Dynamic Voltage and Frequency Scaling for Shared Resources in Multicore Processor Designs


*Department of Electrical and Computer Engineering, Texas A&M University
**Strategic CAD Labs, Intel Corporation

ABSTRACT

As the core count in processor chips grows, so do the on-die, shared resources such as on-chip communication fabric and shared cache, which are of paramount importance for chip performance and power. This paper presents a method for dynamic voltage/frequency scaling of networks-on-chip and last level caches in multicore processor designs, where the shared resources form a single voltage/frequency domain. Several new techniques for monitoring and control are developed, and validated through full system simulations on the PARSEC benchmarks. These techniques reduce energy-delay product by 56% compared to a state-of-the-art prior work.

1. INTRODUCTION

Modern multicore processor designs face challenges across multiple fronts, including: the communication bottleneck, off-chip and on-chip, and the approaching fundamental limits of chip power density. Together these constraints conspire to reinforce one another. Recent designs have resorted to increasing cache size to circumvent the off-chip memory bottleneck. A large cache in turn demands an increase of on-chip communication bandwidth. Indeed, on-chip communication fabrics and shared, last-level caches (LLCs) have grown to occupy a large portion of the overall die area, as much as 30% of chip area in recent Intel chip multiprocessors [10]. Such growth inevitably exacerbates the power challenge.

Networks-on-Chip (NOC) are recognized as a scalable approach to addressing the increasing demand for on-chip communication bandwidth. One study shows that NOCs can achieve 82% energy savings compared to conventional bus design in a 16-core system [9]. Nonetheless, the NOC still accounts for a considerable portion of total chip power, e.g., 36% in MIT RAW architecture [19]. When the workload is small, some cores can be shut down to save leakage power. However, NOC and LLC need to stay active for serving the small workload and therefore their power proportion is even greater. The power-efficiency of NOC and LLC can be improved by Dynamic Voltage and Frequency Scaling (DVFS), with the rationale that power should be provided based on dynamic need instead of a constant level. DVFS has been intensively studied for individual microprocessor cores as well as the NOC [17, 12, 18, 15, 6, 13, 16, 3, 4] 1. Much of this prior work assumes a core-centric voltage/frequency (V/F) domain partitioning. The shared resources (NOC and/or LLC) are then divided and allocated to the core-based partitions according to physical proximity. While such configuration allows a large freedom of V/F tunings, the inter-domain interfacing overhead can be quite large. Furthermore, as these shared resources are utilized as a whole, with cache line interleaving homogenizing traffic and cache slice occupancy, per-slice V/F tunings makes little sense.

This work focuses on a realistic scenario where the entire NOC and LLC belong to a single V/F domain. As such, the interfacing overhead can be largely prevented and there is a coherent policy covering the whole of these shared resources. To the best of our knowledge, only two works [12, 4] have addressed DVFS for such scenario. Liang and Jantsch propose a rule-based control scheme, using network load as the measured system performance metric [12]. Chen et al. examine the motivation and advantages of the single shared V/F domain in detail [4]. They use a PI controller based on AMAT (Average Memory Access Time) and a low-overhead AMAT monitoring technique is proposed. Although both works demonstrate the benefit of DVFS, there are two critical hurdles that have not been well solved. First, the impact of the NOC/LLC V/F level on the chip energy-performance tradeoff is not straightforward. These prior works shy away from this problem by evaluating only parts of the chip system. Second, the chip energy-performance tradeoff is dynamic at runtime while the controls of these prior approaches are based on fixed reference points.

In this paper, we present remarkable progress on overcoming these hurdles. Three new methods are proposed and investigated. First, a throughput driven controller with dynamic reference point is examined. Second is a model assisted PI controller based on a new metric that bridges the gap between the NOC/LLC V/F level and the chip energy-performance tradeoff. The last one is a PI controller with a dynamic reference point based on the new metric. These methods are evaluated in full system simulation on the PARSEC benchmarks [1]. The experimental results show that our techniques can reduce NOC/LLC energy by ~ 80% and ~ 50% compared to a baseline fixed V/F level (no power management) and the state-of-the-art prior work [4], respectively. Simultaneously, we achieve our target of ≤ 5% performance loss. Compared to the competing design [4], the

1Although supply voltage change may affect SRAM read/write stability, Kirolos and Massoud show that DVFS for SRAM is feasible [8].
energy-delay product is reduced by 56%.

2. RELATED WORK

Shang et al. present a pioneering work on dynamic voltage scaling in NOCs [17]. They tune voltage levels for individual links separately according to their utilization history. Mishra et al. propose DVFS techniques for NOC routers [13]. They monitor input queue occupancy of a router, based on which the upstream router changes its V/F level. Son et al. perform DVFS on both CPUs and network links [18]. They target to parallel linear system solving and the V/F levels are decided according to task criticality. Guan et al. propose a voltage island based approach [6], where router queue occupancies are monitored and island V/F levels are tuned accordingly. Rahimi et al. take a similar rule-based approach according to link utilization and queue occupancy [16]. Ogras et al. propose a formal space-state control approach also for voltage island based designs [15]. Bogdan et al. introduce an optimal control method using a fractional state model [3]. In drowsy caches, dynamic voltage scaling is applied at certain cache lines at a time for reducing leakage power [5]. To the best of our knowledge, there is no published work on DVFS which focuses on shared caches in multicore chips.

Liang and Jantsch present a DVFS controller that attempts to maintain the network load near its saturation point, where the load is the number of flits in the network [12]. At each control interval, its policy is to increase (decrease) network V/F level by one step if the network load is significantly greater (less) than the saturation point. This method neglects the fact that chip performance also depends on the distribution besides the amount of network load. A non-uniform distribution may imply certain congestion hotspots, which may significantly degrade chip performance. Even with consideration of the distribution, network load does not always matter. For example, many store operations induce large network load but they are not critical to the overall chip performance. Liang and Jantsch’s method may respond slowly for bursty traffic as only one step V/F change is allowed in each control interval.

Chen et al. introduce a PI controller based on AMAT [4]. AMAT, as they formulate it, including the effects of the private caches, NOC, LLC and off-chip memory, reflects network load and contention inherently, providng an approximation of the latency seen by typical core memory references. Therefore, it captures a more global system effects than a purely network-based metric such as Liang and Jantsch’s approach [12]. Their AMAT metric, however, does not truly separate out the LLC and NOC utility to the core from the effects of the off-chip memory. Thus, applications which frequently miss in the LLC, causing off-chip memory accesses, will lead to high AMAT values, and thus high LLC and NOC frequencies, despite the LLC utility being low in this case. Chen et al.’s DVFS policy uses a PI controller, subsuming that of Liang and Jantsch. This work also describes implementation techniques on how to monitor AMAT of multicore with low overhead.

Both Liang and Jantsch’s [12] and Chen et al.’s [4] approaches suffer from another weakness, they have no systematic approach to decide the reference point for their controllers. In Chen et al.’s work, the reference point is decided empirically according to offline simulations. However, it is very difficult, if not impossible, for a fixed reference point to be appropriate for different kinds of applications. Moreover, neither work provides performance results from full-system simulation to validate their approach.

3. PRELIMINARIES

3.1 Problem Description

We consider a common case in multicore processor design where the entire chip is composed of an array of tiles. Each tile contains a processor core and private caches. The communication fabric is a 2D mesh NOC with one router residing in each tile. There is a shared LLC partitioned into slices and distributed uniformly among tiles. The NOC and the LLC together are referred to as the uncore system in this paper. The system is illustrated in Figure 1 where NI denotes Network Interface with cores.

![Figure 1: A multicore processor design where the uncore (NOC+LLC) forms a single V/F domain.](image)

The problem we attempt to solve is formulated as follows.

Uncore Dynamic Voltage and Frequency Scaling: within a set time window, find the voltage/frequency level for the uncore such that the uncore energy dissipation is minimized while the chip performance, in terms of total application runtime, has negligible or user-specified degradation.

Please note that our problem formulation has a key difference from previous works on NOC DVFS, which try to optimize the performance of NOC itself, not considering directly its utility to the system. In contrast, we have the more challenging goal of optimizing uncore energy under the constraint of entire system performance. We present the first work we are aware of in this area with such formulation. The uncore energy we try to minimize includes both dynamic and leakage energy and their models are well-known.

3.2 Options for DVFS Policy

Broadly speaking, there are two categories of approaches for DVFS: open-loop control and closed-loop control. Open-loop control decides control variables based on the current system state and a system model obtained either theoretically or through machine learning. The behavior of a multicore system is typically very complex. Even if a decent model is available, its behavior depends on environmental parameters that are highly dynamic and thus are very difficult to reliably predict.

Closed-loop control adjusts control variables with consideration of observed output. It includes several options: rule-based, PID (Proportional-Integral-Differential) control, state-space model-based control and optimal control. In a rule-based approach, a set of ad hoc rules are determined, such as simply increase (decrease) uncore V/F level if the network performance is poor (good) according to given metric. Based on the observed output error, PID control adjusts the system to track a target output. State-space model-based control formally synthesizes a control policy that guarantees system stability [15]. Optimal control [3] decides
system operations by solving an optimization problem and sometimes can be applied with a state-space model.

We adopt PID control due to its simplicity, flexibility, low implementation overhead and guaranteed stability. In a discrete-time system, where the output is a time-varying function $y_i$ and the control variable is $u_i$ for control interval $i$, the error function is defined by $e_i = y_{ref} - y_i$, where $y_{ref}$ is the reference point or target output. Since differential control is sensitive to data noise, we drop the differential term. A PI controller can be described by:

$$u_i = u_{i-1} + K_I \cdot e_i + K_P \cdot (e_i - e_{i-1})$$  \hspace{1cm} (1)

where $K_P$ and $K_I$ are constant parameters. Please note that the problem described in Section 3.1 is an optimization problem, which implies a dynamic goal instead of a steady target in typical PI controls. To bridge this gap, it is very important to use a dynamic reference point, as shown in subsequent sections, instead of fixed reference as in [4].

4. DESIGN DESCRIPTION

4.1 Throughput-Driven DVFS

One technique we propose is to use a throughput metric such that a naturally dynamic reference is enabled for the DVFS PI control.

Throughput is the amount of data processed by the uncore per unit time. It can be measured by $R_{U,Out}$, which is the rate of data flowing out of the uncore to cores. Injection rate $R_{U,In}$, on the other hand, is defined as the cores’ requested data rate. Figure 2 depicts the throughputs for various injection rates with respect to varying uncore frequency. As the uncore frequency increases, we observe that the throughput increases asymptotically to the given injection rate. In a transient period, e.g., control interval $i$, if the uncore frequency is not high enough, the throughput $R_{U,Out,i}$ can be different from data injection rate $R_{U,In,i}$ of the same interval. We have then a PI controller with state (output) variable $R_{U,Out}$ and reference point $R_{U,In}$, which is dynamically decided during operations. Intuitively, such controller ensures that $R_{U,Out}$ becomes $R_{U,In}$ and it possibly saves energy by setting the uncore V/F to a lower level than the maximum level.

![Figure 2: Throughput vs. uncore frequency.](image)

Finding the optimal uncore V/F level here, however, is not trivial when the uncore V/F level is higher than necessary. For example, in Figure 2, where the given injection rate is 0.4 and the current normalized uncore frequency is 0.9, the controller should reduce the frequency to 0.4 to save energy. In such a case, unfortunately, the error $(R_{U,In} - R_{U,Out})$ becomes zero and the PI controller will not change the uncore V/F level. To avoid this, we magnify $R_{U,Out}$ by a very small percentage $\delta$, e.g., 1%, resulting in a slightly negative value of the error. By doing so, the controller moves the V/F level down to the saturation point for the given injection rate without significantly hurting the throughput.

The proposed controller requires care to evaluate the injection rate $R_{U,In}$, especially when the program phase changes to increase $R_{U,In}$. Typically, a core stalls after issuing a number of requests yet to be served. The maximum value of outstanding requests is determined by the capacity of its MSHR (Miss Status Handling Register) for out-of-order processors, or it is 1 in case of in-order processors. At this point, the core has to wait till a request is served by the uncore, and then it will resume injecting further requests. If there are a sizable number of such stalled cores, the overall injection rate does not grow enough to increase the uncore frequency, although each core actually starts generating more requests than before. To address this issue, the number of stalled cores and the duration of waiting are considered. If the number of cycles when there are more than $N$ cores waiting their uncore requests is $M$, and the size of control interval is $P$, the effective injection rate, which is also the reference request rate for the controller, is estimated by $R_{ref,i} = R_{U,In,i} \cdot (1 + \frac{M}{P})$ where $i$ is the index of control interval. Then, the overall error function for the throughput-driven controller is:

$$e_i = R_{U,In,i} \cdot (1 + \frac{M}{P}) - R_{U,Out,i} \cdot (1 + \delta)$$ \hspace{1cm} (2)

where $\delta$ is a small number, e.g., 1%. One can use a greater value of $N$ to achieve more energy savings but at the higher performance cost. The value of $N$ is set to 3 in our experiment for the best energy-performance trade off.

4.2 Latency-Based DVFS

4.2.1 Composite Uncore Metric – Critical Latency

The throughput-driven DVFS described in Section 4.1 largely solves the dynamic reference problem. Although the metric used by the technique captures the performance of the uncore well, it is still oblivious of the overall chip performance. Ideally, the metric should reflect both the uncore performance and its criticality to the overall chip performance. We therefore define a new metric - critical latency, expressed by

$$\Gamma = \eta \cdot \lambda_U$$ \hspace{1cm} (3)

where $\eta$ is the criticality factor and $\lambda_U$ is the uncore latency.

The uncore latency should account for the latency in the network and LLC. Subtracting the reply return from request inject time, one can obtain the packet latency; however, this latency may contain off-chip memory latency in the event of an LLC miss. This memory latency should be excluded from consideration since it is not affected by the uncore DVFS. In certain cases, the LLC miss can be very high and the overall data latency is dominated by the memory latency. In this case, increasing uncore V/F does not help to improve chip performance while causes more power dissipation. The uncore latency can be described by

$$\lambda_U = \frac{(\sum_{j=1}^{N_{packets}} \lambda_{packet,j}) - \lambda_{Mem} \cdot N_{LLC\_Misses}}{N_{packets}}$$ \hspace{1cm} (4)

2We intentionally do not differentiate between packets which lead to coherence traffic and those which do not. Coherence related traffic can increase LLC latency and is sensitive to uncore V/F state.
where $\lambda_{\text{packet}, j}$ is the total round-trip latency for packet $j$, $\lambda_{\text{ref}}$ is the memory access latency, $N_{\text{LLC Misses}}$ is the number of LLC misses in a control interval and $N_{\text{packets}}$ is the number of packets in the same interval.

In microprocessors, both store and load data induce network traffic and potentially LLC access. These two types of packet requests have different impact on the overall chip performance. Often, a long latency load can block the execution of instructions that need the data, and therefore is performance critical. In contrast, a store operation can often run in parallel with subsequent instructions and is rarely critical. Thus, the criticality factor of uncore performance includes $\text{Loads} \cdot \text{Fraction}$, which is the number of load instructions per cycle. We scale $\text{Loads} \cdot \text{Fraction}$ by the L1 miss rate, assuming L1 is the only level of private cache, because loads which hit in the private caches never enter the uncore and are not affected by uncore performance. Therefore, we have

$$\eta = L1\text{Miss} \times \text{Loads} \cdot \text{Fraction} \quad (5)$$

### 4.2.2 Model Assisted PI Controller

The error function in the critical latency-based PI controller is

$$e_i = \Gamma_{\text{ref}} - \Gamma_i \quad (6)$$

where $\Gamma_i$ is the monitored critical latency for control interval $i$ and $\Gamma_{\text{ref}}$ is the reference point for the PI controller. Although $\Gamma$ should be correlated with the overall chip performance, its target value is not obvious. Of course, one may select one empirically from offline simulations. However, the offline testcases might not behave the same as online cases.

To address this problem, we propose a model assisted PI control method. From Equation (3)-(5), we can see that $\Gamma$ is approximately a linear function of $\lambda_{\text{packet}}$, which is in turn proportional to the uncore clock period $T_u$. Hence, we have

$$\Gamma = \alpha \cdot T_u + \beta \quad (7)$$

where $\alpha$ and $\beta$ are coefficients independent of $T_u$. Within each program phase, the program execution behaviors are generally consistent so that the variations of $\alpha$ and $\beta$ are often limited. Based on $(T_{i+1}, \Gamma_i)$ of current interval and $(T_{i-1}, \Gamma_{i-1})$ of the previous interval, we can estimate the values for $\alpha$ and $\beta$. Then, we can predict the $\Gamma(T_u)$ function of the next control interval.

This prediction can guide reference V/F toward a more aggressive or more conservative level. We first find three different reference points empirically, one is normal, another one is aggressive and the other one is conservative. Then, we dynamically choose among them at runtime according to the uncore frequency $\frac{1}{T_{i+1}}$ computed from the predicted $\Gamma(T_u)$ function. By default, the normal reference point is employed. If the model based frequency is significantly higher (lower), the reference point is changed to the aggressive (conservative) one. If there is no frequency change in two consecutive control intervals, we cannot obtain an update on $\alpha$ and $\beta$ values. In this case, we continue to use the reference of the previous control interval.

### 4.2.3 Latency-Based PI Control with Dynamic Reference

The model-assisted controller is conceptually superior to that proposed by Chen et al [4], where one fixed reference point is used. The improvement, however, can be limited as the model-based prediction may be inaccurate. To fix this problem, we examine an alternate PI controller that allows a truly dynamic reference point. For the uncore latency $\lambda_{U}$, defined in Equation (4), we can determine our desired target. The $\lambda_{U}$ mainly consists of the propagation latency in the network, the serialization latency, the queuing latency in the network and the LLC access latency. When there is no congestion, the queuing latency should approach zero. Ideally, we want to keep the network lightly loaded. Thus, we can define a reference uncore latency as

$$\lambda_{\text{ref}} = (1 + \rho) \cdot (2 \cdot (\lambda_{\text{hop}} \cdot N_{\text{hops}} + L_{\text{packet}}) + \lambda_{\text{LLC}}) \quad (8)$$

where $\rho$ is a constant selected to be 0.1, $\lambda_{\text{hop}}$ is the propagation (without queuing) latency per hop, $N_{\text{hops}}$ is the average number of hops for packets in a uniformly random traffic, $L_{\text{packet}}$ is the average packet length and $\lambda_{\text{LLC}}$ is the LLC access latency. The packet length $L_{\text{packet}}$ is in terms of flits and to account for the serialization latency. The coefficient 2 in Equation (8) is to cover the round-trip. The underlying reason for including the $\rho$ is the same as having the $\delta$ in Equation (2). The $\delta$ allows the throughput to be slightly lower than the injection rate, i.e., very limited network congestion. Likewise, the $\rho$ allows the queuing latency to be slightly above zero, which is also equivalent to a very small network congestion. Please note that $\lambda_{\text{hop}}$ is a constant and can be pre-characterized offline. Overall, the reference for the critical latency becomes:

$$\Gamma_{\text{ref}, i} = \eta_i \cdot \lambda_{\text{ref}} \quad (9)$$

As the criticality factor $\eta_i$ varies from interval to interval, this reference is dynamic with respect to packets at runtime.

### 4.3 Stability of PI Control with Dynamic Reference

Whether the reference is static or dynamic does not affect the stability analysis for a PI control. Thus, the stability analysis proposed by Chen et al. [4] can be directly applied here to guarantee the stability of our PI controllers.

### 4.4 Design Implementation

The implementation of the proposed DVFS methods mainly include: (1) information collection at each tile; (2) a central controller that aggregates the collected information and performs control policy computation; (3) information transportation from tiles to the central controller. For the critical latency-based DVFS controls, several registers are required to save relevant information at each tile. The bit-width of each register is decided by the range of the data to be saved. Here we show a design for our experiment setting. We use three 16-bit registers to save the numbers of load instructions, private cache hits and private cache misses, respectively. Additionally, there is a 20-bit register for accumulating the total request latency. Another 12-bit register is used to count the number of LLC misses. A 16-bit register is needed to save the uncore request count. Last, there is a 16-bit counter to keep track of control interval. Overall, 112 bits of registers are required for each tile.

In modern multicore processor designs, e.g., Intel's Nehalem architecture [11], there exists a Power Control Unit (PCU), which is a small processor dedicated to chip power management. Thus, the control of our DVFS can be implemented using PCU without additional hardware. The PCU retains a lookup table with each entry containing the data for a given tile. The PCU also needs to keep the reference points and parameters for the PI control. Typically the PCU has plenty of storage space for these needs. At the end of each control interval, the PCU computes the critical
Table 1: Simulation setup.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core Frequency</td>
<td>1GHz</td>
</tr>
<tr>
<td>#processing cores</td>
<td>16</td>
</tr>
<tr>
<td>L1 data cache</td>
<td>2-way 256Kb, 2 core cycle latency</td>
</tr>
<tr>
<td>L2 cache (LLC)</td>
<td>16-way, 2MB/bank, 32MB/total, 10 core cycle latency</td>
</tr>
<tr>
<td>Directory cache</td>
<td>MESI, 4 uncore cycle latency</td>
</tr>
<tr>
<td>NoC</td>
<td>4 x 4 2D mesh, X-Y DOR, 4 fits depth/VC</td>
</tr>
<tr>
<td>Voltage/Frequency</td>
<td>10 levels, voltage: 1V-2V, frequency: 250MHz-1GHz</td>
</tr>
<tr>
<td>V/F transition</td>
<td>100 core cycles per step</td>
</tr>
</tbody>
</table>

5. EVALUATION

5.1 Experiment Setup

The baseline architecture in our experiments is a 16-tile chip multiprocessor with a 2-level cache hierarchy similar to the Sun SPARC T3 processor. Here, each tile is composed of a processing core with 1-level of private cache, a network interface, an NOC router and a partition of the shared L2 cache (LLC) and directory. Each core is an in-order Alpha ISA processor. Table 1 summarizes our experimental configurations and parameters.

For the transmission of status information from the cores to the PCU, we use a method similar to that proposed by Chen et al. [4]. When a packet is sent out from a tile, the 96 bits information (by excluding the control interval counter) is scaled to 64 bits and is embedded in the header flit. If the packet reaches or passes by the tile where the PCU resides, the data is scaled back to 96 bits and downloaded to the lookup table. Within each control interval, later data from a tile overwrites the old data from the same interval. Since we do not use additional network or send dedicated packets, the data transportation overhead is fairly low. Chen et al. [4] showed that a single monitor tile can obtain sufficient sample data in a control interval of 50K clock cycles.

For the throughput-driven DVFS, the registers are required only at the PCU to count the number of requests issued (14 bits) and the number of requests served (14 bits). These counters are incremented whenever its corresponding event occurs. We have a register for the number of cores currently stalled (4 bits) which counts the number of signals from each tile enabled when the core is stalled. We need another register (16 bits) to evaluate \( M \) of Equation (2) which is incremented when the number of currently stalled cores surpasses a certain number. Because this technique requires the PCU to have more timely measurement data, the PCU updates these registers via 3 directly connected lines from each tile. For each tile, each line signals when a packet request is issued, a request is served or the core is stalled, respectively. Overall, the implementation overhead of the throughput-driven DVFS in smaller CMPs is much lower than that of the latency-based controls, though its scalability in very large CMPs may be limited by the requirement of directly connected status wires.

5.2 Energy and Performance Comparisons

Figure 3 shows the energy comparison for the different methods. The figure shows, the critical latency defined by Equation (3) leads to significantly more power savings than that from AMAT [4]. This is especially obvious for the case canneal. The model-assist technique can further reduce power dissipation. Our best method can provide additional 50% power reduction over prior techniques [4].

Figure 4 provides a comparison of the performance impact of the different methods. Except dedup, the performance degradation from all of our techniques is quite limited. In the worst case of dedup, model assist and our best control progressively improve the performance. Overall, the performance loss from all our methods is around only 5%. In Table 2, the normalized energy-delay product among all cases are listed. The progressive improvement from the new metric (critical latency), model assist and dynamic reference can...
be observed. Compared to AMAT+PI [4], our best method reduces the energy-delay product by 56%.

<table>
<thead>
<tr>
<th>Method</th>
<th>Energy x Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>1.0</td>
</tr>
<tr>
<td>AMAT+PI</td>
<td>0.5</td>
</tr>
<tr>
<td>CL+PI</td>
<td>0.31</td>
</tr>
<tr>
<td>CL+ModelAssist</td>
<td>0.28</td>
</tr>
<tr>
<td>Throughput</td>
<td>0.27</td>
</tr>
<tr>
<td>OurBest</td>
<td>0.22</td>
</tr>
</tbody>
</table>

Table 2: Normalized energy-delay product on average for all PARSEC cases.

To analyze controller sensitivity to parameters, we varied the parameters to obtain different solutions for the throughput-based and our best methods. The results are plotted in Figure 5 together with those from AMAT+PI and CL+ModelAssist. Points in Figure 5 indicate the average results among all benchmarks using the same set of parameters. Our target performance loss is 5%, thus we choose the point from the OurBest curve which is closest to 1.05. The solutions at lower-left envelope for each technique are Pareto optimal as they imply either high performance or low energy consumption. One can see that OurBest, the latency-based PI control with dynamic reference, achieves the best performance-energy tradeoff regardless of parameter set point. On the other hand, please note that the throughput-driven DVFS has much lower implementation overhead as shown in Section 4.4.

Figure 5: Normalized system performance (with the baseline result as 1) for PARSEC benchmarks.  

6. CONCLUSIONS

In this work, we investigated DVFS for shared resources (NOC/LLC) in multicore processor designs. Several metrics and policies are developed. The proposed techniques are evaluated on public domain architecture benchmarks with full-system simulation. The results show quite large energy savings and improvement over recent previous work.

7. ACKNOWLEDGMENTS

This work is supported by Intel.

References

SUPPLEMENTAL MATERIAL

In this section we provide some additional analysis, exploring how the control algorithm works and its sensitivity to control interval size.

In the DVFS control, one practical issue is how to decide the control interval size for the control. We use 50K core clock cycles, which is long enough for V/F change and short enough for fine-grained power control. We performed simulations on two PARSEC benchmarks with different control interval sizes and the results are summarized in Table 3. One can see that the results are not sensitive to moderate interval size changes.

<table>
<thead>
<tr>
<th>Interval size (# cycles)</th>
<th>Energy × Delay freqmine</th>
<th>x264</th>
</tr>
</thead>
<tbody>
<tr>
<td>12K</td>
<td>0.468</td>
<td>0.151</td>
</tr>
<tr>
<td>25K</td>
<td>0.470</td>
<td>0.145</td>
</tr>
<tr>
<td>50K</td>
<td>0.462</td>
<td>0.136</td>
</tr>
<tr>
<td>75K</td>
<td>0.479</td>
<td>0.147</td>
</tr>
<tr>
<td>100K</td>
<td>0.500</td>
<td>0.145</td>
</tr>
</tbody>
</table>

Table 3: Impact of different control interval sizes. The Energy × Delay values are normalized with the baseline result as 1.

The model assisted PI control described in Section 4.2.2 is based on Equation (7), which is loosely derived from Equation (3-5). We tried to obtain some supporting evidence through experiments. For the PARSEC benchmark fluidanimate, we simulate with different constant uncore V/F levels throughout the entire ROI. The average critical latency versus uncore clock period results are plotted in Figure 6. The results confirm that critical latency has approximately linear dependence on the uncore period.

In Section 4.2.2 and 4.2.3, we argue that it is very hard to accurately predict the system behaviors. Figure 7, 8 and 9 show some simulation results for a segment of x264, an application from the PARSEC benchmarks. Its horizontal axis is in terms of control interval of 50K clock cycles, i.e., each data point is an average over 50K cycles. Despite such smoothing, the data still show drastic changes from interval to interval. These results confirm that it is very difficult to accurately predict the behavior of a multicore system.