Achieving Uniform Performance and Maximizing Throughput in the Presence of Heterogeneity

Krishna K. Rangan†‡ Michael D. Powell‡ Gu-Yeon Wei† David Brooks†

†Harvard University
33 Oxford St., Cambridge, MA 02138
{kkrangan,guyeon,dbrooks}@eecs.harvard.edu

‡Intel Massachusetts
77 Reed Road, Hudson, MA 01749
{krishna.rangan,michael.d.powell}@intel.com

Abstract

Continued scaling of process technologies is critical to sustaining improvements in processor frequencies and performance. However, shrinking process technologies exacerbates process variations – the deviation of process parameters from their target specifications. In the context of multi-core CMPs, which are implemented to feature homogeneous cores, within-die process variations result in substantially different core frequencies. Exposing such process-variation induced heterogeneity interferes with the norm of marketing chips at a single frequency. Further, application performance is undesirably dictated by the frequency of the core it is running on. To work around these challenges, a single uniform frequency, dictated by the slowest core, is currently chosen as the chip frequency sacrificing the increased performance capabilities of cores that could operate at higher frequencies. In this paper, we propose choosing the mean frequency across all cores, in lieu of the minimum frequency, as the single-frequency to use as the chip sales *frequency. We examine several scheduling algorithms implemented below the O/S in hardware/firmware that guarantee minimum application performance near that of the average frequency, by masking process-variation induced heterogeneity from the end-user. We show that our Throughput-Driven Fairness (TDF) scheduling policy improves throughput by an average of 12% compared to a naive fairness scheme (round-robin) for frequency-sensitive applications. At the same time, TDF allows 98% of chips to maintain minimum performance at or above 90% of that expected at the mean frequency, providing a single uniform performance level to present for the chip.*

1. Introduction

Continued scaling of process technologies is critical to sustaining improvements in processor performance and energy-efficiency. Indeed, Intel's Tick-Tock schedule for releasing processors with ever-increasing performance relies as much on shrinking process technologies as it does on innovations in microprocessor architecture for continued performance and power improvements [4]. However, shrinking process technologies exacerbate process variations–the deviation of process parameters from their target specifications–and can significantly arrest the benefits of scaling. Process variations affect two key transistor parameters, gate length and threshold voltage, that impact the speed and power of the chip. Large variation of these key transistor parameters results in high variation of transistor speeds across the chip.

These variations, in conjunction with large die size and high core counts, result in a spread of maximum frequencies (fmax) across different cores in a chip. An example of observed fmax variation in silicon at three voltages for an 80-core Intel test chip [6] is shown in Figure 1. At a supply voltage of 1.2 V, there is a 28% variation between the fastest core's frequency of 7.3 GHz and slowest core's frequency of 5.7 GHz, and this grows to 59% at 0.8V. While it may be possible to increase the frequency of slower cores through voltage scaling, slower cores may experience sufficient process variation that no feasible voltage increase will raise their frequency to near that of the fastest cores. Power-budget constraints, particularly if all cores share one or a few voltage domains, and the narrow range between minimum and maximum operating voltages (Vmin and Vmax) in modern processes limit the potential of voltage increases to mitigate process variations and the resulting inter-core frequency variation. While not targeting process variations, Intel ® Turbo Boost Technology provides an example of these power-budget limitations; the voltage and frequency of one or a few cores can be increased to improve single-thread performance but only at the expense of powering-off other cores [3]. Turbo Boost illustrates that power budgets are already at the point where all or even most cores cannot run at both maximum frequency and elevated voltage.

The conventional solution to manage inter-core frequency variation is to ignore the variation and set the frequency of the entire chip to be that of the slowest core. We call this solution *min-max* because the chip operates at the minimum maximum frequency among all cores. Min-max sacrifices substantial performance to achieve simplicity by running all cores at the frequency of the slowest core.

An alternative solution to manage inter-core frequency variation allows each core (or group of a few cores) to operate in its own frequency domain. With one frequency domain per core, we can recover all performance opportunity sacrificed by min-max because each core can run at its maximum frequency (within the chip-level power budget). However, *heterogeneous-core-frequency* (HCF) configuration disrupts the appearance of uniform performance for all cores, causing fairness problems in multi-process situations. HCF implementations interfere with processor marketing strategies that are based on a single performance number (typically *sales* frequency). For a variety of reasons we discuss in Section 2.3, a single performance number is desirable to CPU vendors.

Clearly, there is a need not only to exploit HCF configurations to boost performance of applications, but also to provide a single, uniform-level of performance to the O/S and end-users, as though the processor were simply operating at a single, higher frequency. Note that the issues of process fairness and the appearance of a single uniform frequency are intrinsically linked because the appearance of a single frequency across all cores implies that each process is receiving an appropriate share of resources. In this paper, we show that the appearance to users and the O/S of a single uniform performance metric (frequency) across cores can be approximated while reaping the performance benefits of allowing individual cores

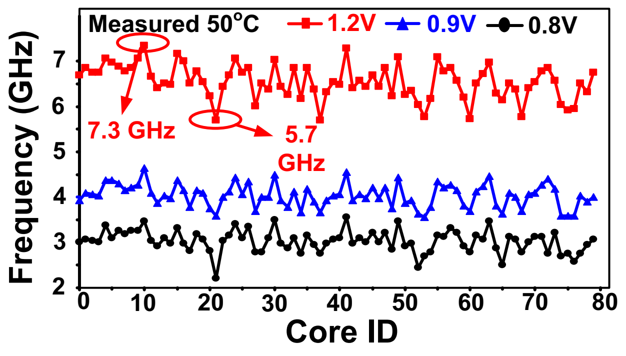


Figure 1: Measured process variations on Intel's 80-core processor showing substantially different core frequencies [6] (used with permission).

to be clocked at frequencies higher than that allowed by min-max. We propose to achieve both goals with the use of thread scheduling policies implemented at fine temporal scheduling windows, below the O/S in the hardware/firmware. Our scheduling policies seek appearance of uniform, single frequency (for a HCF chip), near the arithmetic mean of all cores' frequencies, which we call *apparent mean frequency* (AMF). Performance guarantees near the AMF frequency, which is higher than the min-max frequency, increase the number of premium chips that can be sold for higher prices.

One natural solution to achieve fairness in scheduling while clocking each core at its individual maximum frequency is naively fairness-driven, round-robin scheduling of processes across each core in the chip. Round-robin scheduling creates the appearance of a uniform, single-frequency at AMF since each process spends equal time on each core. Unfortunately, as we demonstrate later, brute-force round-robin reschedulings, largely oblivious to application performance requirements, miss opportunities to substantially increase throughput without sacrificing performance of any process. On the other hand, if maximum throughput is our goal, then an alternative to round-robin is throughput-driven scheduling. However, our results show that throughput-driven scheduling increases average system throughput at the cost of hurting performance of some processes. Thus, throughput-driven scheduling is unfair, as certain processes experience performance below the desired AMF-level, and is inconsistent with our goal of maximizing performance yet providing a single, uniform-level of performance for all processes.

To achieve fairness and high throughput under HCF configurations, we propose *throughput-driven fairness* (TDF) scheduling. TDF combines the opposing policies of fairness-driven and throughput-driven scheduling in a manner transparent to the O/S and end user, implemented at the hardware-level. The fairness-driven component estimates each process' throughput at the AMF and avoids schedules that would reduce throughput below that level. The throughput-driven component estimates each process' throughput on high-frequency and low-frequency cores and opportunistically schedules to increase throughput. In effect, TDF provides the appearance of a single uniform frequency to processors, near AMF frequency, and significantly boosts average performance.

The main contributions of this paper are:

1. We evaluate the throughput of chips running all cores at the min-max frequency compared to chips exploiting heterogeneous frequencies, and illustrate that using round-robin scheduling achieves fairness and the appearance of uniform performance across cores on a chip with heterogeneous frequencies.
2. We propose a throughput-driven scheduling policy which increases throughput up to 20% over round-robin scheduling and a fairness-driven scheduling policy that approximates the fairness of round-robin without relying on continuous brute-force process migration.
3. We propose *throughput-driven fairness* (TDF) scheduling which combines throughput and fairness considerations. TDF improves the throughput of workloads that are sensitive to frequency choices by 12% on average.
4. We demonstrate that TDF policy significantly increases the number of premium chips that guarantee performance near AMF-level instead of min-max-level: 90% of the chips *exceed* AMF-level total throughput while 98% of the chips have minimum performance within 10% of the desired AMF performance-level. In contrast, without our TDF policy, only 2% of chips exceed AMF-level throughput, and only 80% of the chips have minimum performance within 10% of the AMF-level.

The rest of this paper is organized as follows. In Section 2, we discuss process variation and its impact on core frequencies, impact of core-frequency variation on workload performance, and motivate the need for hardware- or firmware-based process management. Section 3 discusses fairness-driven, throughput-driven, and TDF scheduling policies. Section 4 describes our simulation methodology, and Section 5 presents our results. We discuss related work in Section 6 and conclude in Section 7.

2. Motivation: Impact and Handling of Process Variation on CMPs

Process variations have always occurred in microprocessor manufacturing, but their impact and management has changed over time. In this section, we discuss impact of process variations on multi-core processors, effects on application performance, and the need for an O/S and end-user transparent solution.

2.1 Process Variations

Process variations include both systematic and random inconsistencies in the silicon and lead to variations in transistor size and threshold voltages [24]. Ultimately, the impact is variation in transistor latency and power.

In the era of single-core microprocessors, process variations were managed across die. While intra-die variations may have been present, clocking components of a single microprocessor core at different frequencies would have introduced synchronization logic and complexity [25]. Instead, the core (and thus chip) were clocked at the fastest speed the entire core could support. Manufacturers used speed binning to market the faster microprocessors at a faster clock speed and sell them at a higher price.

The introduction of *chip multi-processors* (CMPs) increases interest in intra-die variations. If different cores in the chip support different maximum frequencies, it is tempting to increase performance by running the faster cores at higher clock speeds. Because the interconnects between cores (e.g., bus, on-chip-network, or shared cache) tend to be asynchronous with the core clock, it is not overly complicated to have different clock speeds for each core. There may be little benefit to supporting multiple clocks in a chip with few cores (e.g., two cores), but as core counts increase, the inter-core variation and thus potential performance increase becomes larger. In fact, current multi-core production systems already

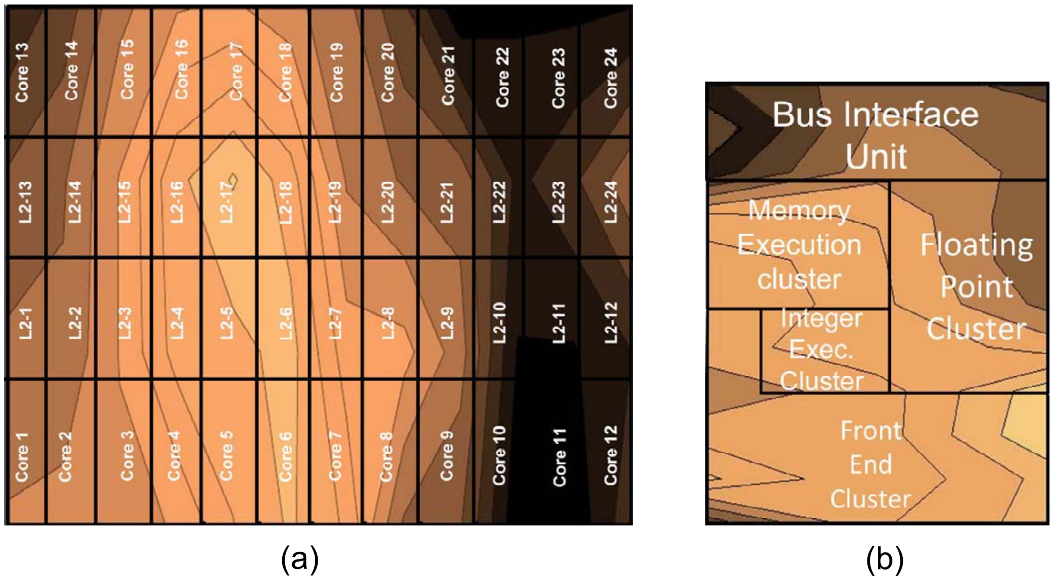


Figure 2: (a) Contour map of variations in the 24-core CMP. (b) Floorplan of Atom-like core [8].

feature per-core frequency control [2]. Of course, actually utilizing a chip with a wide variation in core clock frequencies is a challenge, as it disrupts the appearance of uniform performance as discussed in Section 1.

We illustrate the impact of process variations on a multi-core chip with an example. We evaluate a chip with 24 Intel ® Atom ™ Processor-like cores that share 24 L2 banks. We model the process variation in a 22 *nm* process using the VARIUS model [24]. A representative chip is shown in Figure 2(a). The range from lightest to darkest colors represents a transistor-latency change of about 40%. The core-block layout (based on [8]) and representative intra-core process variation are shown in Figure 2(b). As can be seen in the figure, there exists a wide distribution of core frequencies. Were these single-core parts, they could be cut at the core boundaries and each sold at a different frequency. But multi-core demands either (a) the loss of performance potential by using the minimum-maximum frequency among the cores (min-max) for all cores, or (b) some other solution.

Over a large population of die, the impact of process variation can be substantial. Figure 3a (top) shows the distribution of core frequencies across a single representative 24-core chip as well as the average of those frequencies. Figure 3a (bottom) shows the frequency core distribution of a population of 100 24-core chips. The top and bottom lines show the frequency of the slowest and fastest core (i.e., the bottom line is the min-max frequency that would conventionally be used for the entire chip). The middle line shows the average frequency of all cores for each chip, which we call the Apparent Mean Frequency (AMF). While the AMF is an artificial construct (because it is impossible to clock the entire chip at that frequency), the average frequency is useful to illustrate the potential additional performance of the chip. AMF also is a useful target for creating the appearance of a single performance level.

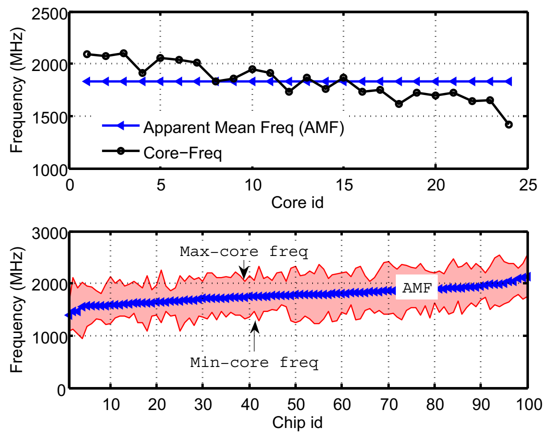
2.2 Process-Variation-Induced Performance Heterogeneity

Exposing inter-core process variations to the O/S or software results in a new challenge of non-uniform performance across the cores. In this section, we present examples of how these variations manifest in applications. Figure 3b shows the performance of two applications from SPEC 2006 [28], the high-instructions-per-cycle (high IPC) application *calculix*, and the low-IPC application *libquantum* on a 24-core CMP as portrayed earlier in Figure 2. For this set of simulations, each core runs at its individual maximum frequency quantized to 100 MHz increments.

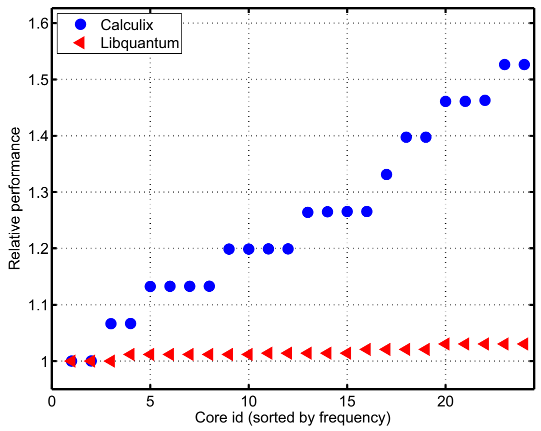
We show performance of these two applications on each core normalized to their performance on the slowest frequency core (the x axis shows core identifiers sorted by frequency). The performance of the slowest core is equivalent to what the performance of *all* cores would be if a single global min-max frequency were used. HCF allows some cores to achieve more than 50% throughput improvement for *calculix*. Average *calculix* throughput across all cores (not shown in the figure) is 25% higher than that of the slowest core. This average represents the throughput at the AMF. The low-IPC *libquantum* experiences smaller variation, indicating *libquantum* is not particularly sensitive to core frequency. Such unpredictability of workload throughput on an individual core presents a challenge for the hardware (and/or O/S) if we wish the user to see a consistent level of performance.

2.3 Hardware-Based Scheduling to Provide the Appearance of Single Frequency

We must address the inconsistent performance across cores while still providing a level of throughput higher than that of min-max. Previous scheduling policy proposals, which we discuss in Section 6, target performance, power and aging improvements in process-variation- or defect-induced heterogenous systems at O/S-level or coarser intervals. In contrast, we seek O/S and end-user transparent solutions to addressing performance inconsistency. CPU vendors highly desire this goal for many reasons. First, processor marketing strategies are based on a single performance number, and transparent scheduling policies simply allow for increased performance without side-effects. Second, CPU vendors do not desire to divulge low-level manufacturing information about process variations. Third, conventional operating systems do not expect a spectrum of core frequencies to be imposed by hardware. O/S transparent solutions allow CPU vendors to boost performance independent of O/S—by avoiding development delays or deploy-



(a)



(b)

Figure 3: (a) Distribution of core frequencies on a single chip (shown in figure 2(a)) [top] and across 100 chips [bottom]. Apparent Mean Frequency (AMF) is the average frequency of the chip. (b) Performance of two applications on 24 cores normalized to their performance on slowest core: *calculix* (compute-bound application) is highly sensitive to the running core; *libquantum* (memory-bound application) exhibits smaller performance differences among cores.

ment inertia in system software. Finally, recent studies show that program phase behavior exists at granularities finer than the O/S scheduler ticks [13, 23]. Thus, O/S and end-user transparent behavior is important for both transparency reasons, and maximizing throughput on HCF architectures. We employ hardware-/firmware-based scheduling policies at granularities finer than the O/S scheduler tick to achieve our goal.

2.4 Beyond Multiple Frequency Domains

An extension of multiple frequency domains is to have multiple voltage and frequency domains (VFS). Although fine-grained VFS might allow voltage of slow cores to be increased enough to bring their frequency closer to that of faster cores, such reduction in frequency variance comes at the expense of chip complexity and increased power and may not be feasible within a power budget. Moreover, our scheduling policies to maintain fairness would extend to VFS configurations at a given power budget.

VFS or simpler per-core frequencies should not be confused with Dynamic Voltage Frequency Scaling (DVFS), a runtime option to improve power-performance efficiency. In contrast to the option of DVFS, the physical characteristics of a chip impose process-variation-induced fmax differences on the hardware and the O/S by presenting a non-uniform chip.

3. Process Scheduling Policies for Heterogeneous Core Frequencies

As discussed, HCF can restore the performance lost from min-max, but can disrupt the appearance of uniform performance for all cores, causing fairness problems in multi-process situations. In this section, we discuss scheduling policies aimed towards improving throughput, fairness and both. Before presenting scheduling policies, we recap the performance and fairness considerations of min-max and static HCF configurations, using Figure 4 which illustrates a conceptual 4-core processor that experiences process variations. Figure 4(a) presents the performance of each core. Figure 4(b) plots the average performance observed across all cores on the y axis versus the *minimum performance level* (MPL)—performance of the slowest process in the mix, relative to all processes running on cores at the desired AMF performance level. Higher MPL indicates performance of the slowest application is closer to the AMF performance-level, and is better.

Min-Max frequency configuration. The (completely overlapped) circles corresponding to the first x-axis label in figure 4(a) illustrate the performance of four copies of a process running on a four core system. Clearly all four copies achieve the same performance, as the underlying cores offer homogeneous performance.

Core disabling. An extension of min-max is to simply disable slow cores and sell the chip with a higher single frequency but fewer cores. Core disabling is *not* shown in the figure. Like min-max, all remaining cores will achieve the same performance level, but that level will depend on the aggressiveness of core disabling. However, total throughput will be reduced because there are fewer cores (further discussed in Section 5.1).

Static HCF. This configuration allows each core to operate at its individual maximum frequency and statically assigns processes to each core. The circles corresponding to the second x-axis label in Figure 4(a) show that the four copies of processes, assigned to four cores, now exhibit varying degrees of performance, depending on the core they run on. Average performance approaches the performance-level of virtual AMF (shown by the dashed line). However, MPL (performance of slowest process) is the same as in the min-max configuration because of the same lowest-frequency core (Figure 4(b)).

Clearly, our goal must be not only to improve the average performance, but also to improve MPL, as though all processes were always running on a core near AMF performance-level. We will examine scheduling policies aimed at this goal, introducing the remaining x-axis labels of the figure at that time. First, we discuss thread migration, which is central to our scheduling policies.

3.1 Thread Migration

In HCF configurations, thread migration provides a way to manage the diverse collection of applications to boost throughput without sacrificing significant performance of any single application. Our scheduling policies employ thread migration to achieve throughput and fairness goals.

Implementation. We envision thread migration-based scheduling policies running in a embedded microcontroller. Current production systems execute firmware code in embedded microcontrollers, to perform a variety of core performance, power-monitoring

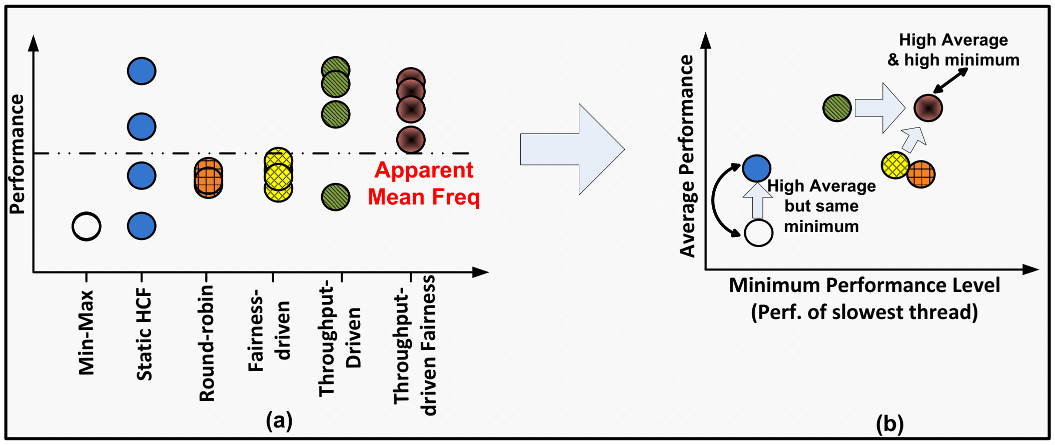


Figure 4: (a) Illustration of various mechanisms to boost throughput on a 4-core system experiencing variations. Performance at the (virtual) AMF frequency is shown by the dashed line. (b) Tradeoff of average and Minimum Performance Level (MPL) for various mechanisms. The slowest application in the mix determines MPL.

and control tasks [16]. The microcontroller has access to core performance and status registers used to make scheduling decisions.

Transparency. As stated in Section 2.3, O/S and end-user transparent scheduling is a highly desired goal for CPU vendors. O/S-transparent solutions allow for processor marketing based on a single (but higher) frequency, avoid system-software development/deployment delays and mask low-level process variation-related information from end users. Our goal of presenting cores operating at a single AMF performance-level requires O/S to believe that applications are operating on the scheduled logical core. To achieve O/S transparency, we rely on a programmable APIC number that migrates with the thread, a small change suggested in [22], in addition to on-die microcontroller implementation of our scheduling policies.

Scheduling interval. O/S and end-user transparency, and thread migration scheduling interval are intrinsically tied: the appearance of a single frequency in a O/S quantum requires making scheduling decisions at intervals that are significantly shorter than the O/S scheduler ticks. All of our policies use relatively frequent, hardware/firmware-based migration [22, 23], of processes to cores to achieve improved throughput and fairness. However, choosing too small a scheduler interval to migrate applications (such as 50,000 cycles), incurs high overheads, as the processes continuously move register state and lose cache state with every migration (more details on the migration costs are presented in Section 4.2). Furthermore, ensuring fairness at granularities finer than O/S scheduler ticks is generally not required, except for some special purpose real-time or embedded systems. We experimentally studied the effect of thread migration interval on our architecture (results not shown for brevity), and determined that cache overheads level off as migration intervals approach 500,000 cycles. Our scheduling policies are evaluated with a 1 million cycle scheduling interval, or tick, that is based off of a system clock. For the remainder of the paper, *scheduling tick* refers to this 1-million cycle hardware-based scheduling interval, not the O/S scheduling interval, unless otherwise noted. It is important for fairness- or throughput-oriented scheduling policies to not be computationally complex, allowing for re-evaluation of processes to cores at the finer time scales we target.

3.2 Fairness-Driven Policies

We model two fairness-driven policies discussed below.

3.2.1 Round-Robin Scheduling

A natural solution to achieving fairness in HCF implementations is round-robin scheduling of processes across each core in the chip. By running on every core in the chip, processes experience performance-levels comparable to running on a system of homogeneous cores at the apparent-mean frequency (without accounting for the migration costs). This policy is illustrated in Figure 4(a), which shows all processes experiencing nearly the same performance. In addition, MPL improves relative to min-max and static (Figure 4(b)). We note that our implementation of round-robin policy distributes evenly the time that a process spends executing under the available performance levels (frequencies), instead of blindly and always migrating at each scheduling tick. A process will never be migrated to a neighboring core operating at a frequency that is equal to the current execution core.

3.2.2 Fairness-Driven Scheduling

The naively fairness-driven, round-robin policy explained above suffers from two major drawbacks. First, it is incorrectly oblivious to application performance requirements. Second, continuous process migrations results in performance degradations—arising from events such as cache misses, loss of core and other architected register states—with every migration. An improved solution that works around these drawbacks is prediction-based fairness-driven policy that we simply refer to as fairness-driven: by predicting process' performance at the desired AMF frequency, and scheduling to meet its AMF performance-level, brute-force reschedulings of round-robin may be avoided, without compromising the uniform performance-level.

Fairness driven policy works as follows. For the initial scheduling ticks, this policy employs round-robin, and tracks instruction rate in BIPS of processes by reading each core's performance counters. After obtaining BIPS from cores at different frequencies, this policy makes a prediction of process' performance at the virtual AMF core—the desired performance metric. In order to predict performance at the virtual AMF, we experimented with a variety of regression models, and found that a simple linear interpolation model works well, yet provides a low-overhead implementation. We call the predicted AMF performance-level for processes *f-pred*. The scheduler rank orders *f-pred* values from all processes, and assigns the highest-frequency core to the process with the (measured average) BIPS farthest below its *f-pred* value. The scheduler forces processes that have not migrated for four consecutive scheduling ticks, to migrate to a core at a different frequency. This is useful to recompute *f-pred* to account for the program phase changes. The fairness-driven policy cuts down on unnecessary migrations to boost performance compared to round-robin (Figure 4(a)), yet maintains round-robin-like MPL (Figure 4(b)). In practice, we observe slightly lower MPLs compared to round-robin due to prediction inaccuracies (results in Section 5.3).

3.3 Throughput-Driven Scheduling

In HCF configurations featuring a mix of applications with diverse behaviors, overall system throughput can be increased by identifying compute-bound program phases, and greedily executing the frequency sensitive phases on high-frequency cores. For example, as we observed earlier in Figure 3b (Section 2.2), the difference in compute requirements between *calculix* and *libquantum* can be exploited to schedule the high-IPC application *calculix* on to high-frequency core for increased system throughput. In throughput-driven policy, the first step is to predict the performance requirements of applications during each scheduling interval. Our performance prediction technique relies on past performance history of the application to arrive at the predicted performance for the next scheduling tick. We investigated a variety of prediction mechanisms, and experimentally determined that last-value predictor works best (consistent with previous findings [12]). The last-value predictor uses the measured BIPS over the prediction period of 100,000 cycles, which we experimentally determined to provide the best prediction accuracy, for use as the predicted value for the next scheduling tick. We call this predicted value *t-pred*. As the next step, the scheduling policy rank orders processes based on their *t-pred* value, and pairs them with cores rank-ordered by frequency.

As can be seen, this policy targets only maximum throughput, taking full advantage of the range of available clock frequencies in a chip (Figure 4(a)). However, this comes at the cost of lower MPL (from starved applications), compared to fairness-based policies (Figure 4(b)). We note that more complex throughput oriented policies have been studied for significantly more complex heterogeneous configurations, such as cores with structural defects, variations in static power or frequency [27, 30, 31]. These complex policies re-evaluate assignment of threads at O/S-level or coarser intervals and do not target appearance of uniform performance level. However, our scheduling policies target frequency-only chip heterogeneity, and need to be computationally simple and scalable, to make scheduling decisions on finer intervals. As we show in Section 5.3, this simple mechanism performed well for our studies.

3.4 Throughput-Driven Fairness Scheduling

The goal of our final policy, throughput-driven fairness scheduling that we simply refer to as TDF, is to bring together throughput and fairness benefits. That is, TDF seeks to squeeze performance out of heterogeneous workload configurations on HCF hardware configurations (Figure 4(a)), while providing the appearance of single uniform minimum performance-level (Figure 4(b)) to the user and the O/S.

TDF scheduling policy works as follows. For each process, it reads in the process' fairness-driven BIPS value, *f-pred* (Section 3.2), and last-value predictor value, *t-pred* from throughput-driven scheduling (Section 3.3). It then uses a weighted average of these two BIPS values, to produce a figure for each process. Processes with higher figures are assigned to faster cores. Throughput- and fairness-policies can be weighted differently to favor one or the other; for example, 100% throughput-weighted policy ends up being the same as the throughput-driven policy while a 50%-50% split weights the policies equally. We experimentally evaluate different weights as inputs to throughput and fairness algorithms (in Section 5.2), and use the best combination of weights for final results.

4. Experimental Framework

Careful modeling of within-die process variations, and realistic performance modeling of 24-core Atom-like system concurrently executing processes at different frequencies are essential to completely evaluate our proposed approaches. Our performance modeling infrastructure precisely models cost of draining core pipeline during migrations, data-transfer across clock domains, Non-Uniform Memory Access (NUMA) to the L2 cache shared across all cores, MESI coherence protocol, and contentions to L2 arising from simultaneous core accesses. This section presents the overall experimental framework and data gathering methodology.

4.1 Modeling Variations

Process variations are caused by fluctuations in device channel dimensions and dopant concentrations. Gate length variations (L_{eff}) can change the effective driving capability of the transistor, as well as the threshold voltage, as a result of the short channel effect. Random dopant variations can also change the threshold voltage (V_{th}) of the device. These two parameters in turn affect the gate delay and the leakage power. In this work, we capture the effect of within-die variations using the VARIUS model [24] which generates different V_{th} and L_{eff} values for each unit of the floorplan in Figure 2(b). Systematic variation is characterized by a spatial correlation, so that adjacent areas on a chip have roughly the same systematic component values. Random variation occurs at the level of individual transistors. It is modeled analytically with a normal distribution with $\mu = 0$ and standard deviation σ_{ran}. Since the random and systematic components are normally distributed and independent, their effects are additive, and the total σ is $\sqrt{\sigma_{ran}^2 + \sigma_{sys}^2}$. Each individual experiment uses a batch of 100 chips that have different V_{th} (and L_{eff}) maps generated with the same σ (standard deviation), μ (mean), and ϕ (correlation range). Our parameters are summarized in Table 5b. We assume that the random and systematic components have equal variances, similar to the model used by [24]. Gate length has a correlation range close to half of the chip's width. Since the systematic component of V_{th} variation directly depends on the gate length variation, we assume $\phi = 0.5$ for V_{th}.

4.2 Performance Simulation

Results from the parameter variation modeling are used to set core frequencies in the performance model. Core frequencies are quantized in 100 MHz increments. Our performance model features 24-core Atom-like microprocessor implemented in the detailed Asim simulation framework [7]. The parameters of our simulated system are shown in Table 5a. We also model an embedded microcontroller that runs our scheduling polices (summarized in Table 5c), and initiates thread migrations. During the scheduling tick, the microcontroller picks processes to migrate, and the corresponding cores receive a *quiesce* signal; following that, cores halt fetch and complete executing the instructions in the pipeline. Simulations precisely account for this pipeline drain cost that can range from hundreds to thousands of cycles depending on pipeline activity. In addition, we assume a 100-cycle overhead to migrate architectural register state between cores, similar to previous work [22], under the direction

Number of cores	24
L1 D-cache	Private, 24 KB, 8-way, line size is 64 bytes, latency 1 cycle
L1 I-cache	Private, 32 KB, 8-way, line size is 64 bytes, latency 1 cycle
L2 cache	Unified, shared, 1 MB, 16-way (per core), line size is 128 bytes
L2 latency	28 cycles if data found in local bank; varies for remote banks
Coherence	MESI
TLB entries	16
Branch predictor	256-entry Branch Target Buffer
Instruction buffer	16 entries for each core
Pipeline width	2 (2 Integer, FP ALUs, 1 SIMD unit)

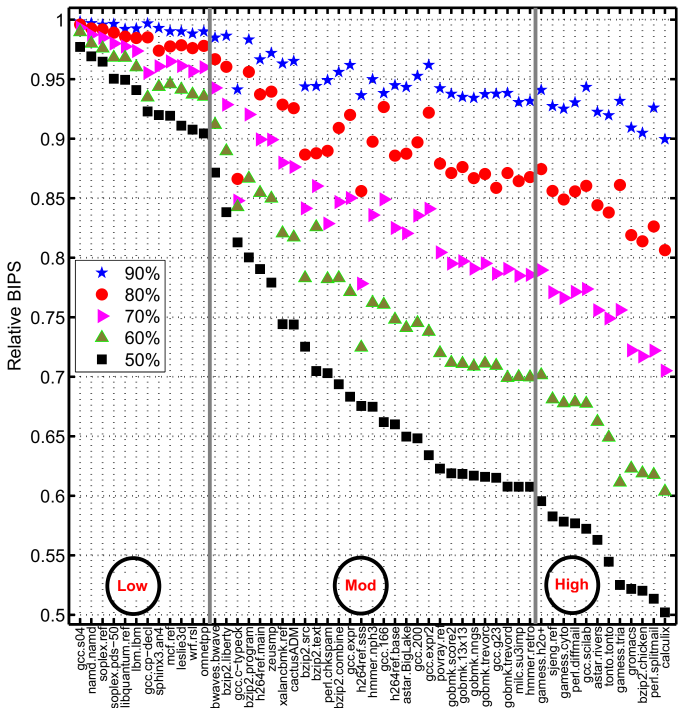
(a) Simulator parameters.

Tech: 22nm; Nominal frequency: 2.27GHz; Core Size: 1.56 x 3.9 mm^2
Number of chips used in each experiment : 100
V_{th}: 300mV, at V_{dd}=0.8V, T=80C, ϕ: 1cm; α: 1.3
V_{th}'s σ/μ: 0.21 ($\sigma_{ran}/\mu = \sigma_{sys}/\mu = 0.15$)
L_{eff}'s σ/μ: 0.105 ($\sigma_{ran}/\mu = \sigma_s ys/\mu = 0.07$)

(b) Process, Voltage and Temperature Parameters

Scheduling Policy	Description
Min-Max	Statically assign (no thread migration) processes to cores in *min-max* frequency setting
Static	Statically assign (no thread migration) processes to cores in Heterogeneous Core Frequency (HCF) setting (Section 3)
Round-robin	Fairness by round-robining processes every scheduling tick (Section 3.2.1)
Fairness-driven	Prediction-based approach to achieving fairness (Section 3.2.2)
Throughput-driven	Assignment of processes to cores to increase throughput (Section 3.3)
TDF	Weighted combination of Fairness- & throughput-driven policies (Section 3.4)

(c) Summary of scheduling policies



(d) Frequency sensitivity of SPEC CPU 2006 workloads

Figure 5: (a-c): Simulation, process-variation model parameters & scheduling policies. (d) Frequency-sensitivity based workload groupings, shown as *low*, *mod* and *high* categories (discussed in Section 4.3), used in our evaluations.

of the microcontroller. We assume that the register state can be migrated using the core C6 state array [1] or a similar small SRAM of a few KB. Although other mechanisms have been suggested to reduce the architectural state transfer penalty further [23], large intervals between migrations in our policies make such optimizations unnecessary. The simulator models data-transfer between clock domains, and accounts for delays introduced by synchronization buffers. L2 accesses that may be serviced by the cache bank local to the core or by remote banks exhibit variable latencies, and are accurately modeled. Following migrations, processes moving to a new core rely on MESI coherence protocol to handle on-demand transfer of data.

We use SPEC CPU 2006 applications for our evaluations [28]. We use 100-million instruction pinpoint regions for every application [20], choosing the pinpoint with the highest weight in the overall application. All simulations are run for 15 *ms* after warming up the caches prior to the pinpoint region. This simulation interval is on the order of one O/S quantum, which is the granularity at which we aim to present the appearance of fairness and uniform performance.

4.3 Workload Groupings

We simulated SPEC CPU 2006 on various core frequencies to determine each applications' frequency sensitivity. Figure 5d plots the performance of applications normalized to their performance at 100% frequency of 2.27 GHz. Workloads are shown on the x-axis and BIPS relative to a maximum frequency on the y axis. From top to bottom, the data series represent running the application at 90%, 80%, 70%, 60%, and 50% of the maximum frequency. Workloads are sorted based on decreasing relative throughput at 50% of the maximum frequency. For example, the workloads on the far right perform poorly at low frequency and are thus highly sensitive.

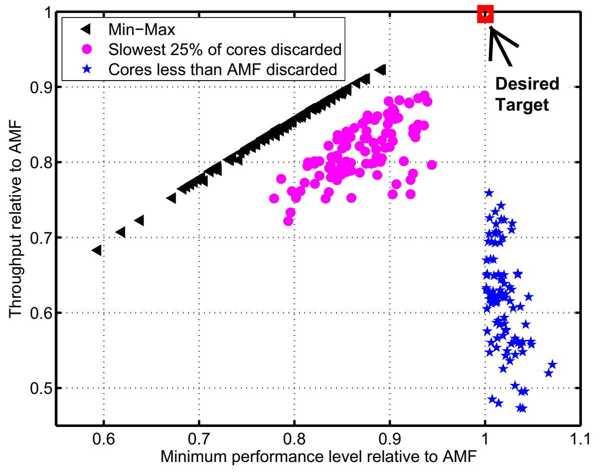
Figure 5d also shows the division of workloads into three groups, low, moderate (mod), and high frequency sensitivity. As can be seen, the low category contains 12 lowest-sensitivity applications, and in our 24-core model, we run two copies of each application. In the high category, we run 2-copies of 12 highest-sensitivity applications. In addition, we examine combinations of these three categories to form low-moderate, moderate-high, and low-high workload categories picking 12 applications from each category. We present results using all six categories in Section 5.

4.4 Measuring Performance

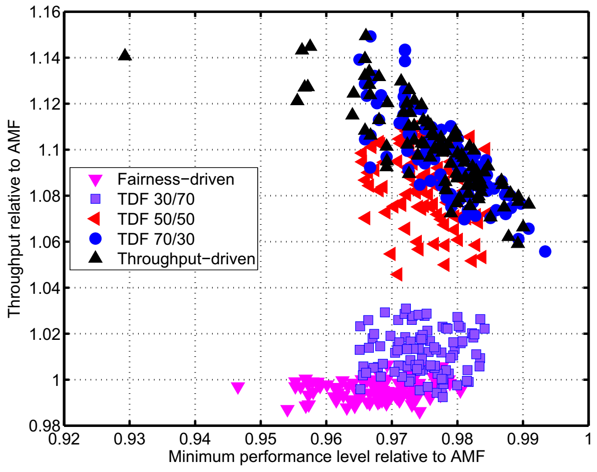
To understand the throughput and fairness aspects of various mechanisms, we present performance in two dimensions. First, the y-axis in our results (Section 5) presents total throughput all processes, each process' throughput relative to its throughput at the AMF. Let ti represent throughput of a thread i measured in billions of instructions per second (BIPS), and ti_{amf} represent the threads' AMF throughput over the same measurement interval. For our configurations involving 24 threads running on 24 cores, we obtain throughput relative to AMF as shown by the expression below:

$$\frac{\frac{t1}{t1_{amf}} + \frac{t2}{t2_{amf}} + \ldots + \frac{t23}{t23_{amf}} + \frac{t24}{t24_{amf}}}{24}$$

This value is a weighted speedup [26] with the weights being AMF performance, and accounts for inherently low-IPC processes as much as it does for high-IPC processes during our measurement interval. Second, the x-axis in our results presents the minimum-performance-level (MPL) - the throughput of the single worst pro-



(a) Disabling cores to achieve AMF performance-level.



(b) TDF analysis with different fairness & throughput weights.

Figure 6: (a) Cores can be disabled to improve minimum performance-level, but throughput suffers by doing so. (b) Performance of TDF with different fairness and throughput weights as inputs. Throughput-driven (bottom legend) has no notion of fairness, and fairness-driven (top legend) has no notion of throughput.

cess relative to that process's AMF throughput, as shown by the expression below:

$$minimum\left(\frac{t1}{t1_{amf}}, \frac{t2}{t2_{amf}}, \ldots, \frac{t23}{t23_{amf}}, \frac{t24}{t24_{amf}}\right)$$

Note that poor MPL cannot be obscured by well-performing high-throughput processes because MPL reflects only one thread. Although weighted speedup metrics partially convey this type of data in one dimension, our two-dimensional presentation is appropriate here because we are concerned about fairness for the individual process with the worst performance impact. A 1-dimensional weighted-speedup could still conceal starvation of one or more processes.

5. Results and Analysis

We discussed several policies in Section 3 aimed at squeezing performance out of HCF configurations, yet maintain a uniform performance level. In this section, we present experimental studies to quantify the benefits of these policies across the variety of workload groupings discussed in Section 4.3. For all results, all scheduling overheads—migration costs arising from pipeline drain, cache losses, architected state transfer costs, performance losses arising from prediction inaccuracies for throughput, fairness and TDF policies—are completely accounted for as discussed in Section 4.2.

As discussed in Section 3, it is viable to simply disable slow cores in a chip to boost MPL. Core disabling is a straight-forward extension to min-max, and we evaluate it first because of the simplicity of this approach to boost MPL.

5.1 Core Disabling to Achieve AMF

Disabling cores to boost minimum performance-level is viable but comes at the cost of reduced chip throughput. Figure 6a compares chip throughput and MPL, for our collection of *lowhigh* workloads discussed in 4.3 as a representative example. The x-axis shows MPL, which is the throughput of the slowest process relative to its throughput at the (artificial) AMF (Section 4.4). Higher MPL values brings the performance of slowest application closer to AMF, and are better. The y-axis shows total throughput for all active cores (desired target is annotated in the figure). The symbols each represent one chip out of a population of 100 for the configurations shown in the legend.

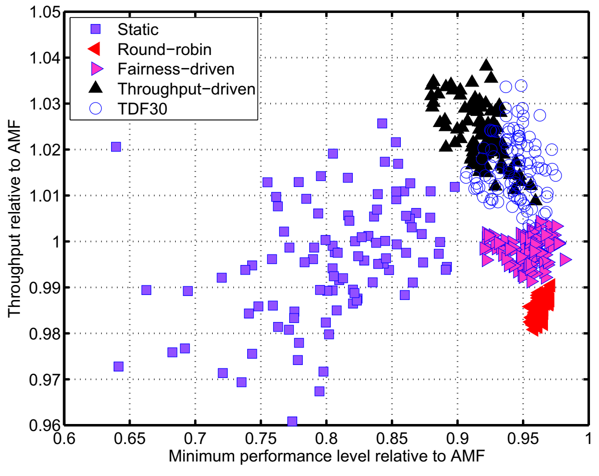
As shown in the figure, discarding the slowest 25% of cores on each chip helps significantly improve the MPL for the majority of chips (x-axis value is higher). Further, as expected, discarding cores that are below AMF guarantees AMF performance-level (or above) for all chips by definition, since all cores slower than AMF are discarded. However, discarding cores results in significant loss of chip throughput. Many chips using discarding have total throughput (y axis) that is lower than that for the vast majority of chips if min-max were used. When all cores slower than AMF are discarded (i.e., about half of the cores), total throughput is quite low. As a result, we do not discuss core disabling further and do not include it in our evaluations. As we will show next, our proposed policies retain the throughput advantages of having all cores active while boosting MPL close to the AMF performance-levels.

Before we examine our scheduling policies on a variety of workload configurations, we analyze fairness and throughput trade-offs of TDF using our collection of *lowhigh* workloads. This helps us narrow the scope of the results to be presented later.

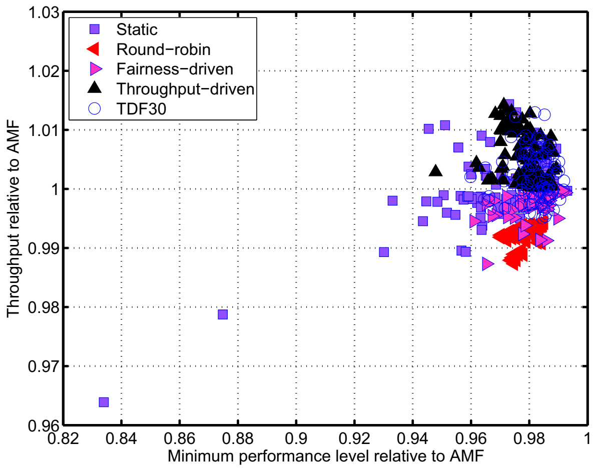
5.2 Throughput-Driven Fairness Analysis

In this section, we evaluate performance of TDF algorithm (described in Section 3.4) using different weights as inputs for throughput and fairness components, and understand trade-offs. Figure 6b presents these results using our present collection of *lowhigh* workloads; note that the axes scales are different from Figure 6a. Each symbol represents a different configuration. The TDF X/Y cases represent an X% weighting of throughput and a Y% weighting of fairness that are both inputs to the algorithm. We omit round-robin scheduling from this figure to avoid clutter.

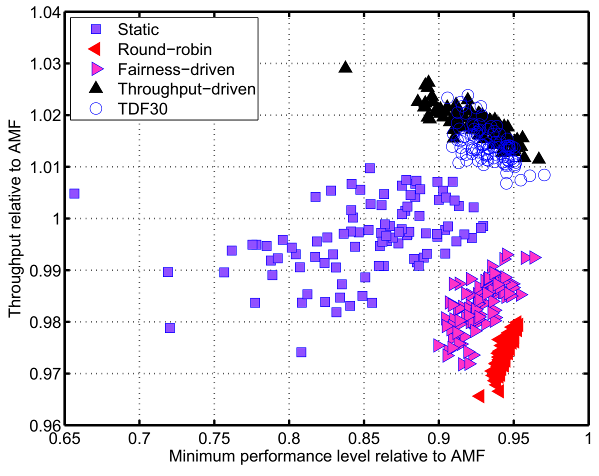
At the extreme of fairness-driven, or TDF 0/100, (shown in the inverted triangles at the bottom of the figure), targets only fairness using predictions for each thread as discussed in Section 3.2.2. It achieves high MPLs; for all but one chip the MPL is above 95% of the AMF throughput. However, it has the following drawbacks: a) even modest prediction-errors on high-sensitivity applications



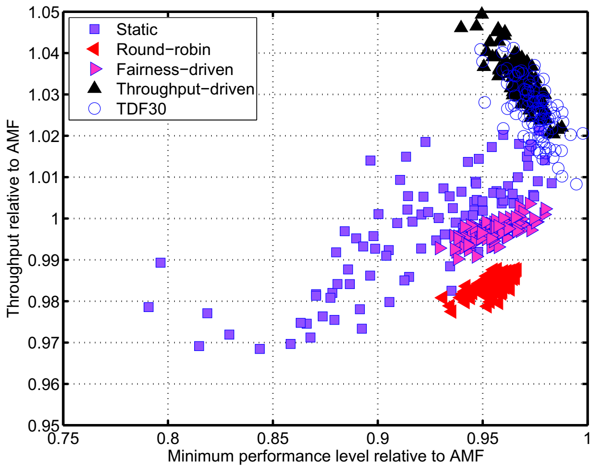
(a) High-sensitivity workloads



(b) Low-sensitivity workloads



(c) Moderate-sensitivity workloads



(d) Mix of Low- & Moderate-Sensitivity workloads

Figure 7: Performance of different approaches on workload groupings discussed in Section 4.3 (Figure 5d). Note that the x- and y- axes scales vary in order to provide reasonable presentations of the data.

result in lower system MPL for those chips; and b) it does not exploit high- and low-workload mix to boost throughput.

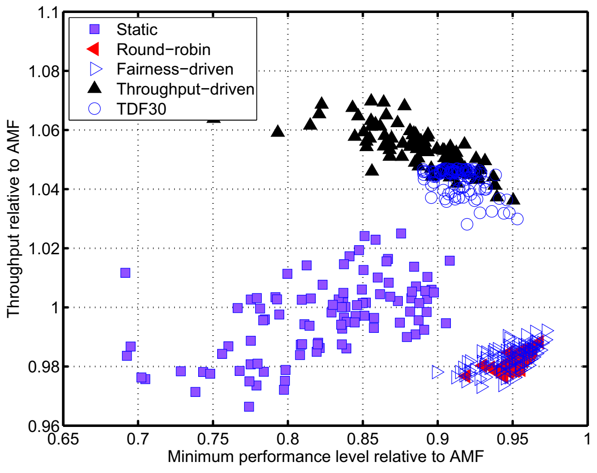
Reducing the fairness weight to 70% in the algorithm, shown in TDF 30/70 scenario, allows the high-sensitivity applications to seek cores to improve throughput. This effect can be seen by comparing the throughput values that are higher for TDF 30/70. Further, chips with low MPLs stemming from prediction-errors do better. We continue to see throughput improvements in 50%-50% case, while preserving MPL. At 70% throughout weight, shown by TDF 70/30 scenario, throughput-driven mechanism works aggressively to achieve throughput at the expense of increasing the MPL spread. At the extreme of purely throughput-driven, TDF 100/0, we see widely scattered MPLs, including one at only 93% of the AMF throughput.

Because TDF 70/30 achieves high throughput while maintaining high MPL—all chips are above 96% of the AMF throughput—our remaining analysis uses this configuration. We observed TDF 70/30 to perform well for the other workload groupings. For the remainder of the paper, we call TDF 70/30 simply TDF30.

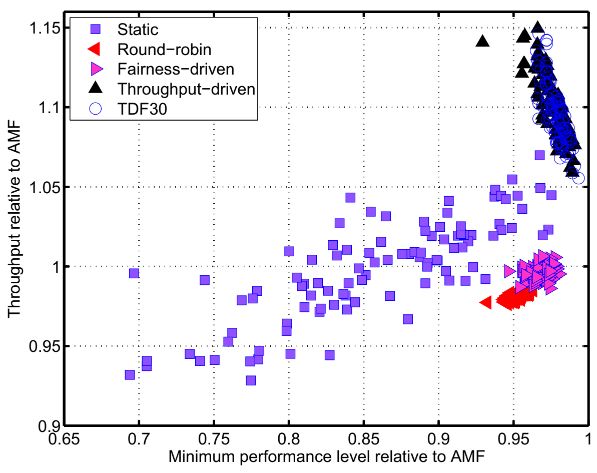
5.3 Workload Mix Analysis

In this section, we present results of our scheduling policies using frequency-sensitivity based workload groupings discussed in Section 4.3. The basic format of the figures in this section are the same as before (discussed in Section 4.4): the x-axis shows MPL—performance of the slowest application relative to its performance at the (artificial) AMF. Higher MPL values brings the performance of slowest application closer to AMF, and are better. The y-axis shows total throughput for all active cores. Each symbol represents one chip out of a population of 100 for the configurations shown in the legend (scheduling policies summarized in Table 5c). We expect to see the best throughput improvements on heterogeneous workloads that have a mix of sensitivities because the scheduling policy can exploit the variety to increase throughput.

High-sensitivity workloads. High-sensitivity workloads, which typically tend to be compute-bound, have strong preferences for high-frequency cores. As a result, static scheduling exhibits extremely poor MPL, the lowest of all workload categories we evaluated (Figure 7a). Round-robin allows all processes to utilize high-frequency cores for a fraction of time and significantly improves MPLs. However, overall throughput for round-robin is low. Additional swap overheads incurred by round-robin are avoided by



(a) Mix of Moderate- & High-Sensitivity workloads



(b) Mix of Low- & High-Sensitivity workloads

Figure 8: Performance of different approaches on workload groupings discussed in Section 4.3 (Figure 5d). Note that the x- and y- axes scales vary in order to provide reasonable presentations of the data.

fairness-driven while still maintaining a tight distribution of chip MPLs. Throughput-driven approach, which aggressively schedules processes predicted to provide high-performance during the scheduling tick, squeezes additional performance out of these workloads. By doing so, other applications in the mix suffer and MPLs shifts unfavorably down. TDF30, by giving some weight to fairness considerations, avoids this scheduling imbalance and recovers MPL.

Low-sensitivity workloads. Figure 7b presents results for a collection of *low-sensitivity* workloads for the various configurations shown in the legend. Low-sensitivity workloads, which are often memory-bound, are more tolerant to core frequency differences. As a result, this collection of workloads offers the best-case scenario for static HCF scheduling. As can be seen, barring a few chips, the majority of chips can provide a high MPL for static. Round-robin scheduling, by allowing all workloads to spend equal amount of time on all cores, removes the outliers found in static scheduling, and tightens the spread of MPL for all chips. Fairness-driven, throughput-driven and TDF30, have little differences in their MPL distributions. Also, in this scenario, throughput benefits are minimal for all scheduling policies, simply because the applications do not have core preferences. While the low-sensitivity workloads are not particularly interesting, all of our scheduling policies perform as well or better than the reference static and round-robin policies in spite of the overheads introduced by our policies.

Moderate-sensitivity workloads. This set features workloads with moderate frequency sensitivity and varying execution behavior. Throughput-driven exploits execution variations, and schedules processes to improve throughput (Figure 7c), at the cost of reducing MPLs. Even so, by exploiting intra-process phase changes, MPL is significantly improved compared to static configuration. TDF narrows the MPL spread for this set of workload preserving throughput benefits. Using TDF policy, all chips achieve at least 90% of AMF performance-level, providing clear benefits compared to static HCF scheduling.

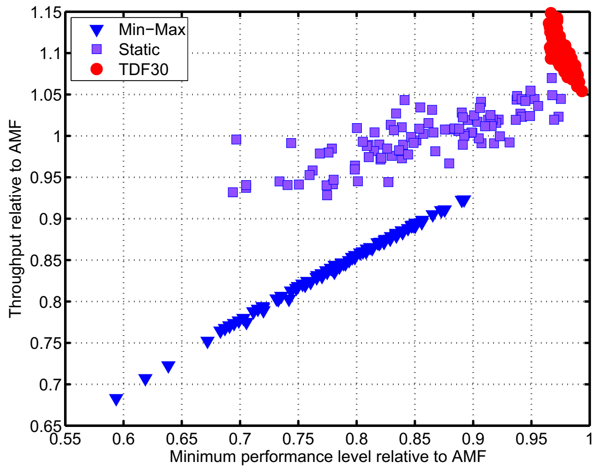
Low- & Mod-sensitivity workload combination. Differences in workload demands for this set results in advantages to throughput-driven policy compared to the earlier category of only moderate-sensitivity workloads (Figure 7d). Although throughput-driven continues to cater to applications with need for high-frequency cores, presence of other low-sensitivity applications, which are mostly core agnostic, results in no significant MPL reduction compared to the other fairness-based policies. It is consistent with our expectations that availability of workloads at different performance levels in general helps TDF, as is also seen in the *mix* categories below, which can manage applications to cater to both throughput and fairness.

Mod- & High-sensitivity workload combination. This set of workloads (Figure 8a) combine the effect of moderate and high-sensitivity categories discussed above. Similar to the high-sensitivity category, static significantly suffers from throughput and fairness losses amidst a collection of applications of which many benefit from high-frequency choices. Further, aggressive scheduling of throughput-driven affects its MPL, a trend we saw in high-sensitivity category above. TDF significantly tightens the spread of MPL in this category, yet the throughput is within 1% of the purely throughput-based implementation.

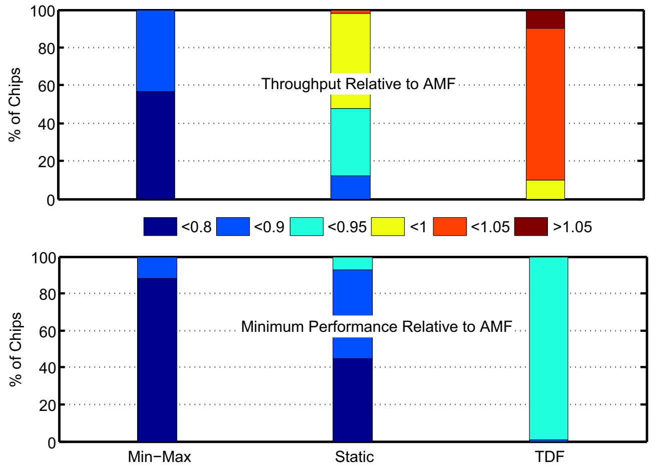
Low- & High-sensitivity workload combination. Our final set of workloads (Figure 8b) features both compute-bound and memory-bound applications, offering the best case scenario for our policies. Aggressive throughput-driven scheduling squeezes performance from high-sensitivity workloads, without affecting MPL in the comfort of other frequency-agnostic workload choices. TDF performs well by removing the few outlier chips and preserves MPL similar to fairness-driven approaches. Overall, we obtain 20% performance benefits from TDF compared to naive round-robin, while maintaining the same MPL.

5.3.1 Workload Mix Analysis Wrap-up

We observe that TDF policy balances throughput and fairness well across a wide variety of applications. The collection of workloads we examined earlier, a mix of low-sensitivity applications, provides the best-case scenario for static configurations and a limited opportunity for throughput or fairness policies. We showed that, even accounting for all costs, proposed scheduling perform as well or better than static throughput and MPL for this category. Other workload categories provide significantly more average performance and MPL. Across all workloads, TDF30 achieves a minimum performance level at or better than 90% of the AMF throughput (with a few exceptions just below 90 in Figure 8a). The final set of low- and



(a) Comparison of Min-Max, Static HCF & TDF



(b) Percentage of chips at various throughput & MPL relative to AMF

Figure 9: Summary of benefits: (a) Static HCF recovers throughput lost in Min-Max but suffers from low MPL. TDF exhibits highest MPL and throughput. (b) TDF scheduling enables 90% of the chips to *exceed* AMF throughput (top), and over 98% of the chips have MPL within 10% of the desired AMF level (bottom).

high-sensitivity workloads illustrate the benefits of TDF, by boosting performance up to 20% over a pure fairness-driven, round-robin solution while maintaining high MPL.

5.4 Summary of Benefits

Figures 9a brings together Min-Max, Static HCF and TDF approaches. For these studies, we return to our mix of *lowhigh* sensitivity applications. As shown in the plot, most chips using Min-Max have significantly lower throughput than the potential AMF performance level. Further, MPL with respect to the AMF performance-level is significantly lower because one or a few slow cores limit the frequency of the entire chip. We observe that Static HCF recovers throughput lost in Min-Max. However, Static suffers from low MPL as was shown in the previous section, largely due to the assignment of some frequency-sensitive applications to slow cores. TDF, which caters to the needs of frequency sensitive applications while ensuring a level of minimum performance for all applications, exhibits the highest MPL and the highest overall throughput.

Finally, Figure 9b summarizes the speed bin distributions of chips for min-max, static, and TDF. The top graph shows the throughput relative to the AMF across the chip populations (equivalent to y axis on previous graph), while the bottom graph shows the MPL relative to the AMF (equivalent to x axis on previous graph). From Figure 9b(top), we observe that static HCF enables 80% of the chips to provide within 10% of the AMF-level throughput, and 2% to exceed AMF-level throughput. In contrast, TDF enables 90% of the chips to *exceed* AMF-level throughput. The MPL plot shows that, using TDF, over 98% of the chips can guarantee their minimum performance to be within 10% of the desired AMF performance-level (a small sliver representing one slow chip is not visible due to the scale). This represents a significant increase in high frequency premium chips compared to static HCF: only 9% of the chips can guarantee that MPL. This trend is also true for other workload categories as discussed at the end of the last section. Note that while the naive round-robin policy is not shown on these graphs, that policy would result in both total and minimum throughput very near the AMF, negating the throughput benefits of TDF. TDF provides a strong balance of high total throughput and consistent performance across the population of chips.

6. Related Work

Our contributions are related to research in the areas of process variations, heterogeneous- and variation-aware scheduling, and thread migration.

Process variations. Nikolic et al. provide a good overview of process variations [18]. Pang et al. built a test chip in 90 *nm* process to characterize lithography induced variations [19]. Their results show that variations are normally distributed. Bowman et al. present a FMAX model derived from D2D and WID statistical process models, and verify it using wafer sort data for 250*nm* microprocessor [5]. Using the FMAX distribution model, they show that the WID variation directly impacts FMAX mean and the D2D variation affects FMAX variance. Humenay et al. [11] estimate core frequencies in their work, and suggest adaptive body bias and supply voltage to reduce variations. Instead, we exploit variations to boost performance.

Scheduling techniques (Variation-aware, heterogeneous, & SMT). Throughput-driven fairness evaluation in this paper is related to numerous other work on thread-scheduling in SMT, heterogeneous, and variation-aware architectures. In the context of SMT processors, Luo et al. present thread starvation techniques while boosting overall system throughput [15]. Mutlu et al. propose fairness mechanisms for multiple applications sharing DRAM memory system [17]. Tiwari et al. study aging-driven scheduling to slow down aging of chips or to boost frequency [29]. Winter et al. evaluate O/S-level scheduling policies for unpredictably heterogeneous multi-cores affected by various manufacturing and wear-out defects [30]. Their more recent work [31] expands the performance and power analysis to heterogeneous many-core systems. In contrast, our policies implemented at fine temporal windows seek appearance of a single, uniform-level of performance to processes in an O/S transparent manner.

Teodorescu et al. [27] propose application scheduling and power management policies to improve performance and energy-efficiency. Although their scheduling policies use the same general intuition— applications that are memory-bound benefit less from being scheduled on high-frequency cores—they do not treat software-process fairness as a first-order consideration, and their evaluations are focused on improving throughput and ED^2 without considering if assignments are unfair to different processes. On systems featuring per-core DVFS, Isci et al. [13] propose using a global power manager to re-evaluate core power levels every 500 μs using run-time application behavior. Herbert et al. [10] study frequency and power heterogeneity from process variations and evaluate DVFS algorithms. Kumar et al. [14] propose using single-ISA, heterogeneous CMPs varying in resources and complexity to efficiently accommodate diverse set of applications. In comparison to these efforts, the goal of this work is to maximize performance yet provide the appearance of a single performance-level to running processes.

Thread migration. Heo et al. [9] propose core hopping to combat thermal problems, and argue that the interval between core migrations can be smaller than typical O/S context swap times for maximum benefit. We argue the same for performance and to provide a uniform performance level to O/S. Powell et al. [21] propose using thread migration to prevent local hot spots that can cause chip failure. Recently, Powell et al. [22] propose migration to move threads away from defective cores. They assume similar architected state transfer costs. Rangan et al. [23] propose thread motion, a fine-grained implementation of thread migration. However, their evaluations are limited to cores that share first-level caches.

7. Conclusion

Ignoring inter-core frequency variations, and setting chip frequency to be that of the slowest core sacrifices substantial performance to achieve simplicity. Alternatively, per-core frequency configurations result in the loss of uniform performance-level to O/S and end-users, and poses scheduling challenges. Our scheduling policies restore the appearance of single, uniform performance-level to the users and the O/S while allowing for individual cores to be clocked at frequencies higher than that allowed by the slowest core. Detailed analysis show that our scheduling policies guarantee minimum application performance near that of the mean frequency of the chip, in lieu of its minimum frequency, by masking process-variation induced heterogeneity. Our throughput-driven fairness policy brings together fairness and throughput considerations, and improves throughput by an average of 12% compared to round-robin for frequency-sensitive applications. TDF policy allows 90% of chips to have total throughput *above* the throughput expected at the average frequency. At the same time, TDF allows 98% of chips to maintain minimum performance at or above 90% of that expected at the mean frequency, providing a single uniform performance level to present for the chip.

Acknowledgments

Special thanks to Meeta Gupta for help with various aspects of this work. We are also thankful to Carl Beckmann, Arijit Biswas, Nitin Borkar, Saurabh Dighe, Joel Emer, Brian Greskamp, William Hasenplaugh, Glenn Holloway, Aamer Jaleel, Kris Konigsfeld, Athanasios Papathanasiou, Harish Patil, Steve Raasch, Paul Racunas and VJ Reddi for their comments and suggestions. We are also grateful to the anonymous reviewers for their feedback. This work is partially supported by National Science Foundation grants CCF-0048313, CSR-0720566 and CCF-0702344. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

References

[1] Intel Core 2 Duo Processor and Intel Core 2 Extreme Processor on 45-nm Process for Platforms Based on Mobile Intel 965 Express Chipset Family Datasheet. http://www.intel.com/design/mobile/datashts/316745.htm.
[2] Intel Core i7-900 Desktop Processor Extreme Edition Series on 32-nm Process: Datasheet, Volume 1. http://download.intel.com/design/processor/datashts/323252.pdf.
[3] Intel Turbo Boost Technology. http://www.intel.com/technology/turboboost/index.htm.
[4] Intel's Tick-Tock Model. http://www.intel.com/technology/tick-tock/index.htm.
[5] K. A. Bowman and S. G. Duvall. Impact of Die-to-Die and Within-Die Parameter Fluctuations on the Maximum Clock Frequency Distribution for Gigascale Integration. *IEEE JSSC*, 2002.
[6] S. Dighe, S. Vangal, et al. Within-Die Variation-Aware Dynamic-Voltage-Frequency Scaling Core Mapping and Thread Hopping for an 80-Core Processor. In *ISSCC*, 2010.
[7] J. Emer et al. Asim: A Performance Model Framework. In *IEEE Computer*, pages 68–76, 2002.
[8] G. Gerosa et al. A Sub-2 W Low Power IA Processor for Mobile Internet Devices in 45 nm High-k Metal Gate CMOS. *IEEE JSSC*, 2009.
[9] S. Heo, K. Barr, and K. Asanovic. Reducing Power Density through Activity Migration. In *ISLPED*, 2003.
[10] S. Herbert and D. Marculescu. Variation-Aware Dynamic Voltage/Frequency Scaling. In *HPCA*, 2009.
[11] E. Humenay et al. Impact of Process Variations on Multicore Performance Symmetry. In *DATE*, 2007.
[12] C. Isci et al. Long-Term Workload Phases: Duration Predictions and Applications to DVFS. *MICRO*, 2005.
[13] C. Isci et al. An analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget. In *MICRO*, 2006.
[14] R. Kumar et al. Single-ISA Heterogeneous Multi-Core Architectures for Multithreaded Workload Performance. In *ISCA*, 2004.
[15] K. Luo et al. Balancing Thoughput and Fairness in SMT Processors. In *ISPASS*, 2001.
[16] R. McGowen et al. Power and Temperature Control on a 90-nm Itanium Family Processor. *IEEE JSSC*, Jan. 2006.
[17] O. Mutlu and T. Moscibroda. Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors. In *MICRO*, 2007.
[18] B. Nikolic and L. Pang. Measurements and Analysis of Process Variability in 90nm CMOS. In *International Conf on Solid-State and Integrated Circuit Technology*, 2006.
[19] L. Pang and B. Nikolic. Impact of Layout on 90nm CMOS Process Parameter Fluctuations. In *Symposium on VLSI Circuits*, 2006.
[20] H. Patil et al. Pinpointing Representative Portions of Large Intel Itanium Programs with Dynamic Instrumentation. In *MICRO*, 2004.
[21] M. Powell et al. Heat-and-run: Leveraging SMT and CMP to Manage Power Density Through the Operating System. In *ASPLOS*, 2004.
[22] M. Powell et al. Architectural Core Salvaging in a Multi-Core Processor for Hard-Error Tolerance. In *ISCA*, 2009.
[23] K. Rangan et al. Thread Motion: Fine-Grained Power Management for Multi-Core Systems. In *ISCA*, 2009.
[24] S. Sarangi et al. VARIUS: A Model of Process Variation and Resulting Timing Errors for Microarchitects. *IEEE Transactions on Semiconductor Manufacturing*, 2008.
[25] G. Semerano et al. Energy-Efficient Processor Design Using Multiple Clock Domains with Dynamic Voltage and Frequency Scaling. In *High Performance Computer Architecture*, 2002.
[26] A. Snavely and D. Tullsen. Symbiotic Jobscheduling for a Simultaneous Multithreading Processor. In *ASPLOS*, 2000.
[27] R. Teodorescu and J. Torrellas. Variation-Aware Application Scheduling and Power Management for Chip Multiprocessors. In *ISCA*, 2008.
[28] The Standard Performance Evaluation Corporation. Spec CPU2006 suite. http://www.specbench.org/osg/cpu2006/.
[29] A. Tiwari and J. Torrellas. Facelift: Hiding and Slowing Down Aging in Multicores. In *MICRO*, 2008.
[30] J. A. Winter and D. H. Albonesi. Scheduling Algorithms for Unpredictably Heterogeneous CMP Architectures. In *Dependable Systems & Networks*, 2008.
[31] J. A. Winter et al. Scalable Thread Scheduling and Global Power Management for Heterogeneous Many-Core Architectures. In *PACT*, 2010.