A Novel Power management for CMP Systems in Data-intensive Environment

Pengju Shang
Computer Engineering and Computer Science Dept.
University of Central Florida
Orlando, USA.
Email: shang@eecs.ucf.edu

Jun Wang
Computer Engineering and Computer Science Dept.
University of Central Florida
Orlando, USA.
Email: jwang@eecs.ucf.edu

Abstract—The emerging data-intensive applications of today are comprised of non-uniform CPU and I/O intensive workloads, thus imposing a requirement to consider both CPU and I/O effects in the power management strategies. Only scaling down the processor’s frequency based on its busy/idle ratio cannot fully exploit opportunities of saving power. Our experiments show that besides the busy and idle status, each processor may also have I/O wait phases waiting for I/O operations to complete. During this period, the completion time is decided by the I/O subsystem rather than the CPU thus scaling the processor to a lower frequency will not affect the performance but save more power. In addition, the CPU’s reaction to the I/O operations may be significantly affected by several factors, such as I/O type (sync or unsync), instruction/job level parallelism; it cannot be accurately modeled via physics laws like mechanical or chemical systems.

In this paper, we propose a novel power management scheme called MAR (modeless, adaptive, rule-based) in multiprocessor systems to minimize the CPU power consumption under performance constraints. By using richer feedback factors, e.g. the I/O wait, MAR is able to accurately describe the relationships among core frequencies, performance and power consumption. We adopt a modeless control model to reduce the complexity of system modeling, MAR is designed for CMP (Chip Multi Processor) systems by employing multi-input/multi-output (MIMO) theory and per-core level DVFS (Dynamic Voltage and Frequency Scaling). Our extensive experiments on a physical test bed demonstrate that, for the SPEC benchmark and data-intensive (TPC-C) benchmark, the efficiency of MAR is 93.6 – 96.2% accurate to the ideal power saving strategy calculated off-line. Compared with baseline solutions, MAR could save 22.5 – 32.5% more power while keeping the comparable performance loss of about 1.8 – 2.9%. In addition, simulation results show the efficiency of our design for various CMP configurations.

Keywords-CPU; power management; data-intensive workloads; I/O wait;

I. INTRODUCTION

Multicore processors also known as CMP have become the mainstream in the current processor market because of tremendous increase in transistor density and advances in semi-conductor technology. At the same time, the limitations in ILP (Instruction Level Parallelism) coupled with the power dissipation restrictions encourage us to enter the “CMP” era for both high performance and power saving’s sake [1]. However many crucial application domains still have demand for single thread (core) performance growth [2]; and even without that, the increasing number of transistors on a single die or chip reveals super-linear growth in power consumption [3]. In this paper, performance is granted as the first priority: we try to minimize the processor’s power consumption while maintaining the required performance quality.

In recent years, many power management strategies [4]–[10] have been proposed for CMP processors based on DVFS (Dynamic Voltage and Frequency Scaling) [11]–[13]. Most of these works focus on compute-intensive or memory-intensive applications, hence they try to balance the power consumption and performance based on the CPU or memory boundness of the running workloads, or the busy/idle ratio of the CPU. This is reasonable for these non-I/O intensive applications, because the processes waiting for I/O to complete will be suspended by the operating system and the core will be given to other waiting threads [13] to avoid the I/O wait time. However, in the face of today’s emerging data-intensive applications, I/O wait time in processors is non-negligible [13], and could affect the performance of CPU’s power management methods, especially in the presence of many synchronized I/Os. During the CPU’s I/O waiting period, the performance will be decided by the I/O subsystem rather than the CPU, hence this is a chance to lower the CPU frequency and save more power without a performance penalty.

Previous works have some limitations either in maintaining the required performance or power saving efficiency. For example, [11], [12] focus on the non-I/O intensive cases, and their power management methods are based on CPU utilization, which is defined as $U = 100% - (\% of time spent in idle tasks)$; the latter part is calculated by idle time overall time. In addition, most of power management implementations are based on Linux operating systems [14], in which the “iowait” field is separated from the “busy” (the sum of user, sys, and nice fields) and “idle” fields. If we apply $U$ into data-intensive environment where the I/O operations are non-negligible, we could miss the I/O wait phases for deeper power saving. For example, assuming there are two data-intensive threads running on one CPU core as shown in Figure 1. Two threads have an overlapped I/O portion that would make the core enters its I/O wait phase. If we could scout both I/O wait (35%) phase and idle (10%) phase in a timely fashion, we can lower the CPU frequency to save power.
without sacrificing the performance. However, if we adopt the aforementioned $U = 100\% - \frac{idle\ time}{overall\ time} = 90\%$ in this case, the I/O wait phase is categorized as busy status and not used for power saving.

On the other hand, some works consider I/O factor in their power management solutions. [14] divides every workload into “on-chip” part and “off-chip” part; the I/O wait time is categorized into the “off-chip” part along with the idle time, both of which are irrelevant to CPU’s frequency. They calculate the workload characteristics as $k = \frac{\delta \cdot f_{onchip}}{\delta \cdot f_{onchip} + \delta \cdot f_{offchip}}$, where $\delta$ is the user specified performance loss and $f_{max}$ is CPU’s maximum frequency setting. Their solution works well when the workload is uniform, e.g., they can scale the lowest frequency when the workload is I/O-bound ($k = 0$) or scale the highest frequency when the workload is CPU bounded. However it cannot handle the data-intensive applications where the computation and I/O operations are non-uniformly distributed. Consider the same example in Figure 1, we assume it takes place in one sampling period, and $k = \frac{35\% + 25\%}{35\% + 25\% + 10\% + 30\%} = 55\%$. If no performance loss is allowed ($\delta = 0$), [14] picks the highest frequency. In fact, we could scale down a lower frequency $f_{new}$ to save more power without compromising performance.

The challenge is how to calculate $f_{new}$ since the relationships among frequency, performance, and power consumption is too complex to be modeled when I/O factor is taken into account.

In this paper, we first launch extensive experiments which show that in a CMP system: 1) scaling down the core’s frequency during its I/O wait time can provide more opportunities to save power without sacrificing performance; 2) core’s waiting time for I/O operations to complete is unpredictable, unmodel-able, and depends on several factors, such as I/O type (sync or unsync), process or application level parallelism; 3) there is no model we could find that accurately describes the relationship between the CPU’s frequency and overall performance when I/O wait time exists, because CPU frequency and I/O wait time are decoupled. As a result, power management solutions for data-intensive applications demand that: 1) considerations of each core’s I/O wait status and its working and idle statuses be made; 2) accurate quantification of each status (e.g., busy, idle, iowait) for accurate power-saving decisions; 3) precise description of the relationships among frequency, performance and power consumption when I/O wait factor is considered.

Then we propose an empirical rule-based power management strategy named MAR (modeless, adaptive, rule-based) for CMP systems. Our design can precisely control the performance of a CMP chip to the desired set point while saving as much power as possible at run-time. There are two primary contributions of this work.

- Comprehensive factors: while most existing control theory based works (close-loop controllers) only consider incomplete CPU statistics, MAR is designed strictly based on comprehensive experiments measuring the impacts of all the core’s working status (e.g. user, nice, sys, idle, iowait, irq and soft irq), and especially the I/O factor.
- Rule-based control: While most existing power saving works adopt model predictive control theories, MAR applies formal rule-based control theory [20] because the system (relationships among frequency, performance, and power) is too complex to be modeled when I/O wait factor is incorporated. In addition, the model-free nature of rule-based control method avoids the troublesome effort to develop accurate system models, and the risk of design errors caused by statistical inaccuracies or inappropriate approximations.

The rest of this paper is organized as follows. Section II discusses CMP behaviors to learn the relationships among frequency, power and performance. Section III describes the design of MAR. Section IV provides the implementation details and Section V presents extensive experiments measuring the impacts of all the core’s working status (e.g., user, nice, sys, idle, iowait, irq and soft irq), and especially the I/O factor.

II. TASK I: LEARNING THE CORE’S BEHAVIORS

In this section, we exploit the behaviors of each core in a CMP processor to learn the relationship among power consumption, performance, and frequency settings, as shown in Figure 4.

As widely shown in previous works, CPU power consumption and performance are both highly related to CPU frequency [14], [24]. The cubic relationship between power consumption and processor frequency, which is $CPU\ Power \propto f^3$ ($f$ is the core frequency), is well-documented and shown in Figure 4.

However, the relationship between performance and frequency is difficult to be modeled: the same frequency setting may results in different response time (rt) or execution time (et) for different types of applications. The performance is related to both processor’s frequency and the workload characteristics. On the other hand, the behavior of the CPU is able to illustrate the characteristics of the running workloads. More specifically, each core in a CMP has 7 working statuses [13], [22]:

- user: normal processes executing in user mode;
- nice: nice processes executing in user mode;
- system: processes executing in kernel mode;
- idle: idle times;
- iowait: waiting for I/O to complete;

<table>
<thead>
<tr>
<th>Thread1</th>
<th>Compute</th>
<th>I/O</th>
<th>Compute</th>
<th>Idle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread2</td>
<td>Compute</td>
<td>I/O</td>
<td>Compute</td>
<td>Idle</td>
</tr>
<tr>
<td>Core</td>
<td>Busy</td>
<td>I/O wait</td>
<td>Busy</td>
<td>Idle</td>
</tr>
<tr>
<td></td>
<td>30% On-chip</td>
<td>35% Off-chip</td>
<td>25% On-chip</td>
<td>10% Off-chip</td>
</tr>
</tbody>
</table>

Fig. 1. Two data-intensive threads are running on a core, the I/O wait phase and idle phase could be used for power saving.
The relationship between performance and frequency can be denoted as a composition of the running workload. As a result, a relationship between performance, power consumption and frequency is necessary to know the meanings of the core’s statuses which could describe the CMP’s behavior. So to confirm, without considering I/O wait latency, the basic B-I model works well for CPU-intensive and memory-intensive workloads is because of the well-developed techniques reducing the cache miss penalty. However, the huge speed gap between I/O devices and processors cannot be effectively eliminated because of the well-developed techniques to reduce cache miss penalty. On the other hand, for the I/O intensive or data-intensive workloads, e.g. I/O bomb and TPCC, B-I model which does not consider the I/O impact will result in up to a 45% error in the model. The reason why B-I works well for CPU-intensive and memory-intensive workloads is because of the well-developed techniques to reduce cache miss penalty.

### Table I

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>L1D miss</th>
<th>L2 miss</th>
<th>Mispredictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc</td>
<td>14.21</td>
<td>3.17</td>
<td>5.11</td>
</tr>
<tr>
<td>mcf</td>
<td>130.15</td>
<td>36.73</td>
<td>15.79</td>
</tr>
</tbody>
</table>

The durations of the core’s 7 statuses completely exhibit the composition of the running workload. As a result, a relationship between performance and frequency can be denoted as Equation (1).

\[
\text{Execution time} = F(\text{frequency, workload}) = F(\text{freq, core' smetrics}) = F(\text{freq, user, nice, sys, idle, iowait, irq, softirq}) \tag{1}
\]

We launch various applications on our test bed to learn the curve of Equation (1), e.g., I/O bomb from Isolation Benchmark Suite (IBS) [23], gcc and mcf benchmark from SPEC CPU 2006 suite version 1.0 [22]. TPCC running on PostgreSQL [23]. I/O bomb uses the IOzone benchmark tool to continuously read and write to the hard disk (by writing files larger than main memory which ensures that we are not just testing memory); mcf is the most memory bound benchmark in SPEC CPU2006; gcc is cpu-intensive, as shown in Table I. TPCC is a standard On-Line-Transaction-Processing (data-intensive) benchmark. The configuration details of these benchmarks can be found in Section IV. Our CPU for the experiment is the Quad-Core Intel Xeon E5345 2.27GHz processor, with 2 x 4MB L2 cache and 1.333MHz FSB. The supported frequencies are 800MHz, 1.6GHz, 2.27GHz.

#### A. Per-Core

Because we are using per-core level DVFS for power management, it is necessary to know the meanings of the 7 statuses for each core. We first enable only one core in the CMP processor and assign one process to run the benchmarks so that we can avoid the noise from task switches among cores. Figure 3 shows the overall execution time (et) of the 4 benchmarks at different frequency settings. B-I (busy-idle) model is a simple method [10], [11] to model the relationship between et and frequency settings. In this model, et(new) = et(old) · (percent(busy)new · f(new) + percent(idle)) · f(new), where percent(busy) is the CPU busy percentage and percent(idle) is the idle percentage; f and f(new) are the two different versions of CPU frequency settings. The predictions are based on the CMP behaviors when frequency is set as 800MHz.

Figure 3 illustrates the prediction results. It is surprisingly to find that for the first two workloads, e.g. gcc (CPU-intensive) and mcf (memory-intensive), B-I model is accurate enough with less than 3% deviation, which is different from some previous works’ [9], [10] results. We believe this is caused by different test bed and cache-miss-penalty-reducing techniques [47]. On the other hand, for the I/O intensive or data-intensive workloads, e.g. I/O bomb and TPCC, B-I model which does not consider the I/O impact will result in up to a 45% error in the model. The reason why B-I works well for CPU-intensive and memory-intensive workloads is because of the well-developed techniques reducing the cache miss penalty [47]. However, the huge speed gap between I/O devices and processors cannot be effectively eliminated [43], which leads to the B-I model’s prediction errors for I/O bomb and TPCC benchmarks.

We also show the statistics of the 7 statuses during the benchmarks’ running in Figure 3. For gcc and mcf, most of the execution time is in user mode; the cache miss and mispredictions of mcf have negligible impact on the CMP’s behavior due to the built techniques reducing the cache miss penalty. For I/O bomb, I/O wait is the main latency; for data-intensive benchmark TPCC, the lower frequency will hide some of the I/O wait latency, but the latencies in both user and iowait modes can not be ignored. For all four cases, the irq and softirq latency are negligible. As a result, “user+nice+sys”, “idle” and “iowait” are the three most important working statuses which could describe the CMP’s behavior. So to confirm, without considering I/O wait latency, the basic B-I...
and record the statistics for each core. In Figure 4, we show the traces for core0. We omit “irq” and “softirq” based on the results of section II-A, and we treat “user, nice, sys” as a group denoting the real “busy” status. When the frequency is 2.27GHz, all the workloads are processed in parallel in “phase1”, the I/O wait latency could be hidden by the processor-level parallelism. However in “phase2”, when there are little available processes to schedule, the I/O wait latency will emerge. After all work is complete, the core will stay idle in “phase3”. The traditional B-I based power management is only able to discover the chances in “phase3” and to save power by lowering the processor’s voltage and frequency at the phase. However in fact, “phase2” also provides opportunities to save more power: we can lower the frequency in order to parallelize the CPU and I/O works as much as possible. As shown in the lower part of Figure 5, we can use “800MHz” to finish all the workloads at roughly the same time while only consuming 4.4% power when compared to the case using 2.27GHz frequency.

We admit that a heavy disk utilization may not necessarily result in I/O wait if there are enough parallel CPU-consuming tasks. However, the new emerging data-intensive applications and applications lead to higher chances of having I/O latency [28]–[30], and high I/O wait is also common in other applications [15], [31]. As a result, I/O wait latency should be exploited in power-saving projects.

C. Analysis of I/O Wait

The iowait time is the duration of time when the processor is waiting for the I/O operation to complete, however we cannot simply consider it as the sub-category of CPU idle time. When there are only CPU idle and busy statuses, increasing the CPU frequency will linearly decrease execution time; however when taking into account the I/O wait time we have two new cases as shown in Figure 6. In case 1 where the CPU-consuming tasks and I/O tasks are synchronous (sequential) or blocking each other [32], the I/O wait time could be treated as idle time, hence we can use the traditional B-I method, as discussed in Section II-A, to model the relation between execution time and frequency. In case 2 where the two types of workloads are running in parallel but not well aligned, scaling CPU frequency will not affect the overall execution time

Based on our comprehensive experimental results, we find that the I/O wait ratio (iowait time/overall time) could be used to distinguish the two cases with an about 5% error rate (for our specific system configuration). We introduce two “thresholds” for I/O wait ratio: \( t_{\text{up}} \) and \( t_{\text{down}} \) to quantify the Equation 1 as the following Equation 2.

\[
\begin{align*}
\text{When scaling up freq, if } & \text{I/Owait} < t_{\text{up}}, \text{ or} \\
\text{when scaling down freq, if } & \text{I/Owait} < t_{\text{down}} : \\
\text{(Case 1): } & r_{\text{new}} = \frac{1}{\frac{1}{r_{\text{old}}} + (t_{\text{up}} - t_{\text{down}})P_{\text{core}}} \\
\text{Otherwise: } & (\text{Case 2): } r_{\text{new}} = r_{\text{old}}
\end{align*}
\]

\[ (2) \]

Since we are discussing cases where I/O wait time exists, the I/O part lasts longer than the CPU-consuming part; otherwise, there will be no I/O wait time.

---

1core0 and core1 are on the same die so they have to be scaled together, this also applies to core2 and core3. Hence there are 3 \times 3 = 9 different settings in total.

2Since we are discussing cases where I/O wait time exists, the I/O part lasts longer than the CPU-consuming part; otherwise, there will be no I/O wait time.
In this section, we introduce the design, analysis, optimizations of our rule-based power management.

A. MAR Control Model

In order to perform good power management, we have to know the relationships among CPU frequency, power consumption, and overall performance. However, as we mentioned in previous sections, after considering I/O wait factors, the relationship between the frequencies and overall performance is too complicated to be modeled (because the CPU frequency and I/O time is decoupled). Hence we adopt the formal rule-based, modeless, control system to manage the power consumption in CMP systems. The rules are derived from experimental results and could be self-tuned for different system configurations.

Fig. 6. Two cases when I/O wait time exists. “Core bounded” area represents the busy status.

Fig. 7. The overall architecture of MAR power management

MAR is designed as a MIMO controller shown in Figure 6. Let SP denote the sampling period. RRT represents the required response times and cb represents the realtime core boundness of the workloads (core’s busy ratio, $w_{cb}$).

Equation 2 can be used to complete Figure 6. Our rule-based power management controller MAR will be designed according to the two relationships in Figure 6.

III. TASK II: A MODELESS, ADAPTIVE, RULE-BASED (MAR) CONTROLLER DESIGN

In this section, we introduce the design, analysis, optimizations of our rule-based power management.

A. MAR Control Model

In order to perform good power management, we have to know the relationships among CPU frequency, power consumption, and overall performance. However, as we mentioned in previous sections, after considering I/O wait factors, the relationship between the frequencies and overall performance is too complicated to be modeled (because the CPU frequency and I/O time is decoupled). Hence we adopt the formal rule-based, modeless, control system to manage the power consumption in CMP systems. The rules are derived from experimental results and could be self-tuned for different system configurations.

Fig. 6. Two cases when I/O wait time exists. “Core bounded” area represents the busy status.

In this section, we introduce the design, analysis, optimizations of our rule-based power management.

A. MAR Control Model

In order to perform good power management, we have to know the relationships among CPU frequency, power consumption, and overall performance. However, as we mentioned in previous sections, after considering I/O wait factors, the relationship between the frequencies and overall performance is too complicated to be modeled (because the CPU frequency and I/O time is decoupled). Hence we adopt the formal rule-based, modeless, control system to manage the power consumption in CMP systems. The rules are derived from experimental results and could be self-tuned for different system configurations.

Fig. 6. Two cases when I/O wait time exists. “Core bounded” area represents the busy status.

In this section, we introduce the design, analysis, optimizations of our rule-based power management.

A. MAR Control Model

In order to perform good power management, we have to know the relationships among CPU frequency, power consumption, and overall performance. However, as we mentioned in previous sections, after considering I/O wait factors, the relationship between the frequencies and overall performance is too complicated to be modeled (because the CPU frequency and I/O time is decoupled). Hence we adopt the formal rule-based, modeless, control system to manage the power consumption in CMP systems. The rules are derived from experimental results and could be self-tuned for different system configurations.

Fig. 6. Two cases when I/O wait time exists. “Core bounded” area represents the busy status.

In this section, we introduce the design, analysis, optimizations of our rule-based power management.

A. MAR Control Model

In order to perform good power management, we have to know the relationships among CPU frequency, power consumption, and overall performance. However, as we mentioned in previous sections, after considering I/O wait factors, the relationship between the frequencies and overall performance is too complicated to be modeled (because the CPU frequency and I/O time is decoupled). Hence we adopt the formal rule-based, modeless, control system to manage the power consumption in CMP systems. The rules are derived from experimental results and could be self-tuned for different system configurations.

Fig. 6. Two cases when I/O wait time exists. “Core bounded” area represents the busy status.

In this section, we introduce the design, analysis, optimizations of our rule-based power management.

A. MAR Control Model

In order to perform good power management, we have to know the relationships among CPU frequency, power consumption, and overall performance. However, as we mentioned in previous sections, after considering I/O wait factors, the relationship between the frequencies and overall performance is too complicated to be modeled (because the CPU frequency and I/O time is decoupled). Hence we adopt the formal rule-based, modeless, control system to manage the power consumption in CMP systems. The rules are derived from experimental results and could be self-tuned for different system configurations.

Fig. 6. Two cases when I/O wait time exists. “Core bounded” area represents the busy status.

In this section, we introduce the design, analysis, optimizations of our rule-based power management.

A. MAR Control Model

In order to perform good power management, we have to know the relationships among CPU frequency, power consumption, and overall performance. However, as we mentioned in previous sections, after considering I/O wait factors, the relationship between the frequencies and overall performance is too complicated to be modeled (because the CPU frequency and I/O time is decoupled). Hence we adopt the formal rule-based, modeless, control system to manage the power consumption in CMP systems. The rules are derived from experimental results and could be self-tuned for different system configurations.

Fig. 6. Two cases when I/O wait time exists. “Core bounded” area represents the busy status.

In this section, we introduce the design, analysis, optimizations of our rule-based power management.

A. MAR Control Model

In order to perform good power management, we have to know the relationships among CPU frequency, power consumption, and overall performance. However, as we mentioned in previous sections, after considering I/O wait factors, the relationship between the frequencies and overall performance is too complicated to be modeled (because the CPU frequency and I/O time is decoupled). Hence we adopt the formal rule-based, modeless, control system to manage the power consumption in CMP systems. The rules are derived from experimental results and could be self-tuned for different system configurations.

Fig. 6. Two cases when I/O wait time exists. “Core bounded” area represents the busy status.

In this section, we introduce the design, analysis, optimizations of our rule-based power management.

A. MAR Control Model

In order to perform good power management, we have to know the relationships among CPU frequency, power consumption, and overall performance. However, as we mentioned in previous sections, after considering I/O wait factors, the relationship between the frequencies and overall performance is too complicated to be modeled (because the CPU frequency and I/O time is decoupled). Hence we adopt the formal rule-based, modeless, control system to manage the power consumption in CMP systems. The rules are derived from experimental results and could be self-tuned for different system configurations.

Fig. 6. Two cases when I/O wait time exists. “Core bounded” area represents the busy status.

In this section, we introduce the design, analysis, optimizations of our rule-based power management.

A. MAR Control Model

In order to perform good power management, we have to know the relationships among CPU frequency, power consumption, and overall performance. However, as we mentioned in previous sections, after considering I/O wait factors, the relationship between the frequencies and overall performance is too complicated to be modeled (because the CPU frequency and I/O time is decoupled). Hence we adopt the formal rule-based, modeless, control system to manage the power consumption in CMP systems. The rules are derived from experimental results and could be self-tuned for different system configurations.

Fig. 6. Two cases when I/O wait time exists. “Core bounded” area represents the busy status.

In this section, we introduce the design, analysis, optimizations of our rule-based power management.

A. MAR Control Model

In order to perform good power management, we have to know the relationships among CPU frequency, power consumption, and overall performance. However, as we mentioned in previous sections, after considering I/O wait factors, the relationship between the frequencies and overall performance is too complicated to be modeled (because the CPU frequency and I/O time is decoupled). Hence we adopt the formal rule-based, modeless, control system to manage the power consumption in CMP systems. The rules are derived from experimental results and could be self-tuned for different system configurations.

Fig. 6. Two cases when I/O wait time exists. “Core bounded” area represents the busy status.
Having only $\xi$ does not provide enough information to seize all the opportunities of saving power. These rules will guide MAR in finding the frequencies to be set in next $SP$. We denote $0 \leq \delta < 1$ as the user-specified performance-loss constraint.

1) $RTT \cdot (1 - \delta) \leq rt \leq RTT \cdot (1 + \delta)$: This is the ideal case from a performance perspective. Traditional solutions may not change the core’s frequency setting, however MAR will do a further check whether $w > th_{down}$.

- If so, the frequency can be scaled down to a lower level to save more power without affecting $rt$.
- If not, scaling the frequency will result in a different $rt$ which is deviated from $RRT$, so we keep using the current frequency.

2) $rt > RRT \cdot (1 + \delta)$: If the real response time does not meet the requirement, MAR checks whether $w > th_{up}$. And thus:

- If $w$ exceeds the scaling up threshold, changing to a higher frequency will not improve the performance. Moreover, higher frequencies will result in a higher $I/O$ wait, which is a waste of core resources. So as a result, MAR will keep the current frequency setting.
- If $w$ is within the threshold, $MAF = \xi \cdot f$, where $f$ is the current core frequency. A higher core frequency could improve $rt$ in this case. Based on Equation 3, we calculate the $f_{new}$ by using Equation 4:

$$f_{new} = \frac{cb}{cb + 1} \cdot f_{old}$$

3) $rt < RRT \cdot (1 - \delta)$: If the measured response time is unnecessarily better than the requirement, there is an additional chance to scale the frequency down to save more power. And thus:

- If $w > th_{down}$, MAR will only scale down the core frequency by one level. The reason for this “lazy” scaling is because it is difficult to know what $w$ will be when using lower frequencies. The new $w$ decides whether we should further scale down the frequency or not.
- If $w \leq th_{down}$, we may be able to scale down the core frequency to just meet the performance requirement while saving more power. MAR adopts aggressive scaling by using the same method shown in Equation 3.

We summarize the rules of MAR in Table 11.

C. Self-Tuning

There are several factors affecting the thresholds $th_{up}$ and $th_{down}$. For example: 1) Throughput of I/O devices. Higher I/O throughput means the same amount of data could be transferred in less “I/O wait” jiffies. 2) On-chip L1/L2 cache hit rates. The lower cache hit rate results in more memory accesses, which is much slower than cache access. Therefore, the overall processing speed of core bounded part (including both cache and memory accesses) becomes slower. 3) Noise in I/O wait time, such as network I/O traffic, file system journaling, paging/swapping, etc. 4) Heat and heat dissipation. When processors run too hot, they can experience errors, lock, freeze, or even burn up. It is difficult to predict the thresholds in this case, hence we adopt self-tuning methods based on the observed system behavior.

The process of self-tuning is shown in Figure 8.

![Figure 8: Self-tuning of I/O wait thresholds](https://example.com/self-tuning可以更好.jpg)

When $rt > RRT$ and $w \leq th_{up}$, rule #3 is used to scale up the core frequency to meet the performance requirement. However, if the $rt_{new}$ after the frequency scaling is same as the $rt$ in last $SP$, we need to lower $th_{up}$.

When $rt < RRT$, rule #1 or #4 is used to scale down the core frequency to save more power. If rule #4 is applied and the $rt_{new}$ does not change after the scaling, the $th_{down}$ should be adjusted to a lower level; if rule #1 is applied and the $rt_{new}$ changes, it means $w$ should be lower than $th_{down}$. Hence we scale $th_{down}$ to a higher level. In the design of MAR, we adopt “lazy” update scheme as shown in Equation 3 in our self-tuning method for the purpose of system stability. In Equation 3, “+” is used when updating the $th_{up}$ or $th_{down}$ to a higher level, “−” is used when updating the $th_{down}$ to a lower level.

$$new_{th} = old_{th} \pm \frac{w}{2}$$

IV. METHODOLOGY

In this section, we show our experimental methodology and benchmarks, as well as the implementation details of each component in our MAR controller. **Processor:** We use an Quad-Core Intel Xeon E5345 2.27GHz processor, with $2 \times 4MB$ L2 cache and 1.333MHz FSB. The four execution cores are in two sockets. Our experiments show that core0 and core1 are in one group and core 2 and core3 are in another group. We change the DVFS levels of the 2 cores in each group together in order to have a real impact on the processor power consumption.

For simplicity, we treat this Quad-Core as a Dual-core, the first group as core0 and the second group as core1. Each core in the processor supports 3 DVFS levels.

**Benchmarks:** We use the 3 stress tests highlighted earlier in the paper (CPU-bomb, I/O-bomb, and memory-bomb) from the Isolation Benchmark Suite (IBS) [23], SPEC CPU 2006.
When using root privilege, we can echo different frequencies cpufreq BIOS and use the as the response time metric. The result can be used /proc/stat period by multiplying the CPU time (first three fields in of instructions which have been processed in the sampling memory-intensive benchmarks have roughly linear relation-

A jiffy is $10^{-6}$ of a second.

The way to measure the real-time response time depends on the benchmarks. In the Isolation Benchmark, the response time can be monitored in I/O throughput. In TPCC, the primary metrics, transaction rate (tpmC), can be used as the response time. However, for the SPEC CPU2006 benchmarks, it is difficult to find any metrics to denote response time because there is no “throughput” concept here. Our previous experiments in Figure 3 show that these CPU-intensive and memory-intensive benchmarks have roughly linear relationships with core frequency. Hence we can calculate the number of instructions which have been processed in the sampling period by multiplying the CPU time (first three fields in /proc/stat file) by the core frequency. The result can be used as the response time metric.

**DVFS Interface:** We enable the Intel’s SpeedStep on the BIOS and use the cpufreq package to implement DVFS. When using root privilege, we can echo different frequencies into the system file /sys/devices/system/cpu/cpu[X]/cpufreq/scaling_setspeed, where [X] is the index of the core. We test the overhead of scaling CPU frequencies in our platform, which is only 0.12 milliseconds on average.

**Power Estimation:** We use the power models from Watch [23] to estimate the processor’s power consumption. Specifically, we first calculate the capacitance $C$ and then calculate the power. After that, we consider the leakage power by multiplying crossover_scaling (1.2), and all other L1 data/inst cache and L2 cache’s read miss/hit and write miss/hit power usage. Hence the power is denoted as $1.2kC \cdot frequency \cdot Voltage^2$ [24], where $k$ is the co-efficient related to the computation intensity of workloads, for example, the $k$ for computation bounded workloads is close to 1, while for I/O intensive workloads is close to 0 [25]. Since we are focusing on the comparison of power consumption among MAR and other baselines, all these common parts such as 1.2, $k$, and $C$ will be eliminated in the calculation. As a result, we simply use the cubic relation [23] to do the estimation: $Power \propto frequency^3$.

**Baselines:** The baselines are three previous CPU power saving works which do not incorporate I/O factors: Relax [13], PID [26], and GPHT [27]. Relax is a simple statistical predictor and used by [13] to assume the next sample behavior (core-boundness) is linearly related to previously monitored behaviors; we set the relaxation factor to 0.5 and the relax window size as 2 based on the empirical value taken from [13]. PID is used in [26] to predict the core’s busy/idle ratio based on previously records by using Proportional Integral Derivative control algorithm; we tune $K_p = 0.4$, $K_i = 0.2$, $K_d = 0.4$ based on [26]. GPHT (Global Phase History Table based) predictor observes the historical patterns of busy/idle ratios from previously observed samples to derive the next phase’s behavior. Previously learned patterns are recorded in a global table which is updated automatically. Based on [27], we set the GPHT depth (history window size) to 4 and the table entries (number of patterns to be recorded) to 512. All these algorithms are implemented in RTAI3.8 [28] to trace the cores’ behavior and predict the I/O wait in next SP.

## V. Experiments

First, we show that the model-based predictors used in the baselines are not suitable for predicting I/O wait ratio. Second, MAR is used to control the power for different types of workloads which including the CPU-intensive, memory-intensive and I/O-intensive benchmarks. The purpose is to show MAR’s performance under specific environments. Third, we compare the two versions of MAR (with/without considering I/O wait) by running the data-intensive benchmarks, in order to highlight the impact of I/O wait in power management schemes. After that, we compare the overall efficiency of MAR and the other baselines. And in end, we briefly show the overheads of the various investigated power management schemes.
A. Modeless-ness of I/O wait

In this section, we prove that the model-based predictors such as Relax, PID, and GPHT are not suitable for I/O wait trajectory learning.

First, we implement Relax, PID, and GPHT to predict the I/O wait ratio based on their predefined models. Figure 9 shows their trajectories of prediction compared with MAR. We also run the test 10 times to collect the “average detecting times”, as shown in the table in Figure 9. It can be seen that MAR has the fastest response time (as shown in the zoomed figure in Figure 9). GPHT could also detect the I/O wait bounce quickly is because of its aggressiveness: when there is no matched history pattern, GPHT assumes the next sample behavior is identical to the last one. But GPHT may result in some severe prediction errors like “A” and “B” spots in Figure 9 due to pattern changes, and GPHT has the highest overhead as shown in Section V-B4. The table in Figure 9 also shows that MAR achieves the shortest settling time after the deviation. The overall response time of MAR outperforms Relax, PID, and GPHT by 2.55, 3.49, and 1.87 times, respectively.

Second, we measure the impact of SP in the prediction accuracy. Figure 10 shows the average prediction errors for all four algorithms when SP = 5s, SP = 10s, and SP = 20s. When using a smaller SP, the trajectory of the core boundness is more unstable so all the predictors have higher average prediction errors; when using larger SP, the core’s behavior is more predictable due to the larger time window in each step. Slow responsive algorithms such as PID do not work well here since they are only good for the workloads with strong locality. GPHT could miss-predict because of few repetitive patterns in our experiment. In summary, MAR obtains the lowest occurrence of prediction errors for two reasons: 1) it incorporates the changing speed of the tracking error, which gives more hints for the coming trend and high responsiveness of the core’s status switches; 2) it adopts the noise filter \( \mu \) as shown in Equation 4 to reduce the unnecessary abrupt fluctuations caused by “phantom” bursty cases which is defined in Appendix A.

B. Power Efficiency

This set of experiments shows the power management efficiency of MAR for different types of benchmarks: gcc, mcf, bzip2, gap, applu, gzip and TPCC.

1) Running homogeneous workloads: In this section, we want to show MAR’s power control performance when homogeneous workloads are running. For each benchmark, we use 4 threads to run 4 copies on our test bed to evaluate the MAR’s performance for each specific type of workload. Here we show the results of power consumption/performance loss of MAR and the baselines: Relax, PID, GPHT and the Ideal case in Figure 11. In the “Ideal” case, we use the ideal DVFS settings which were calculated offline, to achieve the best power saving efficiency and the least performance loss.

Assuming the Ideal case saves the most power, MAR and the other baselines perform well when the workload has no explicit I/O operations. For gcc, mcf, bzip2, gap, applu and gzip, MAR achieves 95.4% efficiency, relative to ideal power management. The other baselines could also achieve similar
control accuracy, but when running the TPCC benchmark the baselines can only achieve a 58.2 – 70.1% efficiency in power saving performance, relative to the ideal case. In contrast, when considering I/O wait time as an additional opportunity to save power, MAR can still achieve 92.5% efficiency power management, relative to the ideal case. MAR outperforms the other baselines by 25.3 – 34.3%. The performance loss of all power management strategies is between 2% – 3%. And, although, MAR has the highest performance loss of 2.8% for the TPCC benchmark, because of our aggressive power saving strategy, it is still in the safe zone.

2) Running heterogeneous workloads: In this section we compare MAR with the other baselines for the case when heterogeneous workloads are running. We launch all aforementioned 7 benchmarks in parallel on our test bed. The database for the TPCC benchmark is locally set up. Figure 12 shows their overall DVFS results and power saving efficiency.

The upper two charts in Figure 12 show the frequency distribution of all management methods. Note that compared with \( SP = 5s \), the trajectory of workload in \( SP = 10s \) case has less fluctuations caused by the “phantom bursts”. The methods lack of considering I/O factors such as Relax, PID and GPHT could not discover as many power-saving opportunities as MAR, especially in the smaller \( SP \) case.

The lower two charts in Figure 12 illustrate the overall power consumptions of all management methods. All the numbers are normalized to MAR which saves the most power. PID and GPHT perform very differently when \( SP = 10s \) and \( SP = 5s \). The reason is that more “phantom bursts” of the workloads (when \( SP = 5s \)) could affect the control accuracy. From the power saving perspective, MAR, on average (\( SP = 10/5s \)), saves 30.6% more than Relax, 25.9% more than PID, 21.2% more than GPHT.

3) The impact of I/O wait latency: In order to highlight the impact of I/O wait latency in power management, we implement an incomplete version of MAR: MAR(-W). MAR(-W) uses the same controller as as MAR does but without considering any of the I/O factors. We use 7 threads to run gcc, mcf, bzip2, gap, applu, gzip, and TPCC in parallel. The comparison of MAR and MAR(-W) is shown in Figure 13.

The results show that MAR is more likely to use lower frequencies than MAR(-W). The reason is that when the I/O wait exceeds the thresholds in the control period, even if the response time is close to RRT, MAR still scales down the core frequency to a lower level to save more power. Compared with MAR, MAR(-W) cannot detect the potential I/O work which is overlapped with the computing intensive work. Based on the cubic relation between frequency and power consumption, when \( SP = 10s \), MAR could save 19.9% more power than MAR(-W); when \( SP = 5s \), MAR saves 31.13% more power. Therefore, MAR outperforms MAR(-W) by about 20 – 30%.

4) Overhead: Table IV shows the overhead of the tested methods. All of the methods are lightweight and consume less than 1% of the CPU’s entire utilization for the sampling period of 10s. The GPHT controller has the highest overhead because it is indexing expensive. In contrast, MAR executes almost 9 times faster than the GPHT controller.

### Table IV

<table>
<thead>
<tr>
<th>Code Size(lines)</th>
<th>MAR</th>
<th>Relax</th>
<th>PID</th>
<th>GPHT</th>
</tr>
</thead>
<tbody>
<tr>
<td>150</td>
<td>0.11%</td>
<td>0.05%</td>
<td>0.09%</td>
<td>0.97%</td>
</tr>
</tbody>
</table>

5) Scalability: In previous subsections we have tested MAR on our testbed, which only has 4 cores and 3 available voltage-frequency settings. In order to show the scalability of MAR, we use a cycle-accurate SESC simulator with modifications to support per-core level DVFS. Each core is configured as Alpha 21264. The processor technology
is 65nm, the L1 data-cache and instruction cache are set as 64K (2 way), and 2M private L2 cache is configured. We enable watchify and cactify to estimate the power change caused by DVFS scaling. In our simulation, we scale up MAR for 8, 16, 32 core processors each with private L1 and L2 hierarchy with cores placed in the middle of the die. Each core in our simulation has 3 DVFS levels (3.88GHz, 4.5GHz and 5GHz). The overhead of each DVFS scaling is set to 20%. The benchmarks we used are randomly selected SPEC 2006 benchmarks; gcc, mcf, bzip2, and the data-intensive TPCC benchmark. The number of processes are equal to the number for cores, e.g., we run 2 copies of each of the 4 benchmarks when there is 8 cores. We first record the maximum power consumption and the best performance of the workloads by setting all the cores to the highest DVFS level. Then we normalize the results of MAR and other baselines to show their power management efficiency and performance losses.

Figure 14 plots the average power saving efficiency and the performance loss of MAR, Relax, PID, and GPHT based per-core level DVFS controllers. All the numbers are normalized to the “performance-oriented” case. With varying numbers of cores the CMP processor with MAR continually saves the most power. On average, MAR outperforms Relax, PID, and GPHT by 31.4, 32.1% and 21.3% respectively under our benchmark configuration. Also, MAR’s and the other baseline’s performance losses are all between 2%-3%, which confirms our observations from our test bed. Our simulation results demonstrate that MAR can precisely and stably control power while achieving good performance for CMPs with varying numbers of cores.

VI. RELATED WORK

In recent years, various power management strategies have been proposed for CMP systems. From the perspective of DVFS level, previous power management schemes could be divided into two categories: chip-level and core-level power managements.

Chip-level power management uses chip-wide DVFS. In chip-level management, the voltage and frequency of all cores are scaled to the same level during program execution by taking advantage of the application phase change. These techniques extensively benefit from application “phase” information that can pinpoint execution regions with different characteristics. Based on the information obtained, calculate the Mem/Uop value which is quantified memory boundness of that task. They define several CPU frequency phases in which every phase is assigned to a fixed range of Mem/Uop. However, these task-oriented power management schemes do not take the advantage from per-core level DVFS.

Core-level power management means managing the power consumption of a core. and collect performance-related information by on-core performance monitoring counter (PMC) hardware. There are several limitations by using PMCs: Each CPU has a different set of available performance counters, usually with different names. Even different models in the same processor family can differ substantially in the specific performance counters available; modern superscalar processors schedule and execute multiple instructions at one time. These “in-flight” instructions can retire at any time, depending on memory access, hits in cache, stalls in the pipeline and many other factors. This can cause performance counter events to be attributed to the wrong instructions, making precise performance analysis difficult or impossible. The metrics used in MAR are simply read from the system monitoring file at run time.

Several recently proposed algorithms are based on open-loop search or optimization strategies, assuming the power consumption of a CMP at different DVFS levels can be estimated accurately. This assumption may result in severe performance degradation or even power constraint violation when the workloads vary significantly from the one they used to do estimation. There are also some closed-loop solutions based on feedback control theory. The key challenge for these
feedback control of power and performance is modeling, but the relationships among frequency, performance, and power consumption are too complex to be accurately modeled when running I/O-intensive workloads.

Some works [3], [5], [1] focus on non-I/O-intensive workloads so they consider the memory-boundness as the scaling reference. Their solution ignored the CPU’s I/O wait time which should not be ignored in data-intensive environment. Other works [13], [27] incorporate CPU’s I/O factor into their power management solutions. They divide every workload into “on-chip” and “off-chip” parts, in which the later one is irrelevant to CPU’s frequency, and I/O wait time is categorized into the “off-chip” part along with the idle time. However, without considering the application level parallelism, they simply quantify the CPU’s I/O wait latency as the application’s required I/O time. This cannot be applied when the I/O time of one application is hidden by parallelizing other CPU-consuming applications or other “on-chip” processes.

Some recently proposed power managements use model predictive controllers, such as MPC, PID control models. These works are working on different levels (cluster, large scale data center, CMP), such as DEUCON [13], [2] and [17]. They make an assumption that the actual execution times of real-time tasks are equal to their estimated execution times, and their online-predictive model will cause significant error in spiky cases due to slow-settling from deviation. Moreover, their control architecture allows degraded performance since they do not include the performance metrics into the feedback. [13] tries to satisfy QoS-critical systems but their assumption is maintaining the same CPU utilization guarantees the same performance. It is not true for the CPU unrelated works, such as the data-intensive or I/O-intensive workloads.

Rule-based control theory [15] is widely used in machine control [25], [16], and it has the advantage that the solution to the problem can be cast in terms that human operators can understand, so that their experience can be used in the design of the controller. It also reduces the development time/cycle, simplifies design complexity as well as implementation, and improves control performance [13].

VII. Conclusion

Power control for multi-core systems has become increasingly important and challenging. However, existing power control solutions can not be directly applied onto CMP systems because of the new data-intensive applications and complicated job scheduling strategies in CMP systems. In this paper, we presented MAR, a modeless, adaptive, rule-based power management scheme in multi-core systems to manage the power consumption while maintain the required performance. “Modeless” reduces the complexity of system modeling as well as the risk of design errors caused by statistical inaccuracies or inappropriate approximations. “Adaptive” allows MAR to adjust the control methods based on the real-time system behaviors. The rules in MAR are derived from experimental observations and operators’ experience, which provide a more accurate and practical way to describe the system behaviors.

“Rule-based” architecture also reduces the design development cycle and control overhead, simplifies design complexity. MAR controller is highly responsive (including short detective time and settling time) to the workload bouncing by incorporating more comprehensive control references (e.g., changing speed, I/O wait). Noise filters are used in MAR to reduce the unnecessary abrupt fluctuations caused by “phantom” bursty cases. Empirical results on a physical testbed show that our control solution can provide precise power control, as well as high power efficiency for optimized system performance compared to four existing solutions. Based on our comprehensive experiments, MAR could outperform the baseline methods by 22.5 – 32.5% in power saving efficiency, and maintains comparable performance loss about 1.8% - 2.9%.

ACKNOWLEDGMENT

The authors would like to express their sincere thanks to the reviewers for their insightful comments, and to Mr. Christopher Mitchell and Mr. Grant Mackey for proofreading. This work is supported in part by the US National Science Foundation grant CCF-0811413, US Department of Energy Early Career Principal Investigator Award DE-FG02-07ER25747, and National Science Foundation Early Career Award 0953946.

REFERENCES


APPENDIX A

Calculating $\Delta ecb$:

It should be noted that the fast responsiveness of MAR may overreact to some small fluctuations, which will not result in real switches. We define them as “phantom”. Therefore, a filter $\mu$ is introduced for robustness purpose: if $|ecb| < \mu$, the burst is phantom which should be ignored, which means the changing speed of tracking error is 0. $\mu$ is derived from our experimental experiences and could be automatically tuned during the runtime. The calculation of $\Delta ecb$ is shown in Equation (3):

$$\Delta ecb = ecb_{i} - ecb_{i-1} \geq \mu$$

$$\Delta ecb = ecb_{i} < \mu$$