Thread Motion: Fine-Grained Power Management for Multi-Core Systems

Krishna K. Rangan†‡ 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}@intel.com

ABSTRACT

Dynamic voltage and frequency scaling (DVFS) is a commonly-used power-management scheme that dynamically adjusts power and performance to the time-varying needs of running programs. Unfortunately, conventional DVFS, relying on off-chip regulators, faces limitations in terms of temporal granularity and high costs when considered for future multi-core systems. To overcome these challenges, this paper presents thread motion (TM), a fine-grained power-management scheme for chip multiprocessors (CMPs). Instead of incurring the high cost of changing the voltage and frequency of different cores, TM enables rapid movement of threads to adapt the time-varying computing needs of running applications to a mixture of cores with fixed but different power/performance levels. Results show that for the same power budget, two voltage/frequency levels are sufficient to provide performance gains commensurate to idealized scenarios using per-core voltage control. Thread motion extends workload-based power management into the nanosecond realm and, for a given power budget, provides up to 20% better performance than coarse-grained DVFS.

Categories and Subject Descriptors

C.1.4 [**Processor Architectures**]: Parallel Architectures—*Distributed architectures*

General Terms

Performance, Design

1. INTRODUCTION

Power dissipation continues to be a primary design constraint in the multi-core chip era. Increasing power consumption not only results in increasing energy costs, but also results in high die temperatures that affect chip reliability, performance, and packaging cost. From the performance standpoint, current and future multi-core systems will have to carefully constrain application performance to stay within power envelopes. For example, power constraints result in reduced per-core throughput when multiple cores are active in current Intel processors [2]. Fortunately, multi-core systems host applications that exhibit runtime variability in their performance requirements, which can be exploited to optimize throughput while staying within the system-power envelope.

Dynamic voltage and frequency scaling (DVFS) schemes seek to exploit runtime variability in application behavior to achieve maximum energy savings with minimal performance degradation. However, traditional DVFS scaling, which is initiated by the operating system (OS), has two primary drawbacks: First, OS scheduler sampling intervals are on the millisecond time scale, while computational requirements can vary on the nanosecond time scale due to events such as cache misses. Hence, OS-driven DVFS is too slow to respond to such *fine* variations in program behavior. Second, multi-core systems execute multiple applications with potentially very different computational needs. Even though the performance advantages of per-core DVFS in multi-core systems have been suggested [11, 15], providing per-core, independent voltage control in chips with more than two cores can be expensive [15]. Moreover, when DVFS is applied across multiple cores, determining a single optimal DVFS setting that simultaneously satisfies all cores will be extremely difficult; some applications will suffer performance loss or power overheads. This problem worsens as the number of cores and running applications increase in future systems.

Clearly, a fast-acting, yet cost-effective mechanism to obtain the benefits of per-core DVFS on systems with a large number of cores is desirable. Trends in current multi-core systems suggest: (1) Even though per-core, independent voltage control is currently impractical, future systems with a multitude of cores can be expected to have a small number of independent voltage and frequency domains [1, 3]. As such, cores that differ in power-performance capabilities will exist. (2) Future high-throughput systems are likely to pack together a large number of simple cores [23, 25, 27] hosting many more applications. Unfortunately, these trends further exacerbate the problems of using conventional DVFS. To address these limitations, we propose a fast, fine-grained power-management approach that we call *thread motion* (TM).

Thread motion is a power-management technique that enables applications to migrate between cores in a multi-core system with simple, homogeneous cores but heterogeneous power-performance capabilities. For example, envision a homogeneous multi-core system where cores differ only in terms of their operating frequency and voltage. Such power-performance heterogeneity offers a way to accommodate a wide range of power envelope levels without limiting the performance of all of the cores together. Instead, it offers a mixture of performance capabilities with a small number of static voltage/frequency (VF) domains. As applications run on these cores, TM enables applications to migrate to cores with higher or lower VF settings depending on a program's time-varying compute intensity. If one application could benefit from higher VF while another is stalled on a cache miss, a swap of these two applications between cores of different power capabilities may provide overall improvements in power-performance efficiency. Compared to slow transition times of conventional regulator-based DVFS schemes, thread motion can be applied at much finer time intervals

ISCA'09, June 20–24, 2009, Austin, Texas, USA

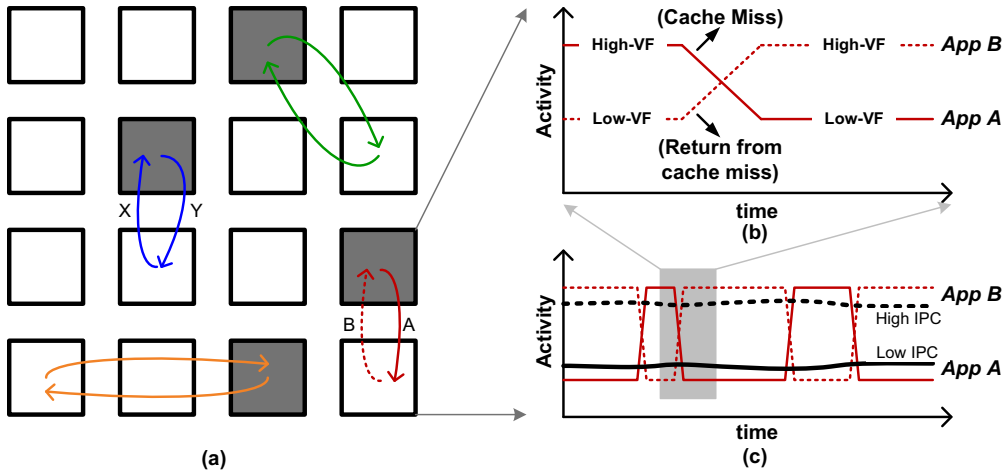


Figure 1: (a) Illustration of thread motion in a multi-core system. (b) Exploiting fine-grained application variability in two running threads. (c) Duty cycling between 2 VF levels to match application IPC.

and applied more often. Another potential benefit of rapidly moving applications between cores is that, by doing so, applications experience a virtual (effective) voltage that is in between the fixed voltage levels of different cores. In fact, thread motion yields the net effect of implementing multiple VF domains while only using two VF levels and keeping system-level costs in check.

While the potential performance benefits of thread motion may be intuitively obvious, it is important to identify and evaluate all of the costs and overheads associated with it. Future multi-core systems offer a plethora of design choices in terms of the number of cores and core complexity. Simple cores with shared resources are likely to be present in future many-core systems featuring tens or hundreds of cores among other possible design alternatives. Hence, we evaluate thread motion in the context of a high-throughput machine featuring clusters of simple in-order cores with some shared L1 cache state (similar to Rock [27]). We study the performance benefits of thread motion constrained to different power envelopes using two approaches: (1) a coarse-grained prediction-driven approach evaluated at fine granularities, called *time-driven* TM; and (2) a last-level cache miss driven approach, called *miss-driven* TM. To understand the cost-benefit tradeoff, we provide a detailed analysis of all power and performance costs associated with TM.

The main contributions of this paper are as follows:

1. Fine-grained program variability motivates the need for fine-grained power management—we analyze SPEC CPU 2006 applications and show that applications can have significant fine-grained performance variability (Section 2). For several applications, we show that variability observed at 300-cycle granularities is an order of magnitude higher than when observed at coarser 10000-cycle granularities.
2. Thread motion on systems with two VF domains can perform nearly as well as having continuous, per-core VF levels but no motion. In other words, thread motion can be used to provide applications with any virtual (effective) power level between the lowest and the highest voltage/frequency settings (Section 3).
3. Thread motion can exploit fine-grained variability in application behavior. Even after accounting for all motion-related costs, up to 20% higher performance can be achieved compared to a design without motion that uses ideal (oracle-based) static VF assignments and equivalent power budgets (Section 5).

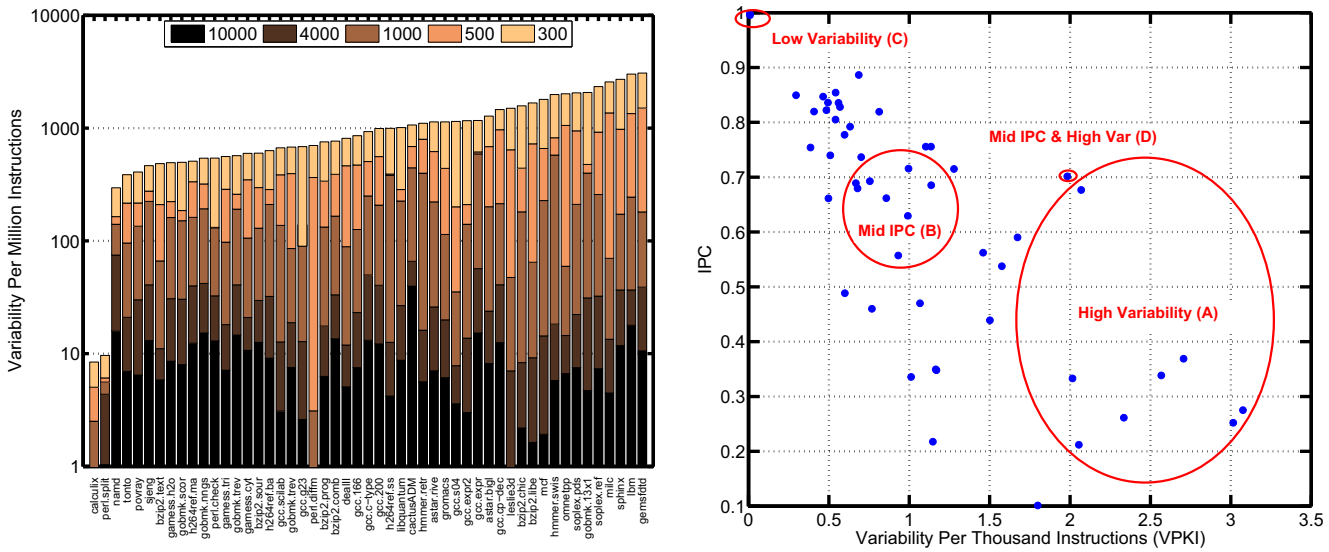
2. MOTIVATION

In this section, we take a closer look at traditional DVFS and elaborate on its limitations. We then introduce the basic concepts of thread motion, illustrate how it can overcome traditional DVFS limitations using synthetic examples, and briefly explain how it is different from other contemporary fine-grained power management schemes. Finally, we present studies of real applications that underscore the importance of fine-grained power management using thread motion.

2.1 Limitations of traditional DVFS

DVFS is a well-known and widely-used power management technique in contemporary microprocessors. In the context of CMPs, power management using dynamic VF scaling seeks to reduce power consumption when cores are idle, boost single-threaded performance in the presence of large workloads, and remap VF settings to improve performance and energy utilization. DVFS algorithms typically use application behavior to arrive at the appropriate VF setting for the core the application is running on. In multi-core systems with DVFS applied with a single power domain (or a small number of domains), the individual needs from all cores are consolidated to arrive at a single chip-wide VF setting, which often compromises performance and/or power efficiency. Otherwise, each core dictates individual settings on systems that support per-core VF, but there are cost and scalability concerns of implementing per-core DVFS on systems with a large number of cores. State-of-the-art systems such as Intel's Core 2 series use power gating to shut down power to the unused cores, but all active cores use the same voltage setting. AMD's Griffin processor does provide dual-power planes for per-core voltage/frequency control [1], but prohibitively high costs of off-chip regulators will likely limit per-core voltage control beyond dual core systems.

DVFS algorithms are typically implemented in the operating system. Consequently, the application phase monitoring and requests for core power mode transitions occur at the millisecond time scale of the OS scheduler. Previous work [11] has recognized the importance of monitoring application phase activity on finer time scales and has proposed using a global power manager framework to re-evaluate DVFS decisions at intervals on the order of hundreds of microseconds. However, all state-of-the-art DVFS-based power management schemes in use (or proposed earlier) incur a large VF transition delay to arrive at the target power mode. The voltage transition delay, which is on the order of tens of microseconds, is



(a) Variability for different intervals

(b) Variability & IPC

Figure 2: Variability and IPC of SPEC CPU 2006 workloads.

due to off-chip voltage regulators that limit how quickly voltage can change, and the frequency transition delay comes from PLL relock times. These transition delays *fundamentally* limit re-evaluation of application behavior and remapping core VFs at finer time scales. In contrast, microarchitectural events such as cache misses introduce application variability at nanosecond granularities. Thread motion seeks to adapt to this microarchitectural variability and extend DVFS benefits to the nanosecond realm by working around the aforementioned VF transition delay.

Illustration of thread motion benefits. We illustrate the basic concepts of thread motion in Figure 1 given a collection of cores with heterogeneous performance. Figure 1(a) assumes two levels of core performance, where dark cores have higher performance. As the performance needs of the applications vary over time, thread motion moves applications amongst the cores. Fine-grained power management with thread motion offers two important benefits: (1) TM provides an opportunity to apply DVFS benefits to program variability arising out of microarchitectural events, which is impractical with conventional DVFS, and boost performance at a given power budget. This is illustrated in Figure 1(b)—fast TM enables application B to benefit from running on the high-VF core when application A is stalled on a cache miss. (2) TM enables fast movement of applications between cores, and by doing so, applications can appear to operate off of a virtual (effective) voltage that is between the fixed voltage levels of the two cores. As Figure 1(c) shows, the high IPC application B spends a larger fraction of its running time on the high-VF core, and realizes an effective VF that hovers near the upper range, between high- and low-VF levels. This has a net effect of achieving per-core DVFS benefits even on systems with only 2 VF levels.

Differences from other fine-grained approaches. Typically, contemporary fine-grained power management involves: (1) clock gating and gating of pipeline stages, but it is important to note that these benefits are orthogonal to the cubic-reduction in power benefits achieved using DVFS; and (2) fine-grained, rapid scheduling of new threads (using an SMT scheduler, for example), when the hardware or the compiler cannot avoid a thread stall. The newly scheduled thread masks the latency of the cache access of the waiting thread. However, systems that employ large degrees of multi-threading to exploit fine-grained variations in program behavior and increase throughput also require significant memory bandwidth and incur high system power penalties [29]. Furthermore, all cores have to run at peak power to always mask the latency of cache accesses, which may not scale in the context of future systems with many more cores. We expect future systems to be power-constrained; that is, not all cores will be at peak power all of the time. The goal of thread motion is to increase system throughput by maximally squeezing performance out of a fixed set of running applications at a given power budget.

2.2 Application Variability

Having looked at synthetic examples to illustrate the basic concepts of TM, we now turn our attention to understanding the behavior of real workloads. Our analysis of fine-grained power management begins by studying the characteristics of SPEC benchmarks. We collected representative program traces of SPEC CPU 2006 IA32/x86 binaries for use with our simulator (complete details of the simulation framework are provided in Section 4.2). Our simulator includes an in-order timing model for the core and a two-level cache hierarchy—separate 16KB L1 instruction and data caches and a unified 2MB L2 cache. For each memory instruction in the trace, the simulator determines hit/miss information and the level of the hierarchy where the hit occurred. Each memory instruction takes a number of cycles equal to the latency of the level in which it hits, and each non-memory instruction takes a cycle to finish. We assume hit latencies of 2, 12, and 150 cycles for L1, L2, and memory accesses. We found that the other hit latencies have little impact on the relative ordering of applications based on our variability metric. We use the IPC information to compute variabilities at various window sizes.

In order to understand variability in the runtime performance needs of applications, we have devised a simple metric. Our variability metric evaluates the magnitude of fine-grained change in application behavior across various window sizes. It is computed as follows. For each SPEC 2006 workload, we track the IPC for several different window sizes, by measuring the ratio of instructions committed to the total number of cycles within a window interval. We define *Interval Variability* as the absolute difference between

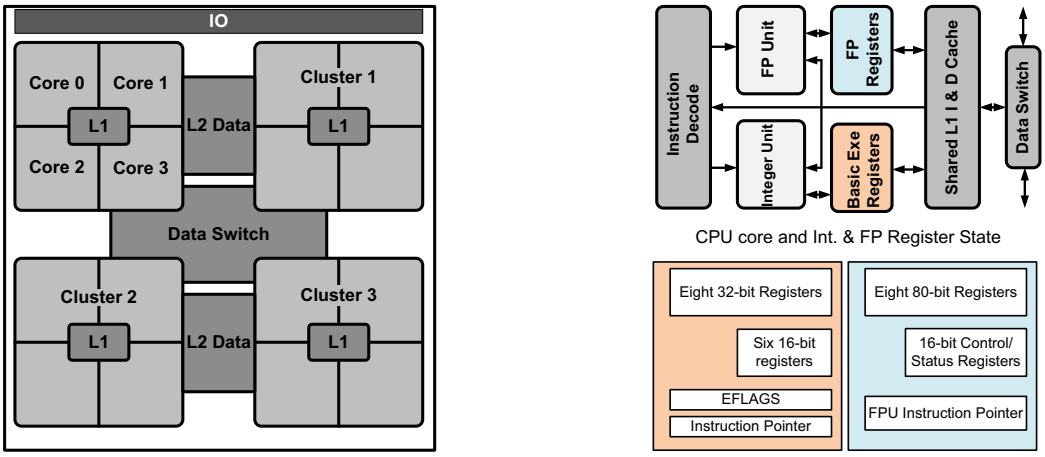


Figure 3: Architecture resembling Sun's Rock which has all cores sharing first-level cache [27]. Also shown are CPU core details and core RF states.

the IPC in an interval compared to the IPC in the previous interval. Total variability in the benchmark is the sum of interval variability across all of the intervals. Variability Per Instruction (VPI) is arrived at by dividing this total application variability by the total number of instructions in the trace. Figure 2(a) shows Variability Per Million Instructions for SPEC 2006 benchmarks and input combinations. Each stacked bar in the figure shows the cumulative variability for each window size (in log scale). As can be seen, variability is more pronounced at smaller window sizes (an order of magnitude higher for several workloads), but is muted at coarser levels. In other words, the amount of application variability is a strong function of the window (or sampling) interval. Smaller intervals help capture cache effects that stall the processor, while coarser intervals that span tens of thousands of processor cycles mask or average out the effects of different microarchitectural events. This variability at fine granularities has an important implication that motivates our work—increased power savings are possible using fine-grained power-management techniques.

In addition to understanding application variability, it is important to understand the relationship between variability and IPC. Figure 2(b) shows the relationship between the overall IPC of an application and variability. Cache effects dictate both variability and IPC, and the metrics are moderately correlated: low-IPC applications tend to have high variability and mid-IPC applications have moderate variability. Understandably, the two applications with highest overall IPC, *calculix* and *perl.splitmail*, are compute intensive with fewer last-level cache misses, and they exhibit very low variability. The figure shows four groups of applications that cover a range of application variabilities and IPCs. We will use these four groups to examine the performance impact of TM in Section 5.4. As we will see later, TM not only helps applications with high variability, but also benefits applications with little variability when the mixture of applications running in the multi-core processor has variability in IPCs (previously illustrated in Figure 1(c)). We will also show that TM can take advantage of variability available at fine resolutions for seemingly static, high-IPC workloads.

3. THREAD MOTION

The goal of thread motion is to enable DVFS benefits at nanosecond time scales by exploiting variations arising from microarchitectural events. However, the benefits of the technique must be carefully studied in the context of all the overheads it incurs. In this section, we take a closer look at TM limitations and introduce the architecture used to evaluate TM. We propose different TM strategies, highlight all the costs incurred, and discuss implementation of TM approaches using a TM manager.

Benefits and limitations. Thread motion enables fine-grained tracking of program behavior that conventional DVFS cannot. As a result, TM can increase the system throughput for different power envelopes. Equally important to the performance benefits, is the ability of TM to provide an effective intermediate voltage with only two VF settings, thus cutting down system costs. As mentioned in Section 1, this paper evaluates TM on a multi-core design setting featuring simple cores, and makes a couple of assumptions. First, TM relies on rapid movement of applications between cores and, hence, is constrained to systems featuring simple homogeneous cores with relatively small amounts of architected state. Second, TM targets power-constrained multi-core systems, which do not have all cores operating at the peak power, featuring cores that differ in their power-performance capabilities through voltage/frequency settings. Implementation of TM on other multi-core alternatives may require additional hardware support and is beyond the scope of this paper.

Architecture. With these assumptions in place, we study TM on a 16-core CMP system featuring simple, in-order, x86 cores sharing 2MB L2 cache. The architecture, shown in Figure 3, consists of cores grouped to form clusters—four cores form a cluster and all clusters share the last-level cache. L1 I&D caches are shared by all cores in each cluster. As mentioned in Section 1, this architectural configuration resembles a production system, Sun's Rock [27], which has small cores sharing first level caches and a 2MB L2 cache. The shared L1 caches enable low-overhead movement of applications within a cluster, but as we will see later, even inter-cluster movement can provide benefits. We elaborate on the architectural parameters along with our simulation framework in Section 4, but use this framework to overview the costs associated with TM and the TM algorithms that can be used.

3.1 TM Framework

In order to extend DVFS benefits to nanosecond time scales, we propose two approaches to implement TM that leverage application variability arising out of microarchitectural events: (1) Our first approach, called *time-driven TM*, is a straight-forward extension of traditional DVFS techniques that re-evaluate program be-

havior at fixed intervals (*TM-intervals*), and re-assigns core VFs to fit the power budget. Time-driven TM examines program behavior at nanosecond granularities and it moves applications to cores with VF settings that best match their needs. (2) Our second approach, called *miss-driven TM*, uses a cache miss as a trigger to evaluate thread motion possibilities, and manages the power overhead of the stalled application by balancing it with the performance needs of active applications. Both schemes exploit fine-grained variations arising from microarchitectural events.

Intra- and inter-cluster TM. The architecture configuration shown in Figure 3 consists of cores grouped to form clusters. Applications can move to a core with a different power level within a cluster (intra-cluster transfer) or migrate to a core in a different cluster (inter-cluster transfer). Inter-cluster transfers pose additional cost and implementation challenges (Section 3.2), but also provide increased benefits by exposing a richer set of available core VF choices (Section 5.2). The TM algorithm, discussed in Section 3.1.1, must carefully balance the benefits of inter-cluster transfers with the anticipated costs.

TM Manager. To implement TM approaches and evaluate their benefits, we model a TM manager. This manager, which runs the TM algorithm, is similar to previously proposed global manager approaches for power management [11]. We envision the TM manager running in a separate embedded microcontroller. Existing production systems already execute firmware code in a dedicated embedded microcontroller to monitor and maximize performance using VF control, subject to runtime power and temperature constraints [21]. The firmware has interface registers to the rest of the system that provides control and status information.

3.1.1 Algorithm

Several previously proposed DVFS algorithms [9, 14] are too computationally complex for the time-scales TM targets. Instead, we implement our TM algorithm as a simple, cost-benefit analysis continuously performed in a loop. It uses the application's predicted IPC as the *expected* IPC for the next TM-interval (Section 3.2.1). We refer to the window of IPC prediction as the prediction period. TM incurs cost overheads that are factored into the expected IPC to arrive at the *effective* IPC of the application, if it were an intra- or inter-cluster transfer participant.

$$effective\ IPC = \frac{instructions\ committed\ in\ prediction\ period}{cycles\ in\ prediction\ period + predicted\ TM\ cost}$$

The TM algorithm maintains a separate table of intra- and inter-cluster *effective* IPCs for each application. For the miss-driven TM approach, upon a miss in an application running on a high VF core, the application with the highest effective IPC that is not already at a high-VF core is chosen as the swapping parter, and TM is initiated. For the time-driven approach, at *TM intervals*, changes in core and application mappings, if any, are applied.

Locally-optimal assignments. The TM algorithm prioritizes moving of applications between cores within a cluster. However, if intra-cluster motion is not possible (for example, all other applications in the cluster are well-suited to their core power levels), inter-cluster migration is performed. In addition to this locally-optimal move methodology, we considered globally-optimal TM schedules, but found little additional benefit, and do not consider them further due to cost and scalability concerns resulting from the low TM time constants.

Policy. Our TM algorithm's assignment policy is geared towards minimizing performance loss; that is, it maximizes the number of committed instructions in the system by assigning an application in a compute-intensive phase to a high VF core. Our chosen policy facilitates straightforward comparisons to static configurations (in which the entire trace is simulated using the pre-determined optimal VF to maximize throughput (Section 4.3)). It is important to note that other policies, such as those that guarantee fairness or that tackle thermal problems, dictate TM for reasons other than increased system throughput, and can also fit into the TM framework.

3.2 Costs of Thread Motion

The TM algorithm requires an estimate for both the predicted IPC on the new core and the costs associated with the TM event. Despite using simple cores that share first-level caches, thread motion incurs latency costs to copy architected state among cores and overheads for inter-cluster transfers.

3.2.1 Prediction

The first step for our TM algorithm is to predict the best VF level for each running application for the next TM interval. This prediction is similar to that used in DVFS control algorithms [12]. Prediction techniques rely on past performance history of the application to predict the metric(s) of interest [6, 12]. We investigated a variety of predictors and chose a simple last-value predictor. We also experimentally determined that a 1000 cycle prediction period provides the best accuracy with an average error of 12.5%.

Inter-cluster TM incurs cache penalties, discussed in the next section. In order to estimate these costs, an estimate of the amount of additional memory traffic that will be incurred because of application migration must be calculated. The algorithm uses the count of memory instructions to evaluate inter-cluster TM. As with the IPC metric prediction, we obtained the best results for the last-value prediction mechanism. The average error was 4% across all workloads for a sample period of 1000 cycles.

3.2.2 Inter-Cluster Cache Penalty

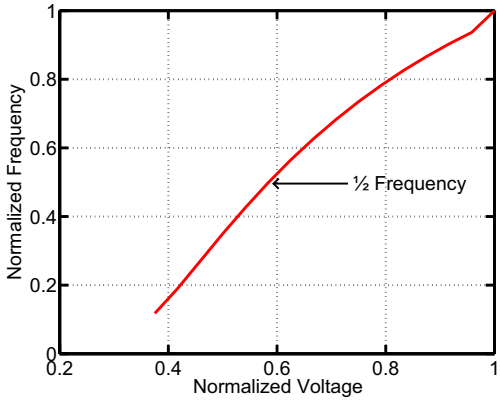
Inter-cluster TM results in loss of L1 cache data for the applications that move. We leverage the existing MESI coherence protocol to handle on-demand transfer of data using the shared-L2 cache for applications that move (similar to [10]). The TM algorithms use memory instruction prediction to evaluate the cost of moving an application to a new cluster. For the in-order cores studied, there is a simple relationship between the number of memory references in the application and performance losses due to cache misses. Inter-cluster transfer includes loss of first-level TLB state as well. However, the shared L2 TLB (behind the first-level TLB) is accessed simultaneously when the first-level cache miss is serviced, and is not in the critical path. Inter-cluster transfer results in additional cache traffic to the last-level cache and has power implications; we study this in detail in Section 5.5.

3.2.3 Register File Transfer Latency

The architected state of the in-order, x86 cores we model contain basic program execution registers—8 32-bit integer registers, 6 16-bit segment registers, EFLAGS register and Instruction Pointer (IP) [4]—and a separate x87 FPU register file with 8 FP data registers, and 6 control/status/IP registers (shown in Figure 3). Special debugging and error reporting registers may be disabled during TM. In addition to the architected register files, x86-microarchitectures often require a large internal register state. By transferring data on macro-instruction boundaries, we avoid having to copy the internal registers. We reserve dedicated cache lines in the shared caches to hold a copy of the core RF state. The first-level caches hold the RF states for all cores that share the cluster and the last-level caches hold the RF states for all cores. This introduces a small overhead; storing 2 KBits for each core RF state results in

Parameters	Values
Separate L1 I&D Caches	16KB and 2-way per core, 32-byte line size, LRU, shared by cores in cluster
L2 Unified I&D Cache	2MB, 16-way, 64-byte line size, pseudo-LRU shared by clusters
L1/L2/Mem latency	2/12/150
L1-TLB	8-entry per core, shared by cores in cluster
Functional Units	Single Issue: 1 Integer ALU, 1 FP ALU
Branch Prediction	Static
Coherence	MESI

(a) Simulation configuration



(b) Voltage-frequency relationship

Figure 4: Table showing simulation configuration parameters. The figure plots the voltage-frequency relationship for the $45nm$ CMOS technology node from HSPICE simulations of an 11-stage FO4 ring oscillator.

1.5% overhead to the first-level caches, and 0.2% overhead to the last-level caches. The cluster-cache interconnect and the shared-L2 caches are used for intra- and inter-cluster motion respectively. State-transfer between RF and caches is performed with core microcode that issues *store* instructions to the cache. Cores participating in TM do a *load* of the register state from the caches on demand after re-starting.

We study three mechanisms that populate RF state to the first-level shared cache. The *lazy-write* mechanism initiates RF state copy of a core after it becomes an intra- or inter-cluster TM participant. This scheme can incur a high RF transfer penalty for every TM event, leading to performance losses of up to 6%. To reduce RF transfer costs, we explore a *shadow-write* mechanism that opportunistically uses cache-idle cycles on single-register access instructions, to write dirty registers to the cache. While this reduces the performance overhead to around 1%, it incurs an high RF and L1 cache power overhead. We consider a more conservative *eager-write* mechanism that initiates transfer of dirty registers following a miss in the first-level cache. The amount of cycles available for the eager-write depends on how long it takes for the L2 cache access to resolve. L2 caches are often phased (serializing tag- and data-access to conserve power), and the miss information would be available sooner than the total latency (which may trigger TM), reducing the number of cycles available for eager-writing. Our eager-write study assumes that the L2 latency of 12 cycles is split into 7-cycle tag-array read and a 5-cycle data-array access (obtained from CACTI simulation). Eager write incurs a 2-3% performance loss, with a reasonable power overhead, and all subsequent results use this scheme. A detailed breakdown of power costs are discussed in Section 5.5.

Inter-cluster state transfers present other latency challenges that the TM algorithm must consider—I-cache data is lost during inter-cluster transfer. Hence, integer and FP instruction pointers always transfer first and initiate an instruction stream fetch in the remote core. In summary, intra- and inter-cluster transfers incur different costs, and the TM algorithm accounts for these costs in evaluating transfer feasibility, and balances costs with anticipated benefits.

4. SYSTEM FRAMEWORK

As discussed in Section 3, we study thread motion on an architecture that resembles Sun's Rock [27] (Figure 3). Four cores, grouped to form a cluster, share the first level I- and D-caches. We model 16KB, 2-way per core of shared L1 caches. It uses LRU replacement and can sustain one load per cycle per core. A similarly configured I-cache provides 8 bytes of instruction stream to each core per cycle. The 2MB, 16-way, unified L2 cache is shared by all four clusters. It uses pseudo-LRU replacement and supports 8 independent requests simultaneously. Table 4(a) lists the architectural parameters used to configure our simulator.

4.1 Core Power Levels

Thread motion relies on having multiple cores that vary in power-performance capabilities. Different VF domains enable heterogeneous power-performance cores, and we model system configurations ranging from 2 to 4 separate VF settings. We use the term VF setting to refer to the combination of frequency and the corresponding voltage needed to sustain the frequency. We model the voltage-frequency relationship for the $45nm$ CMOS technology node based on HSPICE simulations of an 11-stage FO4 ring oscillator (Figure 4(b)) and estimate dynamic power with $freq * volt^2$. This simplistic power-performance relationship is sufficient for this study, because our primary comparison is to coarse-grained DVFS also modeled with the same relationship.

4.1.1 VF Settings

Each VF setting requires a separate regulator to provide the voltage to the core (with a separate power-delivery subsystem), and a per-core PLL to provide the frequency. For example, a four-VF configuration would require four voltage regulators and per-core PLLs. Generating and delivering different voltages can incur potentially high costs. Hence, in CMPs, the goal is to increase performance with fewer VF domains. Given the long time constants associated with off-chip voltage regulators, this study assumes all voltages are fixed except for an ideal scenario where both frequency and voltage change instantaneously.

For this study, we model 2- to 4-VF configurations. All VF configurations are normalized to the maximum operating frequency and voltage (Figure 4(b)); we refer to this baseline case as having a VF of 1. Two VF-domain configurations use a low VF of 0.5 and a high VF of 1. This provides a maximum 50% operating range (similar to a commercial design [17]), and can best accommodate performance-centric and power-centric operating modes of applications using max and min voltages respectively. The three VF-domain configurations use frequencies of 0.5, 0.67 and 1, and the corresponding voltages needed to sustain the frequencies. The four VF-domain configurations use 0.5, 0.67, 0.83 and 1 VF.

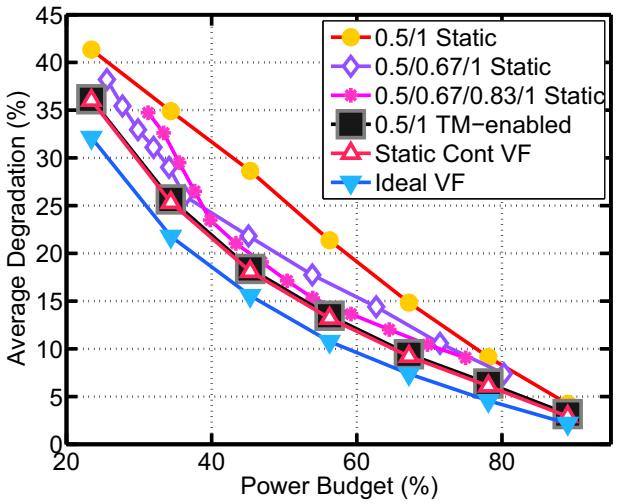


Figure 5: Performance degradation of static and TM-enabled in several VF settings. Ideal VF refers to non-quantized/continuous, per-core VF obtained instantaneously at fine resolutions with no cost. Static Cont VF refers to such VF settings obtained at coarser resolutions.

Performance and reliability of on-chip memories in aggressively-scaled technologies are highly sensitive to the operating voltage. While several circuit and architecture-level solutions have been proposed to enable low-voltage operation [28], we simplify our analysis by assuming the timing critical, shared L1 and L2 caches all reside within the highest 1 VF domain. Moreover, we later show that our TM implementation only uses two VF domains (Section 5.1) with high and low frequencies of 1 and 1/2, respectively. This simple, integer clock ratio obviates complex data-transfer circuitry and provides sufficient timing slack to accommodate level shifter delays for requests to the cache.

4.1.2 Power Budgets

The potential benefits of thread motion are studied in the context of different system-level power budget targets (further explained in Section 4.3). Different power budgets can be achieved by changing the number of cores that connect to different voltage domains. To simplify the analysis, we constrain the possible permutations of these connections for different power budgets. Our choice of VF configurations for a particular budget restricts the core power levels to build up monotonically from low to high, until all cores are at the maximum VF of 1. We distribute VF settings with maximum uniformity across all clusters. For example, for a power budget with half of the cores at high VF and half at low VF, each cluster would have same number of high/low VF cores.

4.2 Experimental Setup

Thorough exploration of thread motion in CMP architectures requires an experimental infrastructure that models core timing and core register states, intra-cluster cache and last-level cache sharing, ability to run multiple VF domains, runtime switching of core VFs, cache contention effects, and power overheads. The starting point for our system model is CMP$im [13], a complex pintool that implements a cache simulator based on the *Pin* [19] software binary instrumentation infrastructure. Pin serves as a functional model providing memory references to CMP$im, and the CMP$im cache simulator processes the incoming instruction and data trace. CMP$im can model multiple levels of private and shared caches and supports cache parameters such as cache sizes, access latencies, sharing bandwidth and associativity. CMP$im models the MESI coherence protocol as well as various cache replacement, allocation, and replacement policies.

Our simulation framework extends CMP$im to perform cache simulations of multi-programmed (MP) workloads. We also augment CMP$im with timing models of simple, in-order cores configured as shown in Table 4(a). Cores have an issue width of 1 and block on a cache miss. Cores are clock-gated when idle, resulting in a 25% reduction of dynamic power consumption. Our simulation environment based on CMP$im fully supports cycle-accurate modeling of cores running on different VFs, modeling core register-file state, various cluster configurations, dynamic core VF transitions, and effects on the shared cache and memory traffic; that is, bus conflicts and cache contention effects are modeled precisely for all evaluated configurations and algorithms.

Simulation of MP workloads is accomplished in two steps. First, CMP$im is run in trace-generation mode to gather traces of representative regions for each SPEC CPU 2006 binary. We gather traces to guarantee run repeatability and ensure fair comparison across all evaluated configurations. We use pinpoint weights [22] obtained using a different infrastructure to provide the representative regions of applications. The generated traces contain 100 million highest-weight pinpoint instructions; in addition, 300 million instructions preceding the pinpoint region is included to warm up the caches. Second, CMP$im is invoked in simulation mode on the collection of traces that compose the MP experiment. Each trace initializes a new 5-stage pipelined, in-order core.

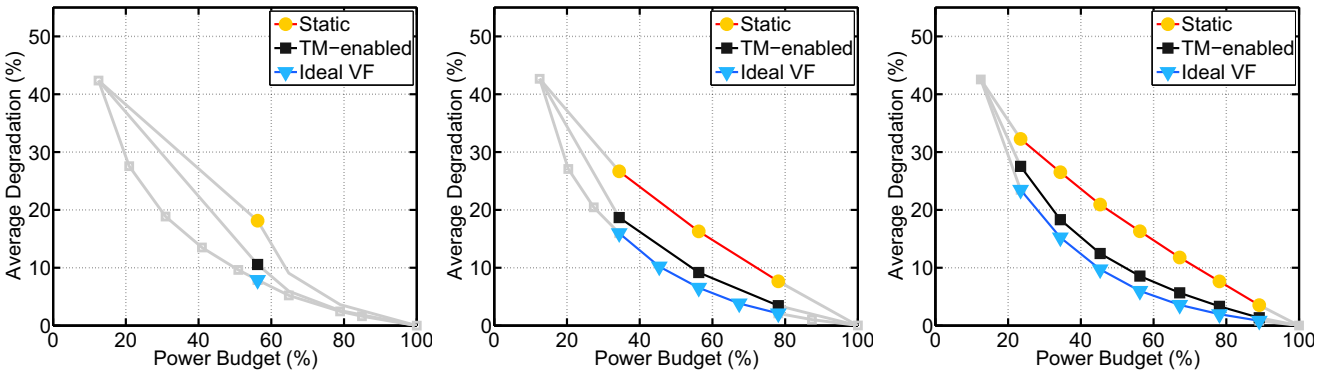
Simulation proceeds by fetching an instruction from the trace, feeding it to the core timing model, and playing data instructions through the simulated cache configuration. This is repeated for the total number of cores every cycle. MP simulation ends when all the cores have finished executing the trace. The cores that finish early loop back on their trace files and re-execute their pinpoint region.

4.3 Measuring Performance

In order to understand the benefits of thread motion, we will compare the performance of machines with TM to conventional machines with OS-driven DVFS and multiple voltage domain scenarios, but no motion. Given its relatively long timescales, our analysis models OS-driven DVFS to effectively operate with *static* VF settings. Simulation of different static configurations occurs in two steps. First, the entire trace for each application is run once to compute the overall IPC. Second, this information is used to schedule each application to its best-matched core; that is, the highest IPC unscheduled application is scheduled to run on the highest remaining core VF. Even though this two-step process to predetermine optimal VF is conservative, it provides a best-case baseline for static approaches. We model *TM-enabled* machines with cost factors related to swap decisions for the two TM approaches (time- and miss-driven TM) detailed in Section 3.1.

On multi-core systems, high non-recurring engineering cost of design motivates using a single design for as many markets as possible, but parameterized by power budgets that lead to different performance classes (laptop vs. server vs. desktop). The goal, then, is to improve performance across a range of power budgets set by a particular VF configuration (i.e. cluster's combination of cores with different heterogeneous VFs). Hence, we evaluate static and TM-enabled configurations across a range of power budgets by measuring relative performance loss. Performance loss at a particular power budget is compared to a best-performance design that operates at the highest VF setting at all times (similar to [11]).

The last level cache (L2) resides on a separate VF domain and accesses to it complete in fixed time [21]. We measure each appli-



(a) 2 Cores Per Cluster **(b)** 4 Cores Per Cluster **(c)** 8 Cores Per Cluster

Figure 6: Effect of available VF choices on thread motion performance, using high-variability workloads.

cation's performance loss (in all schemes) as the ratio of the extra time taken to run the application on a particular power budget to the time taken to run the same number of application's instructions with a 100% power budget (all cores at maximum VF). Running the same number of instructions ensures that the application phases do not bias the results. Net performance loss at each power budget, presented as average degradation, is the sum of losses of individual applications averaged over the total number of simulated cores.

5. RESULTS AND ANALYSIS

Thread motion enables fine-grained temporal tracking of program behavior and, as a result, can offer benefits that are sensitive to the type of workloads being evaluated. Furthermore, TM relies on the differences in power-performance capabilities of cores and, hence, can also be expected to be sensitive to both the number and choices of available VF levels. In order to narrow the study, we first evaluate the effects of two parameters without including TM costs—the number of VF levels and the choice of available VFs. We then analyze the performance in light of TM cost, compare time- and miss-driven TM policies, study a variety of workload configurations, and provide a breakdown of TM power overheads.

5.1 Effects of VF Quantization

Multiple voltage domains incur significant costs in terms of off-chip regulators and power-delivery hardware. A very important benefit of TM is that it can achieve nearly all of its benefits with only two VF domains. To understand this effect clearly, we simulate 2 clusters of the architecture in Figure 3 using 2-, 3-, and 4-VF configurations running a combination of 8 high-variability workloads shown in Figure 2(b). For this study, we also model an *ideal* VF machine in which any application can switch to any VF it wants instantaneously with arbitrarily *fine* resolution while satisfying the different power budgets. In other words, an *ideal* machine could obtain any VF (without quantization effects) best suited for the program phase. Static continuous VF is an ideal VF obtained at coarser granularities.

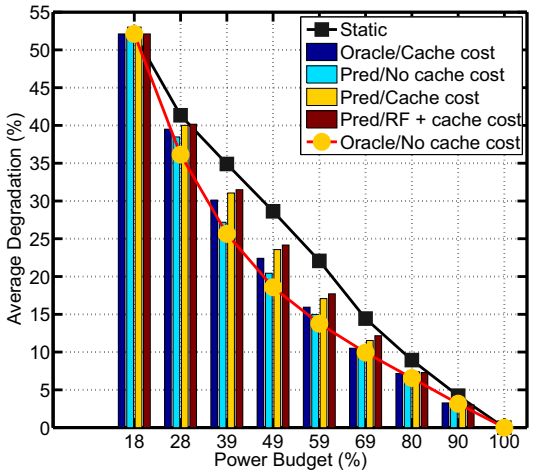
Figure 5 compares thread motion with two VF levels to several other configurations. As discussed earlier, we evaluate the configurations by measuring performance loss over a range of power budgets. For the highest power budget of 100%, all cores are at the maximum VF and there is no performance loss. For the lowest power budget of 12.5%, all cores are at the minimum VF, and all configurations converge to exhibit identical performance degradations. These two extreme cases are *not* shown in this graph. The topmost line in the figure corresponds to a static oracle configuration using two VF levels. We see that as the power budget decreases, average degradation increases as more cores operate at lower VF settings. The next two lower lines show that having additional VF domains benefit the static configuration by reducing performance degradation for a range of power budgets. The static continuous VF, where each core operates at a continuous (non-quantized) VF setting with oracle knowledge of the IPC of running applications, further reduces degradation. The plot also presents results for time-driven TM (with a TM-interval of 200 cycles) with only 2 VF choices, and we find that it performs better than static assignments using 3 and 4 VF choices and is nearly equivalent to continuous static assignment. Although omitted for clarity, simulations of thread motion with 3 and 4 VF choices do not offer performance benefits over having only 2 VF choices. Thus, thread motion can achieve the effect of having multiple VF domains, but with just two voltage levels, saving on costly off-chip regulators. Note that static continuous VF is an idealized mechanism that allows arbitrary selection of voltages and frequencies, but requires a regulator and power grid for each core.

The ability to move between cores at fine granularity means that an application could run on different quantized VF levels for arbitrary amounts of time; hence, during the lifetime of the application, it achieves an effective VF that is different from the fixed VF choices. In this sense, thread motion achieves benefits similar to using hardware dithering [5]. Finally, Figure 5 also compares time-driven TM performance against an ideal dynamic VF—non-quantized, per-core VF obtained instantaneously at 200-cycle TM-intervals—as the best-case performance scenario for fine-grained DVFS. This ideal VF is impossible to achieve and is only shown as a best-case bound.

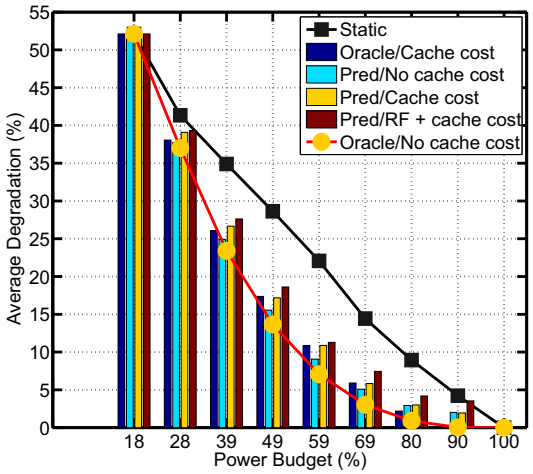
5.2 Effects of available VF choices

The second parameter we study is the sensitivity of thread motion to available VF choices. To construct various choices of core VFs, we model 2-core, 4-core and 8-core per cluster configuration, and disallow inter-cluster thread motion. However, to rule out application behaviors biasing results and to correctly model the cache contention effects, we always run 8 applications and simulate multiple clusters with a total of 8 cores. This necessitates averaging results of multiple clusters. For example, runs from four clusters are averaged to present the 2-core per cluster data. We continue to use our collection of 8 high-variability workloads (shown in Figure 2(b)) for this study. For these evaluations, we dropped the 3- and 4-VF setting choices since they do not offer performance advantages. In addition, we do not show the comparison with static continuous, because it is similar to thread motion.

Figure 6(a) presents the average performance degradation as-



(a) Costs of time-driven TM



(b) Costs of miss-driven TM

Figure 7: Costs for time-driven and miss-driven TM using 8 high-variability workloads.

suming 2 cores per cluster and 2 VF levels. This small number of cores constrains the evaluation to one core at high-VF and one core at low-VF setting. This setting allows the applications to achieve effective frequencies in the range between low and high by moving between the two available choices, which translates to 7.6% less performance degradation compared to static. The left and right most points in the figure have all low and all high cores, respectively. For these power budgets, since all cores are at the same VF, TM-enabled performance is identical to that of the static configuration (these points are grayed out).

For the second processor configuration (Figure 6(b)), we observe that TM performs better for a range of power budgets. Having more cores provides more choices to track application variability and improves performance. We see further improvements in the final processor configuration with 8 cores per cluster, where more configurations cover a wider power budget range, and performance differences per application improves by up to 10%. Static assignments, even when scheduled based on the best possible core using an oracle knowledge of application's IPC, fail to exploit the intra-application variability for all cases. The results of this experiment has an important implication—exposing more VF choices (via the number of available cores) increases TM performance. This motivates us to include inter-cluster TM evaluations in our algorithms despite additional costs.

5.3 Time-Driven vs. Miss-Driven TM

Though time- and miss-driven TM are both fine-grained approaches, they operate on different triggers and incur different costs. Figure 7(a) presents results for a smaller configuration (two-cluster simulation[1]) of the architecture shown in Figure 3, assuming the breakdown of all costs discussed in Section 3.2 for time-driven TM. The two lines plot the static configuration and the TM-enabled configuration with no cost. Each TM cost is shown as a separate bar to isolate its effects. We follow the labeling of (prediction scheme, with or without cache costs) to identify costs. For example, pred/no cost bars mean that last-value prediction was used but the cache costs were ignored. Oracle prediction isolates the effects of loss incurred as a result of lost cache state in inter-cluster transfers. Prediction with no cost isolates the effects of last-value IPC and instruction prediction (over the past sample period of 1000 cycles) on TM benefits. Prediction with cache cost isolates the effect of both costs and assumes instantaneous state transfer. Finally, all costs are factored in the last bar, including the penalty due to state transfer cycles. For the power budgets intervening the max and the min, more TM choices exist and we see an increase in the costs. Even accounting for all costs, for the power budget of 49%, time-driven TM outperforms the static oracle configuration by 4.5%.

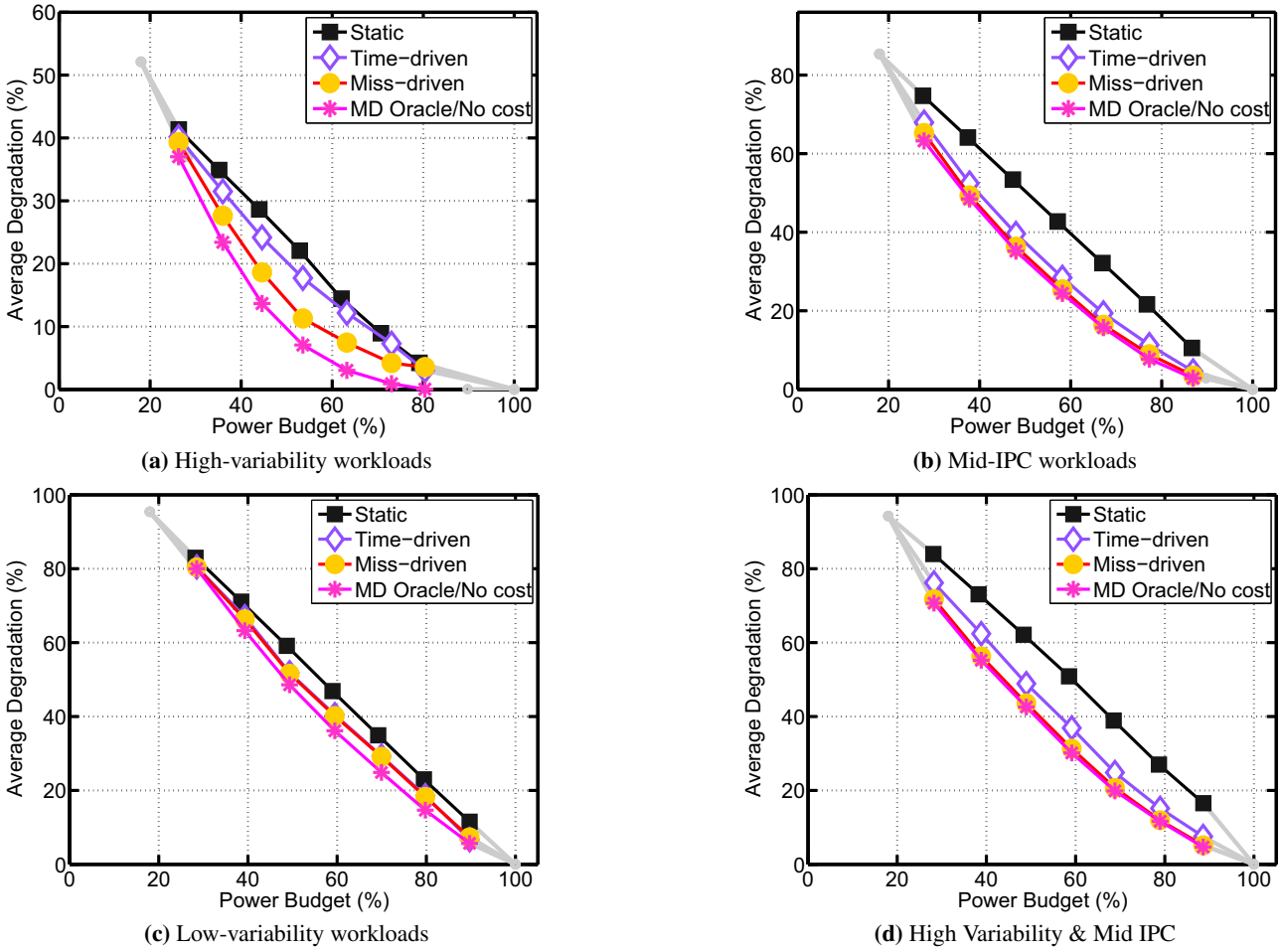
We now look at the costs for the miss-driven approach using the same set of workloads and machine configuration. Figure 7(b) breaks down all related costs. Even though miss-driven approach uses the last-level cache miss as a TM trigger, it still uses instruction-based prediction to evaluate inter-cluster partners (see Section 3.2). So we observe a similar set of cost breakdowns to the time-driven approach. As the figure shows, miss-driven TM outperforms time-driven TM for this set of workloads and, at the power budget of 49%, delivers performance benefits of up to 10% over the static configuration. However, for a higher power budget of 90%, the large number of applications in high-VF cores seek the single low-VF core and the transfer costs of TM make it comparable to a static system.

5.4 Impact of Workloads

As previously discussed, TM benefits are sensitive to the behavior of the workload it tracks. Hence, we evaluate TM performance with respect to four different MP-workload configurations, which are outlined as (A) through (D) in Figure 2(b). Similar to previous experiments, these workloads are run on two 4-core clusters. Each workload result compares four configurations: the static oracle; time-driven TM, including all costs and a TM interval of 200 cycles; miss-driven TM, including all costs; and a miss-driven scheme with no costs and oracle assignments, to present an optimistic bound. All configurations employ clock-gating on core idle cycles.

TM benefits come from two primary factors: (1) the amount of variability in the workload, which translates to how often applications can be managed for better system throughput; and 2) the IPC of the applications in the workload, which translates to how much performance benefit an application can achieve once it moves to a high VF core. The four workload configurations that we consider span combinations of these cases.

[1]Modeling of 4-clusters (16-cores) demands simulation of billions of instructions, and was time-prohibitive for the sweep studies.



(a) High-variability workloads

(b) Mid-IPC workloads

(c) Low-variability workloads

(d) High Variability & Mid IPC

Figure 8: Average degradation for workload configurations (A) through (D) shown in Figure 2(b).

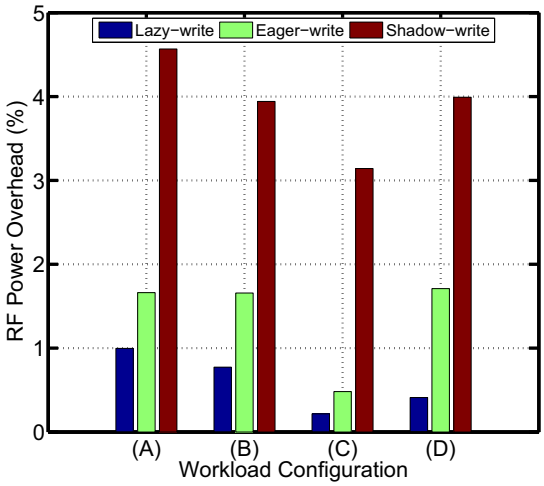
Figure 8(a) presents the results for the workload configuration (A) in Figure 2(b) comprising 8 high-variability workloads (also used for cost analysis). The high variability promotes a larger number of transfers (results not shown for brevity) and incurs higher costs. However, both approaches exploit the large amount of variability in this configuration, resulting in benefits over the static configuration. Miss-driven configuration performs best with up to 10% less performance degradation, inclusive of all costs, over the static counterpart. Since this set mostly contains low-IPC applications, benefits come from more transfers and better utilization of high VF cores.

Figure 8(b) presents the results for the workload configuration (B) in Figure 2(b) comprising mid-IPC workloads (IPC range from 0.5 to 0.8). This set of workloads exemplifies the effect of VF quantization on a CMP system with two fixed voltage domains of high and low. The higher IPC of each application (compared to workload set (A)) results in increased gains for applications that migrate to high VF cores; that is, each transfer to a high VF core results in a larger number of instructions being committed compared to workload (A). Consequently, miss- and time-driven TM perform similarly and are better than static assignments over a range of power budgets, reducing performance degradation by up to 18%.

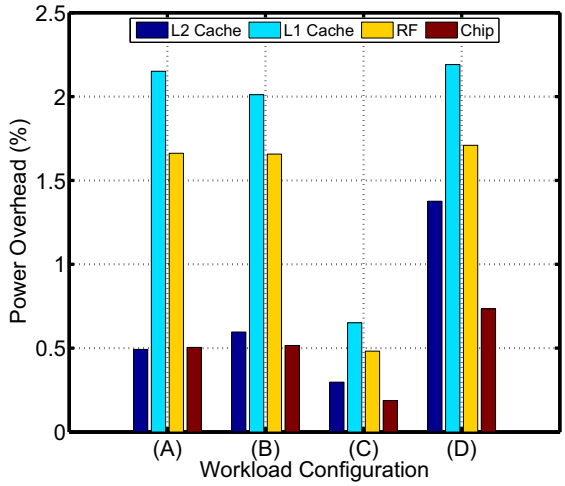
Figure 8(c) presents the result of running 4 copies each of two applications that have almost no variability and high IPC—*calculix* and *perl.splitmail*—shown in Figure 2(b) as configuration (C) on two, 4-core clusters. While this configuration appears to not benefit from TM, the high-IPC of each application translates into the highest number of committed instructions per transfer. In other words, during the rare memory misses that do occur, TM is very effective because it can advantageously run the unstalled application on the high VF core. However, the number of transfers is limited, and as can be seen from the figure, this group offers modest improvements over static (up to 7.2%) compared to the other configurations we have looked at.

Finally, Figure 8(d) presents the results of running 8 copies of a single application that falls under both the high variability and mid-IPC categories (configuration (D) in Figure 2(b)). This scenario combines both benefits of thread motion—a large number of transfers occur (due to the high variability) and each transfer is relatively fruitful (due to the mid-IPC workloads). This represents the best-case scenario for thread motion. TM achieves improvements with degradation reducing by 15.1% to 20.5% over a range of power budgets compared to static.

As we saw in Figure 2(a), variability is more pronounced when examined at smaller intervals. Even though we present time-driven TM results for 200-cycle TM intervals, we have also evaluated coarser granularities and found that TM results over tens of thousands intervals are comparable to static. At 200-cycle intervals, TM incurs much more register transfer and cache costs as a result of frequent thread movement and yet provides maximum benefit.



(a) RF-transfer mechanisms (miss-driven TM)



(b) Miss-Driven TM on workload configurations

Figure 9: Power overheads for all RF-transfer mechanisms and workload configurations shown in Figure 2(b).

5.5 Power Costs

Analysis of TM benefits is not complete without looking at the power overheads. TM primarily incurs power overheads as a result of: (1) increased accesses to the register files and first-level caches; and (2) additional traffic to the L2 as a result of loss of shared first-level cache state in inter-cluster transfers. This section analyzes these overheads.

Our power models are based on Rock's reported frequency and power data [27] [16]—frequency is 2.3GHz, chip power is 250W and core power is 10W. Our power models assume 10% of the core power to be dynamic RF power [20]. We compute the dynamic register power as a product of registers accessed per cycle, energy per access and the frequency. Figure 9(a) shows the worst-case RF power overhead for the different RF-transfer mechanisms described in Section 3.2.3. The best performing *shadow-write* mechanism also incurs high power costs. On the other hand, only writing registers from cores that participate in TM has the lowest power overhead, but results in high delay overheads. The *eager-write* mechanism provides the best delay-power tradeoffs.

The RF-write mechanisms aimed to speed-up intra-cluster motion resulting in additional accesses to L1 that increase L1 power consumption. Even inter-cluster transfers affect first-level cache—inter-cluster TM results in reduced number of hits and additional misses to the first-level cache. In typical first-level cache implementations, the tag and data arrays are accessed simultaneously, so the hit power is the same as the miss power. Hence, inter-cluster TM does not change the power consumption of the L1 caches. Our power model to calculate L1 data cache power overhead assumes L1 power to be 15% of the core power [20].

In contrast to L1 utilization, L2 incurs additional traffic that would increase L2 power. We assume a phased L2 implementation that trades off access latency for low power, by serializing access to the data array following a hit in the tag-array, which results in different hit and miss powers. Our power model assumes L2 power to be 25W, or 10% of the chip power, using data from the cache implementation in [7] (which estimates it to be 7% of chip power). To compute the L2 hit and miss powers, we break down the L2 power into clock, data-array access, tag-array access, and data-transfer power numbers using the data provided by Grodstein *et al.* [7], and utilize the hits per cycle data from simulation.

Chip power for the simulated configuration is a combination of total core power, L1 and L2 cache power, or the aggregate power of components we model. Figure 9(b) plots the worst-case caches, RF, and chip power overheads for all workload configurations for miss-driven TM compared to a static configuration. Overall, the power overhead is negligible, up to 0.75% for the entire chip across all workload configurations analyzed. As can be seen, the workload configuration (D) in Figure 2(b), which gains the most benefit from TM, also incurs the highest power overhead.

6. RELATED WORK

Thread motion, a methodology for fine-grained power management, is related to prior work on dynamic, active power management. Isci *et al.* [11] propose using a global power manager to re-evaluate core power levels every 500μs using run-time application behavior. They evaluate different policies for global CMP power management. Our policy to maximize the number of committed instructions by TM is based on their MaxBIPS policy. Herbert *et al.* [9] reinforce the benefits of DVFS, but observe that hardware, per-core VF islands are not cost effective. Li *et al.* [18] consider a similar L2 miss-driven policy for reducing supply voltage; this scheme is linked into a voltage-control mechanism for certain pieces of the processor core. Juang *et al.* [14] propose using coordinated VF settings instead of local, per-core VF to improve power-performance. In contrast, thread motion targets much smaller time scales to maximize performance across different power budget constraints.

Core hopping, a coarser time scale application of thread motion, has been previously proposed to combat thermal problems. Heo *et al.* [8] argue that the interval between core migrations can be smaller than typical OS context swap times for maximum benefit. However, their ping-pong times that are still on the order of microseconds is much longer than the nanosecond granularities that TM targets. Powell *et al.* [24] propose using thread migration to prevent local hot spots that can result in chip failure. Their thread migration frequency, proportional to thermal time constants, is orders of magnitude higher than TM frequency. Shayesteh *et al.* [26] propose using helper engines to reduce thread migration latencies on CMP systems. The larger structures in the helper engines are decoupled from the processor microcore to help cut down the cost of state transitions during state migration. Their mechanism is one

possible approach to implement our methodology on systems with non-shared caches or complex cores.

Finally, fine-grained power management techniques using voltage dithering [5] and using on-chip regulators to obtain fast, per-core DVFS [15] have been previously proposed. Local voltage dithering has been shown to handle workload variations at fine granularities, but applying voltage dithering on an entire chip results in delays on the order of microseconds. On-chip regulators are another promising approach to achieve fast, per-core power control. However, it is not clear how well their on-chip regulators intended for low-power systems can scale to higher-power system. We show that TM achieves results similar to these other works with lower overheads.

7. CONCLUSION

Our workload studies show that variability observed at finer granularities is masked for larger sample sizes. This suggests a power-management mechanism that can operate at fine time scales can offer higher power-performance efficiency. We present thread motion as a fine-grained, power-management technique for multi-core systems consisting of simple, homogeneous cores capable of operating with heterogeneous power-performance characteristics.

We show that moving threads between cores with only two VF domains can perform nearly as well as having continuous, per-core VF levels. Analysis of TM on the architectural configuration of a modern processor like Rock shows that it can be highly effective. Performance benefits corresponding to reductions in average degradation by up to 20% are observed when compared to a static, OS-driven DVFS scheme. TM provides applications the flexibility to adapt time-varying fluctuations in computing needs to high- and low-performance cores available. Thorough evaluations show that TM offers benefits despite all of the penalties associated with motion, and power overheads are negligibly small.

Acknowledgments

We are thankful to Aamer Jaleel for providing us the initial infrastructure, and to Carl Beckmann for his suggestions. We are also grateful to the anonymous reviewers for their comments and suggestions. This work is partially supported by National Science Foundation grants CCF-0048313 and CSR-0720566. 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] AMD Turion X2 Ultra Dual-Core Processor. http://multicore.amd.com/us-en/AMD-Multi-Core.aspx.

[2] Intel Turbo Boost Technology. http://www.intel.com/technology/turboboost/index.htm.

[3] Nehalem Microarchitecture. http://www.intel.com/technology/architecture-silicon/next-gen/.

[4] *Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture*, 2008.

[5] B. Calhoun and A. Chandrakasan. Ultra-Dynamic Voltage Scaling (UDVS) Using Sub-Threshold Operation and Local Voltage Dithering. *IEEE Journal of Solid-State Circuits*, Jan. 2006.

[6] E. Duesterwald, C. Cascaval, and S. Dwarkadas. Characterizing and Predicting Program Behavior and its Variability. In *Parallel Architectures and Compilation Techniques*, 2003.

[7] J. Grodstein et al. Power and CAD considerations for the 1.75MByte, 1.2GHz L2 cache on the alpha 21364 CPU. In *Great Lakes Symposium on VLSI*, 2002.

[8] S. Heo, K. Barr, and K. Asanovic. Reducing Power Density through Activity Migration. In *International Symposium on Low Power Electronics and Design*, 2003.

[9] S. Herbert and D. Marculescu. Analysis of Dynamic Voltage/Frequency Scaling in Chip-Multiprocessors. In *International Symposium on Low Power Electronics and Design*, 2007.

[10] E. Ipek, M. Kirman, N. Kirman, and J. Martinez. Core Fusion: Accommodating Software Diversity in Chip Multiprocessors. In *International Symposium on Computer Architecture*, 2007.

[11] C. Isci, A. Buyuktosunoglu, C. Cher, P. Bose, and M. Martonosi. An analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget. In *International Symposium on Microarchitecture*, 2006.

[12] C. Isci, A. Buyuktosunoglu, and M. Martonosi. Long-Term Workload Phases: Duration Predictions and Applications to DVFS. *IEEE Micro*, 2005.

[13] A. Jaleel, R. Cohn, and C. Luk. CMP$im: Using Pin to Characterize Memory Behavior of Emerging workloads on CMPs. In *Intel Design, Test and Technologies Conference (DTTC)*, 2006.

[14] P. Juang, Q. Wu, L. Peh, M. Martonosi, and D. Clark. Coordinated, Distributed, Formal Energy Management of Chip Multiprocessors. In *International Symposium on Low Power Electronics and Design*, 2005.

[15] W. Kim, M. Gupta, G.-Y. Wei, and D. Brooks. System level analysis of fast, per-core DVFS using on-chip switching regulators. In *Symposium on High-Performance Computer Architecture*, 2008.

[16] G. Konstadinidis et al. Implementation of a Third-Generation 16-Core 32-Thread Chip-Multithreading SPARC Processor. In *IEEE International Solid-State Circuits Conference*, 2008.

[17] D. Krueger, E. Francom, and J. Langsdorf. Circuit Design for Voltage Scaling and SER Immunity on a Quad-Core Itanium Processor. In *IEEE International Solid-State Circuits Conference*, 2008.

[18] H. Li, C. Cher, T. N. Vijaykumar, and K. Roy. Combined circuit and architectural level variable supply-voltage scaling for low power. *Transactions on Very Large Scale Integration Systems*, 2005.

[19] C.-K. Luk et al. Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. *SIGPLAN Conference on Programming Language Design and Implementation*, 2005.

[20] S. Manne, A. Klauser, and D. Grunwald. Pipeline Gating: Speculation Control for Energy Reduction. In *International Symposium on Computer Architecture*, 1998.

[21] R. McGowen et al. Power and Temperature Control on a 90-nm Itanium Family Processor. *IEEE Journal of Solid-State Circuits*, Jan. 2006.

[22] H. Patil et al. Pinpointing Representative Portions of Large Intel Itanium Programs with Dynamic Instrumentation. In *International Symposium on Microarchitecture*, 2004.

[23] D. Pham et al. The Design and Implementation of a First Generation CELL Processor. In *IEEE International Solid-State Circuits Conference*, 2005.

[24] M. D. Powell, M. Gomaa, and T. Vijaykumar. Heat-and-run: Leveraging SMT and CMP to Manage Power Density Through the Operating System. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2004.

[25] L. Seiler, D. Carmean, E. Sprangle, et al. Larrabee: A Many-Core x86 Architecture for Visual Computing. *ACM Transactions on Graphics*, 2008.

[26] A. Shayesteh, E. Kursun, T. Sherwood, S. Siar, and G. Reinman. Reducing the Latency and Area Cost of Core Swapping through Shared Helper Engines. In *IEEE International Conference on Computer Design*, 2005.

[27] M. Tremblay and S. Chaudhry. A Third-Generation 65nm 16-Core 32-Thread Plus 32-Scout-Thread CMT SPARC Processor. In *IEEE International Solid-State Circuits Conference*, 2008.

[28] C. Wilkerson et al. Trading Off Cache Capacity for Reliability to Enable Low Voltage Operation. In *International Symposium on Computer Architecture*, 2008.

[29] S. Williams et al. Optimization of Sparse Matrix-Vector Multiplication on Emerging Multicore Platforms. In *International Conference for High Performance Computing, Networking, Storage and Analysis*, 2007.