2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)

Fine-Tuning the Active Timing Margin (ATM) Control Loop for Maximizing Multi-Core Efficiency on an IBM POWER Server

Yazhou Zu* Daniel Richins* Charles R. Lefurgy† Vijay Janapa Reddi*‡

*The University of Texas at Austin
{yazhou.zu, drichins}@utexas.edu

†IBM Research
lefurgy@us.ibm.com

‡Harvard University
vj@eecs.harvard.edu

***Abstract*—*Active Timing Margin (ATM)* is a technology that improves processor efficiency by reducing the pipeline timing margin with a control loop that adjusts voltage and frequency based on real-time chip environment monitoring. Although ATM has already been shown to yield substantial performance benefits, its full potential has yet to be unlocked. In this paper, we investigate how to maximize ATM's efficiency gain with a new means of exposing the inter-core speed variation: fine-tuning the ATM control loop. We conduct our analysis and evaluation on a production-grade POWER7+ system. On the POWER7+ server platform, we fine-tune the ATM control loop by programming its *Critical Path Monitors*, a key component of its ATM design that measures the cores' timing margins. With a robust stress-test procedure, we expose over 200 MHz of inherent inter-core speed differential by fine-tuning the per-core ATM control loop. Exploiting this differential, we manage to double the ATM frequency gain over the static timing margin; this is not possible using conventional means, i.e. by setting fixed $<v, f>$ points for each core, because the core-level $<v, f>$ must account for chip-wide worst-case voltage variation. To manage the significant performance heterogeneity of fine-tuned systems, we propose application scheduling and throttling to manage the chip's process and voltage variation. Our proposal improves application performance by more than 10% over the static margin, almost doubling the 6% improvement of the default, unmanaged ATM system. Our technique is general enough that it can be adopted by any system that employs an active timing margin control loop.**

***Keywords*-Active timing margin, Performance, Power efficiency, Reliability, Critical path monitors**

I. Introduction

Active Timing Margin (ATM) is a technique to improve microprocessor power efficiency as transistor scaling comes to its end. By dynamically setting the pipeline timing margin according to the chip's runtime environment, the timing margin provisioned by ATM matches the pipeline's real-time needs; overprovisioning is reduced compared to the conventional static timing margin model where a fixed amount of margin is added to target worst-case conditions, such as severe voltage or temperature variation [1]–[3]. To trim down timing margin, ATM uses a control loop for each core that monitors the pipeline's cycle-level timing margin consumption with canary circuits and tweaks the chip's voltage and frequency at fine granularity. Because of its lucrative efficiency improvements and amenity to implementation, ATM has been adopted widely [4]–[11].

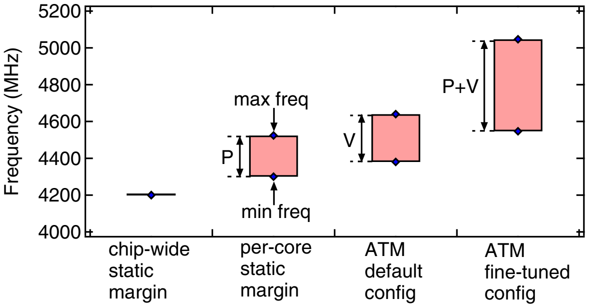


Figure 1: Fine-tuning ATM exposes both process (P) and voltage (V) variation, and improves frequency compared with the default ATM config and the per-core $<v, f>$ static margin setpoints.

In this paper, we maximize ATM's efficiency gain by exploiting each core's own silicon speed and each application's characteristics. *ATM's per-core configurable control loop provides a new opportunity to expose the inter-core speed variation and to derive more performance gain than the conventional multicore process variation*, i.e., calibrating static frequency levels separately for each core [12]–[16].

In the conventional approach, the per-core $<v, f>$ setting relies on static margins and thus requires guarding against worst-case voltage variation, such as the di/dt effect and DC voltage drop across the chip's power delivery path, each of which can consume 3% of the V_{dd} [17]. But because ATM can handle these adaptively, it gains performance by exploiting the inherent inter-core variation in the processor.

In Fig. 1, we illustrate the performance enhancement and heterogeneity exposed by "fine-tuning" the ATM control loop for each core. On the tested POWER7+ platform, we (re)configure ATM via its Critical Path Monitors (CPMs). The CPM is the chip's programmable canary circuit that measures the timing margin [4], [18]. Interfaces similar to the CPMs exist on other ATM systems for test-time calibration of margin measurement accuracy and for configuring margin reduction aggressiveness [9], [19], [20]; an example is the Power Supply Monitor (PSM) on AMD processors [7].

Fig. 1 exposes the pros and cons of the different ap-

2378-203X/19/$31.00 ©2019 IEEE
DOI 10.1109/HPCA.2019.00031

proaches. Starting with the baseline case where there is no ATM mode in the chip, under a chip-wide static margin (i.e., first bar), all cores have a fixed frequency of 4.2 GHz.

Setting the static margin for each core (second bar) with fixed $<v, f>$ improves performance by exposing the fast cores; we estimate the fastest cores can run around 4.5 GHz, based on prior art's voltage noise characterization [17].

Next, the default ATM (third bar) carefully programs each CPM to provide uniform core performance, following the conventional contract between processors and users. When idle, all cores run near 4.6 GHz, higher than static margin's fastest cores because of ATM's highly effective mitigation of di/dt effects [4]. However, when high power workloads are run, the induced DC voltage drop across the power delivery grid can create long-term steady degradation of the supply voltage delivered, eroding timing margin and reducing ATM's frequency gain [17], which lowers the worst-case performance to around 4.4 GHz. Setting fixed $<v, f>$ points for each core requires that this worst-case be guarded against, whereas ATM handles it adaptively and frequency only suffers when power consumption is high.

Fine-tuning (fourth bar) at the per-core ATM control loop level exposes similar inter-core speed variation as static per-core $<v, f>$ setpoints, but it provides much higher performance under typical conditions because of ATM's adaptive margin provisioning capability. Fine-tuning ATM also removes any margin left not trimmed in the default system, which further pushes processor efficiency to the extreme. For instance, when the chip is idle, power consumption and DC voltage drop is minimal, pushing the fastest core to nearly 5 GHz, 10% higher than the fastest static margin core.

While fine-tuning the ATM control loop provides high frequency gain, it exacerbates variability and induces performance predictability issues. In the worst case, e.g., when DC voltage drop is maximized while running eight high power `daxpy` threads, the slowest core, which runs at 4.7 GHz under idle conditions, slows down to 4.5 GHz, a 500 MHz drop from the fastest 5 GHz case. Thus, application performance can vary widely, depending on the core chosen for execution and any co-located workloads.

Fig. 2 shows the variation of `SqueezeNet`, a compute-bound image classification inference task. Under the static margin, the 4.2 GHz fixed frequency delivers consistent inference latency of 80 ms regardless of co-running workloads. Under fine-tuned ATM, inference latency improves by 7.5% to 15% depending on workload schedules. The best schedule, which puts `SqueezeNet` onto the fastest core and leaves other cores idle, increases frequency to 4.9 GHz and reduces latency to 68 ms, 2X the performance gain of the worst schedule, which puts `SqueezeNet` onto the slowest core and co-locates high power workloads with it.

Inspired by the benefits in Fig. 2, in this paper we detail how to fine-tune ATM at the core level to robustly reveal each core's performance limit and to expose inter-core speed differences. We perform extensive hardware measurement to analyze ATM operating limits under different application scenarios, which leads to a low-overhead solution for deploying ATM systems with their highest speed at scale, while delivering controllable application performance in the presence of the exposed process and voltage variation. We present a software solution to proactively fine-tune and manage ATM. In summary, we make the following contributions:

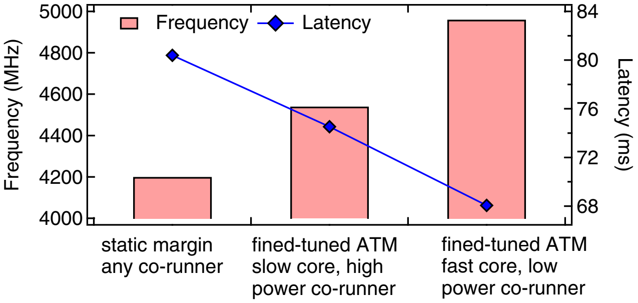


Figure 2: `SqueezeNet` inference latency on a POWER7+ core under different timing margin settings. Aggressively fine-tuning ATM, and co-locating it with "friendly" low-power applications enhance performance.

- We propose fine-tuning the ATM control loop at the individual core level to automatically expose inter-core speed variation and improve performance.
- We analyze the operating limits of ATM under various conditions, based on which we propose a test-time procedure to robustly fine-tune per-core ATM.
- We develop workload scheduling and throttling to controllably boost application performance by over 10% in the presence of significant speed variability.

The outline is as follows. Sec. II explains ATM and its implementation on our POWER7+ experimental platform. Sec. III describes how to fine-tune ATM to improve performance and expose speed variation. Sec. IV-VI present our analysis on fine-tuning ATM to its operating limits under different application scenarios. Sec. VII introduces our proposal to deploy and manage fine-tuned ATM systems for predictable performance. Sec. VIII compares prior art and Sec. IX concludes the paper.

II. Active Timing Margin Background and Setup

Microprocessors must ensure correct execution under exceptional circumstances, such as big di/dt events or extreme temperature conditions that significantly slow down circuits [3], [23], [24]. The traditional *static timing margin* approach adds a fixed time buffer implemented as a higher voltage which ensures that, even under exceptional cases, the signals still propagate by the end of each clock cycle. However, exceptional events occur very rarely, making static margin unnecessary most of the time [25], [26]. Consequently, the excess voltage is usually wasted.

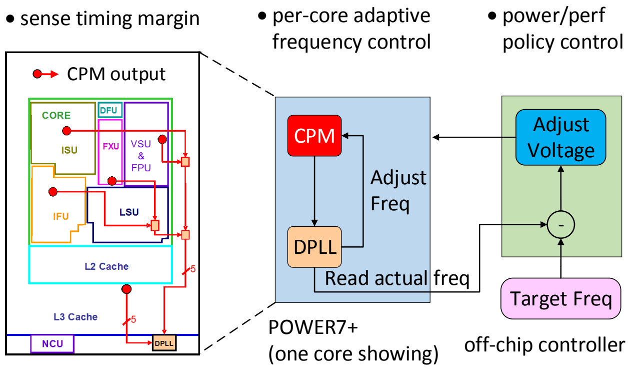


Figure 3: In POWER7+, Critical Path Monitor (CPM), Digital Phase Locked Loop (DPLL), and off-chip voltage controller work synergistically to let ATM provide just enough margin [21].

ATM addresses the waste. ATM control adapts the timing margin dynamically to meet the needs of the chip environment [3], [4], [6], [8], [10]. The buffered voltage in a static margin is converted to either power savings (a lower voltage) or performance enhancements (a higher frequency).

For the purposes of our ATM study, we use a two-socket POWER7+ server. Each processor features eight out-of-order cores. The server runs Red Hat 6.4 operating system, and all workloads are compiled with GCC 4.8.5. We study ATM on two eight-core POWER7+ processors. The POWER7+ supports efficiency management both in coarse-grained dynamic voltage and frequency scaling (DVFS), which adjusts p-states from 2.1 GHz to 4.2 GHz by controlling V_{dd} with a static timing margin, and in fine-grained ATM that further tunes V_{dd} and frequency around each p-state. Each processor features eight out-of-order cores with four-way simultaneous multi-threading (SMT) for 64 logical cores. We conduct our study on both processors to sweep more cores and identify trends across different chips.

Like other ATM processors, POWER7+'s ATM consists of three parts: (1) the timing margin sensor, (2) the adaptive frequency control loop, and (3) the overclocking/undervolting policy controller, as depicted in Fig. 3. We focus on the first two parts and disable undervolting to convert ATM's reclaimed timing margin into frequency gain on each core.

Timing Margin Sensor is the foundation of ATM. It monitors the excess timing margin either by directly measuring the available slack in a clock cycle [22] or by measuring circuit delay time [7]. It outputs integer measurements every cycle to provide real-time chip monitoring. In practice, a set of timing margin sensors are usually dispersed across the chip to capture spatial variation of different parasitic effects, such as temperature variation or core-focused di/dt effect.

In the POWER7+, the timing margin sensor is implemented as a Critical Path Monitor (CPM). A CPM mimics real circuit delay with a set of synthetic paths and monitors the timing slack after the synthetic paths complete execution. There are five CPMs in each core, integrated inside the instruction fetch unit, instruction scheduling unit, fixed point unit, floating point unit, and last level cache (Fig. 3). The worst of the five CPM measurements is reported every cycle.

Adaptive Frequency Control Loop is a loop that operates between the timing margin sensor and an agile clock generator. Each cycle, the measured timing margin is sent to the clock generator, which compares the margin against a threshold and adjusts the clock frequency at short intervals.

An emergency timing event, such as a fast-occurring di/dt effect, can drive the margin lower than the threshold (a timing margin violation). In response, the clock generator compensates by gating the clock for one cycle or—for a lower penalty—by reducing the clock frequency. The round trip time of the feedback loop must be less than several cycles to deal with fast occurring voltage noise [10]. Under typical conditions, the margin is higher than the threshold.

To allow room for flexibility and maximizing effectiveness, the POWER7+ features a control loop for every core. It uses a digital phase-locked loop (DPLL) to slew frequency at a fine granularity, enabling timely feedback [4], [27].

Off-chip Voltage Control determines whether to turn ATM's reclaimed margin into power savings via undervolting or into higher performance via overclocking. Often the goal is to reach a certain frequency target and convert the remaining timing margin into power savings. The POWER7+ off-chip controller reads a 32 ms sliding window average frequency of the slowest core of a chip and decides how much V_{dd} can be reduced for the entire chip without hampering the user-specified frequency target.

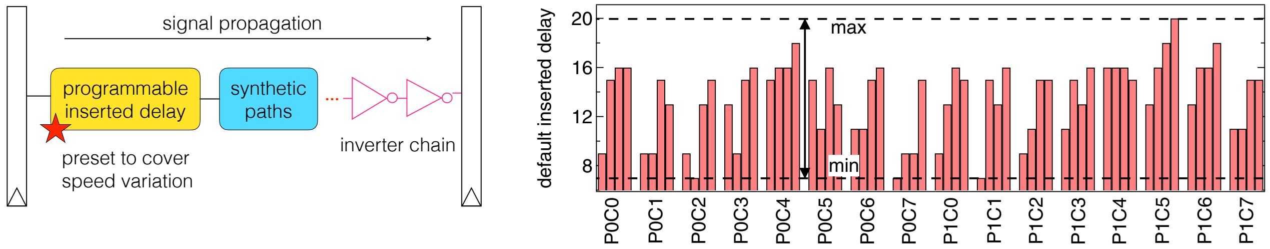
We convert all of ATM's reclaimed timing margin into frequency and keep V_{dd} unchanged, which bypasses the restriction on undervolting wherein a chip's worst-case core limits the amount of undervolting. Overclocking allows each core to independently adapt to its conditions and can fully expose a chip's inter-core speed differential, potentially producing more performance benefit. We let ATM boost each core's frequency at V_{dd} 1.25 V, the 4.2 GHz P-state.

III. Fine-tuning Core-level Active Timing Margin Operation

We explain how to fine-tune ATM operation in each core to extract more margin and increase frequency. To understand ATM's limit when aggressively fine-tuning its operation, we propose a systematic procedure to characterize how the processor behaves under different scenarios. The insights we gain when executing this procedure are instrumental for deploying fine-tuned ATM systems at scale.

A. Programming CPMs to Reconfigure Margin Reclamation

We configure the POWER7+'s Critical Path Monitors (CPMs) to fine-tune ATM's margin reclamation behavior. By design, CPMs are programmable to set how aggressively ATM trims the margin and, more importantly, to cover speed



(a) CPM has three cascaded parts: programmable inserted delay, synthetic paths, and inverter chain.

(b) Pre-set inserted delay of the CPMs in two POWER7+ chips.

Figure 4: (a) CPM's inserted delay is a programmable offset that adjusts timing margin measurement by selecting the number of inverters used [18], [22]. By default, it is pre-set by manufacturers to cover speed variation at different locations. (b) The wide variation of pre-set inserted delays in the two tested POWER7+ chips indicates significant process variation (where P0C0 means Processor 0 Core 0).

variation and deliver uniform performance to users. We use this interface to fine-tune each core's ATM control loop.

Fig. 4a shows a CPM uses three stages to measure margin: (1) inserted delay, (2) synthetic paths, and (3) an inverter chain. The inserted delay is a configurable circuit. A user can specify the number of inverters a signal passes through to select its timing delay length. The synthetic path simulates a pipeline circuit's delay with a set of paths, including AND, OR, and XOR gates and wires. The final inverter chain quantifies the time left after the signal propagates past the inserted delay and synthetic path by counting the number of inverters a signal passes. The inverter count is a CPM's final output and is sent to the DPLL for clock adjustment.

Before a processor is shipped, each CPM's inserted delay is pre-set at test-time with a default value that serves as extra "protection" for the control loop to function robustly. The pre-set delay makes CPMs report less margin than they could have, leaving some margin not trimmed as protection. The pre-set delay also smooths out the speed differences between different corners of a chip by adding more delay to fast corners in order to fill the empty time after a circuit finishes switching and adding less delay to slow corners.

Fig. 4b shows the preset inserted delays in each core of the two POWER7+ chips (we exclude CPMs in the LLC because it lies in a different clock domain). Intuitively, each unit of the delay represents some amount of timing. Under static margin at 4.2 GHz, reducing the inserted delay by one step lets the CPM detect one to three units more timing margin, equivalent to the speed variation caused by 20-60 mV V_{dd} difference [17], [18]. The magnitude of the delay shows the amount of "protection" built into the default ATM system. The pre-set inserted delays range from 7 to 20, nearly a 3X range, indicating significant silicon speed variation.

We fine-tune ATM operation by re-programming CPMs' inserted delays with smaller values, which makes the control loop perceive more margin and generate a higher frequency. To reduce the search space, we change all CPMs in each core with the same step size. In the POWER7+, this is done by sending specialized commands to the service processor.

Fig. 5 shows, for four example cores, how ATM converts more margin into frequency as the CPM inserted delay is reduced. The default delay (0 steps of reduction) makes all core run around 4.6 GHz, delivering uniform performance because the pre-set delay smooths out speed variation. Programming the inserted delay to a smaller value (higher delay reduction) decreases the time to the end of the synthetic path, leaving more margin to be counted by the inverter chain. The DPLL loop harnesses the excess margin by overclocking. We find reducing the inserted delay can safely push some cores' frequency to over 5 GHz, a 20% improvement over the static timing margin baseline.

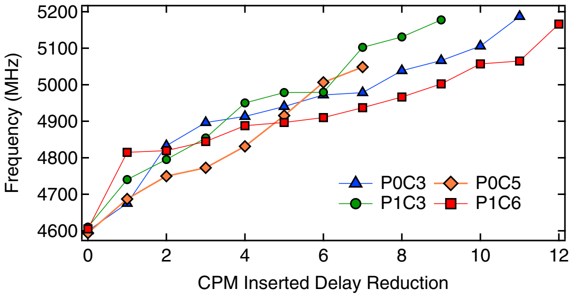


Figure 5: We fine-tune each core's ATM control loop by reducing the core's CPM inserted delay, which makes the control loop perceives more margin and automatically increases frequency.

B. *Characterizing ATM Limits*

To reliably use a fine-tuned ATM system's performance benefit, we follow a systematic methodology to analyze ATM system's operation limit under different scenarios. Fig. 6 outlines our procedure. We analyze an ATM chip on a per-core basis. An idle system is our starting point for conducting our analysis; micro-benchmarks (uBench) cover major paths in a core; and single-threaded benchmarks representing real use cases. Our insights gathered in these analyses serve as the foundation for devising a low-cost automatic ATM fine-tuning procedure that helps enable shipping and deploying fine-tuned ATM systems at scale.

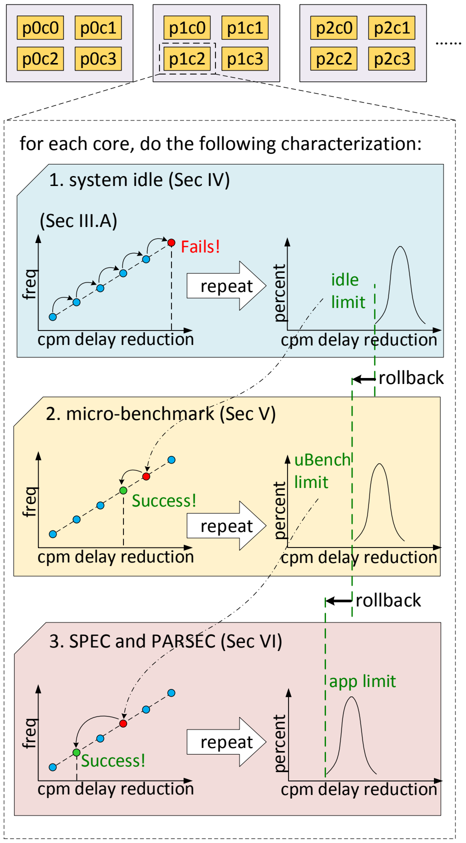


Figure 6: Our ATM characterization methodology iterates over each core and follows a step-by step approach, going from the simplest system idle scenario to the complex real-world workloads.

System Idle Running background operating system tasks, an idle system imposes the least amount of stress on the processor. And as such, understanding each core's ATM operating limits under system idle provides us with valuable insight into inherent the core-to-core level differences.

Micro-benchmarks (uBench) Traditionally, micro-benchmarks are used to measure the performance of individual processor modules, such as the branch predictor, floating point unit, and caches. For ATM fine-tuning, micro-benchmarks' serve to touch only one part of the core, avoiding complex microarchitectural interactions. We thus use uBench to conduct a more comprehensive analysis of core-to-core microarchitecture level variation.

Realistic Workloads For the final step, we analyze the system with complex applications from SPEC CPU 2017 and PARSEC. These benchmarks cover a wide spectrum of program space in the real world and have diverse architecture behavior [28], [29]; hence they can touch more corner-case timing paths or create more active di/dt effects than uBench, all of which threatens the safe execution of aggressively fine-tuned ATM. The single-threaded workloads help identify application-, chip-wide-, and core-level heterogeneity.

In each of the above setups, failure may occur as a result of a timing violation, manifested as an abnormal application termination (e.g., segmentation fault), silent data corruption (SDC), or a system crash. For SDC related errors, we rely on SPEC and uBench's built-in result checking tools for guaranteeing execution correctness. All these failures can occur because either the CPM's delay has become so short that it does not capture real circuit delays or system noise events, such as the di/dt effect, overwhelm the control loop's ability to respond in time. Because the effects that cause ATM failure might be not fully deterministic, we repeat the failure experiments in each setup multiple times to produce a distribution of ATM operating limits. We expect the distributions to be tight because timing violations are not entirely random. The distributions provide insight into ATM margin reclamation capabilities, so we study them in detail.

Our methodology progresses through increasing workload complexity. We often need to roll back the CPM delay setting that was successful in a previous, less complex setup to a less aggressive point, reflecting a workload setup's unique impact on ATM's operation. Although the worst-case might determine practical ATM configurations for real applications, the middle point analysis provides useful insights on what affects the fine-tuned ATM control loop's operation.

IV. Idle System Characterization

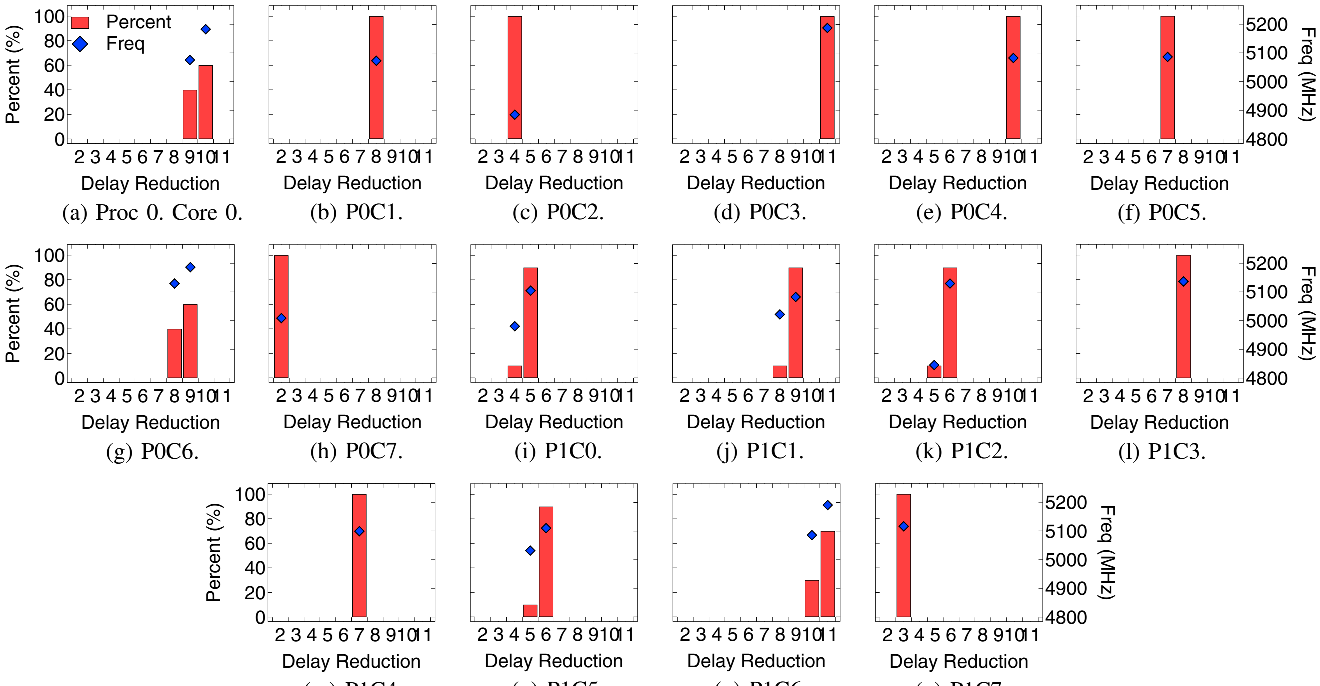
Understanding ATM's margin reclamation limits in an idle system sets a starting point for further complex analysis. With no application code running, the system exerts minimal stress on ATM's reconfigured control loop, enabling us to use ATM to expose the silicon's inherent maximum speed.

Running only the operating system, we build a distribution of the most aggressive yet safe CPM configuration points for each core, depicted in Fig. 7 by the amount of CPM delay reduction from the chip's default setting, along with the resulting frequencies. As expected, the distributions are tight, covering no more than two configurations. Each core's *idle limit* is the lowest (most conservative) CPM delay reduction plotted, e.g. 9 in Fig. 7a. These are summarized in Table I.

The different core-to-core idle limits reveal lucrative performance potential of fine-tuning ATM aggressively for each core (Sec. IV-A), and the significant core-to-core performance variation (Sec. IV-B) which is partly caused by the non-linearity in CPM measurement capability (Sec. IV-C).

A. Significant Performance Potential

For most cores, the inserted delay can be aggressively reduced, making ATM's control loop see more timing margin for reclamation. As Fig. 7 shows, more than half the cores (e.g., P0C0 and P0C1) can tolerate reductions of at least seven steps of CPM inserted delay, elevating frequencies to over 5000 MHz: a 7% improvement over default ATM's 4600 MHz and a 20% improvement over static margin's 4200 MHz baseline, showing fine-tuned ATM can substantially improve performance.



(a) Proc 0. Core 0. (b) P0C1. (c) P0C2. (d) P0C3. (e) P0C4. (f) P0C5.

(g) P0C6. (h) P0C7. (i) P1C0. (j) P1C1. (k) P1C2. (l) P1C3.

(m) P1C4. (n) P1C5. (o) P1C6. (p) P1C7.

Figure 7: The most aggressive CPM delay reduction that ensures system idle safety distributes over a narrow range (red bar, left y axis). The lower bound of each core's distribution is the core's *idle limit*, usually entailing over 5000 MHz frequency (blue mark, right y axis).

	P0C0	P0C1	P0C2	P0C3	P0C4	P0C5	P0C6	P0C7	P1C0	P1C1	P1C2	P1C3	P1C4	P1C5	P1C6	P1C7
idle limit	9	8	4	11	10	7	8	2	4	8	5	8	7	5	10	3
uBench limit	9	8	4	10	9	7	8	2	4	8	5	5	6	4	10	2
thread normal	8	7	4	9	8	6	7	2	3	7	5	4	5	3	8	2
thread worst	6	6	3	6	6	5	5	2	3	3	5	3	3	2	6	2

Table I: ATM reconfiguration limits under system idle, uBench, and real-world application. Data is collected on two eight-core (C) POWER7+ processors (P). ATM limits are reflected as the number of stepped reduced from CPM's default inserted delay configuration.

B. Exposed Inter-core Frequency Variation

Programming the CPM to change ATM operation yields different frequency levels for each core, despite the performance improvement. E.g., at the idle limit P1C2 runs at about 4850 MHz but P0C3 achieves about 5200 MHz. Even within a chip, there is a wide range (e.g., P0C2 and P0C3). The core-to-core frequency variation is essential for application performance management, which we discuss later.

The core to core differences are understood to be a result of manufacturing process variations [15], [16], i.e., some core's circuits are faster due to imperfection in the lithography process. For instance, as Fig. 7 shows, P0C3 can safely reduce its CPM delay by 11 steps, while P0C7 can only mitigate its delay by two, reflecting the varying amount of timing margin available for reclamation, which is caused by the two cores' speed difference.

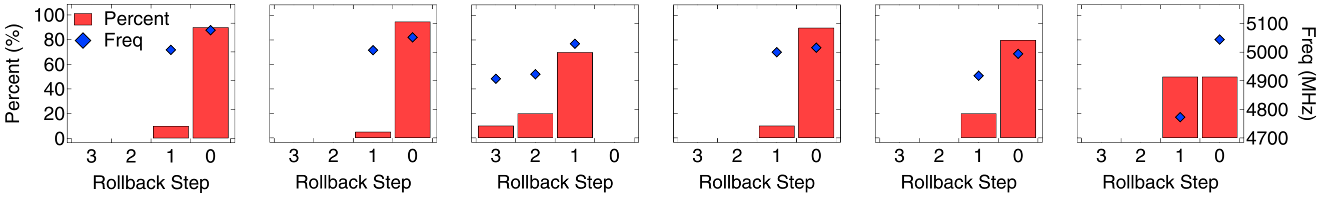
However, because on the POWER7+ each core's performance is unlocked via ATM control loop's automatic harness of available timing margin, the ATM control loop also plays a critical role in the inter-core performance variation.

C. Nonlinearity of CPM Configuration

The CPM inserted delay's configurable inverter chain is designed to have linear timing delay graduation for timing margin measurement. However, the manufacturing process makes it have non-linear graduation when configured to measure timing margin. The non-linearity magnifies the inter-core performance heterogeneity.

The inserted delay's non-linear configuration manifests as significant idle limit variation between cores. Consider P0C4 and P1C7, which are both able to increase frequency from 4600 MHz to 5100 MHz but do so with very different CPM changes: P0C4 reduces the delay by ten steps, while P1C7 only needs two steps. Hence, although the two cores have similar excess timing margins, P0C4's CPM encodes smaller timing delays in each step than P1C7.

Within each core, CPM's non-linearity makes the timing margin encoded by one CPM delay step vary. Fig. 5 shows that P1C6's frequency increases by over 200 MHz when going from step zero to one, jumping from the baseline 4600 MHz to over 4800 MHz. But in going from step one to



(a) P0C3, `daxpy`
(b) P0C4, `daxpy`
(c) P1C3, `stream`
(d) P1C4, `stream`
(e) P1C5, `coremark`
(f) P1C7, `coremark`

Figure 8: For the six cores shown above, CPM inserted delay needs to be rolled back from its idle limit for the uBench to run correctly, indicating the core's idle limit is too aggressive and fails to capture some long delay paths in the core.

two, there is an almost negligible change in frequency. Similarly, the frequency is nearly unchanged when increasing the CPM delay reduction from step five to six for P1C3, but reducing the delay by one additional step (i.e., going from six to seven) increases the frequency by over 100 MHz.

As another example, in Fig. 7k, reducing P1C2's CPM delay by six is too aggressive and can crash the system; rolling back its delay by one step ensures safety but at the cost of 300 MHz. P1C1 (Fig. 7j) similarly needs its CPM delay reduction rolled back by one step (from nine to eight) for safe operation but at the cost of only 100 MHz. Though P1C2 could operate safely at a higher frequency, the large CPM jump forces the 300 MHz drop and amplifies the differences between the two cores.

In summary, the non-linear configuration of the CPM and ATM control loop demands that fine-tuning ATM operation be carried out carefully on a per-core basis because no single CPM configuration works uniformly for all cores.

V. Micro-bench Characterization

While idle system characterization reveals insights on the performance benefits and the inter-core variation of a fine-tuned ATM multicore, it does not evaluate the system's behavior under stress from real-world applications. Before using more complex applications, we use micro-benchmarks (uBench) as a valuable tool that controls program behavior to analyze individual processor components [30]. Because uBench imposes more stress than idling, the CPM configuration tends to be more conservative, creating a practical point for processor deployment in the real world.

A. Workload Selection

We evaluate system behavior under aggressive ATM reconfiguration using three uBench programs. These programs collectively cover all main parts of the microarchitecture, as well as the dispersed CPMs in a core.

We use `coremark` [31] to stress the core's control, branch, and integer units; `daxpy` to stress the floating point unit; and `stream` [32] for its ability to generate cache misses and exercise the load-store unit. Prior work has used such benchmarks to exercise the functional units and validate the ATM [4], [21]. We check the programs' run results to evaluate processor execution correctness. All incorrect runs manifest as system crashes or abnormal application exits.

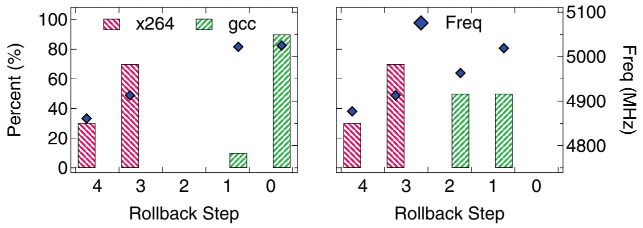
Using these benchmarks ensures we challenge a reconfigured ATM by touching more paths than system idle. Meanwhile, these uBench programs create little system noise, especially the di/dt effect. They have controlled, smooth program behaviors and avoid complex microarchitectural activity such as periodic pipeline flush, which is the root cause of workload-induced voltage droops [1], [2], [25], [26], [33]. The di/dt effect is dangerous for aggressively reconfigured ATM because its fast drooping voltage can prevent the control loop from engaging in time [10], resulting in application failure.

B. Why Some Cores Fail

We start the uBench characterization from the idle limit because it is the point that sustains stable system state. If this initial starting point fails, the CPM inserted delay is rolled back to have a longer timing delay to make ATM harness timing margin more conservatively until the program runs correctly. We find most cores' idle limits sustain correct uBench execution, which indicates they can safely accommodate the major paths activated by the instructions used by uBench programs.

For the server's two physical processors, uBench characterization exposes six cores that fail for the three programs. Fig. 8 shows the distributions of reintroduced delays for these cores, using the "rollback steps" relative to the idle limit, which reflects the stress impact from uBench program execution compared with system idle. For those six cores, rollback ranges from one to three steps and sustains all uBench workloads.

All three programs, despite their different characteristics, show similar behaviors on the six problematic cores. The implication is that the microarchitecture blocks that limit ATM fine-tuning are the common structures used by all programs, such as instruction fetch and scheduling, rather than specific modules stressed by each application (e.g., the FP unit). We also find *uBench limit* sustains voltage and power stress-test, which will be detailed later in this paper. We therefore use the *uBench limit* as a reference point for further characterization with realistic applications.



(a) P0C3

(b) P1C6

Figure 9: `x264` stresses ATM more heavily and needs a more conservative CPM configuration compared to `gcc`, as indicated by the larger CPM rollback that is required for `x264` over `gcc`.

VI. Realistic Workload Characterization

To run real applications, today's ATM systems add some protection margin to CPM's uBench limit configuration [4]. To conservatively guarantee execution correctness, the added margin can be up to 50% of the static guardband. But this leaves room for improvement as demonstrated by the 2X frequency gain during our system idle characterization.

However, adding additional guardband as a conservative precaution ignores the application-dependent behavior and can waste valuable performance benefit. In this section, we profile with a variety of integer and floating point workloads from SPEC CPU 2017 and PARSEC 3.0 [34]. We use these workloads because their result-checking tool provides a convenient method for checking execution correctness. Understanding per core ATM operating limits under these heterogeneous workloads offer helpful insights for deploying fine-tuned ATM chips in real-world use cases.

Fig. 9 shows `x264` often requires significant CPM delay rollback from the uBench limit, whereas `gcc` needs relatively little, allowing ATM to more aggressively boost frequency. The rollback reflects an application's unique system noise effects. Configuring ATM for the worst application in all cases, e.g., `x264`, wastes ATM's margin reclamation potential when running more benign workloads. This is the approach taken by today's deployed ATM processors, which still rely on a safety margin as large as 50% of the original static guardband [4].

To get a complete picture of the behavior of aggressively configured ATM cores on different workloads, we profile CPM rollback from the uBench limit for all $< app, core >$ pairs in Fig. 10. We use the weighted average CPM rollback as it quantifies the application's unique stress level. Two applications may have quite different delay reduction distributions even when they show the same lower bound in their CPM delay profile.

From the individual rows in Fig. 10, we see that each workload imposes a different amount of stress but does so consistently across cores. For instance, `x264` and `ferret` needs much more conservative ATM setting than `gcc` and `leela`, indicating these workloads exert higher pressure on ATM's control loop.

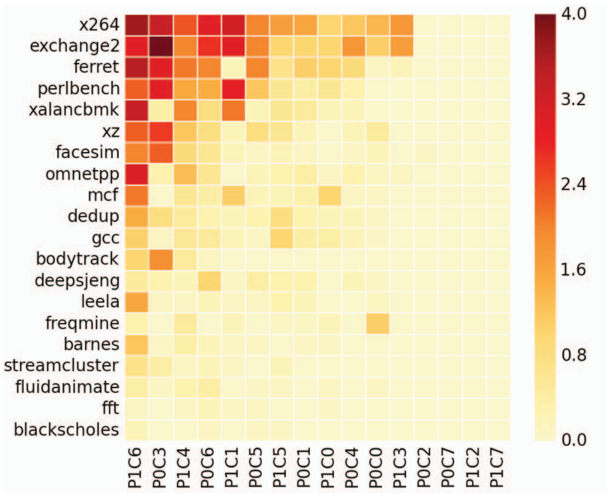


Figure 10: Application's average CPM delay rollback from the core's uBench limit. The top workloads stress ATM heavily and need more delay rollback for less aggressive margin reclamation.

From the individual columns in Fig. 10, we see that different cores exhibit varying levels of "robustness," where we define robustness as the immunity to CPM rollback from the core's CPM uBench limit. The cores on the right of Fig. 10 have the highest robustness, requiring the least rollback across all applications, indicating their ATM control loops can deal with the system effects of any application. We anticipate they will continue to be robust on untested applications since the profiled workloads already cover different behaviors [28].

The reason for certain applications and cores being more vulnerable after aggressive ATM fine-tuning is a combination of the core's inherent speed and the running application's characteristics. We conducted a best-effort static instruction analysis on the applications and concluded that more detailed insight into the running instructions is needed to predict each application's best-fit CPM setting on each core. For instance, `gcc` covers a much richer set of instructions than `exchange2`, likely touching more corner timing paths, yet stresses ATM much less. As another example, `x264` has similar performance counter profiles as `leela`, but their rollback requirements differ substantially. We therefore defer the root-cause analysis and the prediction of applications' heterogeneous CPM configuration to future work and focus on the variations already exposed.

To summarize, in Table I we denote by *thread-worst* the most conservative CPM configuration limit for all profiled workloads. It represents the most severe application stress in our profiling. We denote by *thread-normal* a more aggressive setting that supports most medium and light applications.

VII. Managing a Fine-tuned ATM System

In this section, we discuss how to deploy and manage a fine-tuned ATM system in the field in the presence of significant variability. Fine-tuning improves application performance because frequency is higher. However, pushing ATM to its operating limit requires an execution correctness guarantee, and the varying frequency levels of different cores and application scenarios create an obstacle for the processor in delivering a promised performance level to end users. Hence, we discuss how to determine CPM settings for each core to robustly expose variation and show how to schedule and throttle co-running workloads to deliver predictable performance for latency sensitive critical applications.

A. *Deploying Fine-tuned ATM Configuration*

The insights we gather from the operating limit study of a fine-tuned ATM system under idle, micro-benchmarks, and realistic workload scenarios are useful for understanding the performance opportunity from exposed inter-core variability, but the overhead of our procedure is high for finding a processor's fine-tuned configuration for real-world deployment.

Because programs have heterogeneous CPM settings on different cores, one can try to predict each application's best CPM setting on each core. However, such a prediction scheme demands perfect prediction accuracy because any misprediction can lead to system failure or incorrect execution. Achieving this accuracy is difficult because it relies on deep knowledge of a program's di/dt behavior as well as the circuit paths touched by the program, all of which derives from the dynamic instruction streams and may incur high overhead [26]. We leave CPM prediction for future work.

Rather than predict CPM settings, we propose a test-time stress-test procedure to identify ATM fine-tuning limits while guaranteeing correctness.[1] During test-time, we iterate over each core and run worst-case workloads, such as a di/dt stressmark [35], [36], power stressmark [37], and ISA test suites to create high voltage noise and high operating temperatures and to cover all potential circuit paths. The combined stress-test finds each core's limit ATM configuration, providing a guarantee of correctness for any realistic workload because, by definition, a stress-test pushes the system beyond the requirements of any other workload.

In our work, we try our best to create a stress-test with a voltage virus that repeatedly and synchronously throttles all cores' instruction issue rate to operate only one out of every 128 cycles while simultaneously running 32 `daxpy` threads. The `daxpy` workloads create high power consumption, raising chip power to 160 W and temperature to 70°C; the issue throttling creates a synchronous power surge across the chip, inducing concurrent di/dt effects from adjacent cores, representing worst-case voltage noise [4], [10]. We recognize that better power stressmarks can be constructed using more systematic procedures [37], but we do not expect power and temperature to be the limiting factors for ATM operation because these are long-term effects and are well within ATM control loop's nanosecond-level response time. Though the realistic workload characterization in Sec. VI covers a variety of instructions, in practice, chip vendors have tailored ISA verification suites that provide wider coverage and execute in less time.

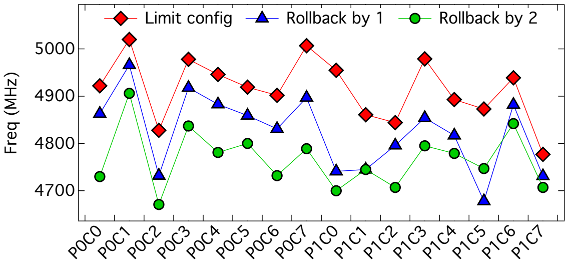


Figure 11: To ensure execution correctness, fine-tuning ATM goes through worst-case stress-test during test time. Vendors can optionally roll back stress-test ATM configurations, providing additional safety guarantee. Either way, speed variability is exposed.

On the two POWER7+ processor chips we tested, the *thread worst* CPM configurations sustain correct execution under all our stressmarks. To provide additional correctness guarantees, the vendor can optionally rollback the stress-test-determined ATM limit by several steps.

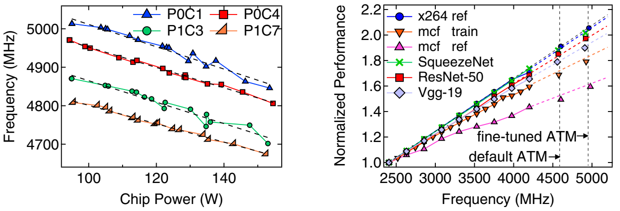
Fig. 11 shows the core frequencies across the two POWER7+ chips after executing the above test-time procedure. At their limit, P0C1 and P0C7 have over 200 MHz speed differential. Rolling back each core's CPM from the limit by one or two steps keeps the same inter-core variation trend and provides an additional safety guarantee. In the management scheme we propose, we will use the limit *thread-worst* configuration, though the conclusions we present and the scheme we propose can be applied to more conservative (rolled back) configuration points.

B. *Predicting Core Frequency*

To manage ATM's performance variability, we develop a predictor that informs frequency and performance for a candidate schedule. We develop this predictor by modeling each core's runtime average frequency $\overline{f}$, as a linear function of the transistors' supply voltage, V_{chip}. Among different dynamic effects, long-term stable effects such as temperature variation and DC voltage drop caused by high power determines core frequency —infrequent, transient di/dt events are handled transparently by the ATM control loop.

Because past research has shown that speed is only modestly affected by temperature [3], we base our model strictly on IR voltage drop. Subtracting the IR voltage drop, which is proportional to current and hence power consumption,

[1] The approach and evaluation presented here is an example of the process we recommend, and not meant to be literally the exact steps to follow. For instance, the stressmarks we use in the paper are different from what we use in production. Nonetheless, the general approach we discuss is useful.



(a) ATM fine-tuned core frequency predictor.

(b) Application performance predictor.

Figure 12: Single-threaded application performance can be predicted linearly with core frequency, and core frequency can be modeled using chip power, following Eq. 1.

we derive a linear relationship between ATM's dynamic frequency and the chip's total power consumption as shown in Eq. 1. The value b represents a core's static CPM setting, while $k' \cdot \overline{P}$ represents the dynamic variation, dominated by the IR voltage drop.

$$\begin{aligned} \overline{f} &= k \cdot \overline{V_{chip}} = k \cdot (V_{vrm} - R \cdot \overline{I}) \\ &= k \cdot (V_{vrm} - R \cdot \frac{\overline{P}}{V_{vrm}}) \\ &= -k' \cdot \overline{P} + b \end{aligned} \tag{1}$$

Fig. 12a shows the linear model fitted for each core's fine-tuned CPM configuration. The data points align with Eq. 1's predictions. Each additional watt degrades the frequency by about two MHz. In practice, each core stores its frequency prediction model and the model is indexed by the chip's total power consumption during job scheduling and runtime.

C. Delivering Predictable Performance

Frequency directly affects application performance. Fig. 12b shows application performance scales linearly with frequency, with different coefficients depending on the workload's memory behavior. A memory-bound workload, such as `mcf`, enjoys less performance improvement from higher frequency compared with a compute-bound workload like `x264` because cache misses limits the compute throughput. Thus, we build a performance predictor for each application, using frequency as the input. In this way, thread performance on each core can be inferred by the chip's total power, using each core's frequency predictor as the intermediate step.

On a fine-tuned ATM system, each application's performance depends on both the core it runs on as each core has different CPM configuration which leads to varying frequency levels, and the applications running on other neighboring cores, as all applications contribute to the chip's total power which in turn affects each core's frequency through the DC voltage drop on the shared power delivery path. For some `critical` applications that the users are interested in, it is crucial that they get mapped to the ATM fine-tuned cores that are high performance and robust. Meanwhile, it is also crucial that the co-located `background` applications are adequately managed so that the total chip power does not exceed the level that hampers critical app's core frequency. To handle this issue, we propose a scheme to selectively throttle `background` application performance to control total chip power, and indirectly frees up frequency potential for `critical` applications.

Mem behavior	Critical	Background
Intensive	resnet, vgg, ferret, fluidanimate	mlp, gcc, facesim, lu_cb, streamcluster, etc.
Non-intensive	squeezenet, seq2seq, babi, bodytrack, vips	blackScholes, x264, swaptions, raytrace,

Table II: Classifying critical and background applications, based on their memory subsystem interference behavior.

We use the applications in Table II for evaluation. The `critical` workloads are user-facing and require high performance for lower latency. They include deep learning inference (CNN, RNN, and LSTM models), object detection, real-time image processing, content similarity search, etc. The `background` workloads can tolerate lower performance and include workloads such as stock price estimation, 3D image rendering, compression, compilation, and machine learning training. For our work, we focus on the performance issue caused by the ATM system's shared power delivery problem and excludes performance interference from the memory subsystem which is a general issue for all multicores. Thus, we avoid co-locating two memory-intensive workloads at the same time to simplify the problem.

Fig. 13 shows our scheme. It takes into account the core-to-core performance and robustness variability caused by process variation, and the inter-core frequency interference caused by dynamic voltage variation on the power delivery subsystem. First, the user selects how the operator would like to set the different cores' CPM configuration.

Under a typical situation, the default policy is to use the chip's ATM configuration found through the per-core stress-test at test time, such as the *thread-worst* in Table I. The default policy provides good reliability guarantee through the worst-case tests, and meanwhile provides high performance. On POWER7+, the `critical` and `background` workloads all execute correctly under thread-worst. We evaluate our scheduler using the default policy as it provides a good trade-off between reliability and performance.

For higher performance, the user can select an "aggressive" governor, which chooses an application's most aggressive CPM configuration that guarantees correct execution. This can be done with per-application prediction or repetitive profiling of an application in a tier of testing servers before shipping the application to production server clusters. Using the core's best-fit ATM configuration provides more performance benefits at the risk of system failure. We put

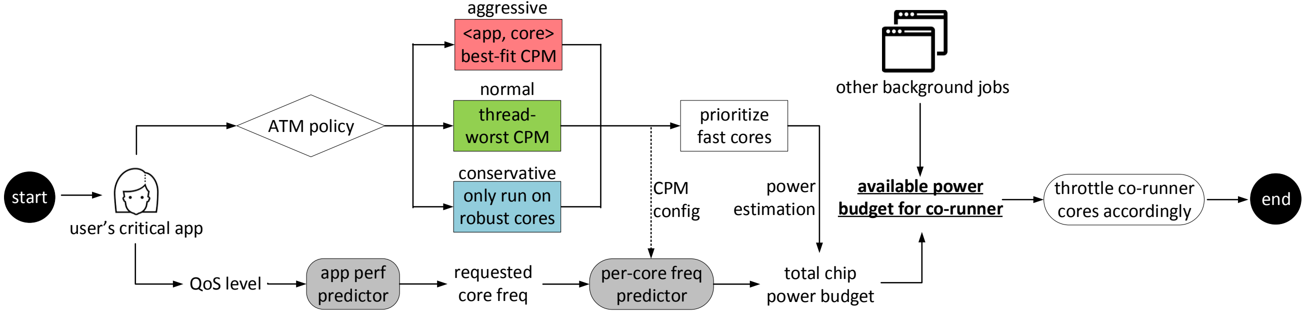


Figure 13: Managing a fine-tuned ATM system needs to integrate the per-app performance predictor and per-core frequency predictor, so that `critical` application performance can be satisfied by using faster cores and maintaining an estimated chip power budget.

exploration on the "aggressive" governor to future work.

For higher robustness, the user can select a "conservative" governor, which only schedules `foreground` workloads onto robust cores picked by ATM characterization. The robust cores are scarce and may not provide the highest performance, but they have the highest guarantee of correct execution. The conservative policy is best for unknown applications or when application correctness is paramount.

The OS then automatically sets each core's CPM setting according to user-selected policy. The faster cores after CPM reconfiguration are selected for running `critical` application. In parallel with CPM reconfiguration, the scheme reads user-specified QoS target for the `critical` application and infers the chip power needed to meet the performance goal using per-application performance predictor and per-core frequency predictor. To meet the QoS goal, total chip power under `critical` and co-running `background` workloads cannot exceed the calculated power budget.

The manager subtracts the estimated power of the `critical` workloads from the total chip power budget to get the power envelope available for the co-running `background` jobs. The `background` jobs can then be scheduled to the same chip under this envelope by carefully tuning their power consumption. On POWER7+ where V_{dd} is shared for all cores, we adjust power consumption by changing core frequency. Depending on the power envelope, we can 1) allow workloads to use aggressive ATM that has the highest frequency, 2) set cores to different DVFS states' frequency levels or 3) use power-gating to disable cores.

D. Performance Improvement

We evaluate our solution(s) against the static margin and the default ATM. Some customers turn off ATM for predictability, so the static margin is one of the fair baselines we compare with for evaluation. The system is running the stock DVFS OS governors that already strive to improve system efficiency. Therefore, our results include that comparison implicitly. Since ATM systems are still new and rare, there is little other prior work to compare against directly.

Our evaluation is carried out when all cores are scheduled to run an application. In practice, power gating idle cores when not enough workloads are available can further free up chip power and boost the performance of target workload [17]. For all our tests, die temperature is maintained under 70°C, and no side effect on-chip cooling is observed.

Fig. 14 summarizes the performance benefit of managing a fine-tuned ATM system. To highlight the frequency interference impact, we use one core to run `critical` application, which suits many applications, such as LSTM and RNN inference. We co-locate all `critical` and `background` applications on processor 0 (P0) of our two-socket server.

Under static margin, the default DVFS governors make POWER7+ processors clock at a fixed 4.2 GHz to run applications, providing predictable but low performance.

The default ATM improves performance for all cores, but not with the highest efficiency. An unmanaged system ignores the sensitivity of core frequency to total chip power. ATM may be indiscriminately activated on all cores, both for `critical` and `background` workloads, which significantly raises total chip power, eroding timing margin and reducing all cores' frequency, thereby diminishing the `critical` application performance. This unmanaged system still increases frequency due to ATM's harnessed margin, but the improvement is restricted to 6.1% on average.

Fine-tuning the ATM control loop provides more frequency gain, but an unmanaged system prevents the processor from the maximum benefit. Compared with the default ATM, an unmanaged processor system may carelessly assign the slowest core after CPM reconfiguration to a `critical` job, limiting the peak performance that can be achieved. The unmanaged system may also let all co-located `background` workloads run under their highest frequency, increasing total chip power and reducing `critical` workload frequency. However, in this scenario `critical` applications still enjoy 10.2% improvement over static margin because fine-tuning ATM unlocks large frequency gains.

A managed ATM system can opt to maximize the performance of `critical` applications. Specifically, `critical` applications get assigned to the fastest cores, and `background` application power is minimized by applying the lowest p-state. In this way, `critical` applica-

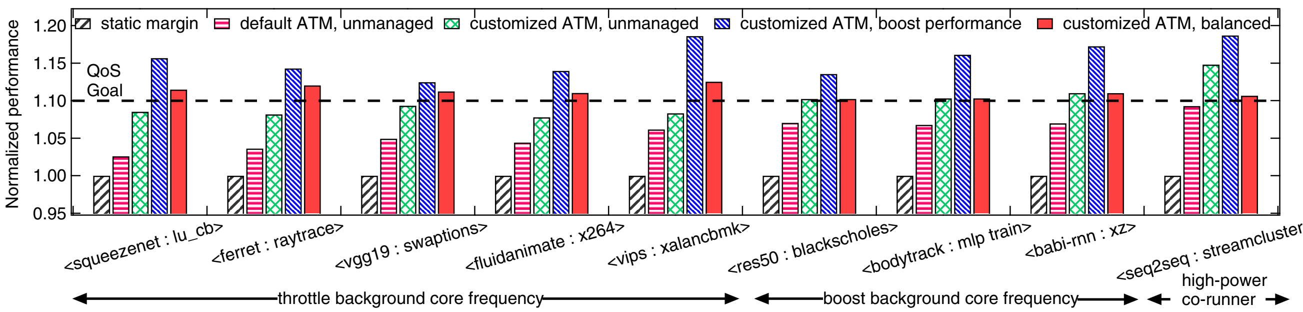


Figure 14: `Critical` application performance co-located with `background` workloads under different settings, shown as $<critical : background>$ pairs. Aggressively fine-tuning ATM, together with low-power co-running `background` workloads, boosts performance by 15.4% on average. With proper management, a 10% performance improvement goal is guaranteed for `critical workloads` by throttling co-runner's core frequency to main total chip power below budget.

tion frequency is maximized, at the cost of `background` workload performance. On average, `critical` workload performance improves by 15.2% on a real physical system.

Alternatively, a managed ATM system can opt to balance `critical` and `background` jobs by letting `critical` applications just meet their performance goal and maximizing `background` performance under that promise. Supposing the user targets 10% performance improvement for a `critical` workload over the static margin run, our managed system then throttles `background` core frequencies by the minimal amount to control total chip power, letting the frequency of the core running the `critical` workload reach the level that delivers target performance.

In Fig. 14 the performance of `squeezenet`, `ferret`, `vgg19`, and `fluidanimate` exceeds the 10% improvement target when the chip aims at maximizing their performance. However, their performance drops below the target when the chip puts all cores into fine-tuned ATM states. A balanced point can be obtained by controlling `background` workload frequency. The frequency of co-located `lu_cb`, `raytrace`, `swaptions`, and `x264` is set to 4.2 GHz.

In contrast, `seq2seq` outperforms the 10% improvement goal when its co-located `streamcluster` runs under fine-tuned ATM. This is because `streamcluster` consumes little power even when the frequency is high. The extra available power budget can be exploited by swapping `streamcluster` with a more power-hungry co-runner, `lu_cb`, with core frequency properly throttled.

The other `critical` and `background` combinations meet the QoS when ATM is aggressively fine-tuned for all cores. The high-frequency gain of ATM fine-tuning provides this benefit. For these cases, no core throttling is needed.

In summary, a system that is based on core-level ATM fine-tuning and ATM-aware application power management provides 5% to 10% steady performance improvement over the original ATM system. This result is notable because the improvement comes on a production-grade system where even a 1% performance gain is considered significant.

VIII. Related Work

Prior work exists on process variation, inter-core performance heterogeneity [12]–[16], [38], [39]. Our work exploits ATM's configurability to expose the variability and provides more performance gain than previous static margin methods.

Timing margin has been studied in the past [25], [26], [33], [40]–[43]. That work set out to understand and mitigate voltage variation. Several works expanded the study to multicore and GPUs [1], [2], [20], [23], [43]–[49]. These works provided valuable insights on how to optimize static margin, but few of them are adopted in real processors because of the strict requirement in production systems.

Active Timing Margin (ATM) is adopted in today's chips to reduce margin [4]–[11], primarily focusing on its power reduction benefits. Our work is the first to study ATM's performance variability on multicores, from both static process and dynamic voltage variation. The knob we tune on POWER7+ is generally available for all ATM systems, thus our insights have wide applicability for future multicores.

IX. Conclusion

As traditional optimization techniques are being exhausted, processor designers are increasingly turning to more exotic solutions such as Active Timing Margin (ATM) to improve efficiency. Though already available in commercial silicon, realizing ATM's full alluring efficiency benefits requires taming its inherent variability. In this paper, we have (1) demonstrated that ATM can be aggressively fine-tuned for higher performance and to expose variability; (2) performed the first complete characterization of the inter-core variation of ATM on multicores; and (3) proposed a scheme to deploy fine-tuned ATM multicore and manage it to offer enhanced and predictable performance. We believe our work paves the way for deploying and managing ATM processors at their full potential in a real environment.

Acknowledgments

The work was supported by NSF CCF-1528045.

References

[1] V. J. Reddi, S. Kanev, W. Kim, S. Campanoni, M. D. Smith, G.-y. Wei, and D. Brooks, "Voltage smoothing: Characterizing and mitigating voltage noise in production processors via software-guided thread scheduling," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2010.

[2] T. Miller, R. Thomas, X. Pan, and R. Teodorescu, "Vrsync: Characterizing and eliminating synchronization-induced voltage emergencies in many-core processors," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2012.

[3] Y. Zu, W. Huang, I. Paul, and V. J. Reddi, "Ti-states: Processor power management in the temperature inversion region," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2016.

[4] C. R. Lefurgy, A. J. Drake, M. S. Floyd, M. S. Allen-Ware, B. Brock, J. A. Tierno, and J. B. Carter, "Active management of timing guardband to save energy in POWER7," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2011.

[5] K. A. Bowman, C. Tokunaga, T. Karnik, V. K. De, and J. W. Tschanz, "A 22nm dynamically adaptive clock distribution for voltage droop tolerance," in *IEEE Symposium on VLSI Circuits (VLSIC)*, 2012.

[6] C. Tokunaga, J. F. Ryan, C. Augustine, J. P. Kulkarni, Y.-C. Shih, S. T. Kim, R. Jain, K. Bowman, A. Raychowdhury, M. M. Khellah, *et al.*, "A graphics execution core in 22nm cmos featuring adaptive clocking, selective boosting and state-retentive sleep," in *International Solid-State Circuits Conference (ISSCC)*, 2014.

[7] A. Grenat, S. Pant, R. Rachala, and S. Naffziger, "Adaptive clocking system for improved power efficiency in a 28nm x86-64 microprocessor," in *International Solid-State Circuits Conference (ISSCC)*, 2014.

[8] K. Bowman, S. Raina, T. Bridges, D. Yingling, H. Nguyen, B. Appel, Y. Kolla, J. Jeong, F. Atallah, and D. Hansquine, "A 16nm auto-calibrating dynamically adaptive clock distribution for maximizing supply-voltage-droop tolerance across a wide operating range," in *International Solid-State Circuits Conference (ISSCC)*, 2015.

[9] T. Webel, P. Lobo, R. Bertran, G. Salem, M. Allen-Ware, R. Rizzolo, S. Carey, T. Strach, A. Buyuktosunoglu, C. Lefurgy, *et al.*, "Robust power management in the IBM z13," *IBM Journal of Research and Development*, 2015.

[10] C. Vezyrtzis, T. Strach, I. Pierce, J. Chuang, P. Lobo, R. Rizzolo, T. Webel, P. Owczarczyk, A. Buyuktosunoglu, R. Bertran, D. Hui, S. Eickhoff, M. Floyd, G. Salem, S. Carey, S. Tsapepas, and P. Restle, "Droop mitigation using critical-path sensors and an on-chip distributed power supply estimation engine in the z14 enterprise processor," in *International Solid-State Circuits Conference-(ISSCC)*, 2018.

[11] T. Fischer, F. Anderson, B. Patella, and S. Naffziger, "A 90nm variable-frequency clock system for a power-managed itanium®-family processor," in *International Solid-State Circuits Conference (ISSCC)*, 2005.

[12] S. R. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas, "Varius: A model of process variation and resulting timing errors for microarchitects," *IEEE Transactions on Semiconductor Manufacturing*, 2008.

[13] R. Teodorescu and J. Torrellas, "Variation-aware application scheduling and power management for chip multiprocessors," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2008.

[14] K. K. Rangan, G.-Y. Wei, and D. Brooks, "Thread motion: fine-grained power management for multi-core systems," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2009.

[15] S. Dighe, S. Vangal, P. Aseron, S. Kumar, T. Jacob, K. Bowman, J. Howard, J. Tschanz, V. Erraguntla, N. Borkar, *et al.*, "Within-die variation-aware dynamic-voltage-frequency scaling core mapping and thread hopping for an 80-core processor," in *International Solid-State Circuits Conference (ISSCC)*, 2010.

[16] K. K. Rangan, M. D. Powell, G.-Y. Wei, and D. Brooks, "Achieving uniform performance and maximizing throughput in the presence of heterogeneity," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 2011.

[17] Y. Zu, C. R. Lefurgy, J. Leng, M. Halpern, M. S. Floyd, and V. J. Reddi, "Adaptive guardband scheduling to improve system-level efficiency of the POWER7+," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2015.

[18] A. J. Drake, M. S. Floyd, R. L. Willaman, D. J. Hathaway, J. Hernandez, C. Soja, M. D. Tiner, G. D. Carpenter, and R. M. Senger, "Single-cycle, pulse-shaped critical path monitor in the POWER7+ microprocessor," in *Low Power Electronics and Design (ISLPED), 2013 IEEE International Symposium on*, IEEE, 2013.

[19] A. Mericas, "Performance characteristics of the power8 processor," in *IEEE Hot Chips(HCS)*, 2014.

[20] C. Berry, J. Warnock, J. Badar, D. Bair, E. Behnen, B. Bell, A. Buyuktosunoglu, C. Cavitt, P. Chuang, O. Geva, *et al.*, "IBM z14 design methodology enhancements in the 14-nm technology node," *IBM Journal of Research and Development*, 2018.

[21] C. R. Lefurgy, A. J. Drake, M. S. Floyd, M. S. Allen-Ware, B. Brock, J. A. Tierno, J. B. Carter, and R. W. Berry, "Active guardband management in POWER7+ to save energy and maintain reliability," *IEEE Micro*, 2013.

[22] A. Drake, R. Senger, H. Deogun, G. Carpenter, S. Ghiasi, T. Nguyen, N. James, M. Floyd, and V. Pokala, "A distributed critical-path timing monitor for a 65nm high-performance microprocessor," in *International Solid-State Circuits Conference (ISSCC)*, 2008.

[23] G. Papadimitriou, M. Kaliorakis, A. Chatzidimitriou, D. Gizopoulos, P. Lawthers, and S. Das, "Harnessing voltage margins for energy efficiency in multicore cpus," in *International Symposium on Microarchitecture (MICRO)*, 2017.

[24] J. Tschanz, N. S. Kim, S. Dighe, J. Howard, G. Ruhl,

S. Vangal, S. Narendra, Y. Hoskote, H. Wilson, C. Lam, *et al.*, "Adaptive frequency and biasing techniques for tolerance to dynamic temperature-voltage variations and aging," in *International Solid-State Circuits Conference (ISSCC)*, 2007.

[25] M. D. Powell and T. Vijaykumar, "Pipeline damping: A microarchitectural technique to reduce inductive noise in supply voltage," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2003.

[26] V. J. Reddi, M. S. Gupta, G. Holloway, G.-Y. Wei, M. D. Smith, and D. Brooks, "Voltage emergency prediction: Using signatures to reduce operating margins," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 2009.

[27] J. Tierno, A. Rylyakov, D. Friedman, A. Chen, A. Ciesla, T. Diemoz, G. English, D. Hui, K. Jenkins, P. Muench, *et al.*, "A dpll-based per core variable frequency clock generator for an eight-core POWER7 microprocessor," in *Symposium on VLSI Circuit Digest of Tech Papers*, 2010.

[28] S. Song, R. Panda, J. Dean, and L. K. John, "Wait of a decade: Did spec cpu 2017 broaden the performance?," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 2018.

[29] C. Bienia, S. Kumar, and K. Li, "Parsec vs. splash-2: A quantitative comparison of two multithreaded benchmark suites on chip-multiprocessors," in *Proceedings of the International Symposium on Workload Characterization (IISWC)*, 2008.

[30] G. Papadimitriou, A. Chatzidimitriou, M. Kaliorakis, Y. Vastakis, and D. Gizopoulos, "Micro-viruses for fast system-level voltage margins characterization in multicore cpus," in *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2018.

[31] "Coremark: https://www.eembc.org/coremark/."

[32] "Stream: https://www.cs.virginia.edu/stream/."

[33] E. Grochowski, D. Ayers, and V. Tiwari, "Microarchitectural simulation and control of di/dt-induced power supply voltage variation," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, 2002.

[34] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Proceedings of the International Conference on Parallel architectures and compilation techniques (PACT)*, 2008.

[35] Y. Kim, L. K. John, S. Pant, S. Manne, M. Schulte, W. L. Bircher, and M. S. S. Govindan, "Audit: Stress testing the automatic way," in *International Symposium on Microarchitecture (MICRO)*, 2012.

[36] R. Bertran, A. Buyuktosunoglu, P. Bose, T. J. Slegel, G. Salem, S. Carey, R. F. Rizzolo, and T. Strach, "Voltage noise in multi-core processors: Empirical characterization and optimization opportunities," in *Proceedings of the International Symposium on Microarchitecture(MICRO)*, 2014.

[37] R. Bertran, A. Buyuktosunoglu, M. S. Gupta, M. Gonzalez, and P. Bose, "Systematic energy characterization of cmp/smt processor systems via automated micro-benchmarks," in *International Symposium on Microarchitecture (MICRO)*, 2012.

[38] X. Liang, R. Canal, G.-Y. Wei, and D. Brooks, "Process variation tolerant 3t1d-based cache architectures," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2007.

[39] A. Bacha and R. Teodorescu, "Dynamic reduction of voltage margins by leveraging on-chip ecc in itanium ii processors," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2013.

[40] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, *et al.*, "Razor: A low-power pipeline based on circuit-level timing speculation," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2003.

[41] M. S. Gupta, K. K. Rangan, M. D. Smith, G.-Y. Wei, and D. Brooks, "Decor: A delayed commit and rollback mechanism for handling inductive noise in processors," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 2008.

[42] M. S. Gupta, V. J. Reddi, G. Holloway, G.-Y. Wei, and D. M. Brooks, "An event-guided approach to reducing voltage noise in processors," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, 2009.

[43] V. J. Reddi, S. Campanoni, M. S. Gupta, M. D. Smith, G.-Y. Wei, D. Brooks, and K. Hazelwood, "Eliminating voltage emergencies via software-guided code transformations," *ACM Transactions on Architecture and Code Optimization (TACO)*, 2010.

[44] M. S. Gupta, J. L. Oatley, R. Joseph, G.-Y. Wei, and D. M. Brooks, "Understanding voltage variations in chip multiprocessors using a distributed power-delivery network," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, 2007.

[45] A. Bacha and R. Teodorescu, "Using ecc feedback to guide voltage speculation in low-voltage processors," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2014.

[46] I. Pierce, J. Chuang, C. Vezyrtzis, D. Pathak, R. Rizzolo, T. Webel, T. Strach, O. Torreiter, P. Lobo, A. Buyuktosunoglu, *et al.*, "Power supply noise in a 22nm z13 microprocessor," in *International Solid-State Circuits Conference (ISSCC)*, 2017.

[47] J. Leng, Y. Zu, and V. J. Reddi, "Gpu voltage noise: Characterization and hierarchical smoothing of spatial and temporal voltage noise interference in gpu architectures," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 2015.

[48] J. Leng, A. Buyuktosunoglu, R. Bertran, P. Bose, and V. J. Reddi, "Safe limits on voltage reduction efficiency in gpus: a direct measurement approach," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2015.

[49] R. Thomas, K. Barber, N. Sedaghati, L. Zhou, and R. Teodorescu, "Core tunneling: Variation-aware voltage noise mitigation in gpus," in *International Symposium on High Performance Computer Architecture (HPCA)*, 2016.