Coordinating Processor and Main Memory for Efficient Server Power Control

Ming Chen, Xiaorui Wang, and Xue Li
Department of Electrical Engineering and Computer Science
University of Tennessee, Knoxville, TN 37996
{mchen11, xwang, xli44}@utk.edu

ABSTRACT

With the number of high-density servers in data centers rapidly increasing, power control with performance optimization has become a key challenge to gain a high return on investment, by safely accommodating the maximized number of servers allowed by the limited power supply and cooling facilities in a data center. Various power control solutions have been recently proposed for high-density servers and different components in a server to avoid system failures due to power overload or overheating. Existing solutions, unfortunately, either rely only on the processor for server power control, with the assumption that it is the only major power consumer, or limit power only for a single component, such as main memory. As a result, the synergy between the processor and main memory is impaired by uncoordinated power adaptations, resulting in degraded overall system performance. In this paper, we propose a novel power control solution that can precisely limit the peak power consumption of a server below a desired budget. Our solution adapts the power states of both the processor and memory in a coordinated manner, based on their power demands, to achieve optimized system performance. Our solution also features a control algorithm that is designed rigorously based on advanced feedback control theory for guaranteed control accuracy and system stability. Compared with two state-of-the-art server power control solutions, experimental results show that our solution achieves up to 23% better average performance than one baseline for CPU-intensive benchmarks and doubles the performance of the other baseline when the power budget is tight.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Design studies; C.5.5 [**Computer System Implementation**]: Servers

General Terms

Design, Management, Performance

Keywords

Power control, server, power capping, memory, data center

ICS'11, May 31–June 4, 2011, Tuscon, Arizona, USA.

1. INTRODUCTION

Power management has become one of the first-order considerations in modern enterprise data centers in recent years. The constant quest for high performance leads to high peak power consumption when the system is fully utilized. The power problem is further exacerbated by the wide adoption of high-density servers and the increasing demand for high-capacity, high-bandwidth main memory subsystems. This is especially true for supercomputers, which are usually equipped with very large memory size. As a result, the power supply and cooling facilities become expensive and bulky, which may hinder the deployment of new servers in the data centers in turn. More importantly, high peak power may cause system failures due to power capacity overload and thermal violations. In response, power control techniques have been proposed at different levels in data centers [25][31]. For example, a data center may allocate its power budget to different Power Distribution Units (PDU) and then, a group of racks. Further, the power budget of a single server may be allocated by the rack in which the server resides. Likewise, the server-level budget may be further divided among multiple components in a server. Hence, it is important that power is controlled at all the levels.

Power control at the server level faces several major challenges. *First*, multiple components need to be manipulated simultaneously to control the power consumption of a server. The widespread adoption of multi-core processors and the rapid increase of applications' memory footprints have dramatically increased the demand on memory bandwidth and capacity. As a result, it is no longer valid to assume that the processor is the only major power consumer in a server. For example, Lin et al. [19] reported that the main memory systems in a multi-core server box, which has $32GB$ of Fully-Buffered DIMMs, might have power consumption in the same range as the processor. Hence, we need Multi-Input-Multi-Output (MIMO) strategies to coordinate both processor and memory for server power control. *Second*, the components in a server are usually heterogeneous. Thus, we cannot simply allocate power proportional to the activities of each individual component since the power consumption of a single activity varies for different components. For example, an instruction dispatched in a processor may contribute a different amount of power from a memory request. Therefore, we need to optimize power allocation based on performance indicators that can characterize the real power demands of the components. *Third*, workloads in different components in a server are usually synergetic. For example, processor frequency downscaling may decrease the number of memory requests so that the memory power consumption decreases accordingly. Therefore, the synergy among components should be carefully addressed during power allocation at the server level. *Fourth and most importantly*, the workloads of different components are

unpredictable at design time and may vary significantly at runtime. As a result, power control algorithms cannot rely on static power models or open-loop estimations. They must be self-adaptive to workload variations for improved server performance.

In recent years, various power control solutions have been proposed for high-density servers. A recent solution [15] relies only on the processor for server power control, with the assumption that it is the only major power consumer in the server. As a result, that solution is limited to those small form-factor servers with small-size memory systems. Another recent paper [5] proposes to shift power between the processor and main memory proportionally to the number of activities. However, their solution relies on power estimation based on measured activities and off-line profiled power models, which may result in either power violations or performance degradation when the workload varies. In addition, their power actuation method relies only on throttling the number of activities and does not exploit low-power states in the processor and main memory to minimize idle power. As a result, it has unnecessarily high idle power and a limited capacity of power adaptation.

In this paper, we propose a novel server power control solution that can precisely control the server power consumption to a desired budget. We periodically coordinate the processor and main memory to achieve improved performance, based on the memory queue level, by dynamically adjusting the voltage/frequency of the processor and placing memory ranks into different power states. Our coordinated solution is systematically designed based on Model Predicative Control (MPC) theory, which is an advanced optimal MIMO control theory. Compared with two state-of-the-art server power control solutions, our solution has the following improvements. First, our solution, on average, achieves up to 23% better performance than one baseline for CPU-intensive benchmarks and doubles the performance of the other baseline when the power budget is tight. Second, our solution significantly improves the power adaptation capacity, which is the range between the maximum and minimum power consumption that the server can have, due to coordinating the power states of both processor and main memory. As a result, our solution can still manage to conduct power control even when the power budget becomes very tight (*e.g.*, due to thermal emergency without having to shutdown the server). Third, our solution has better power control accuracy so that it is more robust and less vulnerable to workload variations. Specifically, the contributions of this paper are three-fold:

- We mathematically model the power dynamics of the memory system by varying the number of memory ranks in low power states based on a recently proposed memory power actuation method.
- We design and analyze a MIMO power control algorithm to control the total power of a high-density server to a budget, while coordinating the processor and main memory for improved server performance.
- We propose a new technique that uses the memory queue level as the power demand indicator to optimally allocate power between the processor and main memory.

The reminder of this paper is organized as follows. Section 2 introduces the power adaptation methods and provides a high-level description of the coordinated power control architecture. Section 3 presents the system modeling, design, and analysis of our power controller. Section 4 introduces the simulation environment and discusses the impact of DDR3 technology on our solution. Section 5 presents the results of our experiments. Section 6 discusses related work and Section 7 concludes the paper.

Table 1: DRAM power states

Power state/Transition	Power/Delay
Active standby (ACT)	104.5 mW
Precharged (PRE)	76 mW
Active powerdown (APD)	19 mW
Precharged powerdown (PPD)	13.3 mW
Self refreshing (SR)	5 mW
PRE $\rightleftharpoons$ PPD	6 ns
APD $\rightleftharpoons$ ACT	6 ns
SR $\rightarrow$ PRE	$\geq$ 137.5 ns

2. SYSTEM DESIGN

In this section, we first introduce the power adaptation methods used in this paper. We next provide a high-level description of the coordinated power control architecture.

2.1 Power Adaptation for the Main Memory

To effectively enforce the power budget of a server, it is important to find efficient power actuators for the processor and main memory. In this paper, we adapt the processor power consumption by conducting the widely used *DVFS* technique. Our power adaptation method for the main memory leverages the fact that modern memory devices have multiple power states that retain stored data. Each power state consumes a different amount of power, whereas the transitions between states involve different performance overheads. As an example, Table 1 lists the power states, power consumption, and transition overheads of DDR2 SDRAM chips with 1Gb capacity [24]. Diniz et al. [3] have shown that dynamically adjusting the power states of different memory devices can efficiently limit memory power consumption. By making a compromise between power saving and transition overhead based on Table 1, we utilize two power states: the precharged (PRE) state and precharged powerdown (PPD) state. The PRE state is an idle mode in which all banks are precharged and awaiting row activation commands to service a request, while the PPD state is the lowest power mode, other than the self-refreshing mode, in which all banks are precharged [11]. Hereinafter, we refer to the PRE state as *active state* and the PPD state as *sleep state*, for simplicity. We use the number of ranks in the active state as the actuator to dynamically change the memory power consumption. We place each memory rank either in the active state or in the sleep state. A rank in the sleep state needs to be activated first to service any arriving memory requests, which involves some overhead. We define the *active ratio* as the number of memory ranks in the active state normalized to the total number of ranks in the memory system. For example, if we keep 4 out of 16 ranks in the active state, the active ratio is 0.25.

The active ratio is kept constant within each control period while it is re-calculated at the end of each control period. The memory scheduler, shown in Figure 1, maintains a Most Recently Used (MRU) queue which records the most recently accessed rank at the head of the queue, and a status array which records the power state of each rank (*i.e.*, active state or sleep state). Its working process is based, generally, on an approach proposed by a recent paper [3]. We briefly introduce it as follows.

- At the end of each control period, based on the difference between the ratios in the current and previous control periods, some sleeping ranks at the head of the MRU queue are transitioned to the active state, or some active ranks from the tail of the MRU queue are transitioned to the sleep state.
- Within a control period, when a memory request arrives, if the accessed rank is in the sleep state, the active ratio is kept

unchanged by firstly switching an active rank to the sleep state and then activating the accessed rank. Before the accessed rank becomes active, the memory access is held in the memory queue. To avoid frequent transitioning, a rank is chosen to be switched only if the time it has been in the active state is longer than the *break-even time* [3].

Please note that the active ratio mechanism does not conflict with existing memory power management solutions implemented in nowadays' commercial servers (*e.g.*, transition memory ranks into the sleep state when they are idle for a certain period of time.). Our solution only throttles the maximum number of active ranks. If the number of ranks in the active state is smaller than our decision, nothing needs to be done. If it is greater, some extra ranks have to be put into the sleep state to make sure the power budget is not violated.

2.2 Coordinated Power Control Architecture

Figure 1 is a simplified illustration of a memory controller with multiple memory channels that can sustain multiple memory requests at a given time. The memory requests generated by the processor are sent to a memory queue in the memory controller after a certain bus delay. The memory scheduler converts the physical address of the requests into the memory address via an address mapping scheme. It then forwards them to the corresponding channels if the transaction queue in the channel has space. The channel scheduler reorders and schedules the requests in the transaction queue based on a scheduling algorithm (*e.g.*, Read or Instruction Fetch First). Finally, the scheduled requests are converted into a sequence of DRAM commands and serviced by the DRAM system.

As shown in Figure 1, the key components in the server power control loop include a power controller and a power monitor at the server level, a queue monitor and an active ratio modulator in the memory controller, and a DVFS modulator in the processor. The control loop is invoked periodically. At the end of each control period: 1) The power monitor (*e.g.*, a power meter) measures the average server power consumption and sends the value to the power controller. The total server power consumption is the *controlled variable*. 2) The queue monitor samples the memory queue level (*i.e.*, the number of memory requests waiting in the queue) multiple times in each control period. The average queue level is calculated at the end of each control period and used as a power demand indicator for the power controller to coordinate the processor and main memory for improved server performance. 3) The power controller collects the power value and queue level, calculates the new frequency level for the processor and the new active ratio for the main memory, respectively. The processor frequency level and active ratio are the *manipulated variables*. 4) The DVFS modulator changes the processor frequency level, and the active ratio modulator notifies the memory scheduler to power up/down memory ranks as introduced in Section 2.1, accordingly.

3. COORDINATED POWER CONTROLLER

In this section, we present the system modeling, design, and analysis of the coordinated power controller.

3.1 System Models

In order to have an effective controller design, it is important to model the dynamics of the controlled system, namely the relation between the controlled variable (*i.e.*, server power consumption) and the manipulated variables (*i.e.*, processor frequency level and memory active ratio). Since the total power is the sum of the processor power and the main memory power[1], we have two power models: the processor power model and the memory power model. We first introduce some notation. T_s is the control period. $f(k)$ is the average processor frequency in the k^{th} control period. $r(k)$ is the active ratio of the main memory. $\Delta f(k)$ is the difference between $f(k+1)$ and $f(k)$, *i.e.*, $\Delta f(k) = f(k+1) - f(k)$. $\Delta r(k)$ is the difference between $r(k+1)$ and $r(k)$, *i.e.*, $\Delta r(k) = r(k+1) - r(k)$. $F_{max} = 1$ and F_{min} are the maximum and minimum processor frequency (normalized to the maximum) while $R_{max} = 1$ and $R_{min} = 0$ are the maximum and minimum active ratio, respectively. $p_p(k)$, $p_m(k)$, and $p(k)$ are the power consumption of the processor, the main memory, and the whole server, respectively.

Memory Power Modeling. Based on the power adaptation method of the main memory introduced in Section 2.1, we need to model the dynamics of the relationship between the memory power consumption (*i.e.*, $p_m(k)$) and active ratio (*i.e.*, $r(k)$). However, a well-established physical equation is unavailable between $p_m(k)$ and $r(k)$. Therefore, we use a standard approach to this problem called *system identification* [6]. Instead of trying to build an analytical equation between $p_m(k)$ and $r(k)$, we infer their relationship by collecting data in the simulation environment introduced in Section 4 and establish a statistical model based on the measured data.

First, we examine the relationship between $p_m(k)$ and $r(k)$ based on experiments with 12 randomly selected workloads from SPEC CPU2000. Figure 2 plots the memory power consumption with the standard deviation under different active ratios for 5 workloads. Each data point in the curve is the average of 10 values which are sampled every 64 million CPU cycles (*i.e.*, the control period introduced in Section 4). The data points in the same curve are produced by forwarding the same number of instructions of the workload. As shown in Figure 2, the memory power consumption decreases when the active ratio decreases. The reduction of the memory power has two parts: the reduction of idle power due to the decreased number of ranks in the active state and the reduction of activity power due to the reduced available memory bandwidth. As shown in the figure, the curves of CPU-intensive workloads (*e.g.*, *gzip* and *mesa*) are closer to a linear model than those of memory-intensive workloads (*e.g.*, *mcf*, *swim*, and *art*). This is because the decrease of the active ratio for CPU-intensive workloads has a smaller impact on the bandwidth, compared with the memory-intensive workloads. Based on these experiments, we make three important observations:

1. There exhibits an approximately linear relationship between the memory power consumption and active ratio for each workload, except some offset.
2. The slope of all curves varies within a limited range, which is $[34.42, 53.68]$, based on our analysis.
3. The memory power consumption is stable with a fixed active ratio within a certain phase. The maximum standard deviation of all the data points in Figure 2 is only 6.63 W, compared with the average power of $95.63W$.

Using the system identification approach for each randomly selected workload, we find that a linear model fits very well for all of them (smallest $R^2 > 94\%$). Therefore, it is valid to assume that there exists a model between the memory power consumption and active ratio as follows:

[1] In this paper, since we do not change the power states of other components in a server, we assume that their power consumption can be approximated as a constant and is thus eliminated in the difference power model (5) introduced later.

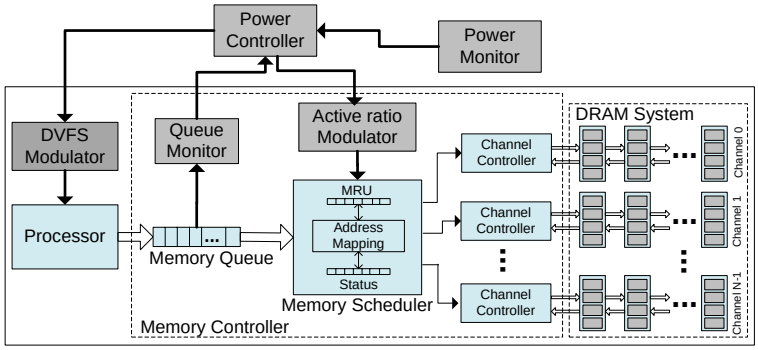


Figure 1: Coordinated Power Control Architecture

$$p_m(k) = k_r r(k) + C_i, \tag{1}$$

where k_r is a generalized parameter that may vary for different workloads and C_i is a constant representing the offset. The dynamic model as a difference equation is

$$p_m(k+1) = p_m(k) + k_r \Delta r(k). \tag{2}$$

To validate our system model, we first stimulate the main memory system with pseudo-random white-noise input to change the active ratio in a random manner. We then compare the actual power consumption with the value predicted by our model. Figure 3 plots the predicted power and measured power by running *swim* which has the largest standard deviation. We can see that the predicted power is adequately close to the actual power of the memory system, even for the workload with a significant power variation.

Please note that we are not trying to find a fixed k_r for all workloads, but to validate the linearity observed in our experiments by using 12 randomly selected workloads from SPEC CPU2000 (5 of them shown in Figure 2). As we prove in Section 3.4, only if the workload exhibits an approximately linear memory power model and the slope k_r (together with the slope for the processor power model as we discuss later) varies within a certain range, the stability of our coordinated power control algorithm can be guaranteed. A key advantage of the control-theoretic design approach is that it can tolerate a certain degree of modeling errors and can adapt to online model variations based on dynamic feedback [6]. As a result, our solution does not need to rely on power models that are 100% accurate, which is in sharp contrast to open-loop solutions that would fail without an accurate model.

Processor Power Modeling. Raghavendra et al. [25] and Lefurgy et al. [15] have shown that the processor power is approximately linear to the DVFS level. In this paper, we model the processor power in the same way as the memory power by using system identification. The power consumption of a processor is modeled as:

$$p_p(k) = p_p(k-1) + k_f \Delta f(k), \tag{3}$$

where k_f is a generalized parameter that may vary for different workloads running in the processor. The detailed analysis of the processor power model can be found in [15].

Server Power Modeling. The server power consumption is the sum of power of all components in the server:

$$p(k) = p_p(k) + p_m(k) + p_o, \tag{4}$$

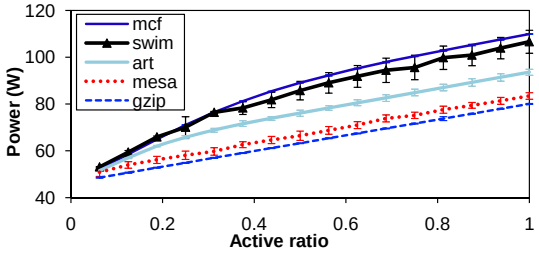


Figure 2: Relationship between memory power and active ratio

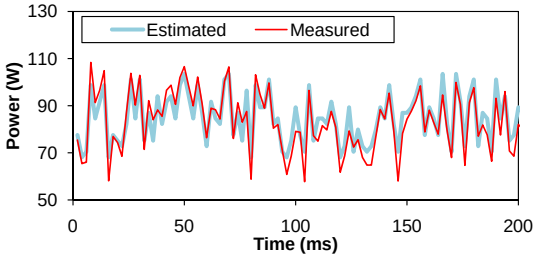


Figure 3: Memory Power Model Validation

where p_o is the power consumption of other components, which can be eliminated in a difference power model. Based on (2) and (3), we can get the difference power model as follows:

$$p(k+1) = p(k) + \mathbf{K}\boldsymbol{\Delta}\mathbf{v}(k), \tag{5}$$

where $\mathbf{K} = \begin{bmatrix} k_f & k_r \end{bmatrix}$ and $\boldsymbol{\Delta}\mathbf{v}(k) = \begin{bmatrix} \Delta f(k) & \Delta r(k) \end{bmatrix}^T$. The actual values of k_f and k_r at runtime may change for different workloads and are unknown at design time. In Section 3.4, we prove that a system, controlled by the controller designed with the estimated parameters, can remain stable as long as the variations of k_f and k_r are within allowed ranges.

3.2 Controller Design

We apply *Model Predictive Control* (MPC) theory [21] to design the controller based on the system model (5). MPC is an advanced control technique that can deal with coupled MIMO control problems with constraints on the plant and the actuators. MPC enables us to combine power prediction, optimization, constraint satisfaction, and feedback control into a single algorithm. This property makes MPC well suited for coordinating the processor and main memory for server power control.

A model predictive controller optimizes a *cost function* defined over a time interval in the future. The controller uses the system model to predict the control behavior over P control periods, which is referred to as the *prediction horizon*, based on the feedback $p(k)$

from the power monitor. The control objective is to select an input trajectory that minimizes the cost function while satisfying the constraints. An input trajectory includes the control inputs in the M control periods, which is referred to as *control horizon*. The controller includes a least squares solver, a cost function, a reference trajectory, and a system model. At the end of every control period, the controller computes the control input $\mathbf{\Delta v(k)}$ that minimizes the following cost function under constraints.

$$V(k) = \sum_{i=1}^{P} \|p(k+i|k) - ref(k+i|k)\|^2_{Q(i)} + \sum_{i=0}^{M-1} \|\mathbf{\Delta v(k+i|k)} + \mathbf{v(k+i|k)} - \mathbf{V_{max}}\|^2_{\mathbf{R(i)}} \quad (6)$$

where $\mathbf{V_{max}} = \begin{bmatrix} F_{max} & R_{max} \end{bmatrix}^T$, $Q(i)$ is the *tracking error weight*, and $\mathbf{R(i)}$ is the *control penalty weight vector*. The notation $x(k+i|k)$ means that the value of variable x at time $(k+i)T$ depends on the conditions at time kT. The first term in the cost function represents the *tracking error*, *i.e.*, the difference between the total power $p(k+i|k)$ and a reference trajectory $ref(k+i|k)$. The reference trajectory defines an ideal trajectory along which the total power $p(k+i|k)$ should change from the current value $p(k)$ to the set point P_s (*i.e.*, power budget of the server)[21]. The second term in the cost function (6) represents the *control penalty*. The control penalty term causes the controller to optimize system performance by minimizing the difference between the highest power states of the processor and the main memory, $\mathbf{V_{max}}$, and the new power states, $\mathbf{v(k+i+1|k)} = \mathbf{\Delta v(k+i|k)} + \mathbf{v(k+i|k)}$, along the control horizon. The control weight vector, $\mathbf{R(i)} = \begin{bmatrix} R_c & R_m \end{bmatrix}_{1\times M}$, is tuned, based on the memory queue level, to shift power to either the processor (*i.e.*, R_c) or the main memory (*i.e.*, R_m), which will be discussed in Section 3.3. This control problem is subject to two sets of constraints. First, both the DVFS level of the processor and the active ratio of the main memory should be within allowed physical ranges. Second, the total power consumption should not be higher than the desired power constraint.

Based on the above analysis, the problem of server power control has been modeled as a constrained MIMO optimal control problem, which can be easily transformed to a standard constrained least-squares problem [21]. Please note that the computational complexity and runtime overhead of an MPC controller can be significantly reduced by optimizing the original MPC algorithm. For example, a hardware implementation of an improved MPC algorithm with 4 inputs (we have 2 inputs in the paper) and 1 output (e.g., the server power) only takes $4.7\mu s$ at a clock frequency of $20MHz$ [12]. Compared with the power control period (*i.e.*, $20ms$) as discussed in Section 4.1, the computation overhead can be considered small ($< 0.02\%$). Therefore, it is feasible to use MPC in practice for server power control.

3.3 Weight Allocation in Coordinated Controller

The power controller discussed above can precisely control the total server power consumption to the desired budget by solving an optimization problem. However, it cannot guarantee that the power states of the processor and main memory are coordinated to achieve further improved performance. The experiments in Section 5.4 demonstrate that the system performance, in terms of Instructions Per Cycle (IPC), differs significantly by giving different preferences to the processor and main memory. In this subsection, we introduce a simple, but efficient, coordination scheme that allocates the control penalty weights for the processor and main memory based on the memory queue level, so that improved performance can be achieved. The reason why we use the memory queue level as the indicator is as follows: (1) the memory queue physically exists in many memory controllers [9], so it does not incur any other implementation overhead. (2) More importantly, when compared with other related metrics such as CPU stall time or L2 cache miss ratio, the queue level is a fair indicator for all workloads and can quantitatively reflect the power demand of the memory system. For example, if the queue level increases, it means that the memory requires more power to increase its capability so that the waiting time of the memory requests in the queue can be decreased.

In this paper, we propose an algorithm called *Moving Average* (MA) to assign weights in the coordinated controller as follows. We keep the weight of the processor (*i.e.*, R_c) constant at 1 and adjust the weight of the memory (*i.e.*, R_m) at runtime. At the end of each control period, the controller calculates the moving average of the queue level $q_a(k)$ in a window with a size of L, after receiving the queue level $q(k)$ from the queue level monitor. Specifically, $q_a(k) = \sum_{i=0}^{L-1} q(k-i)$. Clearly, the selection of the window size is a trade-off between a smooth weight allocation for the main memory and the response speed to memory workload variations. Based on our experiments, we found that the window size of 4 works well for most workloads. If the moving average of queue level $q_a(k)$ is higher than the reference level Q_{ref}, we set the weight of the memory as $(q_a(k) - Q_{ref})\alpha + 1$ to indicate that the memory needs more power. If $q_a(k)$ is smaller than Q_{ref}, the weight of the memory is set as $(q_a(k)/Q_{ref})^\beta$, which is lower than 1, to indicate that the processor needs more power. Both α and β are experiential parameters that map the queue level to the weight. Based on our profiling experiments with SPEC CPU 2000, we find that $\alpha = 10$ and $\beta = 2$ have approximately the best results.

The selection of the reference level Q_{ref} also plays an important role in the allocation scheme. Due to the dependencies among instructions, the length of the memory queue usually stops increasing when the processor stalls, instead of increasing infinitely. We define the *saturation level* as the maximum queue level that can be accumulated before the processor stalls. Clearly, the saturation level relies highly on the degree of dependency among the instructions. The stronger the dependency, the smaller the saturation level. If Q_{ref} is too high (*e.g.*, higher than the saturation level), the processor may stall before the queue level can be accumulated to the reference level. Likewise, if Q_{ref} is too low, the preference is more likely to be always given to the processor. Both of these may lead to degraded performance. Based on our profiling experiments, most workloads have a saturation level from 6 to 50. Without loss of generality, we set the reference level Q_{ref} as 4. Note that no reference level can be guaranteed to be always optimal in terms of performance for all workloads, since the interaction between the processor and memory is interleaved among instructions at runtime.

3.4 Control Analysis for Model Variations

A fundamental benefit of the control-theoretic approach is that it gives us confidence for system stability, even when the system power model (5) may change at runtime due to workload variations. We say that a system is *stable* if the total power $p(k)$ converges to the desired set point P_s, that is, $\lim_{k\to\infty} p(k) = P_s$.

We now outline the general steps to analyze the stability of the system controlled by the coordinated power controller, when the actual system model is different from the *estimated* model used to design the coordinated controller. First, given a specific system, we derive the control inputs $\mathbf{\Delta v(k)}$ that minimize the cost function based on the estimated system model (5) with estimated parameters $\mathbf{K}$. Second, we construct the *actual* system model by assuming the actual parameter $k'_f = g_f k_f$ and $k'_r = g_r k_r$ for the processor and

memory, respectively, where g_f and g_r represent the unknown system gain. Third, we derive the closed-loop system model by substituting the control inputs derived in the first step into the actual system model. Finally, we analyze the stability of the closed-loop system by computing the poles of the closed-loop system. According to control theory, if all the poles locate inside the unit circle in the complex space, the controlled system is stable.

Following the steps above, we have proven that the closed-loop system is guaranteed to be stable when $0 < g_f, g_r < 5.3$. This means that a system, controlled by the coordinated controller designed based on the estimated model (5), can remain stable as long as the real system parameters k'_f and k'_r are smaller than 5.3 times of the values used to design the controller. This stability analysis gives us confidence in the performance of our controller since we use the average k_f and k_r of all workloads from SPEC CPU2000 at the controller design time. It is reasonable to consider that our closed-loop system is stable for all workloads.

4. SYSTEM IMPLEMENTATION

In this section, we introduce our simulation environment and discuss the impact of DDR3 technology on our solution.

4.1 Simulation Environment

It would be ideal to test the proposed coordinated solution on a hardware testbed. However, to our best knowledge, currently there are few DRAM devices that are commercially available and provide external interfaces for power state adaptation, though DVFS is widely available in many processors. Therefore, as in other memory power management projects [3][5], we can only use simulations to evaluate the proposed coordinated solution. We integrate two cycle-accurate simulators: SimpleScalar [2] and DRAMsim [29], to simulate both the processor and main memory. SimpleScalar is heavily modified to simulate a quad-core CMP with one thread per core. To accurately simulate memory dependency, we modify the static main memory latency in SimpleScalar and hold all instructions which require main memory accesses from dispatching until the main memory accesses they depend on are returned by DRAMsim. Since the proposed solution is designed for high-end servers with large memory capacity, DRAMsim is configured to simulate a FB-DIMM DDR2 SDRAM system for its high bandwidth. The memory system has a capacity of 32 GB with four channels and specifications are based on the Micron data sheet [22]. The major parameters of the processor and main memory are shown in Table 2. We integrate Wattch [1] with SimpleScalar to estimate the power consumption of the processor. The power calculation in DRAMsim is based on the power model proposed by Micron [23]. We assume that the simulated quad-core CMP has an idle power of 50 W when running at the lowest frequency, based on a recent quad-core Xeon processor from Intel. Hence, the peak power of the simulated system can be as high as 270 W. In this paper, we assume that the processor and memory contribute the majority of the total power consumption in a server. However, our framework described above can be extended to include other components, such as network or disks.

Since the new frequency level periodically received from the controller could be any value that is not exactly one of the four supported DVFS levels shown in Table 2, we implement the first-order delta-sigma modulator proposed by Lefurgy et al. [15]. The modulator approximates the desired value via a series of supported DVFS levels. For example, to approximate $3GHz$ during a control period, the modulator would output the sequence, 2.8, 3.2, 2.8, and 3.2, on a smaller timescale. Apparently, the more subintervals the modulator is invoked within one control period, the better the approximation is, but with a higher overhead. In this paper, we choose to use 20 subintervals to approximate the desired DVFS level. Based on Skadron et al. [27], the DVFS overhead is approximately $10\mu s$. As a result, we choose a subinterval of $1ms$ so that the overhead is up to 1% in the worst case when the DVFS level is changed every subinterval. Therefore, the coordinated control period is $20ms$, which is 64 million CPU cycles in our simulation environment. To simulate the overhead of DVFS, we assume there is no instruction executed during transitions [10].

The server power is calculated every control period (*i.e.*, 64 million CPU cycles) in our simulation environment. Many of today's high-density servers are equipped with built-in power measurement circuit, such as IBM system x and p servers [14]. Those circuits can accurately measure the server power with a sampling period as short as $1ms$, which is much shorter than the sampling period of $20ms$ used in this paper. Therefore, it is realistic to precisely measure server power at runtime at a low cost.

In real systems, the controller can be implemented in the service processor firmware. In our simulation environment, the coordinated controller is implemented as a separate process and communicates with the processor-memory simulator via a pipe. The controller executes the control algorithm presented in Section 3 by calling a Matlab library which implements a standard constrained least-square solver. Our experiments are driven by pre-compiled alpha binaries of SEPC CPU2000. Since our goal is to validate the idea of shifting power between processors and memory for improved performance within a certain power constraint, we generate 4 identical threads of the same workload and run each thread on each core. To test our solution by running other coupled multi-threaded workloads, we need to incorporate advanced workload scheduling algorithms that can guarantee performance fairness by allocating power among different cores, which will be our future work. The performance is accumulated across all the cores in term of IPC. Please note that the length of cycle is normalized to the reciprocal of the highest frequency (*i.e.*, $3.2GHz$). Therefore, the performance metric of IPC is equivalent to instructions per second.

4.2 Impact of DDR3 Technology

In this paper, we aim for high-end servers with high memory bandwidth. We chose FB-DIMM memory architecture for it high bandwidth [7]. However, our solution is not limited to a certain DRAM technology, and can also be applied on other memory systems such as DDR3. In this section, we discuss the impact of DDR3 technology on our solution, which will be part of our future work.

Memory power model. The coordinated power control solution is designed based on the processor and memory power models presented in Section 3.1. A similar memory power model can be derived as long as the system consists of multiple memory ranks whose power states can be transitioned independently, as in a DDR3 memory system. Therefore, it is reasonable to infer that a similar memory power model can be derived for DDR3, but with a different parameter k_m.

High power of FB-DIMM. FB-DIMM memory modules have relatively higher power consumption compared with conventional memory modules, mainly due to the Advanced Memory Buffer (AMB). For each AMB, the idle power (4-5 Watts) takes a major part of the total power (4-8 Watts) [18][19]. However, we do not take advantage of the high idle power of AMB. In our memory power model (1), the high idle AMB power is part of the offset C_i, which is eliminated in the difference model (2). As a result, the power dynamics of the memory system mainly come from the power difference between the different power modes of memory ranks, instead of the high idle power of FB-DIMM. If the coordi-

Table 2: Simulator parameters

Parameters	Values
Processor	4 cores, 8 issues per core
Frequency scaling	3.2GHz at 1.3V, 2.8GHz at 1.15V, 1.6GHz at 0.95V, 0.8GHz at 0.8V
Functional unit	4 IntALU, 2 IntMult, 2 FPALU, 1 FPMult
L1 caches (per core)	64KB Inst/64KB Data, 2-way, 64B line size, 3-cycle hit latency
L2 cache (shared)	8MB, 8-way, 64B line size, 12-cycle hit latency
Memory	4 channels, 4 DIMMs/channel, 8 banks/DIMM, 1 rank/DIMM
Channel bandwidth	667MT/s, FB-DIMM DDR2

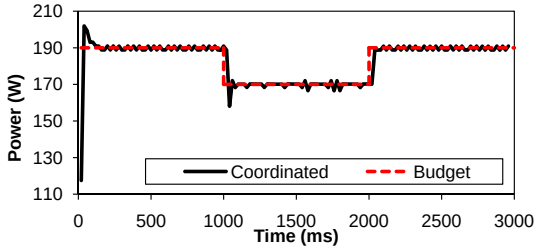


Figure 4: A typical run of the coordinated solution

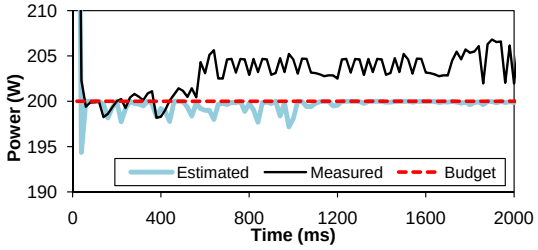


Figure 5: Estimation error of PLI

nated solution is applied to DDR3 memory systems, similar power dynamics can be achieved as long as DDR3 memory chips have a similar power reduction from the PRE state to the PPD state and the available memory bandwidth decreases similarly. Therefore, it is reasonable to believe that DDR3 memory systems may have power dynamics similar to FB-DIMM configured in our testbed. As a result, the coordinated solution can achieve similar performance improvement in DDR3 memory systems.

5. EVALUATION

In this section, we first introduce two state-of-the-art baselines. We then compare the coordinated solution with the two baselines, in terms of power control performance and application performance. Finally, we investigate the impact of weight allocation on server performance.

5.1 Baselines

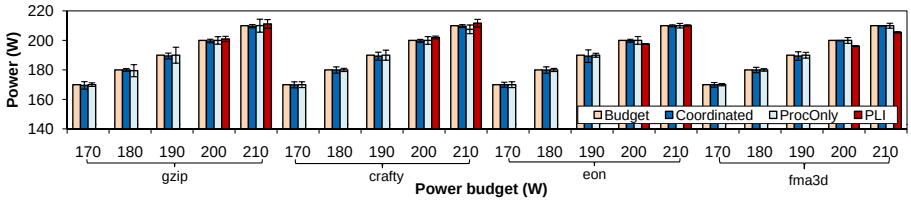
Our first baseline, referred to as *ProcOnly*, is a server power control solution based on feedback control theory, proposed by Lefurgy et al. [15]. ProcOnly represents a typical server power control solution that assumes the processor is the only major power consumer in a server. ProcOnly leverages frequency scaling in the processor to control the power consumption of the whole server to be within a certain power budget. We compare the proposed coordinated solution against ProcOnly to highlight that coordinating the processor and main memory, when power budget is limited, is important to achieve better performance than only considering the processor. A fundamental difference between the coordinated solution and ProcOnly is that ProcOnly only manipulates the CPU frequency while disregarding the synergy between the processor and main memory. In contrast, the coordinated solution manipulates the power states of both the processor and memory, and adaptively adjusts the power states of the two components in a coordinated way.

The second baseline, referred to as *Proportional-by-Last-Interval* (PLI), is a server power control scheme that shifts power between the processor and main memory based on the number of activities, proposed by Felter et al. [5]. PLI profiles the processor power model as a function of the number of dispatched instruction per cycle (DPC), while the memory power is modeled as a function of memory bandwidth. It periodically estimates the server power consumption based on the two off-line power models. Given a power budget, in every period, PLI calculates the maximum number of activities in the processor (*i.e.*, dispatched instructions) and memory (*i.e.*, memory requests) that can occur in the next period without violating the power budget, as the thresholds, proportionally to the measured number of activities in the last period. The power budget is enforced by only running the calculated numbers of activities in the processor and main memory. There are three fundamental differences between PLI and the coordinated solution. *First of all*, PLI is based on an estimation strategy that does not explicitly measure the power but relies on estimation, and thus cannot guarantee budget enforcement when the runtime power model becomes different from the profiled power model. Although the coordinated solution also predicts the power consumption based on profiled power models, the fundamental advantage of the coordination solution is that the prediction is continuously corrected based on feedback information, as discussed in Section 3.2. *Second*, the power allocation of PLI is proportional to the number of activities. In contrast, in the coordinated solution, the power allocation is driven by a performance indicator, the memory queue level. *Finally*, PLI enforces the power budget by directly throttling the number of activities. In contrast, the coordinated solution exploits different power states to effectively reduce idle power for both the processor and memory. When the budget is tight, our solution can allow more system activities by utilizing the reduced idle power. As a result, our solution has a higher capacity of power adaptation and can achieve better application performance.

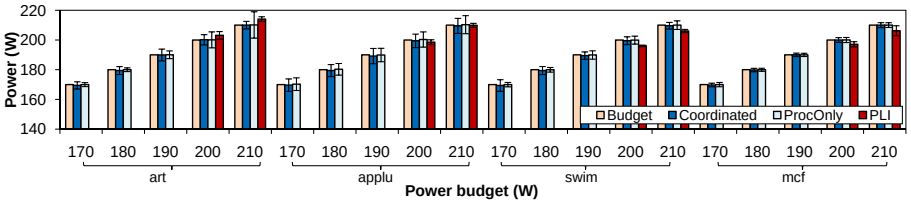
5.2 Power Control Performance

In this experiment, we compare the power control performance of the coordinated solution with the two baselines, in terms of power control accuracy and power adaptation capacity.

Power Control Accuracy. To test the power control accuracy of the coordinated solution in a scenario where the power budget of the system needs to be changed at runtime due to various reasons (*e.g.*, thermal emergencies), we select a workload from SPEC benchmarks that is not used in the profiling of the memory power model in Section 3.1, *twolf*. As shown in Figure 4, the power budget is reduced from $190W$ to $170W$ at time $1,000ms$, and then restored to $190W$ at time $2,000ms$. We can see that the coordinated solution quickly responds to the power budget reduction and



9(a) CPU-intensive workloads



9(b) Memory-intensive workloads

Figure 6: Comparison of power control among the coordinated solution, ProcOnly, and PLI

Table 3: Power adaptation capacity of the proposed coordinated solution and two baselines.

	Coordinated	ProcOnly	PLI
Lowest budget (W)	105	132	191

precisely control the total power consumption of the server to the budget by adjusting the CPU frequency of the processor and the active ratio of the memory.

Figures 6(a) and (b) show the average power consumption under the proposed coordinated solution, ProcOnly, and PLI with standard deviations for CPU-intensive and memory-intensive workloads, respectively. Each bar in the figures is the average of 110 data points after the controllers enter the steady state. The reason why there are no results for PLI at the budgets of $190W$, $180W$, and $170W$ is because PLI is incapable of lowering the power below its idle power (*i.e.*, $191W$). The smaller the standard deviation is, the smaller power variation will be, that is, the less dangerous to have a power violation. We can see that the average power consumption under both the coordinated solution and ProcOnly precisely converges to the budgets. However, the standard deviation of ProcOnly tends to be greater than the coordinated solution. For example, the maximum standard deviation of ProcOnly in all runs is 8.94W compared to 5.1W for the coordinated solution. This is because the coordinated solution has two manipulated variables (*i.e.*, CPU frequency and active ratio) which can handle larger workload variations than ProcOnly, which has only one manipulated variable (*i.e.*, processor frequency). The result is consistent with the stability analysis of the coordinated solution presented in Section 3.4, and the analysis of ProcOnly presented in [15]. The analyses show that the stability range of the coordinated solution (*i.e.*, $(0, 5.3]$) is larger than that of ProcOnly (*i.e.*, $(0, 2)$). Therefore, the coordinated solution is less vulnerable to workload variations and more robust.

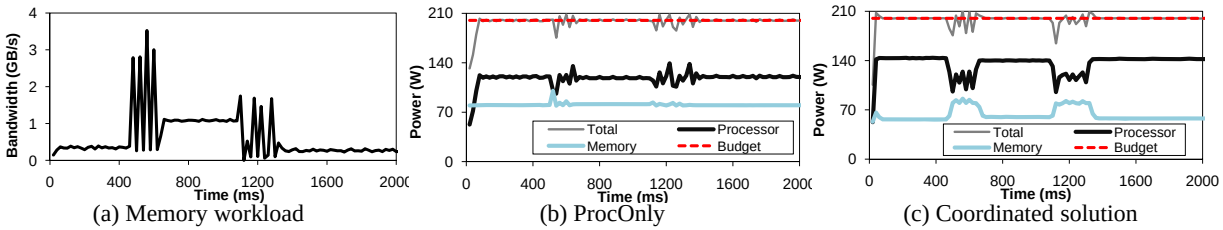
As for PLI, the average power consumption fails to converge to the power budget due to estimation errors in most runs. For example, the average power consumption under PLI is either smaller (*e.g.*, *fma3a* and *mcf*) or larger (*e.g.*, *art*) than the power budget. As a result, PLI either violates the power budget or leads to degraded performance. This is because PLI features a strategy that does not explicitly measure power but estimates power based on measured activities. As a result, it relies heavily on the accuracy of the offline profiling of power models so that it has the risk of violating the power budget. To further verify our analysis of estimation inaccuracy about PLI, Figure 5 plots both the measured and estimated power consumption in a typical run of PLI by using *lucas* ($0.8B$ instructions are forwarded) under a power budget of $200W$. We can see that the actual power consumption (*i.e.*, measured) is successfully enforced to the power budget at the beginning because the power characteristic of activities in the processor and main memory are accurately captured by the power models used by PLI for estimation. After the time around 580ms, the power characteristic varies. However, PLI neglects the variation due to the lack of feedback information and still allocates power between the processor and memory based on the measured number of activities. This results in a constant power violation of approximately $4.5W$, which is highly undesirable and may cause system failures in real systems.

Power Adaptation Capacity. Power adaptation capacity is defined as the power budget range that a control solution can achieve. A higher power adaptation capacity means that a control solution can still manage to conduct power control even when the power budget becomes very tight (*e.g.*, due to thermal emergency). PLI caps power by throttling the number of dispatched instructions in the processor and the number of memory requests in the main memory, instead of exploiting the low-power states. As a result, it has the smallest power adaptation capacity among the three solutions. As introduced in Section 5.1, a fundamental difference between the coordinated solution and ProcOnly is that the coordinated solution utilizes the low-power states of both the processor and main memory while ProcOnly only manipulates the DVFS level of the processor. Therefore, the coordinated solution has the largest power adaptation capacity. Table 3 shows the average minimum power budget that can be achieved by the three power control algorithms using 8 workloads randomly selected from SPEC CPU2000. If the maximum power budget can be achieved is 270 W, we can see that the coordinated solution improves the power adaptation capacity up to 20% and 120%, compared with ProcOnly and PLI, respectively.

5.3 Application Performance

Now we compare the coordinated solution with the two baselines in terms of application performance.

ProcOnly. In this experiment, we run ProcOnly and the coordinated solution by using *gzip* under a power budget of 200 W. Figure 7(a) shows the memory bandwidth of running *gzip* in our simulation environment by forwarding $0.4B$ instructions. Figures 7(b) and (c) show the power consumption of the whole server, the processor, and the main memory in a typical run of ProcOnly and the



(a) Memory workload (b) ProcOnly (c) Coordinated solution

Figure 7: Comparison with ProcOnly during a typical run of gzip

coordinated solution, respectively. We can see tha *gzip* experiences time-varying memory workload from a bandwidth of $0.3GB/s$ to more than $3GB/s$ at runtime. However, ProcOnly only adjusts the power states of the processor, disregarding the variations of memory workload. Consequently, the power consumption of the memory system is unnecessarily high even at the time when the memory system experiences a low workload. In contrast, the proposed coordinated solution adapts to the low memory workload by putting the main memory in low-power states and putting the processor in relatively high-power states. When the memory workload increases significantly, the coordinated solution shifts power from the processor to the main memory. As a result, performance increases by approximately 11% when compared with ProcOnly, due to the coordination between the processor and main memory. Please note that, though the power temporarily violates the budget due to severe workload variations in Figures 7(b) and (c), these overshoots are instantaneous (*i.e.*, in tens of milliseconds) and the system is safe as long as the server power can be controlled back to the desired budget within the designed time interval that the power supply can sustain a power overload.

To more thoroughly investigate the performance comparison between the coordinated solution and ProcOnly, we run experiments by using a variety of workloads under different power budgets. Figures 8(a) and (b) show the performance comparison for CPU-intensive and memory-intensive workloads, respectively. Performance is measured as the average IPC of 120 data points from the beginning of each run. As we can see, the coordinated solution has better performance for all CPU-intensive workloads. The average improvement of the coordinated solution over ProcOnly at each power budget is shown in Figures 8(a) (*i.e.*, up to 23% on average at the budget of $170W$). This is because the coordinated solution can coordinate the power states of the processor and main memory by dynamically shifting power between them. We can also see that the lower the budget, the better improvement that the coordinated solution can achieve over ProcOnly (*i.e.*, the improvement increases from 6% to 23% as the budget decreases from $210W$ to $170W$). This is because when the power resource becomes more constrained, it is more important to efficiently allocate it between the processor and main memory for improved performance. As for the memory-intensive workloads shown in Figure 8(b), the coordinated solution has similar performance to ProcOnly. This is because those memory-intensive workloads have high memory traffic at the majority of time. As a result, the coordinated solution places the memory in high-power states almost all the time which is similar to what ProcOnly does. The reason why the coordinated solution has a very slightly lower performance (around 1%) than ProcOnly for *applu* and *art* is because *applu* and *art*'s memory workload oscillates more than other memory-intensive workloads so that the queue level varies significantly at runtime. As a result, the coordinated solution has a higher overhead by frequently adjusting the power states of the processor and main memory. This set of experiments demonstrates that the coordinated solution achieves considerably better application performance than ProcOnly for CPU-intensive workloads and similar performance for memory-intensive workloads.

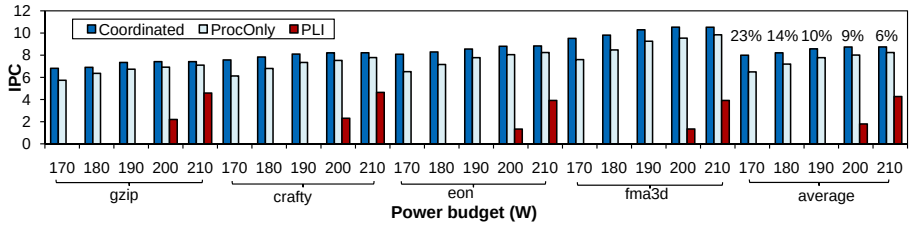
It is important to note that the *runtime* complexity of the proposed coordinated solution is comparable to that of ProcOnly, as discussed in Section 3.2. Please also note that the performance improvement is highly dependent on the percentage of memory power in the total power consumption of the system. The higher the percentage is, the higher the improvement will be. As presented in [14], memory power may have a much higher percentage in many high-end servers than our configuration (*i.e.*, around 50%). Therefore, the performance improvement of the coordinated solution can be even more significant.

PLI. The high idle power places PLI in a disadvantageous situation when the power budget is tight (*e.g.*, lower than the average power consumption). As shown in Figures 8(a) and (b), PLI has much lower performance than both the coordinated solution and ProcOnly. On average across all the budgets, the coordinated solution improves PLI by 110% and 120% for CPU-intensive and memory-intensive workloads, respectively. However, as the power budget increases, differences between PLI and the other two solutions becomes smaller.

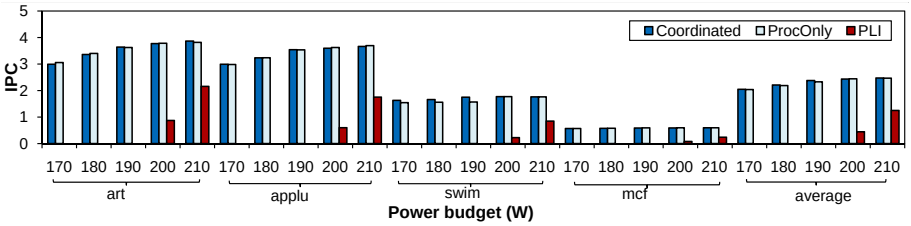
5.4 Weight Allocation Schemes

Now we investigate the control penalty weight $\mathbf{R}(i)$ in the cost function (6) and the impacts of different allocation schemes on the coordination between the processor and main memory. To highlight the importance of the dynamic weight allocation based on the memory queue level, we compare the proposed moving average scheme (MA) with three static allocation schemes. The first one, referred to as *Equal*, gives the same preference to the processor and main memory by assigning an equal weight to them. The other two, referred to as *Proc-preferred* and *Mem-preferred*, always give preference to the processor and main memory, respectively.

To stress test the allocation schemes, we select 8 workloads and run all of them under a tight budget, $150W$, which is only approximately 56% of the system's peak power. We plot the average IPC of 200 control periods from the beginning in Figure 9. We can observe that: 1) For CPU-intensive workloads, MA has better performance than both Equal and Mem-preferred. The reason is that using lower processor power states unnecessarily for CPU-intensive workloads may hurt performance significantly. At the same time, MA has similar performance to Proc-preferred. Note that Proc-preferred has slightly better performance than MA (around 2%) for *gcc*. This is because MA has some overhead to find the best allocation weight based on the memory queue level, which results in slightly worse performance. 2) For memory-intensive workloads, MA has significantly better performance than Proc-preferred. In addition, MA has slightly better performance than both Equal and Mem-preferred (with the average 2% and 6%, respectively). This is because shifting power to the main memory for memory-intensive workloads has less impact on performance than shifting power to the processor for CPU-intensive workloads, since the memory latency dominates the performance bottleneck.



9(a) CPU-intensive workloads



9(b) Memory-intensive workloads

Figure 8: Comparison of performance among the coordinated solution, ProcOnly, and PLI

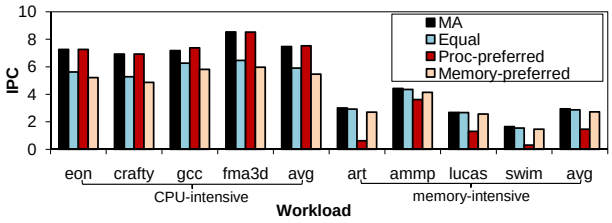


Figure 9: Comparison of weight allocation schemes

In general, MA, which dynamically allocates control penalty weights to the processor and main memory based on the memory queue level, is superior to other static allocation schemes.

6. RELATED WORK

Power consumption has become one of the most important design concerns for computing systems. Much prior work has focused on minimizing the power consumption within a specified performance guarantee. At the cluster level, Verma et al. [28] and Horvath et al. [8] propose different power management schemes to minimize power consumption while guaranteeing the application performance. At the server level, Li et al. develop algorithms for power adaptation between processor and memory so that energy can be minimized [16]. At the component level, energy conserving algorithms have been presented for processors [33], DRAM systems [34], disks [17], and caches [13]. However, all of the solutions cannot provide any explicit guarantees for the power consumption to stay below the desired budget though the performance is guaranteed. Our work is different in that we focus on a different but equally important problem, *i.e.*, power control to avoid power overload or thermal violations.

Several power provisioning strategies have been proposed by [4] to host the maximized number of servers allowed by the limited power supply in data centers. Different power control solutions have been proposed at different levels in data centers [25][31] to maximize application performance. Shang et al. present PowerHerd to control the peak power of interconnection networks [26]. At the server level, Lefurgy et al. propose a server power control solution based on basic feedback control theory by assuming that CPU is the major power contributor in servers [15]. Felter et al. propose to cap server peak power by shifting power between the processor and memory proportionally to the number of activities [5]. At the cluster level, Wang et al. propose a MIMO controller to cap the cluster power consumption [30]. Raghavendr et al. also propose a hierarchical power control solution for server clusters [25]. However, those solutions conduct power control by adjusting only the DVFS levels of the server processors. In contrast, our work coordinates the processor and main memory for efficient server power capping based on an advanced optimal MIMO control theory. Our server-level power control solution can be used in a cluster setting with a cluster-level power control loop, such as the ones in [25] [31]. The cluster-level loop allocates power budget to each server, where our loop precisely controls the server power to the desired budget. As shown in Figure 4, our controller can react quickly to a sudden budget change. The settling time is just several control periods. Since our control period can be as short as $20ms$, the settling time is acceptably short compared with the time interval for a PDU-level circuit breaker to trip, which is about $2s$ for a power overload of 20% over the rated capacity [31].

Power control solutions have also been applied on different server components to address cooling and thermal problems. For example, Diniz et al. propose several policies to limit power consumption for DRAM systems [3]. Lin et al. develop dynamic thermal management (DTM) techniques for FB-DIMM memory systems by DVFS [18]. Various power control solutions have also been proposed to cap the power consumption of a chip multiprocessor with per-core DVFS [10][20][32]. However, all of them focus on a single component in servers and cannot be directly applied to server power control.

7. CONCLUSIONS

Existing power control solutions either rely solely on processor frequency scaling or shift power simply based on estimated system activities. In this paper, we propose a novel server power control solution that can precisely control the power consumption of a server to the desired budget. Our solution shifts power between processor and main memory in a coordinated manner by dynamically adjusting the voltage/frequency of the processor and placing memory ranks into different power states, based on their power demands indicated by the memory queue level, to achieve improved server performance. We compare with two state-of-the-art server power control solutions. One baseline relies only on processor frequency scaling while the other one uses estimated system activi-

ties for power shifting. Our experimental results demonstrate that our solution, on average, achieves up to 23% better performance than the first baseline for CPU-intensive benchmarks and doubles the performance of the second baseline when the power budget is tight. In addition, our solution significantly improves the power adaptation range and has better power control accuracy.

8. ACKNOWLEDGEMENTS

This work was supported, in part, by NSF under CAREER Award CNS-0845390 and Grants CCF-1017336, CNS-0915959, and CNS-0720663, and by ONR under Grant N00014-09-1-0750.

9. REFERENCES

[1] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a Framework for Architectural-level Power Analysis and Optimizations. In *ISCA*, 2000.

[2] D. Burger and T. M. Austin. The SimpleScalar Tool Set, Version 2.0. In *Technical Report CS-TR-1997-1342, University of Wisconsin, Madison*, 1997.

[3] B. Diniz, D. Guedes, W. Meira, and R. Bianchini. Limiting the Power Consumption of Main Memory. In *ISCA*, 2007.

[4] X. Fan, W.-D. Weber, and L. A. Barroso. Power Provisioning for a Warehouse-sized Computer. In *ISCA*, 2007.

[5] W. Felter, K. Rajamani, and T. W. Keller. A Performance-Conserving Approach for Reducing Peak Power Consumption in Server Systems. In *ICS*, 2005.

[6] G. F. Franklin, J. D. Powell, and M. Workman. *Digital Control of Dynamic Systems, 3rd edition*. Addition-Wesley, 1997.

[7] B. Ganesh, A. Jaleel, D. Wang, and B. Jacob. Fully-Buffered DIMM Memory Architectures: Understanding Mechanisms, Overheads and Scaling. In *HPCA*, 2007.

[8] T. Horvath, T. Abdelzaher, K. Skadron, and X. Liu. Dynamic Voltage Scaling in Multi-tier Web Servers with End-to-end Delay Control. *IEEE Transactions on Computers*, 56(4), 2007.

[9] Intel. Intel 845 Chipset: 82845 Memory Controller Hub (MCH) Datasheet. http://developer.intel.com/Assets/PDF/datasheet/290725.pdf, 2002.

[10] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi. An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget. In *MICRO*, 2006.

[11] B. Jacob, Spencer, and D. Wang. *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann, 2007.

[12] T. A. Johansen, W. Jackson, R. Schreiber, and P. Tøndel. Hardware Synthesis of Explicit Model Predictive Controllers. *IEEE Control Systems Technology*, 15(1), Jan. 2007.

[13] S. Kaxiras, Z. Hu, and M. Martonosi. Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power. In *ISCA*, July 2001.

[14] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T. W. Keller. Energy Management for Commercial Servers. *IEEE Computer*, 36(12), Dec. 2003.

[15] C. Lefurgy, X. Wang, and M. Ware. Power capping: a prelude to power shifting. *Cluster Computing*, 11, 2008.

[16] X. Li, R. Gupta, S. V. Adve, and Y. Zhou. Cross-Component Energy Management: Joint Adaptation of Processor and Memory. *ACM Transactions on Architecture and Code Optimization*, 4, 2007.

[17] X. Li, Z. Li, F. David, P. Zhou, Y. Zhou, S. Adve, and S. Kumar. Performance Directed Energy Management for Main Memory and Disks. In *ASPLOS*, 2004.

[18] J. Lin, H. Zheng, Z. Zhu, H. David, and Z. Zhang. Thermal Modeling and Management of DRAM Memory Systems. In *ISCA*, July 2007.

[19] J. Lin, H. Zheng, Z. Zhu, E. Gorbatov, H. David, and Z. Zhang. Software Thermal Management of DRAM Memory for Multicore Systems. In *SIGMETRICS*, 2008.

[20] K. Ma, X. Li, M. Chen, and X. Wang. Scalable Power Control for Many-Core Architectures Running Multi-threaded Applications. In *ISCA*, 2011.

[21] J. M. Maciejowski. *Predictive Control with Constraints*. Prentice Hall, 2002.

[22] Micron. Data Sheet. http://download.micron.com/pdf/datasheets/modules/ddr2/HTF9C64_128x72F.pdf, 2007.

[23] Micron. Micron System Power Calculator. http://download.micron.com/downloads/misc/ddr2_power_calc_web.xls, 2007.

[24] Micron. Data Sheet. http://download.micron.com/pdf/datasheets/dram/ddr2/1GbDDR2.pdf, 2009.

[25] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu. No Power Struggle: Coordinated Multi-level Power Management for the Data Center. In *ASPLOS*, 2008.

[26] L. Shang, L.-S. Peh, and N. K. Jha. PowerHerd: A Distributed Scheme for Dynamically Satisfying Peak-Power Constraints in Interconnection Networks. *IEEE Transactions on Computer-Aided Design of Circuits and Systems*, 25(1), 2006.

[27] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware Microarchitecture: Modeling and Implementation. *ACM Transactions on Architecture and Code Optimization*, 1, Mar. 2004.

[28] A. Verma, G. Dasgupta, T. K. Nayak, P. De, and R. Kothari. Server Workload Analysis for Power Minimization using Consolidation. In *USENIX*, 2009.

[29] D. Wang, B. Ganesh, N. Tuaycharoen, K. Baynes, A. Jaleel, , and B. Jacob. DRAMsim: A Memory System Simulator. *SIGARCH Computer Architecture News*, 33, Nov. 2005.

[30] X. Wang, M. Chen, and X. Fu. MIMO Power Control for High-Density Servers in an Enclosure. *IEEE Transactions on Parallel and Distributed Systems*, 21(10), Oct. 2010.

[31] X. Wang, M. Chen, C. Lefurgy, and T. W. Keller. SHIP: Scalable Hierarchical Power Control for Large-Scale Data Centers. In *PACT*, 2009.

[32] Y. Wang, K. Ma, and X. Wang. Temperature-Constrained Power Control for Chip Multiprocessors with Online Model Estimation. In *ISCA*, 2009.

[33] Q. Wu, P. Juang, M. Martonosi, and D. W. Clark. Formal Online Methods for Voltage/Frequency Control in Multiple Clock Domain Microprocessors. In *ASPLOS*, 2004.

[34] H. Zheng, J. Lin, Z. Zhang, E. Gorbatov, H. David, and Z. Zhu. Mini-Rank: Adaptive DRAM Architecture for Improving Memory Power Efficiency. In *MICRO*, 2008.