In-network Monitoring and Control Policy for DVFS of CMP Networks-on-Chip and Last Level Caches

Xi Chen* Zheng Xu* Hyungjun Kim* Paul Gratz* Jiang Hu* Michael Kishinevsky† Umit Ogras†

*Department of Electrical and Computer Engineering, Texas A&M University
†Strategic CAD Labs, Intel Corporation
*{xchen19, xuzheng0309, hyungjun, pgratz, jianghu}@tamu.edu, †{michael.kishinevsky, umit.y.ogras}@intel.com

***Abstract*—In chip design today and for a foreseeable future, on-chip communication is not only a performance bottleneck but also a substantial power consumer. This work focuses on employing dynamic voltage and frequency scaling (DVFS) policies for networks-on-chip (NoC) and shared, distributed last-level caches (LLC). In particular, we consider a practical system architecture where the distributed LLC and the NoC share a voltage/frequency domain which is separate from the core domain. This architecture enables controlling the relative speed between the cores and memory hierarchy without introducing synchronization delays within the NoC. DVFS for this architecture is more difficult than individual link/core-based DVFS since it involves spatially distributed monitoring and control. We propose an average memory access time (AMAT)-based monitoring technique and integrate it with DVFS based on PID control theory. Simulations on PARSEC benchmarks yield a 33% dynamic energy savings with a negligible impact on system performance.**

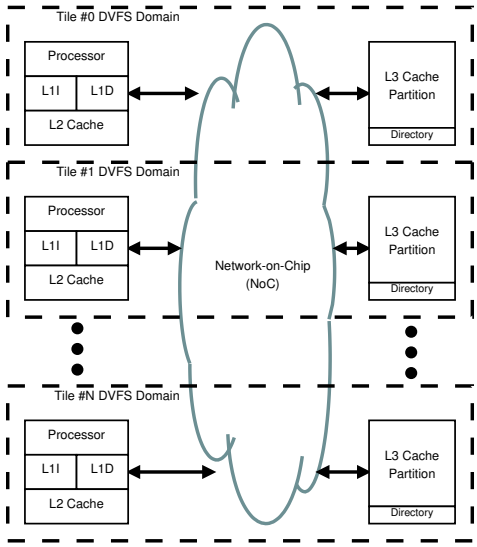
***Index Terms*—Multicore, NoC, dynamic power, memory system**

I. Introduction

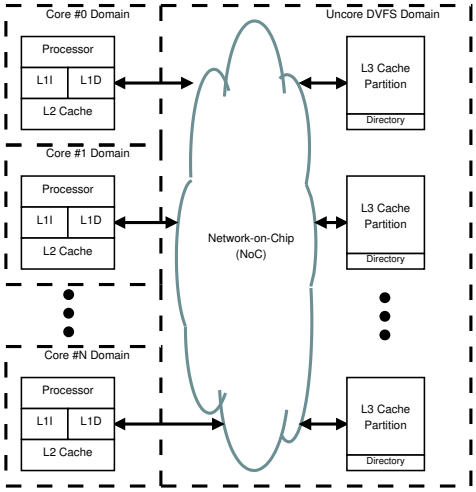
The progress of chip design technology faces two related challenges: power and on-chip communication [13]. A recent study by Google [1] shows that, as power-efficiency improves for server processors, the interconnection network is becoming a major power consumer in the datacenter. Likewise, on-chip communication now forms a power bottleneck in chip multiprocessors (CMPs) given the considerable progress on processor core power-efficiency. In his speech at International Conference on Computer-Aided Design 2011, Chris Malachowsky, co-founder of Nvidia, pointed out that the energy expended delivering data on chip has far exceeded the energy in computation operations. Dynamic voltage and frequency scaling (DVFS) is an effective and popular low-power technique. This paper presents techniques that facilitate efficient DVFS for NoC (Networks-on-Chip), which is widely recognized as a scalable approach to on-chip communication.

In this work, we will focus on DVFS for NoCs for CMPs. Previous work in DVFS for NoCs and CMPs have focused on per-core or per-router DVFS policies, as shown in Figure 1a. Unlike much prior work, we consider a realistic scenario wherein the entire NoC and shared last-level cache (LLC) forms a single voltage/frequency domain, separate from the domains of the cores (see Figure 1b). Latency is a critical characteristic in CMP NoCs [6, 5]. Synchronizing across clock domains is expensive in cycles per hop; placing many clock domain crossings in the interconnect makes the design unscalable by imposing a high cost in latency per hop [18].

Furthermore, we argue placing the shared LLC in one clock domain across the chip is logical because it is one large, partitioned structure. Allowing some portions of the address space to see a penalty in performance due to a given



(a) CMP with V/F domains by tile.



(b) Separate V/F domain for the uncore and cores.

Fig. 1: Logical CMP diagrams highlighting Voltage/Frequency domain partitions.

LLC bank being clocked slower relative to other portions would impact performance determinism and could make the performance of active threads hostage to the DVFS state of idle threads. Unlike the individual cores which are running different threads/programs, the LLC banks have a mostly homogeneous load due to the interleaving of cache lines in the system; in this case, the voltage/frequency domain partitioning like Figure 1a can be inefficient. For example, if one core is

active and makes many LLC requests while the other cores are idle, then, according to the partition in Figure 1a, only the V/F domain for the active core is in high V/F mode. However, this is not sufficient as its data travels in other domains which are in low V/F modes. Therefore, Figure 1b is a more reasonable V/F domain partitioning for a CMP with shared LLC.

DVFS of an entire NoC system is more difficult to manage than the DVFS in previous works, such as that for each link [20] or each core [25]. In these cases, voltage/frequency levels are determined by a single, local observation, (e.g. link congestion [20] or core workload [25]). In contrast, sharing V/F level over the entire network, requires information on the load/activity of many different entities. Therefore, the challenge is how to tune the single V/F level to satisfy the need of the entire network, without impacting performance of the network by flooding it with status information messages.

Two crucial components of a DVFS system are the performance monitor, and the controller which computes the output V/F level according to observed performance. Compared to previous works [20, 25], our challenge is developing online monitoring techniques. That is, how to efficiently and reliably monitor the overall network performance, which is obviously more complex than a single core or link. If many places are monitored, then how to collect the monitoring results to the central power controller with low overhead?

In this paper we address these questions. The key contributions of this work are as follows:

- We introduce several new uncore status metrics to predict the impact of DVFS policy on system performance.
- We propose a novel, extremely low overhead, uncore status monitoring technique. This technique is composed of the following: 1) Per-node metric sampling, 2) Passive in-network status encoding, no extra packets needed, 3) Metric extrapolation to properly scale value weights.
- We introduce an uncore DVFS policy based upon PID (Proportional-Integral-Derivative) control.

Results for our technique on the PARSEC suite show that we achieve a 33% dynamic energy reduction while limiting the system's average memory access time (AMAT) increase to 5%, and overall system performance penalty to $\sim 2.5\%$.

II. Background and Related Work

This section introduces the basics of shared, distributed, last-level caches and their NoC interconnect in CMPs (collectively the "uncore"). We then introduce the basic concepts in NoC performance monitoring and discuss the power and performance constraints on the uncore. Finally we discuss power management using DVFS in the uncore.

A. CMP Uncore Basics

Typical CMPs are composed of a set of cores, consisting of the processor and private lower level caches (level-1 and sometimes level-2), along with an "uncore". The uncore portion of the die refers to all the integrated subsystems on the chip except the cores. More precisely, the LLC, the routers and links of NoC, integrated memory controller, integrated I/O controller etc. constitute the uncore. In other words, the uncore enables communication between the processing cores, and with the LLC, off-chip memory, I/O devices, graphics core and accelerators, if any. Therefore, any miss in the local caches of a core will result in an uncore request. Finding the location of the requested cache line, transferring the cache line to the core or the memory controller as well as controlling the global state of the cache line are all managed by the uncore. In modern LLCs, the banks of the LLC are partitioned and distributed such that a portion of the LLC is co-located with each core. This LLC arrangement has the advantage of improving performance over the prior, monolithic cache designs.

In our baseline design, we assume coherence between the private caches in the cores is maintained via a distributed directory cache in the uncore. The NoC interconnecting the uncore and the cores primarily carries memory system traffic. Particularly, the NoC carries lower-level cache spills and fills, lower-level cache coherence messages, and LLC cache spills and fills. We assume that LLC cache set indices are spread about the partitions of the LLC in a round robin fashion, to ensure that each partition receives approximately the same amount of traffic and no single partition becomes a hotspot.

B. Uncore Power and Performance Implications

The uncore consumes a significant fraction of the whole chip power due to the relatively large proportion of the chip area it consumes. Dynamic power dissipation for CMOS circuits is given by

$$P = \alpha \cdot C \cdot V^2 \cdot f \tag{1}$$

Although the activity factor (α) for the uncore is not necessarily high, its total area and capacitance (C) can be large. Similarly the leakage power, a growing problem in future VLSI process technologies, is also proportional to the area of uncore. Equation (1) also includes two interrelated components – the voltage squared (V^2) and frequency (f). For a given design, increasing the voltage makes transistors to switch faster, allowing the chip to operate at a higher frequency. Conversely, lowering the voltage forces a decrease of the clock frequency to meet timing constraints. *Dynamic voltage and frequency scaling* (DVFS), is a well-known technique which leverages this relationship to lower dynamic power consumption. Lowering the voltage has a quadratic effect on the dynamic power of the circuit being lowered, though this comes at the cost of some performance due to the required decrease in frequency. By lowering the voltage and frequency to match but not exceed the demands of the application, a good DVFS policy can achieve substantial power savings.

Achieving power savings through DVFS in the cores is a comparatively simple problem, considering that the information needed to select an appropriate voltage and frequency are available in a localized place, the CPU core itself (e.g. from performance counters within that core). Determining an appropriate DVFS state for the uncore is a significantly more difficult problem. Because the uncore consists of the LLC and network, the relative criticality of the uncore's performance to the performance of the system is highly dependent on the application's demand for LLC data and inter-thread communication. Applications which are mostly L1 cache resident place little performance pressure on the uncore and the uncore can safely run at a relatively low frequency, while those with frequent L2 cache misses place high demands on the uncore and require the uncore to run at a high frequency.

For purposes of studying uncore DVFS polices, we decompose the problem into three major components. First, because the uncore consists of two very different components, the LLC and NoC, it is unclear what performance metrics are appropriate as an input to the DVFS policy. Second, because the uncore is distributed across the chip, a mechanism must be developed to monitor the status of the uncore performance metrics and inform the DVFS policy. Finally third, once the inputs have been defined and arrive at the DVFS controller,

an appropriate policy must be developed. We explore these components in the remainder of this section.

C. Uncore Performance Metrics

Network communication traffic load and performance can be described via several potential networks metrics. In this section we explore some potentially suitable indicators that may predict network load.

Queue Occupancy and Crossbar Demand: Routers typically have queues, or input FIFOs for temporary storage of data. The occupancy level of a queue naturally indicates local congestion. Hence, queue occupancy has been widely used in adaptive routing [4, 10] and DVFS for individual links [20, 15, 17]. The overall congestion status of a network can be estimated by monitoring queue occupancy of all routers.

A different and slightly overlapping metric for congestion is crossbar demand, which is the number of active requests to an output of a router [4]. Many requests to an output imply a convergent traffic patten, which is likely to become a bottleneck. While queue occupancy measures the degree of downstream congestion, crossbar demand indicates local congestion. In either case, estimating the performance of the network as a whole requires monitoring many if not all routers.

Average Per-Hop Latency (APHL): While queue occupancy and crossbar demand reflect network congestion, packet latency is a direct measure of network performance. Packet latency is defined by the time span from a packet request being made to the arrival to its destination. Although low packet latency is obviously preferred, it is not directly obvious what should be the absolute goal in packet latency. Different types of workload can be expected to produce different average hop-counts and average packet lengths and hence different nominal packet latencies. For the convenience of a normalized comparison, we suggest average per-hop latency (APHL), which is the average latency a flit incurs as it traverses each router along its path through the NoC. For any specific system, there is an unique minimum latency for one hop which is determined by the link delay plus the router pipeline latency. By removing distance traveled and serialization latency, APHL gives a traffic pattern independent metric of network load. As such, APHL's deviance from a given NoC's inherent minimum per-hop latency serves as a clear target for the DVFS tuning.

Average Memory Access Time (AMAT): While the previous metrics merely provide the information of current network congestion, we also propose average memory access time (AMAT) as a metric which provides not only the current network status but also the demand for its performance. When a cache miss occurs on the private caches, the NoC is used to fetch the missing cache line from LLC. Thus, NoC performance translates to memory operation latency. Experimentally we determined that for small AMAT increases, IPC (instructions per cycle) of the cores decreases approximately linearly with AMAT with a slope of .5. Consequently, AMAT provides a good index of required uncore performance. Equation (2) shows a simplified uncore AMAT formula:

$$\begin{aligned} AMAT = {} & HitRate(private) \times Latency(private) \\ & + (1 - HitRate(private)) \times Latency(uncore) \end{aligned} \quad (2)$$

where $HitRate(private)$ is the aggregate hit rate on the private caches (L1 and L2) and $Latency(private)$ is the average access time on those caches. $Latency(uncore)$ is the average access time to the shared LLC (ie. access time to the L3 slice on a remote tile), plus the latency of the memory controller if the required cache block is missing in LLC. For simplicity we assume $Latency(uncore)$ linear with the clock period $(1/f)$ of the uncore, such that Figure 2 shows a representation of AMAT.

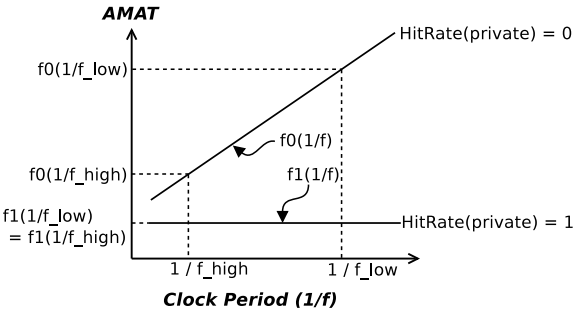


Fig. 2: AMAT with respect to uncore clock period.

Figure 2 shows two extreme cases. f_0 depicts the AMAT with respect to the clock period when $HitRate(private) = 0$, while f_1 shows the AMAT when $HitRate(private) = 1$. f_0 represents the case the most of the memory access results in the private cache miss such that the missing block must be transferred over the network. In this case, the AMAT is highly dependent on the uncore performance, hence decreasing the uncore frequency via DVFS has a strong negative impact on system performance and should be avoided. On the other hand, f_1 represents the case where all memory accesses are served by the private caches. In such a case, decreasing uncore frequency has no impact on system behavior, hence should be done to achieve power savings with little impact on performance. Thus, it is desirable for the DVFS controller to account for AMAT in deciding whether increasing the frequency is worth the increased power or not.

D. Uncore Status Monitoring

In the prior work, network status information has been predominately used locally, particularly for adaptive routing [10, 24, 22, 21], and for localized DVFS policies [20, 15, 17]. Gratz et al. [4] and Ma et al. [12] propose small, light-weight status information networks to provide deeper visibility into the NoC and hence enable better adaptive routing decisions. In these cases, the performance metric obtained from the local router is assumed to provide enough information to enable local decisions, such as which output port to send a packet to, or what DVFS setting for a given router. These techniques, however, provide a limited view of the status of the network as a whole, either decreasing exponentially with distance [4] or limited to one bit of data per node in only the orthogonal directions [12]. There are some other monitoring techniques to inform DVFS policy in NoCs. Yin et al. [26] propose dedicated links and virtual channels for collecting and monitoring system status information; this approach is expensive in terms of design time to create a dedicated interface, power for these links and area for the logic associated. Rahimi et al. [17] propose to monitor network performance based on link utilization; this approach is reasonable for fine-grained, local link DVFS control but does not scale to a full-chip shared resource.

Ideally for global decisions, one would like continuous monitoring of uncore status, whether per-packet statistics to calculate APHL, or per-core memory statistics to calculate average AMAT. Such complete, active monitoring entails a calculation overhead at every tile, which includes counting starting/arrival time of packets, calculating the average, etc. Moreover, the status obtained at each tile must be regularly sent to the central power controller to set DVFS policy, consequently causing increased traffic and congestion.

To avoid increasing traffic and congestion within the primary NoC we propose to leverage unused space in the header flits of existing packets to "piggyback" network status information. This approach has the advantage of being scalable to larger networks because no extra networks or links must be designed. Furthermore, it is passive, ie. it does not perturb the network with extra status packets. A potential disadvantage to this approach is the non-determinism of status message delivery. As we will show, however, this does not significantly degrade the accuracy of this approach.

E. Prior Work in NoC and CMP DVFS

Several groups have explored DVFS in NoCs and/or CMPs. Shang et al. wrote a pioneering work in the use of DVFS for NoCs [20]. They perform DVFS for individual links in NoC. DVFS has also been studied for individual routers [14]. Son et al. explored DVFS for specific application NoCs [23]. In other previous works [15, 7, 17], the voltage/frequency domains within the NoC are assumed to be associated with individual cores processor cores. In each of these works, the DVFS policy is determined by local information. DVFS is also widely studied for processor cores or CMPs. A simple approach is rule-based DVFS [8] that changes voltage/frequency level when monitored performance crosses certain threshold. Rule-based method is improved by including hysteresis [19]. Control theoretic techniques are proposed in [25, 15]. Jung and Pedram proposed a learning-based approach [9].

In this paper, we address DVFS for the uncore as a whole. We will demonstrate that there is a large opportunity for uncore power saving with an accurate but low cost monitoring technique to fetch network information, meanwhile using a simple but effective control algorithm to adjust uncore voltage and frequency based on the monitoring result.

III. AMAT-Based DVFS Policy Description

We propose to estimate current AMAT to provide feedback to the DVFS controller. Each tile keeps track of its status information and a monitor collects the information for AMAT computation. To minimize hardware overhead, we use the existing NoC infrastructure to convey the information, to avoid increasing traffic and congestion, we leverage unused space in the header flits and employ passive monitoring rather than any active system. We propose a single monitor for our 4×4 network; experimental results show this is sufficient for this size network. We assume the uncore contains a Power Control Unit (PCU), which is a dedicated small processor for chip power management as in Intel's Nehalem architecture [11].

A. Data Collection

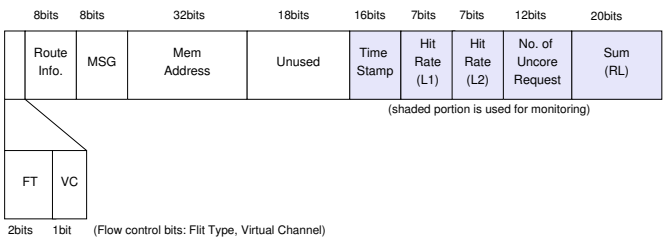


Fig. 3: Header flit bit fields.

In our proposed technique, each tile maintains statistics on its private, lower-level cache's behavior necessary to compute AMAT. These statistics must be sent to the a central, monitoring node for overall AMAT estimation and DVFS policy generation. Rather than injecting more packets into the system, increasing traffic and congestion, each tile "piggybacks" its status information into every packet injected into the network. Under the assumption of 128-bit wide links, and 64-byte cache blocks, we find there are $\sim$ 64 unused bits in the header flit or single flit packets. We leverage these unused bits to encode the status information as shown in Figure 3. Here, *Time Stamp* denotes the time the packet is generated. $HitRate(L1)$ and $HitRate(L2)$ show the hit rate of L1 and L2 caches during the time window. $No.of Uncore Request$ and $Sum(RL)$ correspond to the number of L2 requests sent into the Uncore and the sum request latency during the time window, respectively. The $Sum(RL)$ is the accumulated time span required for L2 requests into the uncore, and includes the access time of the L3, cache coherence resolution time and main memory access time on L3 misses. The additional hardware cost to maintain this status information is discussed in Section III-D.

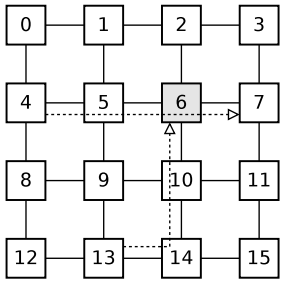


Fig. 4: NoC layout; the monitor resides at tile 6

Figure 4 shows an NoC layout illustrating our monitoring technique. Instead of adding monitors to each tile, we employ a single monitor that collects the data from the whole network. We chose tile 6 as the location of the monitor intuitively because a central location provides the best vantage point to passively collect the desired statistics (an assumption experimentally verified). In addition, the monitor tile should be near the PCU so that the overhead of interconnect between the monitor tile and the PCU is negligible. The monitor grabs status information from all passing packets (e.g. a packet traveling from tile 4 to tile 7) or packets bound to tile 6 (e.g. a packet from tile 13 to tile 6). Data collection is passive, a potential downside of this approach is non-determinism of message delivery, as we will discuss in the next section.

B. Overall AMAT Computation

Once a packet arrives at the monitor, the source node's status information, encoded in the packet's header, is handed over to the PCU (Power Control Unit). The PCU is in charge of computing AMAT on the packet's arrival. The AMAT for each tile is computed as per Equation 3 based on the information gathered.

$$\begin{aligned} AMAT &= HitRate(L1) \times AccessTime(L1) \\ &\quad + (1 - HitRate(L1)) \times Latency(L2) \\ Latency(L2) &= HitRate(L2) \times AccessTime(L2) \\ &\quad + (1 - HitRate(L2)) \times Latency(uncore) \\ Latency(uncore) &= (Sum(RL))/(\text{No. of Uncore Request}) \end{aligned} \tag{3}$$

Note that Eq. 3 is a more detailed version of Eq. 2, where $HitRate(private)$ and $Latency(private)$ are decomposed into their constituent L1 and L2 components. The $AccessTime$ values are constants for a given L1 and L2 cache design and hence need not be sent in the header. Also note, the $Sum(RL)$ the sum of round trip time for Uncore requests, including NoC latency, L3 access time, cache

coherency latency, as well as main memory latency when L3 misses occur.

Our DVFS policy requires the overall, chip-wide AMAT as an input. This overall AMAT is computed for every time window of 50000 cycles. To calculate overall AMAT, we use a weighted average of the per-node AMAT, with respect to the number of memory instructions issued by that tile during the window. Upon packet arrival or traversal of the monitor tile, the PCU calculates the AMAT for the source tile of the packet and stores it in a table with 16 entries; one for each tile. Memory instruction count is also stored in the table. At the end of each time window, the PCU, using the memory instruction counts as weights, computes the weighted average of AMAT across all tiles as the overall AMAT. At the end of every time window, all entries are reset to 0; entries not updated are excluded from overall AMAT computation. Figure 5 depicts this process. The graph shows the AMATs of $Tile_i$ and $Tile_j$, in a two tile system. The cross marks on the time line denote packet arrivals, and the circles are the AMAT for the corresponding tile to be stored in the table. The numbers above the circles denote the memory instruction count. The triangles are the final AMAT values used in overall AMAT calculation. In "Window 1", $Tile_i$ sends two packets, but the data from the first packet is discarded as the second one overrides it. At the end of this window, the PCU first calculates the per-tile AMAT for $Tile_i$ and $Tile_j$ of 80 and 65 respectively from the status information in the final packet from each. It then determines there are 120 memory instructions in $Tile_i$ and 60 in $Tile_j$, and these values are used in the weighted average overall AMAT thus $(80 \times 120 + 65 \times 60)/(120 + 60)$. In "Window 2", the PCU receives no packet from $Tile_i$ thus the overall AMAT is computed only from $AMAT_j$ and it is 120. The system clock is also reset at the end of time window, thus if a packet's time stamp is later than the current system time, the packet is discarded as it is from the previous window.

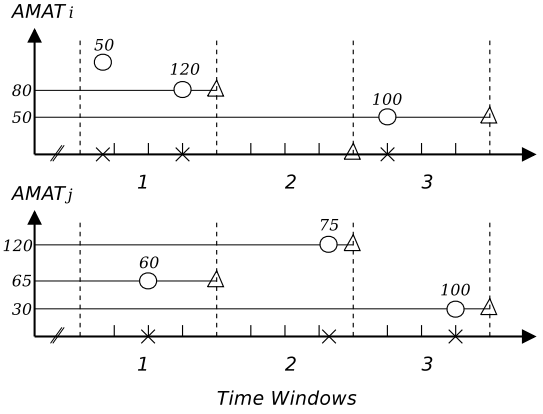
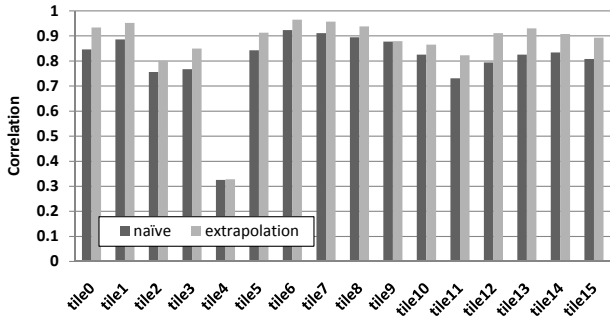


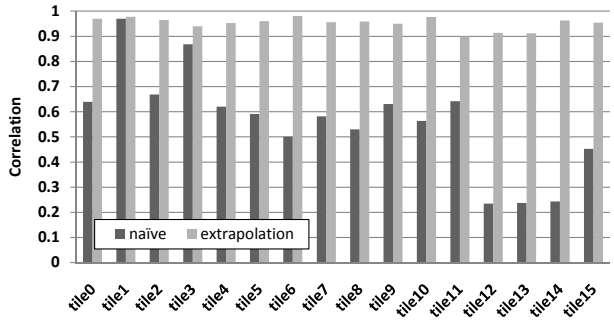
Fig. 5: Overall AMAT Computation

We refer the above method as "Naïve" as it calculates AMAT without accounting for packet arrival time. Unfortunately, this passive monitoring method does not guarantee that each tile's statistics are current. As the final value is evaluated at the end of each time window, it is better to have a packet from a tile near the end of the window. For example, at "Window 3" in Figure 5, $AMAT_i$ and $AMAT_j$ have the same weight as the numbers of memory instructions carried over are the same. However, in the end of the time window, it is very likely that $Tile_i$ eventually has more memory instructions than $Tile_j$ since the count had been determined much earlier. Thus, we introduce a method of linear extrapolation to correct this bias. We assume that the number of memory instructions linearly increases in a time window. With the fact that the memory instruction count is 0 at the beginning of a time window, we can estimate the count in the end of it using a sampled count at any location of the window. We use the "Time Stamp" in the packet to define the relative location within the time window, and that needs to be stored in the table. By this "extrapolation", in "Window 3", the effective memory instruction count of $Tile_i$ becomes 400 while that of $Tile_j$ becomes 133. Finally, the overall AMAT becomes 45 while in "Naïve" it is 40.

Figure 6 compares the performances of the "Naïve" and "Extrapolation" methods. The figure shows the correlation between actual and computed overall AMATs for monitor tiles, tile0-tile15 (tiles numbered as shown in Figure 4). Generally, higher correlations across all tiles indicates a given method provides a better estimate of overall AMAT. For *Canneal* we see that "extrapolation" generally results in a more accurate overall AMAT than "Naïve." For *Vips*, much higher correlation is achieved by "extrapolation" method. *Vips*'s traffic pattern is more highly skewed and hence requires "extrapolation" to produce reasonable results. Note that in both cases tile6 shows the highest correlation among the tiles, thus we select tile6 as our monitor tile which also matches our assumption that central location provides better visibility.



(a) Canneal



(b) Vips

Fig. 6: Overall AMAT from each tile's perspective

C. PID-Based DVFS Policy and Stability Analysis

We choose to implement a DVFS control scheme based on PID (Proportional-Integral-Derivative) control. Compared to rule-based approaches [8, 19], PID control can easily adapt to various application scenarios. Its computation cost is significantly less than learning-based methods [9]. It has been applied for DVFS in processor cores [25]. Moreover, it has theoretic grounds for stability analysis.

The block diagram of PID control system is shown in

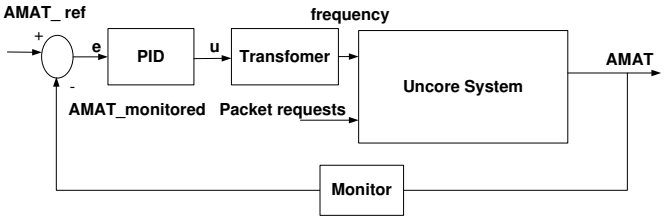


Fig. 7: PID system diagram.

Figure 7. The controller takes two inputs: the reference AMAT and monitored AMAT. The reference AMAT is the control target and can be obtained from empirical data. The control output is updated once per control interval (same as time window for AMAT monitoring). The difference between the two inputs is the error function $e_j = AMAT_{ref} - AMAT_j$ where $AMAT_j$ is the AMAT observed at control interval j. The controller calculates the control output u according to

$$u_j = u_{j-1} + K_I \cdot e_j + K_P \cdot (e_j - e_{j-1}) \quad (4)$$

where K_I and K_P are constant coefficients. Note we only implement the Proportional (P) and Integral (I) terms in our controller, as this is simpler and often more robust than including the Derivative (D) term [25]. Control output u is converted to a V/F setting for the uncore system. In general, AMAT is a nonlinear function with respect to uncore frequency f. We perform a transformation of $u = 1/f$ such that AMAT is approximately a linear function of u.

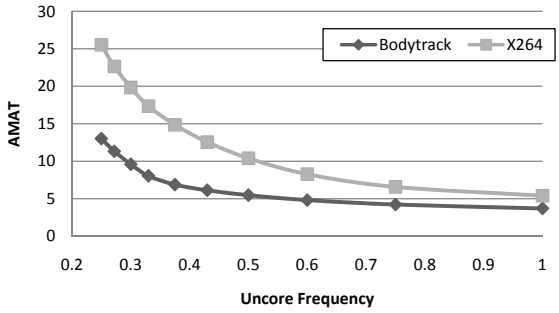


Fig. 8: AMAT versus uncore frequency.

In order to analyze the stability of this control system, we need to obtain an analytical form of the system function. Figure 8 shows simulation results of AMAT versus uncore frequency (f). Then, by curve fitting, we can obtain an approximated expression

$$AMAT = \frac{\eta}{f} + \beta \quad (5)$$

where η and β are two fitting coefficients. Although the values of η and β are specific for each application, the subsequent analysis is general, as long as the $AMAT(f)$ relation conforms to Equation (5). We performed many other simulations and all results follow similar trend as Figure 8.

By performing z-transform, we can get system transfer function as

$$AMAT(z) = \frac{(K_P + K_I) \cdot \eta \cdot z - K_P \cdot \eta}{(1 + K_P \cdot \eta + K_I \cdot \eta) \cdot z - (1 + K_P \cdot \eta)} \quad (6)$$

The characteristic equation for this system is

$$(1 + K_P \cdot \eta + K_I \cdot \eta) \cdot z - (1 + K_P \cdot \eta) = 0 \quad (7)$$

According to control theory [3], the system is stable if and only if the root of the above equation is inside the unit circle of the z-plane. In our case, this requires that $z = (1 + K_P \cdot \eta)/(1 + K_P \cdot \eta + K_I \cdot \eta)$ is within the unit circle. To satisfy this condition, we simply need to find PID coefficient $K_P, K_I > 0$.

When $\rho_{L1,miss} \cdot \rho_{L2,miss} = 0$ (where ρ means rate), AMAT is not affected by f and the above closed-loop based analysis does not hold. However, the chance of $\rho_{L1,miss} \cdot \rho_{L2,miss} = 0$ in practice is very small. Even when it happens, the monitored AMAT is very low and the controlled frequency gradually decreases to its minimum. Therefore, the system is still stable.

TABLE I: Simulation setup

Parameter	Values
Core Frequency	1GHz
#processing cores	16
L1 data cache	2-way 32Kb, 1 core cycle latency
L2 cache	8-way 256Kb, 13 core cycle latency
L3 cache (LLC)	16-way, 2MB/bank, 32MB/total, 15 uncore cycle latency
Directory cache	MESI, 4 uncore cycle latency
Memory access latency	100 core cycles
NoC	4×4 2D mesh, X-Y DOR, 2VCs/port 4flits deep
Voltage/Frequency	10 levels, voltage: 0.5V–1V, frequency: 250MHz–1GHz

D. Implementation Overhead

The DVFS controller needs a modest amount of hardware support. At each tile, one counter is needed to frame the control interval, in our case 50000 cycles, so 16 bits is sufficient. To track each tile's memory operation status the following additional registers are required by each tile: One 20-bit register for L1 hit count, one 12-bit register for L2 hit count, one 12-bit register is required to count the number of L2 misses and one 20-bit register to sum up all of the L2 miss latencies. These registers are updated at the completion of memory instructions, and reset as the time windows end. These registers are used to compute $HitRate(L1)$ and $HitRate(L2)$ prior to encoding in the header flit, this latency is hidden by the packet generation delay. Both AMAT computation and the PID algorithm are composed of a few simple arithmetic calculations, which can be easily handled by the PCU. The PCU has sufficient storage to accommodate the data collected. We assume the PCU is co-located with the monitor tile, if this is not the case the monitor will need additional registers for the temporary storage of collected data prior to sending it to the PCU. Therefore, the overall hardware overhead is 80 bits per tile plus a possible 64 bits at the monitor tile.

IV. Evaluation

In this section we first discuss our methodology and then compare the performance of our proposed technique and several variation versus baseline.

A. Experiment Setup

The testbed architecture in our experiment is a 16-tile CMP as shown in Figure 1(b). Each tile is composed of a processing core with 2-levels of private cache, a network interface (NI) and a partition of the shared L3 cache (LLC). Table I summarizes our experimental configurations and parameters. We use M5 full system simulator to generate PARSEC shared-memory multi-processor benchmark memory system traces [2]. Each trace contains up to 250 million memory operations. These traces are run through a memory hierarchy (L1-L2-LLC+directory) and network simulator based upon Ocintsim [16]. Although trace-driven, open-loop, NoC simulation can introduce error, we expect that for the small changes in AMAT experienced, these errors are minimal. Uncore DVFS is emulated by varying packet injection rate (e.g. slowing down

uncore frequency by a half emulates doubling the injection rate). Please note that uncore DVFS does not affect off-chip memory access time. We assume that memory access time is constant with respect to core frequency. We explored several options for the DVFS control interval between 10000 clock cycles to 100000 clock cycles. In this work present results assuming an interval of 50000 clocks cycles as it provides more than sufficient time to collect traffic information as well as being short enough to capture fine-grain program phase behavior.

In this work we focus on the dynamic energy reduction due to DVFS based off impact on the dynamic power equation (Eq. 1), because dynamic power currently dominates in modern process technologies. DVFS should also provide benefit for static energy consumption though the absolute amount is highly dependent on particular process technology, we plan to explore this in future work.

B. Power and Performance Comparisons

Figure 9 compares the dynamic energy savings and performance degradation of our proposed approach (labeled *Est. AMAT+PID*) versus baseline without DVFS, along with four variations which we discuss in the following subsections. The figure shows our PID technique, informed by estimated AMAT provides an average savings of 33% while reducing the performance of the uncore (measured by impact on absolute AMAT) by 5.3%. We empirically determined that AMAT increases of 5% tend to decrease processor performance (IPC) by $< 2.5\%$. *Blackscholes* shows the best benefit, with a 73% reduction in energy, while *Canneal* shows the least reduction in energy at 1%. Both benchmarks show negligible performance impact. In *Blackscholes* the uncore is not critical to performance, because the L1 and L2 hit rates are high, so voltage and frequency (V/F) can be dropped without impacting performance. *Canneal*, however, has a relatively lower L1 and L2 hit rate and thus the uncore's performance is more critical to application performance. In both cases our algorithm preserves our performance goal of <5% AMAT loss.

PID-Based vs. Rule-Based DVFS: Figure 9 also shows the results for a simple, naïve, rule based approach for DVFS policy (labeled *Rule-Based*). In this approach, V/Fs are associated with specific ranges of AMAT (e.g. if monitored AMAT is a then uncore frequency level is set to b). The advantage to this technique would be that it eschews the overheads of the PID controller, however, its static nature ignores time-varying dynamics of the system. While rule-based DVFS obtains similar energy reduction as PID-based DVFS, it is unable to adapt as well and performance decreases by 10% compared to the 5% by PID-based DVFS.

Monitoring AMAT vs. APHL: APHL (Average Per-Hop Latency) is a simple, direct measure of network performance discussed in Section II-C. Figure 9 also shows results for PID informed by the APHL metric (labeled *APHL+PID*). In some cases, like *Blackscholes*, the power savings from APHL-based DVFS is much less than that of AMAT-based. In such applications, the packet injection rate is not high but constantly above zero-load. Thus, the network is often moderately busy and APHL-based DVFS will not lower V/F level. On the other hand, the L1 and L2 miss rates in these cases are low and therefore moderate increase of APHL does not have significant impact to system performance. AMAT-based DVFS is able to capture such opportunities for energy savings. In other cases, such as *Canneal*, APHL-based DVFS is overly aggressive, causing $> 50\%$ performance degradation. This phenomenon is due to the static APHL target of three *uncore* cycles. When traffic load is not high, lowering uncore frequency may still maintain an APHL close to 3 uncore cycles, however, the network latency in core cycles increases due to the frequency ratio change. This dramatically affects performance when L1 or L2 miss rates are high. Monitoring APHL in core cycles, however, is impractical because of the difficulty in finding appropriate PID reference levels without knowing the dynamically changing network latency relative to system performance. Overall, APHL-based DVFS achieves about the same power savings of about 30% as AMAT-based DVFS, but degradation performance significantly more at about 16%.

Sampled AMAT vs. Perfect Knowledge of AMAT: The AMAT monitoring technique we propose uses a single tile to collect information from its own packets and passing-by traffic. Compared to monitoring all tiles for complete knowledge of AMAT, our technique has much lower overhead, however, this comes at the potential cost of some inaccuracy in AMAT estimation due to incomplete knowledge. Figure 9 also shows results for PID DVFS based upon perfect knowledge of the system AMAT each time window (labeled *Perfect AMAT+PID*). The figure shows estimating AMAT produces results quite close to perfect knowledge. Overall, the difference in energy savings and performance is $< 1\%$.

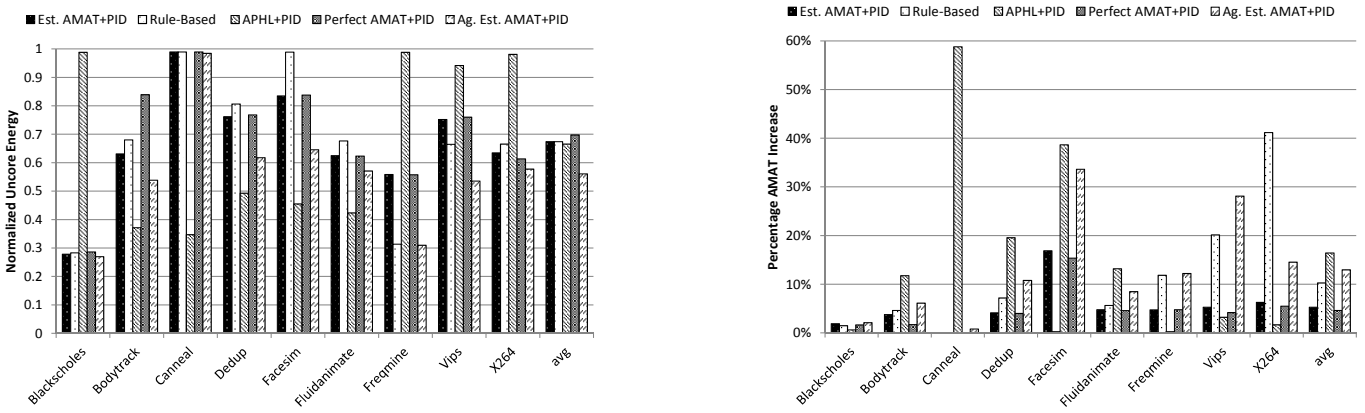
Power-Performance Tradeoff: The tradeoff between power savings and AMAT degradation can be easily adjusted by tuning the $AMAT_{ref}$ to the PID system (see Figure 7). In *Est. AMAT+PID* we set $AMAT_{ref}$ to 2, empirically determined to reach our goal of $< 5\%$ performance degradation. Figure 9 shows results for increasing $AMAT_{ref}$ to 4.2 to achieve a more aggressive power savings (labeled *Ag. Est. AMAT+PID*). As expected, the aggressive PID yields increased energy savings of 44% as well as a greater AMAT degradation of 13%. This comparison confirms that the power-performance tradeoff can be easily managed by our approach. In future research, we will develop techniques that make $AMAT_{ref}$ adaptable to different applications at runtime.

C. Analysis

Here we provide some simulation details to aid developing an intuition on the behavior of our approach. In Figure 10, we show a snapshot of the uncore frequency over time of our *Est. AMAT+PID* system versus an "*Ideal*" DVFS policy. The *Ideal* policy is an unrealistic case where every benchmark is simulated once for each V/F setting and the lowest V/F are chosen for each time window which meet the performance goal of $< 5\%$ performance loss. Generally *Est. AMAT+PID* follows the *Ideal* policy very closely. Where variance occurs, the estimated method is generally more conservative. Initially, both frequencies are high due to high cache miss rate during initialization. After two millions of clock cycles, the lower level caches are filled and the frequency is lowered, reflecting the lower performance criticality of the uncore. There are some frequency spikes arising from occasional traffic spikes due to application phase changes. Surprisingly, the *Est. AMAT+PID* policy is able to track the performance needs of the uncore quite well even during these spikes.

V. Conclusions and Future Work

In this work, we propose a DVFS policy for a CMP's uncore. This policy leverages an uncore performance monitoring based upon AMAT estimation. This technique is integrated with a PID-based DVFS control system. In simulation on



(a) Normalized dynamic energy consumption

(b) Performance degradation

Fig. 9: Energy and performance impact for PARSEC benchmarks. Our proposed method is "Est. AMAT + PID".

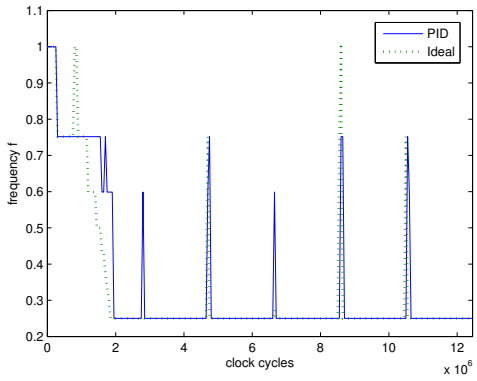


Fig. 10: Simulation snapshot of *Blackscholes* showing DVFS frequency over time, Est. AMAT+PID vs. Ideal.

PARSEC benchmarks, the proposed approach achieves 33% NoC and LLC dynamic power reduction, with only 5% degradation on AMAT, correlated to a $\sim 2.5\%$ system performance drop. We show that the technique has low computation and communication overheads and is practical to implement.

In future research, we will validate this approach on multi-application benchmarks and architectures with significantly more cores. While we believe that future VLSI technology will allow rapid DVFS changes, such as analyzed here, in the future we will explore how latency in Voltage/Frequency changes could impact our results.

Acknowledgments

We would like to thank Pritha Ghoshal, Mark Browning and David Kadjo for their helpful discussions. This research is supported by a gift from Intel Corp.

References

[1] D. Abts, M. R. Marty, P. M. Wells, P. Klausler, and H. Liu, "Energy proportional datacenter networks," in *ISCA*, 2010, pp. 338–347.
[2] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC Benchmark Suite: Characterization and Architectural Implications," in *PACT*, 2008.
[3] M. S. Fadali and A. Visioli, *Digital control engineering: analysis and design*, 1st ed. Academic Press, 2009.
[4] P. Gratz, B. Grot, and S. W. Keckler, "Regional Congestion Awareness for Load Balance in Networks-on-Chip," in *HPCA*, 2008, pp. 203–215.
[5] P. Gratz and S. W. Keclker, "Realistic Workload Characterization and Analysis for Networks-on-Chip Design," in *CMP-MSI*, 2010.
[6] P. Gratz, C. Kim, K. Sankaralingam, H. Hanson, P. Shivakumar, S. W. Keckler, and D. Burger, "On-Chip Interconnection Networks of the TRIPS Chip," *IEEE Micro*, vol. 27, pp. 41–50, 2007.
[7] L. Guang, E. Nigussie, L. Koskinen, and H. Tenhunen, "Autonomous DVFS on supply islands for energy-constrained NoC communication," *Lecture Notes in Computer Science: Architecture of Computing Systems*, vol. 5455/2009, pp. 183–194, 2009.
[8] A. Iyer and D. Marculescu, "Power efficiency of voltage scaling in multiple clock multiple voltage cores," in *ICCAD*, 2002, pp. 379–386.
[9] H.-S. Jung and M. Pedram, "Supervised learning based power management for multicore processors," *TCAD*, vol. 29, no. 9, pp. 1395–1408, Sep. 2010.
[10] J. Kim, D. Park, T. Theocharides, N. Vijaykrishnan, and C. R. Das, "A Low Latency Router Supporting Adaptivity for On-Chip Interconnects," in *DAC*, 2005.
[11] R. Kumar and G. Hinton, "A family of 45nm IA processors," in *ISSCC*, 2009, pp. 58–59.
[12] S. Ma, N. Enright Jerger, and Z. Wang, "Dbar: an efficient routing algorithm to support multiple concurrent applications in networks-on-chip," in *ISCA*, 2011, pp. 413–424.
[13] R. Marculescu, U. Y. Ogras, L.-S. Peh, N. E. Jerger, and Y. Hoskote, "Outstanding research problems in NoC design: system, microarchitecture, and circuit perspectives," *TCAD*, vol. 28, no. 1, pp. 3–21, Jan. 2009.
[14] A. K. Mishra, R. Das, S. Eachempati, R. Iyer, N. Vijaykrishnan, and C. R. Das, "A case for dynamic frequency tuning in on-chip networks," in *MICRO*, 2009, pp. 292–303.
[15] U. Y. Ogras, R. Marculescu, and D. Marculescu, "Variation-adaptive feedback control for networks-on-chip with multiple clock domains," in *DAC*, 2008, pp. 614–619.
[16] S. Prabhu, B. Grot, P. V. Gratz, and J. Hu, "Ocin-tsim: DVFS aware simulator for NoCs," in *SAW-2*, 2010.
[17] A. Rahimi, M. E. Salehi, S. Mohammadi, and S. M. Fakhraie, "Low-energy GALS NoC with FIFO-monitoring dynamic voltage scaling," *Microelectronics Journal*, vol. 42, no. 6, pp. 889–896, Jun. 2011.
[18] E. Rotem, A. Mendelson, R. Ginosar, and U. Weiser, "Multiple clock and voltage domains for chip multi processors," in *MICRO*, ser. MICRO, 2009, pp. 459–468.
[19] G. Semeraro, D. H. Albonesi, S. G. Dropsho, G. Magklis, S. Dwarkadas, and M. L. Scott, "Dynamic frequency and voltage control for a multiple clock domain microarchitecture," in *MICRO*, 2002, pp. 356–367.
[20] L. Shang, L. Peh, and N. K. Jha, "Power-efficient interconnection networks: dynamic voltage scaling with links," *IEEE Computer Architecture Letters*, vol. 1, no. 1, 2002.
[21] A. Singh, W. J. Dally, A. K. Gupta, and B. Towles, "GOAL: A Load-Balanced Adaptive Routing Algorithm for Torus Networks," in *ISCA*, 2003.
[22] A. Singh, W. J. Dally, B. Towles, and A. K. Gupta, "Globally Adaptive Load-Balanced Routing on Tori," *IEEE Computer Architecture Letters*, vol. 3, no. 1, p. 2, 2004.
[23] S. W. Son, K. Malkowski, G. Chen, M. Kandemir, and P. Raghavan, "Integrated link/CPU voltage scaling for reducing energy consumption of parallel sparse maxtrix applications," in *IPDPS*, 2006.
[24] B. Towles, W. J. Dally, and S. Boyd, "Throughput-centric Routing Algorithm Design," in *Symposium on Parallel Algorithms and Architectures*, 2003, pp. 200–209.
[25] 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, pp. 248–259.
[26] A. W. Yin, L. Guang, P. Liljeberg, P. Rantala, E. Nigussie, J. Isoaho, and H. Tenhunen, "Hierarchical agent architecture for scalable NoC design with online monitoring services," in *NoCArc*, 2008, pp. 58–63.