

The number of virtual CPUs, called vCPU count, is an important parameter of a *virtual machine* (VM) because this count limits the performance of the VM. For example, a VM with two vCPUs is capable of utilizing up to two CPUs at a time, so the maximum total CPU utilization of the VM is 200%. However, managing each vCPU causes additional overheads; thus, it could hurt both performance and energy efficiency if VMs have unnecessarily too many vCPUs. We use the ratio of vCPU to active CPU counts (called *virtualization ratio*) as an indicator of this overhead.

Experimental results of PARSEC benchmark programs with different virtualization ratios are reported in Fig. 8. and Fig. 9. The active CPU count is 4; that is, the total CPU utilization is always equal to or less than 400%. The same experiments are repeated for four different vCPU counts: 8, 16, 24, and 32 (corresponding to virtualization ratios of 2, 4, 6, and 8, respectively). Except one program, i.e., *vips*, execution time remains the same when the ratio is 6 or less while E/task of many programs increases noticeably even when the ratio is 4 (Fig. 9. ). This is due to higher overheads of vCPUs management. We suggest keeping the virtualization ratio to be less than or equal to 3.

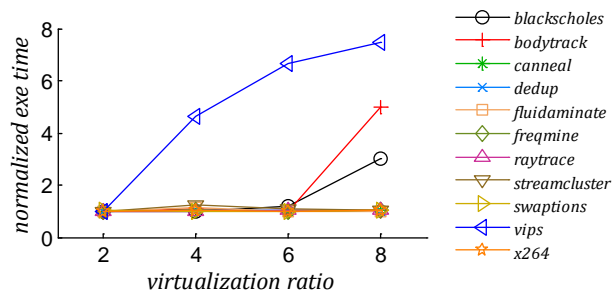


Fig. 8. Consolidation overhead i.e., execution time as a function of the virtualization ratio

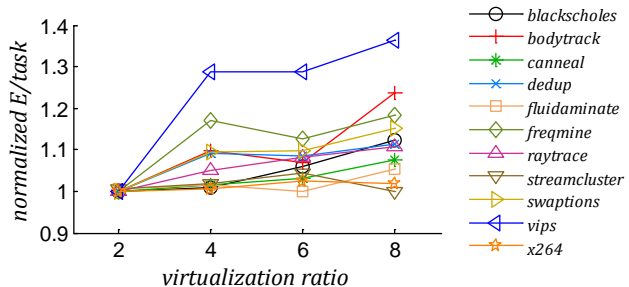


Fig. 9. Consolidation overhead i.e., energy per task as a function of the virtualization ratio

### D. CPU selection policy

The basic idea of the consolidation is to have as fewer active CPUs at any time. In addition to the active CPU count, the CPU selection policy can be important for multi core/processor systems; e.g., choosing CPUs from a minimum number of packages or selecting CPUs uniformly from all packages. The system under test has two packages, so there are two possible selection policies: i) Select all CPUs from one package first and take additional CPUs from the other package if necessary.

ii) Select equal number of CPUs (modulo plus/minus one) from each package.

$CC_0$  and  $PC_0$  states of the *bodytrack* program are shown in Fig. 10. Each plot compares two selection policies: all four CPUs are selected from one package (4CPU-1P) and two CPUs are chosen from each package (4CPU-2P). In our experimental results, core and package may reside in either active ( $CC_0$  and  $PC_0$ ) or deepest sleep ( $CC_6$  and  $PC_6$ ) states. Hence, we only present statistics of active states. The total time spent in each state (called *state residency*) is calculated as the sum of all times spent in the corresponding state by all active cores (for  $CC_0$ ) or packages (for  $PC_0$ ), so these times can be greater than 100%. Normalized workloads are calculated as ratios of actual workloads over workloads that result in 50% total core utilization. As shown in Fig. 10.  $CC_0$  states of the policies close to each other, which is reasonable and expected. On the other hand, the total time spent in the  $PC_0$  state for the first policy (4CPU-1P) is smaller than that for the other (4CPU-2P). That is, time spent in the  $PC_6$  state under 4CPU-1P is greater; therefore, the uncore power dissipation of 4CPU-1P is smaller. According to the discussion in Section VI.B, *bodytrack* is considered to be a non-memory-intensive task.

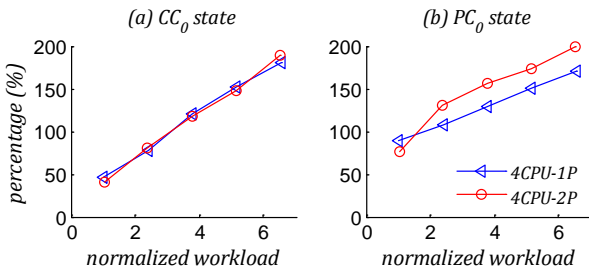


Fig. 10.  $CC_0$  and  $PC_0$  state residencies for the *bodytrack* program

Result of another program, *cannal*, is shown in Fig. 11. Here there is a significant difference in the  $CC_0$  state residency between the two policies. *cannal* is a program to find a chip design with minimum routing cost. It uses cache-aware simulated annealing which creates intensive memory read/write activity. If we use all four CPUs in the same package, then we only use half of the L3 cache compared with the other case (4CPU-2P). This causes much higher cache misses, so both the time spent in the  $CC_0$  state and the application execution time increase. There is negligible difference in the  $PC_0$  state residency. From this result, package-level consolidation is not a good idea for applications requiring extensive data transfers to and from the main memory.

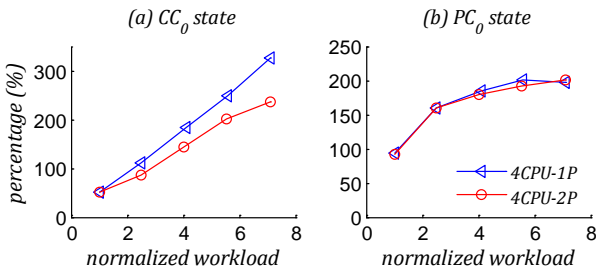


Fig. 11.  $CC_0$  and  $PC_0$  state residencies for the *cannal* program

Normalized E/task comparisons for all PARSEC programs are reported in Fig. 12. Due to run-to-run variations, the average of 15 measurements is presented. E/task difference is less than 3% for most of programs except *bodytrack*, *cannal*, and *x264*. Most significant difference ( $\sim 6\%$ ) is observed for *cannal*. Later in this paper we will present a more sophisticated CPU selection policy to minimize the E/task.

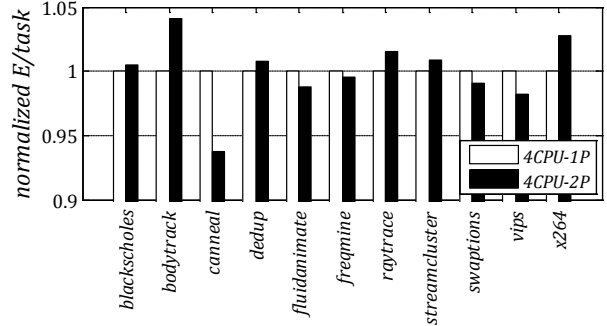


Fig. 12. Effect of simple CPU selection policies on energy consumption per task of various PARSEC programs

### E. Execution time

According to (10), the delay (execution time) of a task increases as the average utilization per core increases. The marginal rate of increase at high utilization is greater than that at low utilization. Hence, if we keep the average utilization lower than a certain threshold, then delay increase by consolidation can be made small. The normalized execution times of PARSEC benchmark programs at various average utilizations are shown in Fig. 13.

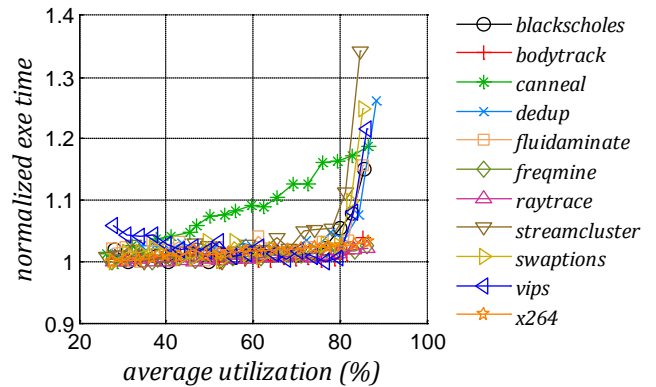


Fig. 13. Execution time of PARSEC benchmark programs as a function of the average utilization per core

Except one program, i.e., *cannal*, the execution time increase is less than 5% at average utilizations as high as 70%. In other words, if we keep the average utilization below 70%, the maximum execution time increase will be less than 5% for most applications. We use this threshold to decide about the degree of consolidation that we do. For example, let's say that the average CPU utilization is 40% with 8 active CPUs. If we consolidate the workloads to four CPUs, then the new average utilization will be approximately 80%, which is greater than our threshold (70%). This implies that there should be at least 5 active CPUs in order to avoid a considerable increase in the

average execution time of tasks. As discussed in the previous section, *canneal* generates extensive memory read/write requests, so the cache miss rate is expected to rise rapidly as the CPU utilization increases. In other words, the execution time of *canneal* increases monotonically with CPU consolidation. More generally speaking, when we have an execution time target (upper bound), aggressive CPU consolidation can result in significant SLA violation for memory-intensive applications.

#### F. E/task and ED/task improvements for PARSEC

In this section we present E/task and ED/task for PARSEC benchmark programs. The average utilization is kept to be less than 70% and the virtualization ratio not to be greater than 3 as discussed in previous sections.

The first interesting metric is E/task, which is reported in Fig. 14. The white, gray, and black bars in the plot show improvements in E/task achieved by DVFS, consolidation, and both DVFS and consolidation, respectively. For all programs, improvement by DVFS is always greater than that by consolidation. Recall that the processor under test supports 7 frequencies from 1.6GHz to 2.4GHz. DVFS can thus effectively reduce E/task by slowing down the clock frequency from 2.4GHz all the way down to 1.6GHz (and accordingly lowering the supply voltage level). Evidently, this action increases the average execution time of tasks; however, this execution time increase does not affect the E/task metric much (this is because energy consumption of the server is dominated by dynamic power and not leakage power). The maximum E/task improvement achieved by consolidation is about 10%. Another observation is that the effects are somewhat additive that is, when we apply both DVFS and consolidation (see the ‘Combined’ results in the figure), the improvement is greater than the other two cases for most programs with the exception of *canneal*. The maximum improvement of the ‘Combined’ technique is greater than 15% (achieved for *dedup*).

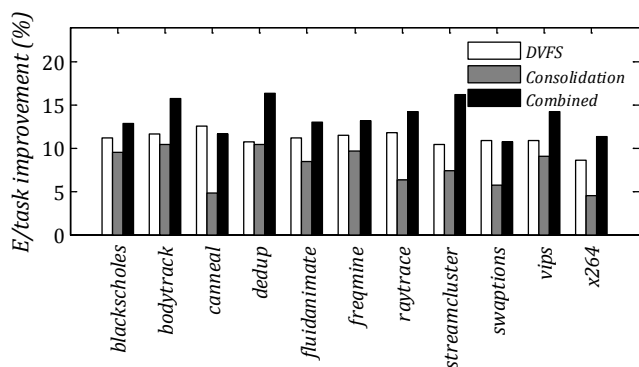


Fig. 14. Energy per task improvement

Surprisingly, we observe very different results for the ED/task metric, as seen in Fig. 15. The ED/task is worsened by DVFS because the task execution time increases significantly as a result of reducing the CPU clock frequency. On the other hand, consolidation maintains its relative energy savings except for the case of *canneal*. This is because the execution time of *canneal* increases monotonically even when the average CPU utilization is kept below 70%. Therefore, the ED/task

improvement of consolidation for *canneal* is much smaller than all other programs. From this result, we can conclude that consolidation is a much more effective solution for delay sensitive applications compared to DVFS (although it loses much of its advantage in memory-bound applications).

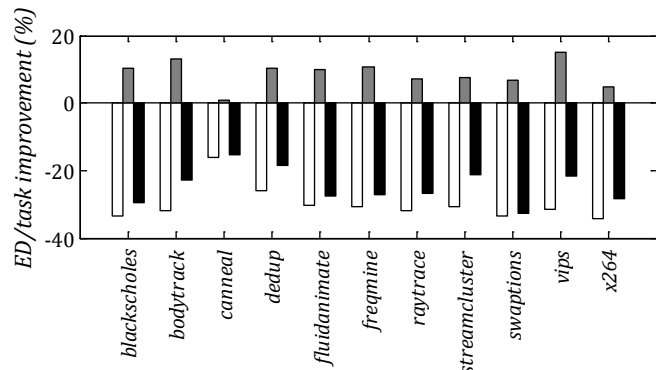


Fig. 15. Energy delay product per task improvement

#### G. CPU consolidation for SPECWeb2009 benchmarks

In the previous section, the relative effectiveness of the CPU consolidation and DVFS was studied for the PARSEC benchmark suite. In this section, results for the SPECWeb2009 are presented. This benchmark suite comprises of I/O bound application programs whose characteristics are very different from those of the PARSEC programs. SPECWeb2009 is a very well developed benchmark suite, and its main purpose is to evaluate a web server (I/O-bound application); hence, we can see how consolidation affects the delay and energy efficiency of I/O-bound applications from SPECWeb2009 results.

The energy efficiency is quantified as ED/packet because delay (i.e., response time) is a critical performance metric in these applications. SPECWeb2009 requires a *simultaneous user session* (SUS) count as an input, which is another way of specifying the workload intensity. The SUS count specifies only the average workload intensity (the instantaneous workload intensity fluctuates a lot). Hence, an online method, which dynamically finds optimal settings for consolidation, is needed. In this section, we start from analyzing characteristics of the SPECWeb2009. After that, four online consolidation algorithms are presented, and results of those algorithms are reported and analyzed.

Web applications are in general not compute-intensive [25]; hence, the average response time is less dependent on CPU clock frequencies as shown in Fig. 16. (a). This is because the response time of web servers is closely related to the I/O processes, such as network and disk access. Likewise, the response time is almost independent of the active CPU count when a sufficiently large number of CPUs is active. The relationship between the power dissipation and clock frequency/active CPU count is shown in Fig. 16. (b). The power dissipation declines as the frequency decreases and/or the active CPU count is reduced. This result implies that both DVFS and the CPU consolidation improve the energy efficiency without any significant performance degradation. In addition, we expect higher power efficiency gains when both

techniques are applied at the same time.

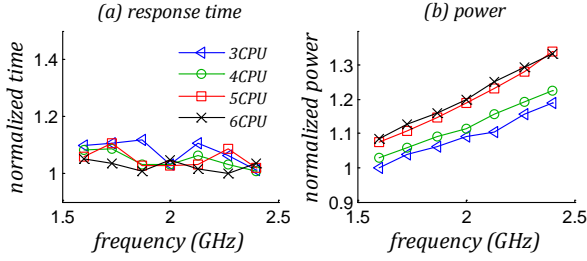


Fig. 16. Response time and power dissipation

When the OS changes the CPU clock frequency, the CPU utilization also changes under the same workload. Therefore, before changing the CPU clock frequency, the corresponding CPU utilization must be estimated in order to prevent the undesirable situation whereby active CPUs are overloaded because the chosen frequency is too low for the given workload. The relationship between the total CPU utilization and CPU clock frequency is depicted in Fig. 17. Note that utilization is the percentage of time that a CPU spends executing user and system space codes. When a task is waiting for an I/O operation to be completed, the task is suspended and CPU does nothing. Hence, this suspension time is not included in the utilization.

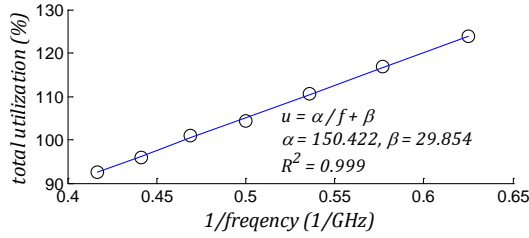


Fig. 17. Frequency vs. total utilization (SPECWeb)

According to the  $R^2$  value, a linear equation is a nearly perfect fit the data points in Fig. 17. The relationship is then as follows:

$$(u - \beta)f = \alpha \quad (14)$$

where  $\alpha = 150.4$ ,  $\beta = 29.9$  and  $0 \leq u \leq 800$  (i.e., there are eight CPUs). Since coefficient  $\beta$  is relatively small, it can be ignored to simplify the relationship. Hence, the equation may be written as follows:

$$f_i u_i = f_j u_j = \alpha \quad (15)$$

#### H. Online CPU consolidation algorithms

As shown in the previous section, both the clock frequency and the active CPU count affect the E/task and ED/task. In this section, we present online algorithms, which perform voltage/frequency setting and consolidation simultaneously. These algorithms monitor the CPU utilization, and change the frequency setting and/or the active CPU count depending on the current workloads. The main idea of these algorithms is to utilize as few CPUs at low frequencies as possible (while meeting the performance constraints); the decision is made by considering the current CPU utilization levels. This approach is reasonable for I/O bound applications because performance degradation is not significant unless the CPU is very highly

utilized [10]. To avoid energy and delay overheads associated with frequent state changes, the proposed algorithms change the system configuration conservatively, that is, if the system is overloaded, these algorithms will immediately increase the frequency and/or the number of active CPUs. If, however, the system is underutilized, they will apply a state change (reduce frequency and/or turn off some CPUs) only if the situation persists for at least some time. We achieve this goal by introducing two different thresholds with hysteresis as described below.

We present four algorithms whose main ideas are quite similar to each other: if the average utilization ( $u_i$ ) of a CPU is greater than an upper threshold ( $u_{high}$ ), these algorithms deploy more computing resources by increasing the clock frequency of the active CPUs and/or by adding to the number of active CPUs. On the other hand, if the average utilization is less than a lower threshold ( $u_{low}$ ), then they will release some computing resources by decreasing the CPU frequency and/or reducing the number of active CPUs. It is necessary to estimate the new utilization level under the new frequency and active CPU count to avoid any performance degradation. Equation (15) does not account for the number of active CPUs ( $c_i$ ) in the system, and hence, it is modified to apply to this new situation:

$$c_i f_i u_i = c_j f_j u_j \quad (16)$$

Because we can change both the CPU frequency and the active CPU count (when needed), we must decide which strategy must be given higher priority: i) Changing the clock frequency first and the CPU count next, ii) Changing the CPU count first and the clock frequency next. Two pseudo codes are presented in Fig. 18. The first function  $min\_cpu()$  finds the minimum CPU count ( $x_c$ ) without any performance degradation. After finding the minimum CPU count, it determines the slowest frequency ( $x_f$ ) with the new CPU count that would still avoid any performance degradation. This function tries to achieve a new CPU utilization close to  $u_{mid}$ , which is the median of high/low thresholds and is calculated as follows:

$$u_{mid} = \frac{u_{high} + u_{low}}{2}, \quad u_{high} = 85\%, \text{ and } u_{low} = 65\% \quad (17)$$

The second function  $min\_freq()$  finds the slowest frequency first, and then finds the minimum CPU count with the new frequency. Again no performance penalty is allowed. The two functions are called when the system is under-utilized (i.e., the current utilization is smaller than  $u_{low}$ ) or over-utilized (i.e., the current utilization is greater than  $u_{high}$ ). For each case, we can choose which function is called, i.e.,  $min\_cpu()$  or  $min\_freq()$ . Therefore, there are a total of four online algorithms, which are shown in Fig. 18.

$$\left. \begin{array}{l} \text{Function } min\_cpu(u_i, f_i, c_i) \{ \\ \quad x_c = \left\lceil \frac{u_i f_i}{u_{mid} f_{max}} c_i \right\rceil; \\ \quad x_f = \left\lceil \frac{u_i c_i}{u_{mid} x_c} f_i \right\rceil; \\ \quad \text{return } (x_c, x_f); \\ \} \end{array} \right\} \left. \begin{array}{l} \text{Function } min\_freq(u_i, f_i, c_i) \{ \\ \quad x_f = \left\lceil \frac{u_i c_i}{u_{mid} c_{max}} f_i \right\rceil; \\ \quad x_c = \left\lceil \frac{u_i f_i}{u_{mid} x_f} c_i \right\rceil; \\ \quad \text{return } (x_c, x_f); \\ \} \end{array} \right\}$$

Fig. 18. Pseudo codes for  $min\_cpu()$  and  $min\_freq()$

The first algorithm (Type1) calls  $min\_cpu()$  function for both the under and over-utilized CPU cases. The Type2 algorithm calls  $min\_cpu()$  when a CPU is over-utilized and  $min\_freq()$  if it is under-utilized. The Type3 algorithm calls  $min\_freq()$  when a CPU is over-utilized and  $min\_cpu()$  if it is under-utilized. The last algorithm (Type4) calls  $min\_freq()$  for both over and under-utilized CPU cases.

We do experiments for three different *SUS* counts and compare the ED/packet and the *quality of service* (QoS) for the aforesaid four consolidation algorithms and two more algorithms (read below). The QoS refers to the percentage of packets whose response time (latency) is less than the pre-defined threshold. This QoS is reported by SPECWeb2009 benchmark suite. In addition to the four proposed algorithms, we provide results for two other algorithms: *base* and *ondemand*. The *base* algorithm means there is no dynamic adjustment of the active CPU count and clock frequency, i.e., all CPUs are active and running at the maximum allowed clock frequency. The *ondemand* algorithm is the default DVFS method used in Linux™, which does not change the active CPU count but changes the CPU frequency.

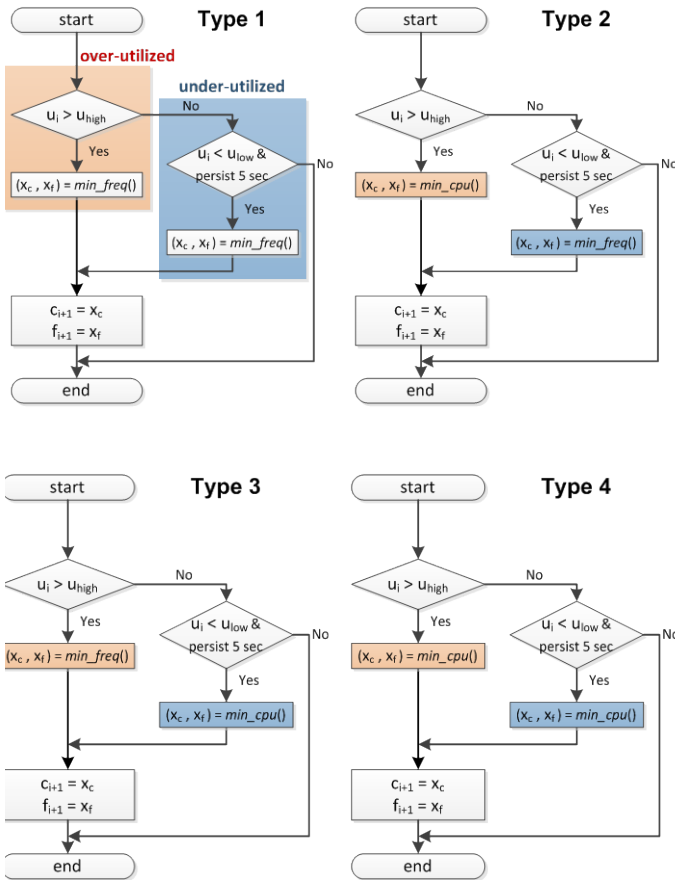


Fig. 19. Four online consolidation algorithms

Experimental results are reported in Fig. 20. Regardless of the *SUS* count, the proposed algorithms always result in smaller ED/packet compared to the *base* and *ondemand* algorithms. Among the four proposed algorithms, Type1 algorithm is the best one in terms of the ED/packet. As the *SUS*

count increases, QoS of all algorithms decreases, but QoS remains greater than 95%; hence, there are no appreciable performance degradation concerns. Note that the magnitude of ED/packet metric also decreases as the *SUS* count increases, which implies that the system consumes less energy for executing a packet. This is because of the energy non-proportionality of the existing server systems (including the one used in this study). From these results, we can state that the Type1 consolidation algorithm is the best. This implies that, at least for the system under experiment, adjusting the CPU frequency has higher impact on the ED/packet metric than changing the CPU count. We compare ED/packet of the *ondemand* and Type1 algorithm in TABLE III. For three *SUS* settings, ED/packet of Type1 algorithm is always smaller than that of *ondemand*. In addition, the difference between the two algorithms increases for larger number of user sessions.

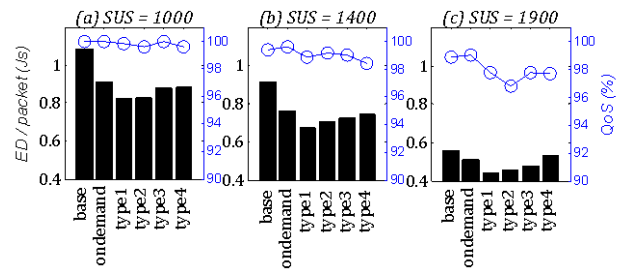


Fig. 20. ED/packet and QoS comparisons

TABLE III  
COEFFICIENTS OF THE DELAY MODEL

<i>SUS</i>	<i>ED/packet</i> (Js)		$\Delta ED/packet(\%)$
	<i>ondemand</i>	<i>Type1</i>	
1000	0.91	0.82	9.44
1400	0.76	0.67	11.83
1900	0.51	0.44	13.65

## VII. CONCLUSION

DVFS has been a promising method for reducing the energy consumption, but the energy saving leverage of DVFS reduces as the supply voltage level decreases with CMOS scaling. In this paper, CPU consolidation is considered as a substitute, or better stated, as a complement. The idea looks simple; however, we need to investigate CPU consolidation under realistic setup to maximize the energy efficiency. The effectiveness of CPU consolidation was thus investigated for different configurations: types of applications, the virtual CPU count, the active CPU count, and the active CPU set. From the investigation we learn a few useful lessons. First, unnecessarily large number of virtual CPUs causes significant performance degradation; hence, the virtual CPU count must be dynamically adjusted. Second, we need to choose different CPU selection policy depending on applications. Third, DVFS outperforms consolidation in terms of E/task improvement. On the other hand, DVFS do not improve ED/task of PARSEC while consolidation does. Forth, the maximum improvement of ED/task for SPECWeb2009 is also achieved when both DVFS and the consolidation are applied. Similarly, biggest E/task improvement of PARSEC is achieved when both techniques are used.

## ACKNOWLEDGMENT

The authors would like to thank Dr. Timothy Kam of Intel<sup>®</sup> Corp. who contributed to the conference version of the paper and who provided helpful feedback and advice regarding the experimental setup, results analysis, and discussions.

## REFERENCES

- [1] L. A. Barroso, and U. Holzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, pp. 33-37, 2007.
- [2] O. Bilgir, M. Martonosi, and Q. Wu, "Exploring the potential of CMP core count management on data center energy savings."
- [3] G. Dhiman, G. Marchetti, and T. Rosing, "vGreen: a system for energy efficient computing in virtualized environments," in Proceedings of the 14th ACM/IEEE international symposium on Low power electronics and design, San Francisco, CA, USA, 2009, pp. 243-248.
- [4] R. Nathuji, and K. Schwan, "VirtualPower: coordinated power management in virtualized enterprise systems," in Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles, Stevenson, Washington, USA, 2007, pp. 265-278.
- [5] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic Placement of Virtual Machines for Managing SLA Violations." pp. 119-128.
- [6] H. N. Van, F. D. Tran, and J.-M. Menaud, "Autonomic virtual resource management for service hosting platforms," in Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing, 2009, pp. 1-8.
- [7] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation - Volume 2, 2005, pp. 273-286.
- [8] G. von Laszewski, W. Lizhe, A. J. Younge, and H. Xi, "Power-aware scheduling of virtual machines in DVFS-enabled clusters." pp. 1-10.
- [9] P. Pillai, and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in Proceedings of the eighteenth ACM symposium on Operating systems principles, Banff, Alberta, Canada, 2001, pp. 89-102.
- [10] M. Pedram, and H. Inkwon, "Power and Performance Modeling in a Virtualized Server System." pp. 520-526.
- [11] J. Leverich, M. Monchiero, V. Talwar, P. Ranganathan, and C. Kozyrakis, "Power Management of Datacenter Workloads Using Per-Core Power Gating," *Computer Architecture Letters*, vol. 8, no. 2, pp. 48-51, 2009.
- [12] I. Hwang, T. Kam, and M. Pedram, "A study of the effectiveness of CPU consolidation in a virtualized multi-core server system," in Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design, Redondo Beach, California, USA, 2012, pp. 339-344.
- [13] H. Goudarzi, and M. Pedram, "Multi-dimensional SLA-Based Resource Allocation for Multi-tier Cloud Computing Systems," in Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing, 2011, pp. 324-331.
- [14] H. Goudarzi, M. Ghasemazar, and M. Pedram, "SLA-based Optimization of Power and Migration Cost in Cloud Computing." pp. 172-179.
- [15] M. Pedram, "Energy-Efficient Datacenters," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 31, no. 10, pp. 1465-1484, 2012.
- [16] L. A. Barroso, J. Dean, and U. Holzle, "Web Search for a Planet: The Google Cluster Architecture," *IEEE Micro*, vol. 23, no. 2, pp. 22-28, 2003.
- [17] S. Shankland. "Google uncloaks once-secret server." <http://www.cnet.com/news/google-uncloaks-once-secret-server-1020958/>.
- [18] "Open computer project: Server/Specs and designs," <http://www.opencompute.org/wiki/Motherboard/SpecsAndDesigns>.
- [19] A. Andreyev. "Introducing data center fabric, the next-generation Facebook data center network," <https://code.facebook.com/posts/360346274145943/introducing-data-center-fabric-the-next-generation-facebook-data-center-network/>.
- [20] L. A. Barroso, J. Clidaras, and U. Holzle, "The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Second edition," *Synthesis Lectures on Computer Architecture*, vol. 8, no. 3, pp. 1-154, 2013/07/31, 2013.

- [21] C. Bienia, "Benchmarking modern multiprocessors," Princeton University, 2011.
- [22] P. Sangyoung, P. JaeHyun, S. Donghwa, W. Yanzi, X. Qing, M. Pedram, and C. Naehyuck, "Accurate Modeling of the Delay and Energy Overhead of Dynamic Voltage and Frequency Scaling in Modern Microprocessors," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 32, no. 5, pp. 695-708, 2013.
- [23] "Xen power management," [http://wiki.xen.org/wiki/Xen\\_power\\_management](http://wiki.xen.org/wiki/Xen_power_management).
- [24] I. Hwang, and M. Pedram, "A Comparative Study of the Effectiveness of CPU Consolidation versus Dynamic Voltage and Frequency Scaling in a Virtualized Multi-Core Server."
- [25] D. Meisner, C. M. Sadler, L. A. Barroso, W. Weber, and T. F. Wenisch, "Power management of online data-intensive services." pp. 319-330.



**Inkwon Hwang (S'10)** received the B.S. degree in electrical engineering from Seoul National University, Seoul, Korea, in 2006 and M.S. degree in electrical engineering from the University of Southern California, Los Angeles, CA, USA, in 2008. He is currently pursuing the Ph.D. degree in electrical engineering at the University of Southern California, Los Angeles, CA, USA. He has been working on power management of cloud computing systems and CPU consolidation.



**Massoud Pedram (F'01)** received the Ph.D. degree in electrical engineering and computer sciences from the University of California, Berkeley, CA, USA, in 1991. He is the Stephen and Etta Varra Professor with the Ming Hsieh Department of Electrical Engineering, University of Southern California, Los Angeles, CA, USA. He holds 10 U.S. patents and has authored four books, 13 book chapters, and more than 140 archival and 380 conference papers. His research interests include low power electronics, energy-efficient processing, and cloud computing to photovoltaic cell power generation, energy storage, and power conversion, and from RT-level optimization of VLSI circuits to synthesis and physical design of quantum circuits. Dr. Pedram is an ACM Distinguished Scientist, and currently serves as the Editor-in-Chief of the ACM Transactions on Design Automation of Electronic Systems and the IEEE JOURNAL ON EMERGING AND SELECTED TOPICS IN CIRCUITS AND SYSTEMS. He has also served on the Technical Program Committee of a number of premiere conferences in his field and was the founding Technical Program Co-Chair of the 1996 International Symposium on Low Power Electronics and Design and the Technical Program Chair of the 2002 International Symposium on Physical Design. He was a recipient of the 1996 Presidential Early Career Award for Scientists and Engineers. He and his students have received seven conference and two IEEE TRANSACTIONS Best Paper Awards.