Addressing Thermal Nonuniformity in SMT Workloads

JONATHAN A. WINTER and DAVID H. ALBONESI
Cornell University

We explore DTM techniques within the context of uniform and nonuniform SMT workloads. While DVS is suitable for addressing workloads with uniformly high temperatures, for nonuniform workloads, performance loss occurs because of the slowdown of the cooler thread. To address this, we propose and evaluate DTM mechanisms that exploit the steering-based thread management mechanisms inherent in a clustered SMT architecture. We show that in contrast to DVS, which operates globally, our techniques are more effective at controlling temperature for nonuniform workloads. Furthermore, we devise a DTM technique that combines steering and DVS to achieve consistently good performance across all workloads.

Categories and Subject Descriptors: C.1.3 [**Processor Architectures**]: Other Architecture Styles—*Adaptable architectures*; C.1.4 [**Processor Architectures**]: Parallel Architectures

General Terms: Algorithms, Design, Experimentation, Performance

Additional Key Words and Phrases: Simultaneous multithreading, clustered microarchitectures, dynamic thermal management, dynamic voltage scaling, adaptive microarchitectures

ACM Reference Format:

Winter, J. A. and Albonesi, D. H. 2008. Addressing thermal nonuniformity in SMT workloads. ACM. Trans. Architec. Code Optim. 5, 1, Article 4 (May 2008), 28 pages. DOI = 10.1145/1369396.1369400 http://doi.acm.org/10.1145/1369396.1369400

1. INTRODUCTION

Dynamic thermal management (DTM) is a microarchitectural approach to maintaining acceptable on-die temperatures while reducing packaging and cooling costs [Huang et al. 2000; Brooks and Martonosi 2001]. The fundamental idea is to employ thermal sensors at the hotspots of the die to detect when a potential thermal violation may arise. This triggers a response by the microar-

This research was supported by the Center for Circuits and Systems Solutions funded by MARCO/DARPA.

Authors' address: Jonathan A. Winter and David H. Albonesi, Computer Systems Laboratory, Cornell University, Frank H.T. Rhodes Hall, Ithaca, New York, 14853; email: {winter, albonesi}@csl.cornell.edu.

Permission to make digital or hard copies part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from the Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2008 ACM 1544-3566/2008/05-ART4 $5.00 DOI 10.1145/1369396.1369400 http://doi.acm.org/10.1145/1369396.1369400

chitecture to ward off the potential violation by throttling the chip resources in some fashion to reduce the power and, over time, the temperature of the affected die area. The temperature trigger is set such that only a small subset of all applications may cause such a thermal emergency necessitating a throttling response. Thus, performance is only minimally affected, while packaging and cooling costs can be significantly reduced compared to a processor in which no detection and throttling mechanism is employed (where the packaging/cooling must be designed for the absolute worst-case condition).

DTM has been an active area of research, both for single and, more recently, multicore systems. While the vast majority of prior DTM research has examined single-threaded processors [Dhodapkar et al. 2000; Huang et al. 2000; Brooks and Martonosi 2001; Lim et al. 2002; Heo et al. 2003; Skadron et al. 2003; Skadron 2004; Chaparro et al. 2004a, 2004b, 2005], more recent efforts have focused on chip multiprocessors (CMPs) and simultaneous multithreading (SMT) processors. While a number of DTM techniques have been proposed for CMPs and SMT, dynamic voltage scaling (DVS) stands out as the most effective approach [Li et al. 2005; Donald and Martonosi 2006; Chaparro et al. 2007]. DVS reduces both the voltage and frequency of the processor enabling an almost cubic reduction in dynamic power, as well as attenuating the static power, making it difficult to beat.

While DVS proves to be most effective when cooling a uniformly hot, CPU-intensive workload, it is much less effective when there are differences in the thermal behavior of the simultaneously running threads. Specifically, in an SMT processor, DVS can penalize the performance of one thread in order to cool a hotspot caused by another. In workloads where one thread is very CPU-intensive and the other is not, or in mixed floating-point/integer benchmark workloads, engaging DVS on an SMT core can often cause the other nonoffending thread to slow down despite the fact that it is not the cause of the hotspot.

To illustrate this phenomenon, we examine an SMT workload consisting of two SPEC CPU2000 benchmarks: *vortex*, a CPU intensive integer application, and *equake*, a less intensive floating-point application. Figure 1 shows the baseline performance of the pair on the clustered SMT architecture we model (described in detail later) when no dynamic thermal management is used. In comparison, Figure 2 shows the performance when DVS is used to cool the processor. In Figures 1, 2, and 12 (see later), performance is measured as billions of instructions per second (BIPS) completed, averaged over 100 K cycle intervals. The periodic performance degradation in Figure 2 is because of intervals during which DVS is employed. Figure 3 illustrates the temperatures of the hottest components of the processor during the DVS simulation. Because of *vortex*'s intensive behavior, the hotspots are consistently the integer ALU and integer multiplier unit on its clusters. Dips in temperature in Figure 3 from engaging DVS, correspond to the performance degradation points in Figure 2. Clearly, *equake* is being penalized by DVS even though its portion of the processor is nowhere near the thermal danger limit of 87°C. This example illustrates how *thermal nonuniformity* among the simultaneously running threads can increase the performance degradation of DVS-based DTM by penalizing a cool thread that runs on the same core as the hot one. A better DTM policy would

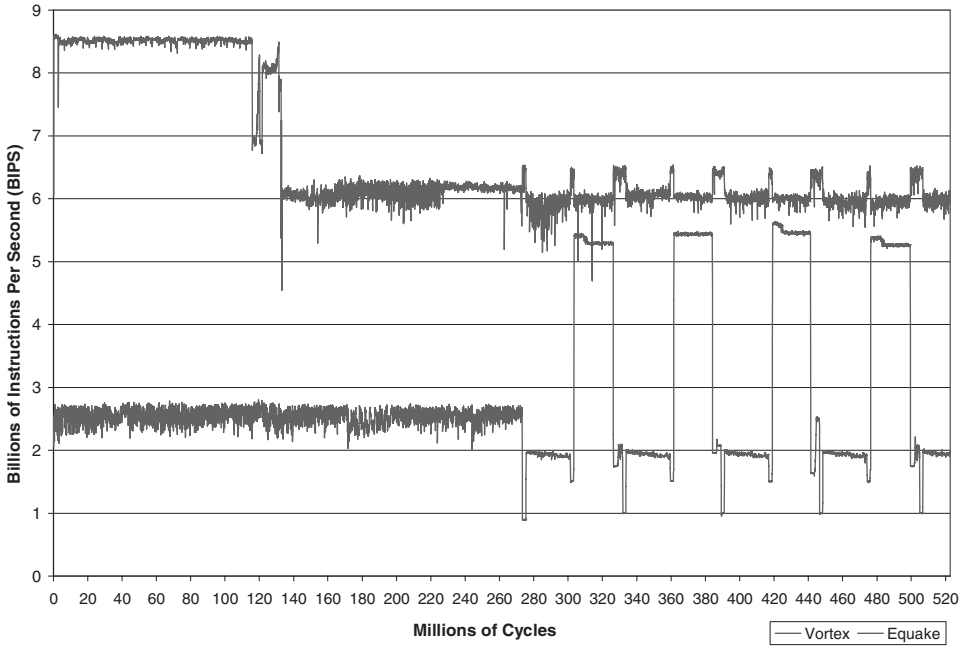

Fig. 1. Vortex/equake—performance with no DTM mechanism.

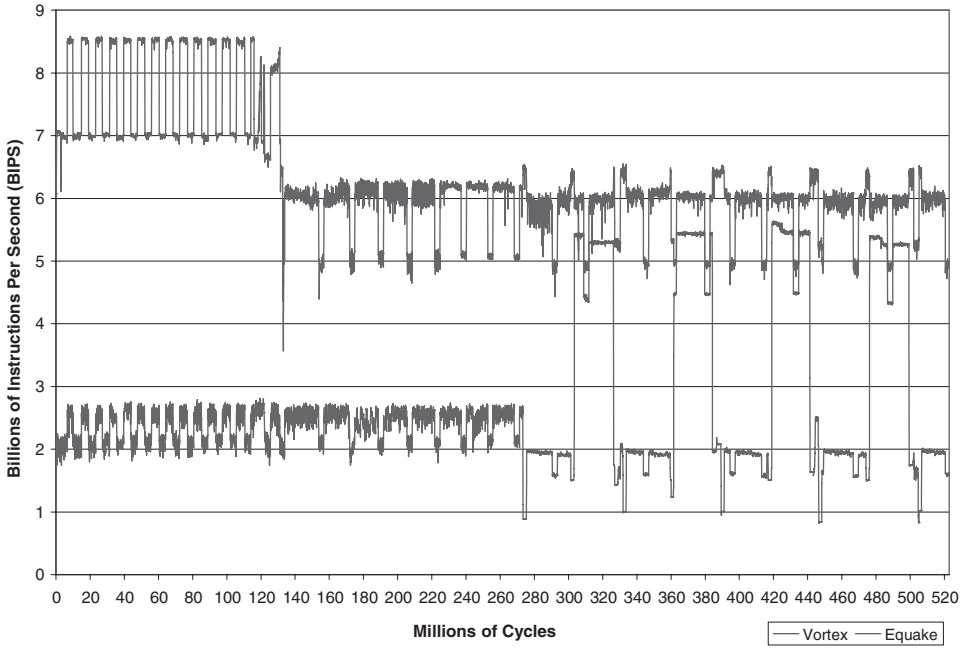

Fig. 2. Vortex/equake—performance with dynamic voltage scaling.

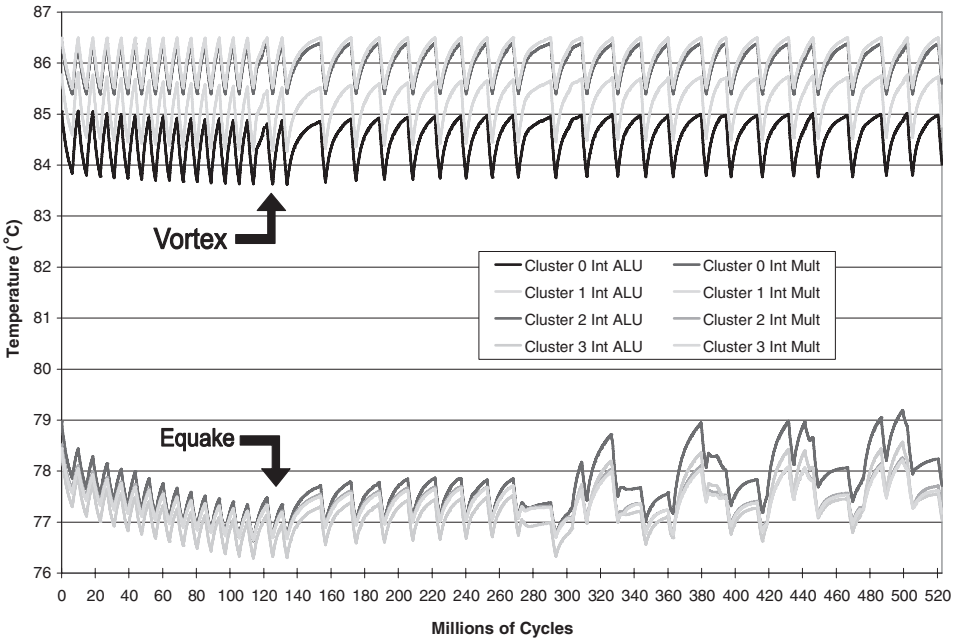

Fig. 3. Vortex/equake—temperature with dynamic voltage scaling.

intelligently manage individual threads and take advantage of the nonuniformity in temperature to eliminate chip hotspots.

In this paper, we develop effective alternatives to DVS for thermally nonuniform SMT workloads. We propose DTM techniques, which utilize the inherent steering mechanism in a clustered SMT microprocessor [Collins and Tullsen 2004; Latorre et al. 2004; El-Moursy et al. 2005] to exploit thermal nonuniformity among the threads to cool the chip more efficiently. For workloads composed of threads with nonuniform heating characteristics, our best DTM algorithm, counter-based steering, prevents all thermal violations with a worst case performance of 1% compared to 6.4% for DVS.

We also propose a "best-of-both-worlds" DTM policy, which utilizes the complementary properties of both steering and DVS-based algorithms. Namely, the steering-based mechanisms exploit the thermal differences of the running threads to reduce hotspots and avoid slowing all threads by globally scaling down the frequency and voltage, whereas DVS is effective in cases where a set of "hot threads" are uniformly heating the back ends (leaving little opportunity for the steering-based mechanisms to exploit temperature differences among the clusters). Moreover, adding steering-based DTM to DVS requires only minor changes to the baseline clustered SMT organization.

Our paper makes a number of novel contributions to the field of dynamic thermal management. This is the first work to explore DTM policies for clustered SMT microarchitectures, a unique design that combines the benefits of both multithreaded and multicore architectures. We discuss an overlooked drawback of dynamic voltage scaling, which is that engaging DVS on an SMT core

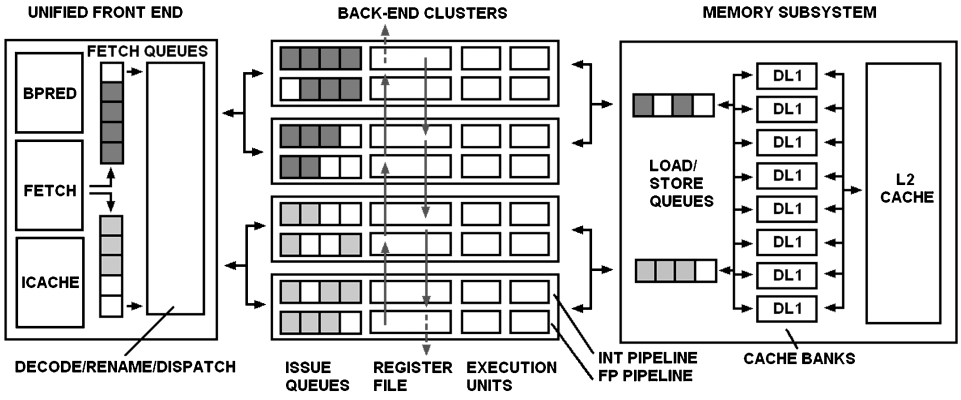

Fig. 4. A clustered simultaneous multithreaded microarchitecture.

will unfairly penalize threads that are not causing the thermal emergency. We show that this is a particularly large problem for nonuniform workloads and propose a novel thermal management technique to address this deficiency. Our algorithm exploits the spatial nonuniformity in temperature *within* a processor core caused by differences in thread pipeline usage to provide very low-cost DTM. Finally, we devise a new kind of thermal management policy that combines two distinct techniques—DVS and counter-based steering, each specialized for a particular class of benchmarks—to provide effective DTM across the full range of multithreaded workloads.

2. THE CLUSTERED SMT MICROARCHITECTURE

The clustered SMT processor, shown in Figure 4, is similar to that explored by El-Moursy et al. [2005]. The execution core, consisting of the issue queues, register files, and the functional units, is divided into multiple clusters with communication paths between the unified front end and the back ends, the back ends and the L1 data cache, and among the back ends for passing operand values. A traditional SMT front end is used to fetch, decode, and rename instructions from multiple threads, and a *steering mechanism* is employed to assign these instructions to back ends. While sharing of back ends among multiple threads is possible, prior research [Raasch and Reinhardt 2003; Latorre et al. 2004; El-Moursy et al. 2005] has shown that the best performance and power characteristics are obtained by largely isolating the threads from each other by assigning them to separate cluster groups. In order to reduce the performance cost of intercluster communication of operand values, instructions from the same thread are usually assigned to a contiguous group of adjacent clusters.

In our architecture, we found that the back-end clusters were the hottest part of the die and focused on alleviating hotspots in this section of the processor. This result is supported by previous research showing that register files and execution units are typically the biggest thermal concern [Heo et al. 2003; Skadron et al. 2003; Donald and Martonosi 2004, 2005, 2006; Li et al. 2005]. However, in other designs the front end could also be a source of thermal emergencies. Chaparro et al. [2005] explore partitioning the front end to control temperature

and the architectural enhancements they propose can be combined with our techniques to provide thermal control across the processor.

Clustered SMT processor designs may appear to go against the current trend toward chip multiprocessors containing many simple, perhaps single-threaded, cores. However, sequential code will not disappear altogether. Some workloads will contain high levels of instruction-level parallelism that is most effectively processed by wide-issue cores, some applications may be extremely difficult to parallelize, and there will always be some programs that software developers have not yet parallelized. Clustered SMT processors provide the flexibility to address these sequential workloads—by providing a low complexity, wide-issue engine when needed—as well as future highly parallel workloads through multithreading. The partitioning of back-end resources greatly reduces the complexity of the design, as well as the power and temperature. In addition, there is industry precedence for designing wide cores with back-end clusters, such as the Alpha 21264/21364, IBM Power4, and the SMT IBM Power5. Clustered SMT cores could be implemented in CMPs, along with narrower cores in an asymmetric configuration, providing a design that is performance- and power-efficient across a large variety of applications.

Moreover, a clustered SMT microarchitecture provides a natural platform for addressing nonuniformity for several reasons. First, the clustering of the hottest components on the die, namely the functional units, register files, and issue queues, into multiple back-end clusters, permits threads, and even different instructions from the same thread, to be largely thermally isolated from other threads or instruction groups, yet run simultaneously. This is in contrast to a traditional SMT microarchitecture in which threads largely share these hot back-end resources. Second, a clustered SMT microarchitecture has a built-in communication mechanism to permit instruction operands to propagate to the cluster in which they are needed. This allows a thread's register values to rapidly move from one cluster to another with low overhead and no additional support required. Finally, the steering mechanism in a clustered SMT provides a simple, yet effective means for temperature management when a thermal emergency arises. If a particular thread, or a subset of that thread's instructions, causes a thermal emergency in a particular back end, then those instructions can simply be steered to a cooler back end for some period of time. Other instructions, possibly from a different thread with less stringent cooling requirements at this moment, may begin to be steered to the hot cluster to make use of its resources, while instructions drain from the prior thread.

In this paper, DTM mechanisms are evaluated for a two-thread, four-cluster microarchitecture. Each back-end cluster is dedicated to a single thread at any time, except for the temporary overlap that occurs when cluster assignments are switched by the steering algorithm. Nominally, each thread is allocated two back ends in the processor. This simplifies the microarchitecture implementation and allows our DTM policies to take advantage of the differences in the thermal heating profiles of the applications. Furthermore, we found that back-end resource utilization was very high, indicating that having an additional thread competing for issue queue slots, issue bandwidth, etc., would not be beneficial.

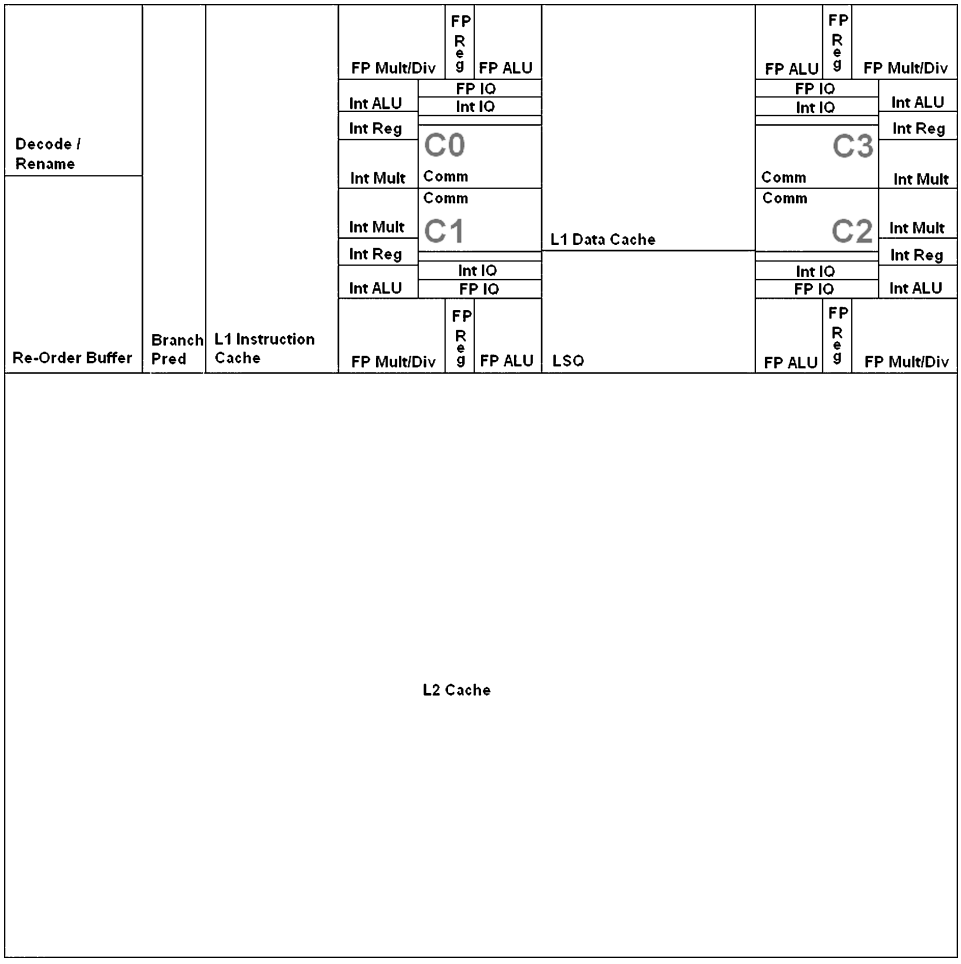

Fig. 5. The floor plan of the clustered SMT microarchitecture.

Figure 5 illustrates the floor plan of our clustered SMT design. The back-end clusters consist of integer and floating-point issue queues, register files, and execution units, as well as a shared set of communication links and a register access window for intercluster operand passing [Zyuban and Kogge 2001]. The communication links provide a ring interconnect for forwarding register values to other back ends, which can forward two values in either direction along the ring each cycle. In addition, there are point-to-point links between each end back and the front end for dispatching instructions, accessing the data L1 cache for stores and loads, communicating branch results, and updating the reorder buffer. Again, the links can send two results in each direction every cycle. The contention and power of all the communication links are modeled. We assumed a centralized, banked L1 data cache in order to simplify the architecture and to allow us to analyze steering policies without the affect of relocating threads away from their associated cache lines. We placed the data cache and the load

store queue in the middle of the clusters on the die to permit a single cycle hop to the cache for loads and stores.

The front end uses the ICOUNT fetch policy [Tullsen et al. 1996] to determine the number of instructions to fetch for each thread each cycle. The baseline front to back end performance-steering mechanism statically dedicates two clusters to each thread [El-Moursy et al. 2005]. As in Balasubramonian et al. [2003], steering of instructions within the same thread to its two clusters is based on load balancing and the location of the instructions' operands. The criticality of each source operand is also considered and ties are broken by sending the instruction to the cluster, whose operand is predicted to be produced last [Balasubramonian et al. 2003]. Further details of the microarchitecture can be found in Table I.

3. METHODOLOGY

The simulation infrastructure is based on the SimpleScalar 3.0 architecture simulator modeling the Alpha ISA [Burger and Austin 1997], augmented with Wattch [Brooks et al. 2000], Hotspot 2.0 [Skadron et al. 2003], and HotLeakage [Zhang et al. 2003] for modeling dynamic power, temperature, and static power, respectively. The simulator has been further modified to support the clustered SMT microarchitecture and dynamic thermal management.

We assumed a processor implemented in 70-nm technology with a clock frequency of 2.5 GHz and a 1.0 V supply voltage. We use HotSpot's default parameters for the thickness of thermal package components, including the die-to-spreader thermal interface material (0.075 mm), the heat spreader (1 mm), and the heat sink thickness (6.9 mm). The ambient temperature is set to 45°C [Skadron et al. 2003]. The *thermal hard limit*, which is the temperature the microprocessor must not exceed to operate properly, is set at 87°C to be consistent with ITRS projections for maximum junction temperature for the 70-nm technology node [Semiconductor Industry Association 2003].

The convection resistance of the heat sink models the quality of the thermal package. It should be set so that with appropriate thermal management, the processor's performance is not degraded severely under worst-case conditions, and so that DTM is not required in the average case. We used a convection resistance of 0.40 K/W for our clustered SMT microarchitecture. One problem with HotSpot is that it inaccurately models the heat flow through the edges of the die [Skadron 2006]. As a result, the blocks near the edge of the chip, particularly the units of cluster three in the corner of the chip, were disproportionately hot relative to their power dissipation. As suggested by Skadron [2006], we addressed this problem by surrounding the die with a ring of dummy blocks about 1 mm $\times$ 1 mm, which allowed floor plan units on the edge to cool more reasonably.

Our leakage power model is based on the code provided for download with HotLeakage [Zhang et al. 2003]. However, we have extended HotLeakage significantly, from the original modeling of the caches and register file, to include static power for all front- and back-end components of the processor. This extension is based on the leakage power estimation method presented by Butts

Table I. Simulated Clustered SMT Microarchitectural Parameters

Unified front-end parameters	
Fetch bandwidth	8 instructions per thread
Branch predictor type	Hybrid of bimodal and 2-level
Bimodal predictor entries	2048, 2-bit counters
Level 1 predictor table entries	1024, history of 10 branches
Level 2 predictor table entries	4096, 2-bit counters
BTB entries	2048, 2-way associative
Branch misprediction penalty	11 cycles
Fetch queue size	16 entries per thread
Decode/rename/dispatch/commit bandwidth	8/8/8/8 instructions per thread
Re-order buffer size	200 entries per thread
Back-end cluster parameters	
Integer/FP issue queue size	20 entries each
Integer/FP register file size	80 entries each
Integer/FP simple ALU	1 of each
Integer/FP complex ALU w/ multiply & divide	1 of each
Register access window	10 slots
Intercluster communication links	2 point-to-point bi-directional links around the ring
Front-end communication links	2 point-to-point bi-directional links to the front end
Memory hierarchy	
Load/store queue size	64 entries per thread
L1 instruction cache	64KB, 4-way, 32B blocks, 8 banks, 1 cycle latency
L1 data cache	64KB, 4-way, 64B blocks, 8 banks, 2 cycle latency
L2 unified cache	4MB, 8-way, 128B blocks, 16 cycle latency
Memory latency	100 cycles

and Sohi [2000], which employs the following main equation:

$$P_{static} = V_{dd} \cdot N \cdot k_{design} \cdot \hat{I}_{leak}$$

Here, P_{static} is the static power of a block in the HotSpot floor plan, V_{dd} is the current processor supply voltage, N is the number of transistors in that processor component, calculated using the area of the floor plan block multiplied by transistor densities for logic and SRAM structures given by Semiconductor Industry Association [2003]. The multiplicative factor accounts for circuit design parameters such as transistor stacking, sizing ratios between the NMOS and the PMOS transistors, and the type of circuit being used. We used a combination of the k_{design} values presented in a chart in Butts and Sohi [2000] for each processor block appropriately matching the types of circuits found in that unit. For example, the issue queues are assumed to be a combination of CAM cells for the wakeup component, SRAM cells to store instruction information, logic for the instruction select stage, and multiplexers for the writeback component and, thus, the k_{design} value used is the weighted average of these. On the other hand, the execution units are assumed to be pure logic and thus have a k_{design} of 11. Finally, $\hat{I}_{leak}$ is the average leakage current of a single transistor for a given technology and temperature.

We model temperature-dependent static power by taking the temperature that HotSpot generates every 10,000 cycle interval and recalculating the $\hat{I}_{leak}$ current term for each block for the next interval. Since we use such a short interval, there is no need to perform iterative leakage calculations because temperatures can only change a few hundredths of a degree in that time frame. HotSpot calculates the starting temperature of the next interval, based on the current interval's temperature and the sum of the dynamic and static power generated over this interval.

When conducting DTM architecture simulations, it is critical to obtain realistic starting temperatures for the components of the thermal model and, in particular, the heat-sink temperature [Skadron et al. 2003]. This is achieved by fast-forwarding four billion instructions per thread, running each benchmark for 500 million instructions on our baseline architecture without DTM, and then using HotSpot to calculate steady-state temperatures for each block. As in Skadron et al. [2003], these steady-state temperatures are scaled so that no block exceeds the emergency threshold and then used as a starting temperature for the DTM simulations. Each DTM simulation also consists of fast-forwarding four billion instructions per thread and then the simulation is allowed to run for 400 million instructions in order to warm-up the caches and branch predictors and allow the DTM mechanisms to begin operating. With the simulation now in a realistic performance and temperature state, each benchmark is executed for 500 million instructions during which time we collect our results. Once a thread completes its 500 million instructions, it continues to run without recording statistics in order to maintain realistic temperature conditions for the other thread and permit continued use of the DTM techniques. Overall performance is measured using the harmonic mean of the IPCs for DTM simulations compared to a baseline run consisting of the same steps, but which uses no DTM mechanism. The harmonic mean was chosen to prevent rewarding algorithms that unfairly constricted the performance of one application in favor of another.

We compare our steering-based DTM policies to dynamic voltage scaling to illustrate their effectiveness against the most popular contemporary approach. DVS scales the overall processor supply voltage and frequency until the temperature cools to an acceptable level. By reducing voltage and frequency, the dynamic power of the processor is reduced almost cubically because of dynamic power's linear dependence on frequency and quadratic dependence on voltage. Static power is also reduced significantly, because as described in the equation above, P_{static} is linearly proportional to V_{dd} and the $\hat{I}_{leak}$ term also has a supply voltage dependence [Zhang et al. 2003].

Our implementation of DVS only employs two voltage levels, the nominal and the low voltage level, as advocated by Skadron [2004], who showed that for DTM, there is virtually no benefit to using multiple voltage steps. After exploring a wide range of low voltages, we found that 0.8 V had the best performance and successfully prevented all thermal violations. At this low voltage, the processor frequency was calculated to be 2.055 GHz by using the following equation [Rabaey 2003]:

$$frequency = k \cdot (V_{dd} - V_t)^{\alpha} / V_{dd}$$

Table II. Baseline Benchmark Set Performance and Thermal Characteristics

Benchmark pairs	Average Number of Clusters in Violation%	Peak Temperature(°C)	Type – IPC (INT/FP – high/low)	Temp. (hot/warm)
Uniform workloads				
applu/apsi	87.84	94.41	FF-HH	HH
bzip2/vortex	100.00	92.74	II-LH	WH
eon/galgel	100.00	92.70	IF-HH	HH
facerec/mesa	100.00	92.68	FF-HH	HH
gzip/vpr	50.00	89.84	II-HL	WW
Nonuniform workloads				
ammp/lucas	24.65	87.84	FF-HL	HW
gcc/mgrid	100.00	93.67	IF-LH	WH
mesa/parser	50.00	94.91	FI-HL	HW
swim/wupwise	88.22	90.55	FF-LH	WH
vortex/equake	50.00	93.86	IF-HL	HW

where V_{dd} is the supply voltage, V_t is the threshold voltage (0.18 V) [Semiconductor Industry Association 2003], α is a technology dependent constant set to 1.5, and k is a fitting constant set to 3.366 in our simulations so as to match our nominal voltage (1.0 V) and frequency (2.5 GHz).

The benchmark sets for our simulations were selected using the 18 hottest SPEC CPU2000 integer and floating-point applications. Each benchmark is represented as equally as possible in the mixes and care was taken to evenly combine benchmarks so that sets had different mixes of floating-point and integer applications, high and low IPC applications, and hotter and colder applications to provide varied workloads for our simulations. One exception is that two low IPC, low heat, benchmarks are not combined, as that pair would lead to a simulation without any need for DTM.

Table II outlines our benchmark sets and their type, performance, and thermal characteristics. It also shows the average number of clusters in thermal violation during the execution of the applications when no dynamic thermal management is employed. For example, a value of 50% means that, on average, one-half the clusters are in thermal violation during the run. In addition, the peak temperature reached during the run without DTM indicates how far the application pairs would exceed the emergency thermal threshold of 87°C.

In spite of the diversity among individual SPEC benchmarks, the most important factor influencing the effectiveness of the DTM techniques on a benchmark pair is the uniformity of the threads' thermal and performance characteristics. The top five pairs in Table II consist of uniform workloads, where both benchmarks are high IPC *and* high temperature, or both benchmarks are integer applications, which thermally stress the same components of the processor. We will show in Section 5 that DVS is the most proficient DTM technique for dealing with this kind of application mix. The bottom five benchmarks are mixed floating-point and integer application pairs or two floating-point applications, where, in either case, one benchmark runs significantly cooler and uses resources less intensively than the other. We focus on these nonuniform benchmarks in the next section and demonstrate that our fine-grain, clustered

SMT temperature-management policies are far more effective than DVS on this workload type.

4. STEERING-BASED TEMPERATURE-MANAGEMENT POLICIES

This section explores a number of steering-based DTM policies. All the mechanisms adhere to the same general framework. When not actively addressing a thermal emergency, the architecture employs the baseline performance steering policy described previously. Simultaneously, the DTM techniques monitor temperatures within each cluster every 10,000 cycles. If the temperature of a back-end component reaches the *trigger threshold*, then the dynamic thermal management mechanism reacts in some manner. When the temperature of the component drops to the *stop threshold*, the thermal emergency is considered alleviated. At this point, if the DTM technique involves throttling some active component, such as avoiding the sending of instructions to a hotspot, the technique is disengaged and processor operation returns to normal. Note that all of the proposed temperature-control mechanisms successfully prevented the occurrence of thermal violations in the back-end clusters.

Our DTM policies can be divided into two general types. Section 4.1 describes *dispatch-gating policies*, which reduce heat by decreasing processor activity. These mechanisms guarantee that a particular maximum temperature will not be exceeded, but incur a high performance penalty. Section 4.2 presents *heat-spreading policies* which seek to balance heat dissipation among the clusters to reduce the occurrences of hotspots. These mechanisms incur a negligible performance cost, but cannot guarantee a safe temperature under all circumstances. Thus, they require a backup fail-safe mechanism, such as dispatch gating. When the spreading policies are successful, however, dispatch gating is not needed, thus avoiding the performance penalty.

4.1 Dispatch-Gating Policies

The dispatch-gating policies all build on the following basic mechanism. The *dispatch-gating trigger threshold* is set to 86.5°C, 0.5°C below the thermal limit, to give the thermal management mechanism some breathing room to operate and to budget for possible temperature sensor error. The value of 0.5°C was empirically determined as the closest value to the emergency threshold that still guaranteed that dispatch gating would succeed in keeping the temperature at a safe level. With dispatch gating engaged, no further instructions are sent from the front end to the issue queues of the hot back end. The hot cluster, receiving no further instructions, will soon run out of work and begin to cool down. In addition, once all the instructions have passed through the cluster's pipeline, dispatch gating clock-gates that back end's resources. This stops all switching activity and thus eliminates the dynamic power of the cluster. Furthermore, to eliminate the leakage power, all structures are power-gated except the register file, which must be left on to preserve register values possibly needed in the future. When dispatch gating succeeds in lowering the peak temperature to the *stop threshold* (85.5°C), the thermal emergency is considered averted and gating is disengaged. Lower values for the stop threshold were considered, such as 84.5

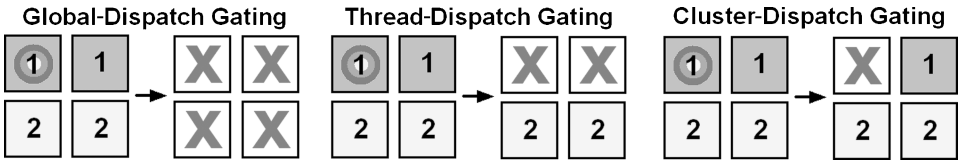

Fig. 6. The three dispatch-gating techniques.

and 83.5°C, in the hope that they would decrease the probability that the hotspot would quickly reheat and require another DTM response causing a ping-pong effect. However, we found as in Chaparro et al. [2007] that hotspots cool with an exponential curve such that keeping dispatch gating on for longer intervals cools less efficiently and reduces performance, because the DTM mechanism remains on for more of the execution time despite being engaged less often.

The difference between the dispatch-gating policies is the granularity at which the back end can be gated. The simplest policy is *global-dispatch gating*, and consists of ceasing dispatch to all the clusters, making it representative of pipeline or resource toggling [Brooks and Martonosi 2001; Skadron et al. 2003; Li et al. 2005]. This policy emulates a thermal management scheme in a nonclustered architecture. *Thread-dispatch gating* distinguishes between the thermal activities of different threads. This technique stops dispatching instructions from a hot thread to the execution engines, but permits other threads to proceed normally.

Cluster-dispatch gating utilizes the clustered nature of our microarchitecture by halting the dispatch of instructions to a specific hot cluster. The thread whose cluster is disabled steers all instructions to its remaining cluster until the emergency subsides. The ability to control the flow of instructions to different parts of the back end is a feature unique to architectures with clustered back ends and enables finer grained DTM control. Figure 6 illustrates how the three dispatch-gating policies operate in the presence of a hotspot in one of the clusters of thread 1.

We compare our steering-based techniques to dynamic voltage scaling, another heat-reduction mechanism that does not require gating. DVS is similarly employed to global-dispatch gating, turning on when the hottest part across all back ends reaches the trigger threshold of 86.5°C and turning off when the stop threshold, 85.5°C, is reached. We assumed that the processor pipeline must stall for 10 μs whenever the voltage and frequency is changed, which is consistent with previous research [Brooks and Martonosi 2001; Skadron et al. 2003; Skadron 2004; Li et al. 2005].

4.1.1 *Results.* Figure 7 shows the performance of DVS and the three dispatch-gating techniques relative to the baseline architecture with no thermal management for the five nonuniform workloads. On average, global-dispatch gating causes a 12.7% performance degradation, compared to thread-dispatch gating with a 7.5% penalty, and cluster-dispatch gating with a 5.0% penalty. Clearly, it is beneficial to dispatch gate at a finer granularity. Because of differences in processor resource-utilization and heating between a workload's benchmarks, it is beneficial to employ thread-dispatch gating, which is able to

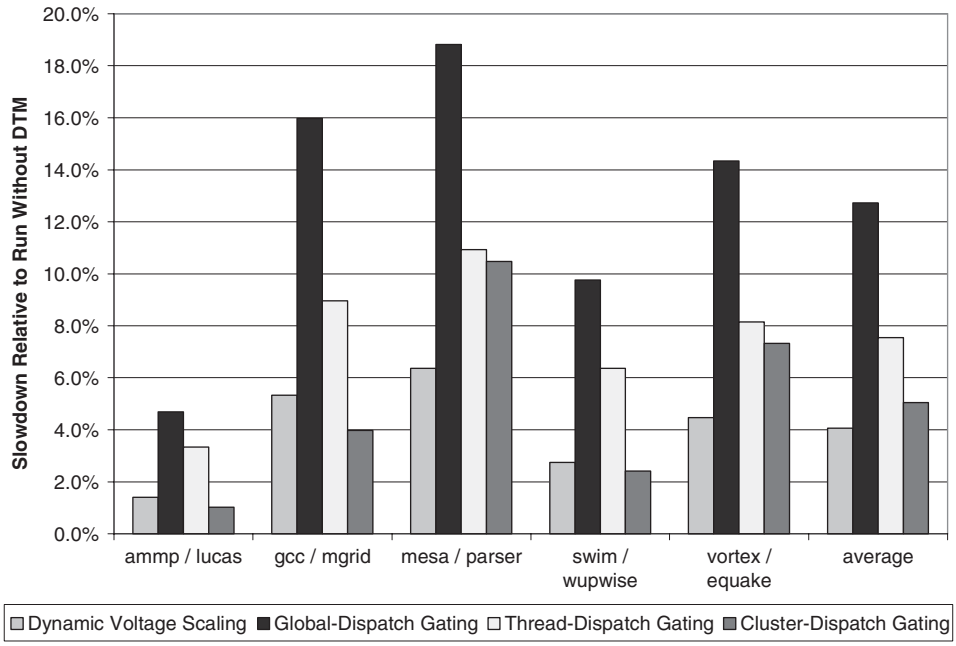

Fig. 7. Performance of the dispatch-gating techniques on the clustered SMT design.

isolate and cool the thread with the hotspot, while allowing the other thread to execute normally when it is not also overheating. The dependence-based baseline performance steering algorithm tries to dispatch dependent instructions to the same cluster as the instructions producing their operands. As a result, back-end clusters of the same thread may exhibit quite different temperature characteristics. Cluster-dispatch gating benefits from cutting power to only the hot cluster, allowing the thread to continue to make forward progress using its other cluster as long as that back end stays cool. While cluster-dispatch gating has lower performance than DVS on three out of the five workloads, DVS is still better overall with a slowdown of 4.1%.

4.2 Heat-Spreading Policies

Heat-spreading policies exploit the different heating patterns of applications to reduce the frequency of thermal crises. By altering the assignment of threads to back-end clusters, these mechanisms cool a hot component of the chip by sending it instructions from a thread that does not heavily utilize that resource. For example, if a processor has a floating-point application that overheats the FP units on its two clusters, steering an integer application to those clusters would cool the FP units while they are inactive. The goal of this approach is to keep all the back-end clusters active all the time, in contrast to policies based on activity migration, which require idling resources [Lim et al. 2002; Heo et al. 2003; Skadron et al. 2003; Chaparro et al. 2004a, 2004b; Ghiasi and Grunwald 2004].

Static heat-spreading policies rearrange the assignment of threads to clusters at fixed intervals regardless of the application thermal behavior. These

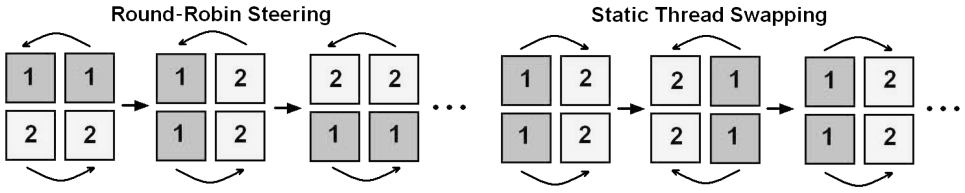

Fig. 8. The two static heat-spreading techniques.

techniques are simple to implement and do not even use temperature measurements (except to determine when to engage dispatch gating—see below). Since static heat-spreading methods do not respond to application behavior, they may potentially miss opportunities to cool threads more effectively. *Dynamic heat-spreading* policies react to thermal conditions on the die by steering instructions or threads that are causing excess heating to cooler areas of the chip. Simultaneously, cooler instructions or threads are steered to the hot components of the processor to allow these areas to avoid thermal emergencies.

Heat-spreading policies are not guaranteed to alleviate thermal emergencies. It may turn out that the "cold" thread enters a phase where it becomes quite hot, preventing the hotspot from cooling down. To ensure that the emergency threshold temperature is not exceeded, all heat-spreading methods must engage a form of dispatch gating as a last resort. The dispatch-gating mechanism engages at the normal dispatch trigger threshold, which is the temperature at which the heat-spreading mechanism is deemed to have failed to address the temperature emergency. Because of the superiority of cluster-dispatch gating, it serves as this fail-safe backup cooling mechanism for all our heat-spreading policies.

4.2.1 *Static Heat-Spreading Policies.* We considered two static heat-spreading algorithms. The first, *round-robin steering*, simply shifts the clusters assigned to each thread by one in a counterclockwise direction after a fixed interval length. For example, if a thread is running on clusters 0 and 1 and the interval length is one million cycles, after one million cycles it will be steered to clusters 1 and 2; after two million cycles it will be steered to clusters 2 and 3.

The second algorithm, called *static thread swapping*, swaps the clusters assigned to one thread with the clusters assigned to another after each fixed-interval length. With two threads, this means that the first thread is steered to clusters 0 and 1 for one interval, then to clusters 2 and 3 for another interval, and then to clusters 0 and 1. The other thread is steered to the alternate set of clusters. We simulated policies with interval lengths ranging from 10,000 to 50 million cycles and found that one million cycle intervals delivered the best performance.

Combining static heat-spreading methods with dispatch gating is very simple. Both techniques are simply run together, with the spreading mechanism changing the assignment of threads to clusters, and cluster-dispatch gating engaging when the dispatch trigger threshold is reached, and disengaging at the stop threshold. Figure 8 illustrates the workings of the two static heat-spreading policies.

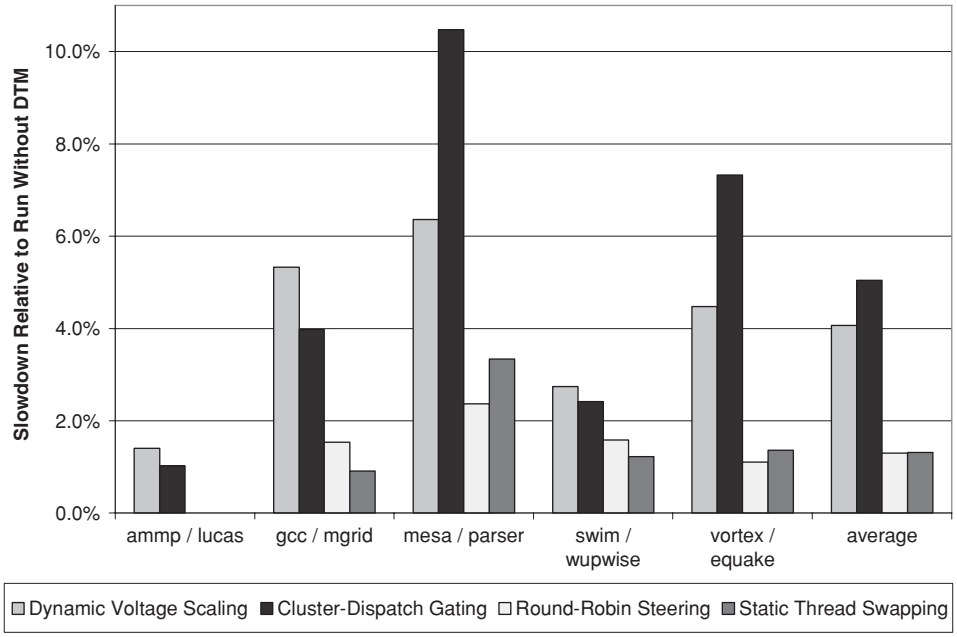

Fig. 9. Performance of the static heat-spreading techniques on the clustered SMT design.

4.2.2 *Static Heat-Spreading Results.* The results for static spreading are shown in Figure 9 along with cluster-dispatch gating for comparison. First, static heat-spreading mechanisms definitely improve upon cluster-dispatch gating, decreasing the average slowdown from 5.0 to 1.3%. In particular, for the *ammp/lucas* workload, static heat spreading is so effective that no dispatch gating is needed and the mechanisms had zero slowdown compared to the baseline. These spreading techniques also have a performance degradation of less than one-third of the slowdown of DVS. While round-robin steering and static thread swapping are about even in slowdown on nonuniform workloads, round robin is a slightly better mechanism across uniform workloads (not shown). The greater success of round-robin was primarily because of its ability to more evenly spread threads among the clusters, since, over the course of the run, each thread is assigned to every pair of neighboring clusters. In static thread swapping, threads either use clusters 0 and 1, or 2 and 3, and thus cooling is not as even.

4.2.3 *Dynamic Heat-Spreading Policies.* The dynamic heat-spreading policies monitor the thermal behavior of the running applications and rearrange the cluster to thread assignment to balance the heating of the back ends. Every 10,000 cycles, the temperature sensors of the back-end clusters are examined. Dynamic spreading is then applied to any back ends with a peak temperature above the *spreading trigger threshold* (85.5°C), starting from the hottest cluster. To give the dynamic spreading techniques a chance to have an effect, the algorithm waits five million cycles before making another change to a back end involved in an earlier application of dynamic heat spreading. Furthermore, this

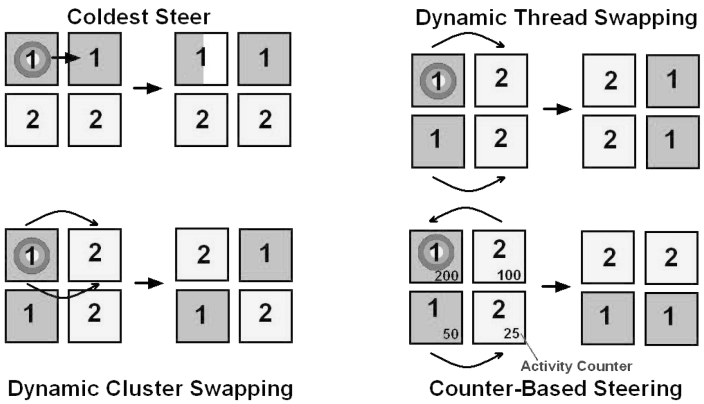

Fig. 10. The four dynamic heat-spreading techniques.

prevents the algorithm from responding to small temperature fluctuations that do not threaten to cause thermal emergencies. Figure 10 provides a graphic explanation of the dynamic heating policies.

The first heat-spreading mechanism, *coldest steer*, is an intrathread-spreading policy that works within a single thread's clusters. It is inspired by the T-thermal algorithm of Chaparro et al. [2004a]. In order to avoid applying the policy unnecessarily and to account for noisy sensors, it requires that the difference in the peak temperatures between the hot and cold thread of a cluster be over a 0.2°C threshold before engaging. When engaged, it tries to send all instructions to the colder cluster of the two that are allocated to the thread. However, if the cold cluster cannot accept any more dispatched instructions because it has no free physical registers or issue queue slots, then instructions are sent to the hot cluster, if it has room. Ideally, this algorithm is successful in directing the bulk of the instructions to the colder cluster, leaving the hot cluster with a light load and a chance to cool off. The relevant thread can still suffer a performance drop, because instructions are no longer steered according to the performance-conscious steering policy of the baseline architecture.

Dynamic thread swapping attempts to cool a hot thread by exchanging the clusters that the hot thread is using with the clusters assigned to the colder thread. Thus, it is a more sophisticated version of static thread swapping policy. The colder thread is determined as the thread with the lowest peak temperature among all its clusters and its instructions are steered to the hot clusters to permit them to cool. The hot thread is steered to the cooler clusters of the other thread, allowing the application to heat up this part of the back end without causing an immediate thermal emergency. Both the hot and the cold thread are steered to their new clusters for at least five million cycles. This prevents the cold thread from being falsely picked as a hot thread right after it was moved to the hot cluster and before the temperature has had a chance to change.

Often, the algorithm will swap two threads and after waiting five million cycles, it will find that the hot cluster is still hot. Before swapping the hot clusters again, the algorithm checks to see if that cluster has decreased in temperature. If the hot cluster is cooler than when it was swapped, the swapping is deemed

to be successful and the thread running on the cluster is not swapped again, despite still being above the spreading threshold. This criterion prevents clusters that are cooling down slowly from being swapped prematurely. However, if the hot cluster was dispatch gated since last being swapped, then thread swapping proceeds normally, as the original swap clearly did not solve the heating crisis. These extra features were experimentally found to improve the algorithm's performance over blindly swapping hot threads again after five million cycles.

The third dynamic spreading algorithm is *dynamic cluster swapping*. This mechanism is very similar to dynamic thread swapping. The only difference is that a single hot cluster can be individually swapped with a single cold cluster. Often the temperatures of the clusters associated with a given thread are quite different. Swapping at the granularity of a single cluster gives the spreading algorithm more flexibility to pick the best back ends to assign to a hot thread and does not force it to reassign clusters that are not causing thermal problems. On the other hand, cluster swapping may lead to configurations where a thread is using noncontiguous clusters in the back end, resulting in higher intercluster communication of register operand values.

Our final technique, *counter-based steering*, employs activity counters for back-end components, in particular, the number of instructions issued, the number of register accesses, and the number of times a functional unit is used. We examined the steering decisions made by the previous three interthread dynamic spreading techniques and found that often the algorithms make the mistake of swapping too often because they cannot tell whether a hot unit is heating up because of the current thread running on it or still hot because of the previous thread that used it. This final algorithm gets around that problem by using activity counters as an indicator of a thread's thermal intensity as in Powell et al. [2004] and Donald and Martonosi [2005, 2006]. The activity of the various back-end components is continuously tracked by hardware counters. When the spreading trigger threshold is reached, the steering algorithm selects the thread with the least activity in the processor component that is overheating and starts dispatching its instructions to the hot cluster and its hottest neighboring cluster, in an attempt to cool those clusters. Meanwhile, the thread that caused the overheating is sent to the other cooler clusters. The counters are then reset waiting for the next hotspot. If the thread with the lowest activity is already occupying the hot cluster, the algorithm chooses not to do anything, because most likely moving threads will just heat the hotspot even more.

As with the static spreading policies, cluster-dispatch gating is engaged as a backup DTM mechanism for the dynamic spreading mechanisms if a hotspot reaches the dispatch trigger threshold and is disabled at the stop threshold. The spreading trigger threshold is set to be the same value as the stop threshold for dispatch gating. This way, as soon as dispatch gating is turned off, spreading is engaged to further cool the hot thread and hopefully prevent the need to use dispatch gating in the future.

When cluster-dispatch gating is employed with counter-based steering, there will be some interference between the two mechanisms. Since counter-based steering observes the activity levels of back-end components, its readings will be

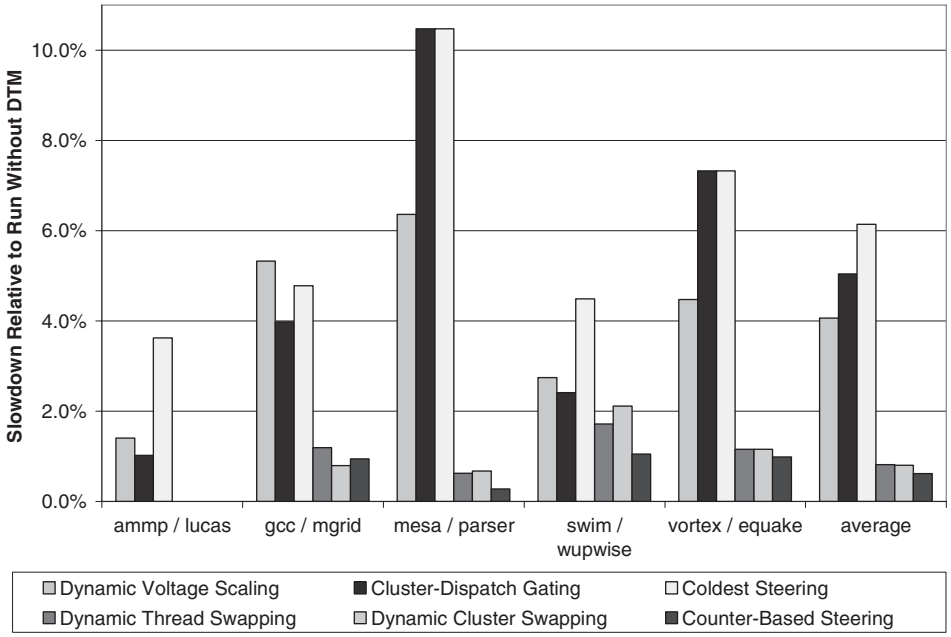

Fig. 11. Performance of the dynamic heat-spreading techniques on the clustered SMT design.

affected by dispatch gating activity, which deactivates clusters for short periods of time. However, we observed that the interference was mostly constructive as clusters that were dispatch gated and thus had lower levels of activity also had lower temperatures, making them good candidates for the steering algorithm to send hot threads to.

4.2.4 *Dynamic Heat-Spreading Results.* Figure 11 shows the dynamic heat-spreading results. Clearly, coldest steer is ineffective, with overall performance worse than cluster-dispatch gating alone. There are three reasons for coldest steer's ineffectiveness in an SMT environment: (1) the two clusters of the hot thread often have almost the same peak temperature, (2) the limited free resources in the cold cluster means that most of the instructions from the hot cluster cannot be moved, and (3) coldest steer prevents the very effective performance-driven steering mechanism from dispatching instructions to minimize communication between clusters and balance the instruction load.

On the other hand, our other three dynamic techniques are very effective at providing low-cost DTM. Dynamic thread and cluster swapping both show a 0.8% slowdown, on average. On individual workloads, sometimes thread swapping is superior, on other benchmarks, cluster swapping is preferable. When it is most important to keep each thread's clusters together to decrease the communication costs of passing register values, thread swapping performs better. On other workloads, cluster swapping takes advantage of differences in the thermal behavior of individual clusters to provide more precise targeting of hotspots and, thus, requires less use of dispatch gating.

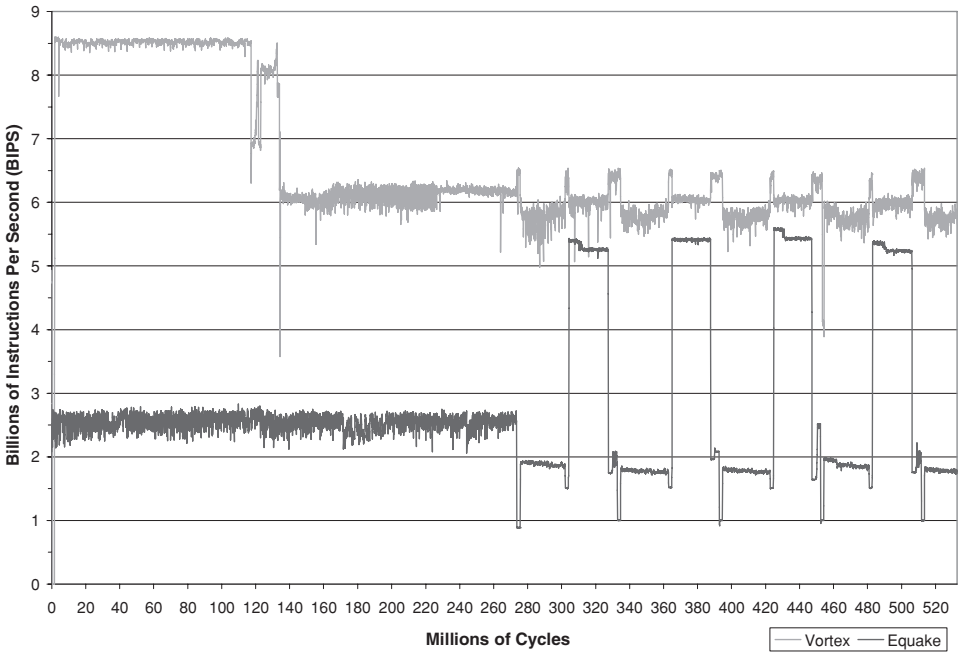

Fig. 12. Vortex/equake—performance with the counter-based steering DTM policy.

Finally, we find that the counter-based steering policy performs best with a worst-case degradation of only 1.0% on nonuniform workloads, compared to 6.4% for DVS (*mesa*/*parser*). To gain a deeper understanding of the workings of counter-based steering, Figure 12 shows the performance of the *vortex*/*equake* workload; Figure 13 shows the temperatures of the hottest resources during the simulation. Note the similarity of Figure 12 (counter-based steering) with Figure 1 (no DTM), indicating little performance loss is incurred. Indeed, the performance loss on this benchmark pair is only 1.0% compared with 4.5% for DVS (Figure 2). Undoubtedly, the counter-based steering policy is better suited than DVS to managing hotspots in a nonuniform workload and Figure 13 indicates why. By switching the thread to cluster assignment intelligently, counter-based steering significantly evens out the temperatures across all the clusters and eliminates most of the need for dispatch gating. Cluster-dispatch gating is called as a backup fail-safe just once, compared to the 27 times that DVS is required. When DVS is engaged, the frequency and voltage of the processor are decreased, harming both benchmark's performance, as shown in Figure 2 by the multiple drops in the BIPS rate.

Overall, the best steering-based DTM policies are round-robin and counter-based steering. While round-robin steering is not quite as effective as counter-based steering, it is a very simple static policy to implement. Counter-based steering is appealing because of its particularly good performance on nonuniform workloads, where it demonstrates a significant advantage over DVS. We, therefore, consider combining it with DVS to achieve a "best-of-both-worlds" DTM technique.

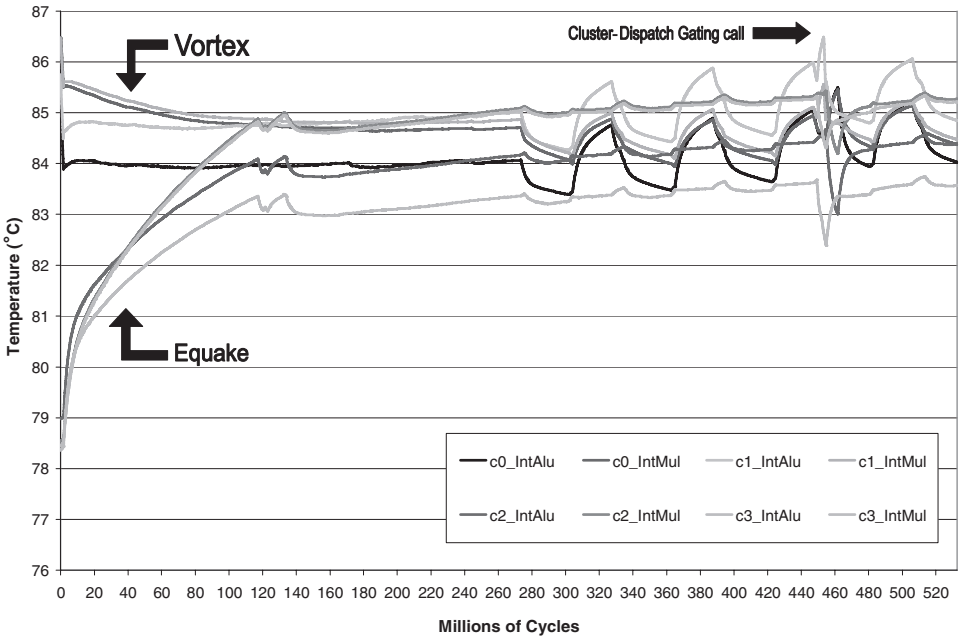

Fig. 13. Vortex/equake—temperature with the counter-based steering DTM policy.

5. COMBINED POLICIES

Given the good match of dynamic voltage scaling to uniform workloads, and steering-based DTM's proficiency on nonuniform benchmark pairs, such as *vortex*/*equake*, it seems natural to combine the two approaches. Thus, we considered a hybrid mechanism where counter-based steering is used to spread the heat and dynamic voltage scaling takes the place of cluster-dispatch gating as the fail-safe mechanism for preventing a thermal crisis. (We implemented other hybrid policies using round-robin and dynamic thread swapping, but found counter-based steering to perform best.) Ideally, this combined policy will harness the temperature-steering mechanism to obviate much of the need to use DVS on nonuniform workloads, while we will get the performance benefits of DVS on the uniform workloads where there is little temperature variation among clusters to exploit. Employing global DVS as a backup mechanism does not cause interference with counter-based steering. Reducing the frequency of the back ends changes the counter values read each interval in an absolute sense, but does not affect the relative differences between component activities that are used to determine which threads are thermally intensive and need to be assigned to cold clusters. While it is possible to combine DVS, cluster-dispatch gating, and counter-based steering, we felt that merging three DTM mechanisms would be too complex to implement in hardware.

Figure 14 compares dynamic voltage scaling, counter-based steering with cluster-dispatch gating (CDG), and a hybrid policy (counter-based steering + DVS) for both uniform and nonuniform workloads. As expected, DVS is superior for uniform workloads, with an overall slowdown of 4.6% compared to 6.4% for

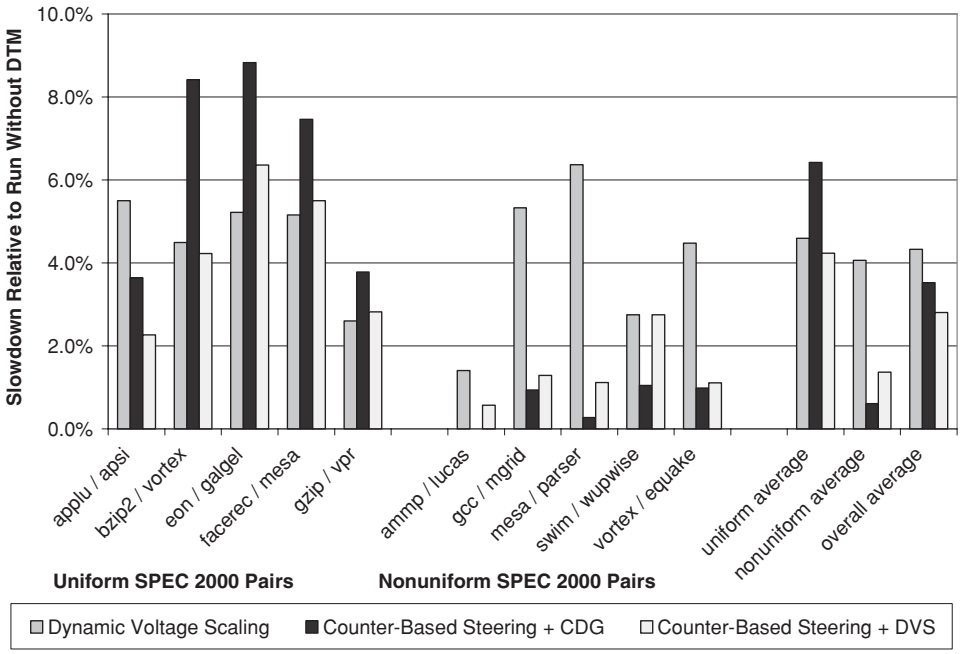

Fig. 14. Overall performance comparison of the DTM techniques on the clustered SMT design.

counter-based steering. Under these uniform conditions when both threads are simultaneously hot, it is difficult to beat the almost cubic power reduction of DVS. However, because of the superiority of counter-based steering on nonuniform workloads, it provides better performance, averaged across all workloads (uniform and nonuniform combined) with a degradation of 3.5%, compared to 4.3% with DVS.

Looking at the hybrid policy, shown in the the last bar in Figure 14, we see that this mechanism is effective at merging the better performance features of DVS and counter-based steering. Specifically, this combination provides strictly better performance than DVS, with degradations on uniform, nonuniform, and overall workloads at 4.2, 1.4, and 2.8%, respectively (compared to 4.6, 4.1, and 4.3% for DVS). Yet, it is still capable of obtaining most of the heat-spreading benefit on nonuniform workloads. Furthermore, our results assume that the processor will see an equal mix of uniform and nonuniform workloads. In the event of an imbalanced workload of all uniform or all nonuniform thread combinations, counter-based steering with DVS provides near optimal performance regardless of the application mix, unlike either technique alone.

6. RELATED WORK

Previous research related to our study can be categorized into a number of different themes. First, there are a few studies that pioneered the need for dynamic thermal management and proposed early solutions in this area. Skadron et al. [2003] released HotSpot, an easy to use architectural-level temperature model, which generated much research activity. A number of more recent works

look at thermal issues on processors with multiple threads either through SMT or CMP. Finally, a few papers analyze DTM on single-threaded clustered microarchitectures.

Huang et al. [2000] were perhaps the first to propose the need for dynamic thermal management. They outlined a framework called dynamic energy efficiency and temperature management (DEETM), which invokes the operating system to monitor chip thermal behavior and engage various techniques to combat high temperatures. Brooks and Martonosi [2001] further strengthen the case for dynamic thermal management and outline the basic components and methodology that all DTM schemes are based upon. Dhodapkar et al. [2000] propose TEM^{2}P^{2}EST, an early thermal model at the architectural level.

Lim et al. [2002] use the TEM^{2}P^{2}EST model to explore a form of activity migration that uses a second low-power in-order pipeline when the main out-of-order pipeline overheats. Heo et al. [2003] examine activity migration in detail and try to determine which microprocessor components are best duplicated. They conclude that it is most critical to replicate the register files and execution units to get the best power/area tradeoff. We similarly focus on the back-end execution engine. However, the important difference between our research and these works on activity migration is that we do not require spare, backup resources for our DTM mechanisms. The extra register files, ALUs, and pipelines required by these prior schemes are inactive most of the time when there are no thermal emergencies, and they add significantly to the cost of the chip.

Skadron et al. [2003] describe and use HotSpot to study a number of DTM techniques on a single-threaded core, which thereafter became the standard tool for DTM research. Skadron [2004] extends that work by combining DTM techniques in a hybrid scheme. Liao et al. [2003, 2005] examine the impact of temperature-dependent modeling of leakage power and thermal run-away on DTM techniques. This work advocates considering the effect temperature and voltage dependence on performance and power and the need for tightly coupled management of power and temperature. As in our work, Powell et al. [2005] use HotSpot to study the asymmetric usage of back-end processor resources and devise algorithms to spread the power and heating more evenly. However, they examine a single-threaded processor and thus only explore intrathread diversity in application behavior. Furthermore, we go beyond utilizing asymmetry to show how DVS and our steering algorithms, which are well adapted to different workload characteristics, can be combined to provide a comprehensive DTM scheme. All these earlier works focus on single-threaded processors.

A number of papers study power and thermal issues on SMT and CMP architectures. A few papers look specifically at the energy efficiency of multithreaded processors. Seng et al. [2000] examine the power efficiency of SMT machines and propose architectural enhancements to optimize energy usage. Similarly, Li et al. [2004] look at the energy efficiency of SMT machines and identify the root causes of multithreading's power advantage. Another study compares the energy efficiency of chip multiprocessors and simultaneous multithreaded machines and finds CMPs to be more compelling [Sasanka et al. 2004]. Managing chip temperature, however, is a distinct issue from power efficiency, because

many schemes for saving energy do not reduce power density, or reduce average power, but have no effect on die hotspots [Skadron et al. 2003].

Other research looks at the more related topic of thermal management on SMT and CMP machines. Ghiasi and Grunwald [2004] examine an asymmetric chip multiprocessor with a power-hungry ILP intensive core and a spare low-power, low-performance core. Under normal operating conditions, the high-performance core executes a single thread. DTM consists of activity migration between these two cores, which is done either proactively or reactively. Again, the need for a spare core results in a large amount of die area that is not used unless the workload is extremely thermally intensive.

Donald and Martonosi [2004] examine the thermal properties of two- and four-threaded SMT and CMP architectures. They find that these processors experience higher temperatures, but have a similar heating distribution across their components to that of a single-thread superscalar machine. Rather than using DTM, the authors propose to mitigate the high temperatures of the issue queue and result bus by changing the floor plan of the processor and by increasing the area of these units to spread out the heat.

In another work, the same researchers investigate the possibility of exploiting varying application behavior to control temperature on an SMT processor [Donald and Martonosi 2005]. Temperature-guided fetch and rename policies are developed. The goal is to restrict the flow of instructions to the thread least likely to heat the integer and floating-point register files, deemed the hottest processor components. Like our work, that paper tries to adjust the instruction flow in an SMT processor to mitigate the heating effects of a hot thread. However, both their policies are toggling policies that reduce activity to reduce hotspots. Our static and dynamic steering techniques utilize the differences in application thermal characteristics to eliminate hotspots, often without reducing performance.

Heat Stroke is a denial-of-service (DoS) attack on an SMT processor in which a malicious thread highly utilizes a particular pipeline resource, creating a hotspot on the chip [Hasan et al. 2005]. In that work, Hasan et al. [2005] assert that such a Heat Stroke attack harms the performance of other threads on the processor when DTM mechanisms, such as stop–go or DVS, slowdown the whole pipeline in order to cool the hotspot. They conclude that it is imperative to isolate the offending thread and restrict only its execution through sedation, which has the same effect as our thread-dispatch gating. We likewise propose thread- and cluster-specific DTM techniques to avoid penalizing threads that are not causing thermal problems. However, while the goal of their work is to stop malicious threads from degrading performance, our hot threads are behaving properly, but happen to use the processor intensively. Rather than simply restrict the execution of these threads, we propose steering-based DTM policies, which take advantage of the variations in applications to improve the thermal behavior of the hot thread by intelligently steering it to clusters cooled by less resource intensive threads. This approach does not punish hot threads, but, instead, increases IPC by lengthening the time before performance-harming DTM must be engaged.

Other research proposes HybDTM, a methodology for coordinating fine-grained hardware techniques and coarser grain software mechanisms to provide more effective thermal management [Kumar et al. 2006]. The authors evaluate their approach on a Pentium 4 processor running Linux in single thread and SMT configurations. Like our work, their two-level approach combines a low-cost mechanism (an OS software technique), which tries to keep the processor as cool as possible and a higher cost mechanism (hardware stop-go) as a backup to prevent thermal emergencies. However, since they experiment with a real processor, they focus on making better use of features that are already implemented, whereas we propose new hardware DTM techniques for future clustered SMT architectures.

Donald and Martonosi [2006] employ formal control theory to perform DTM in a four-core CMP environment. They compare chip-wide and per-core implementations of a basic stop-go policy (equivalent to global-dispatch gating) and dynamic voltage scaling and experiment with migration of threads among the cores. In contrast, our work investigates, in more detail, the DTM opportunities within a single SMT core and shows that there are effective alternatives to dynamic voltage scaling at the per-core level. While we examine how nonuniform heating within an SMT core can be exploited for thermal management, they look at single-thread cores and treat the cores holistically. Future research could explore how our intracore and Donald and Martonosi's intercore policies could work in tandem for even more effective DTM.

Chaparro et al. [2007] also examine thermal management using stop–go, thread migration, and DVS in a chip multiprocessor of single-thread cores. However, these authors evaluate a sixteen-core architecture, compare a number of variants of stop–go and thread migration, and focus on performing sensitivity studies to understand the impact of cool-down interval length, emergency threshold temperature, frequency of DTM decisions, and the quality of the thermal solution. Again, the insight they develop for intercore DTM policies can be combined with our techniques for managing temperature within the core.

Heat-and-run exploits the variation in application resource usage to manage temperature in an architecture consisting of a chip multiprocessor composed of SMT cores [Powell et al. 2004]. By pairing complimentary applications in a multiprogrammed workload, processor-core resources are maximally utilized and heated. These more efficiently used cores can then be cooled using activity migration across the cores of the CMP. While heat-and-run focuses on how to schedule threads to cores to maximize processor use under thermal constraints, our work examines how to manage a given set of threads within the core to reduce the cost of DTM.

Li et al. [2005] compare the thermal behavior of a baseline superscalar processor to that of a two-way SMT and a CMP of two cores. They also compare the effectiveness of a variety of DTM techniques on these architectures, including DVS and throttling of the fetch unit, rename, and the register file. Unlike our DTM policies, they do not employ mechanisms that adapt to application resource usage or exploit nonuniform thermal behavior to cool hotspots.

Chaparro et al. [2004a, 2004b] propose that clustered microarchitectures naturally reduce temperatures in the processor, because they distribute resources and computation among the back ends. They investigate single-thread clustered processors and develop some simple steering mechanisms to mitigate thermal problems. The also propose cluster hopping, a form of activity migration where instructions are only sent to a subset of the back ends in the processor. Unlike our simulated microarchitecture, their clusters are highly overprovisioned for a single-thread workload and, thus, unused clusters can again be thought of as spares. We study a whole new class of adaptive DTM techniques for clustered multithreaded machines, which exploit workload nonuniformity and do not require spare or idle resources. Furthermore, we provide a comparison against DTM mechanisms, such as DVS and global-dispatch gating, to demonstrate the benefits of our steering-based techniques. Finally, orthogonal research to ours explores the thermal benefits of front-end clustering [Chaparro et al. 2005]. The enhancements in that paper could be combined with our work to provide DTM for the entire processor.

7. CONCLUSIONS

In this work, we identify a class of SMT workloads where there is significant room to improve on the performance of dynamic voltage scaling. For nonuniform workloads with mixed integer and floating-point applications or different thermal behaviors among the threads, DVS's global effect cools one thread at the performance expense of the other. To address this deficiency, we propose DTM policies that leverage the built-in partitioning, steering, and thread-migration mechanisms of clustered SMT architectures to provide effective temperature control with low-implementation complexity. On nonuniform workloads, our best policy, counter-based steering, provides effective DTM with only a 1% worst-case slowdown compared to a 6.4% slowdown for DVS over a baseline without thermal management. Furthermore, our steering-based policy is competitive with DVS across all workloads.

Our clustered SMT DTM policies have the additional advantage of not requiring the capability to scale frequency and voltage on a per-core level. This makes them very attractive for implementation in future large-scale chip multiprocessors, where having numerous voltage domains will be undesirable, and possibly infeasible. Furthermore, our steering-based DTM techniques will function effectively in future technologies with ultralow supply voltages, which may pose problems for dynamic voltage scaling.

In order to take advantage of the complementary features of steering-based DTM and DVS, we propose combining counter-based steering with dynamic voltage scaling. In this "best-of-both-worlds" policy, counter-based steering addresses nonuniform workloads by spreading the heat and minimizing the need to use DVS, while uniformly hot benchmark pairs are cooled most effectively by dynamic voltage scaling. As a result, this hybrid policy provides the best overall performance with a slowdown of 2.8% compared to 4.3% for DVS and 3.5% for counter-based steering with cluster-dispatch gating.

REFERENCES

BALASUBRAMONIAN, R., DWARKADAS, S., AND ALBONESI, D. 2003. Dynamically managing the communication-parallelism trade-off in future clustered processors. In *Proceedings of ISCA 30* (June). 275–286.

BROOKS, D. AND MARTONOSI, M. 2001. Dynamic thermal management for high-performance microprocessors. In *Proceedings of HPCA 7* (Jan.), 171–182.

BROOKS, D., TIWARI, V., AND MARTONOSI, M. 2000. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of ISCA 27* (June), 83–94.

BURGER, D. AND AUSTIN, T. 1997. The SimpleScalar Toolset, Version 2.0. *ACM SIGARCH Comp. Arch. News 23*, 3 (June), 13–25.

BUTTS, J. A., AND SOHI, G. S. 2000. A static power model for architects. In *Proceedings of MICRO 33* (Dec.), 191–201.

CHAPARRO, P., GONZÁLEZ, J., AND GONZÁLEZ, A. 2004a. Thermal-aware clustered microarchitectures. In *Proceedings of ICCD* (Oct.), 48–53.

CHAPARRO, P., GONZÁLEZ, J., AND GONZÁLEZ, A. 2004b. Thermal-effective clustered microarchitectures. In *Proceedings of the 1st TACS Workshop* (June).

CHAPARRO, P., MAGKLIS, G., GONZÁLEZ, J., AND GONZÁLEZ, A. 2005. Distributing the frontend for temperature reduction. In *Proceedings of HPCA 11* (Feb.), 61–70.

CHAPARRO, P., GONZÁLEZ, J., MAGKLIS, G., CAI, Q., AND GONZÁLEZ, A. 2007. Understanding the thermal implications of multicore architectures. In *IEEE Transactions on Parallel and Distributed Systems 18*, 8 (Aug.), 1055–1065.

COLLINS, J. D., AND TULLSEN, D. M. 2004. Clustered multithreaded architectures—Pursuing both IPC and cycle time. In *Proceedings of IPDPS 18* (April), 76–86.

DHODAPKAR, A., LIM, C. H., CAI, G., AND DAASCH, W. R. 2000. TEM^{2}P^{2}EST: A thermal enabled multi-model power/performance estimator. In *Proceedings of the 1st PACS Workshop* (Nov.).

DONALD, J., AND MARTONOSI, M. 2004. Temperature-aware design issues for SMT and CMP architectures. In *Proceedings of the 5th WCED Workshop* (June).

DONALD, J. AND MARTONOSI, M. 2005. Leveraging simultaneous multithreading for adaptive thermal control. In *Proceedings of the 2nd TACS Workshop* (June).

DONALD, J., AND MARTONOSI, M. 2006. Techniques for multicore thermal management: classification and new exploration. In *Proceedings of ISCA 33* (June) 78–88.

EL-MOURSY, A., GARG, R., ALBONESI, D. H., AND DWARKADAS, S. 2005. Partitioning multi-threaded processors with a large number of threads. In *Proceedings of ISPASS* (March) 112–123.

GHIASI, S. AND GRUNWALD, D. 2004. Design choices for thermal control in dual-core processors. In *Proceedings of the 5th WCED Workshop* (June).

HASAN, J., JALOTE, A., VIJAYKUMAR, T. N., AND BRODLEY, C. E. 2005. Heat stroke: Power-density-based denial of service in SMT. In *Proceedings of HPCA 11* (Feb.) 166–177.

HEO, S., BARR, K., AND ASANOVIC, K. 2003. Reducing power density through activity migration. In *Proceedings of ISLPED* (Aug.) 217–222.

HUANG, M., RENUA, J., YOO, S., AND TORRELLAS, J. 2000. A framework for dynamic energy efficiency and temperature management. In *Proceedings of MICRO 33* (Dec.), 202–213.

KUMAR, A., SHANG, L., PEH, L.-S., AND JHA, N. K. 2006. HybDTM: A coordinated hardware-software approach for dynamic thermal management. In *Proceedings of the 43rd DAC* (July). 548–553.

LATORRE, F., GONZÁLEZ, J., AND GONZÁLEZ, A. 2004. Back-end assignment schemes for clustered multithreaded processors. In *Proceedings of ICS 18* (June), 316–325.

LI, Y., BROOKS, D., HU, Z., SKADRON, K., AND BOSE, P. 2004. Understanding the energy efficiency of simultaneous multithreading. In *Proceedings of ISLPED* (Aug.). 44–49.

LI, Y., BROOKS, D., HU, Z., AND SKADRON, K. 2005. Performance, energy, and thermal considerations for SMT and CMP architectures. In *Proceedings of HPCA 11* (Feb.), 71–82.

LIAO, W., LI, F., AND HE, L. 2003. Microarchitecture level power and thermal simulation considering temperature dependent leakage model. In *Proceedings of ISLPED* (Aug.), 211–216.

LIAO, W., HE, L., AND LEPAK, K. M. 2005. Temperature and supply voltage aware performance and power modeling at microarchitecture level. In *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, *24*, 7 (July), 1042–1053.

LIM, C. H., DAASCH, W. R., AND CAI, G. 2002. A thermal-aware superscalar microprocessor. In *Proceedings of ISQED* (March). 517–522.

POWELL, M. D., GOMAA, M., AND VIJAYKUMAR, T. N. 2004. Heat-and-run: Leveraging SMT and CMP to manage power density through the operating system. In *Proceedings of ASPLOS XI* (Oct.). 260–270.

POWELL, M. D., SCHUSCHMAN, E., AND VIJAYKUMAR, T. N. 2005. Balancing resource utilization to mitigate power density in processor pipelines. In *Proceedings of MICRO 38* (Nov.), 294–304.

RAASCH, S. AND REINHARDT, S. 2003. The impact of resource partitioning on SMT processors. In *Proceedings of PACT 12* (Sept.), 15–26.

RABAEY, J. M. 1995. *Digital Integrated Circuits: A Design Perspective*. Prentice-Hall, Englewood Cliffs, NJ.

SASANKA, R., ADVE, S. V., CHEN, Y. AND DEBES, E. 2004. The energy efficiency of CMP vs. SMT for multimedia workloads. In *Proceedings of ICS* (June). 196–206.

SEMICONDUCTOR INDUSTRY ASSOCIATION. 2003. International Technology Roadmap for Semiconductors – 2003 Edition. In http://www.itrs.net/Links/2003ITRS/Home2003.htm.

SENG, J. S., TULLSEN, D. M., AND CAI, G. Z. N. 2000. Power-sensitive multithreaded architecture. In *Proceedings of ICCD* (Sept.). 199–206.

SKADRON, K. 2004. Hybrid architectural dynamic thermal management. In *Proceedings of DATE* (Feb.), 10–15.

SKADRON, K. 2006. Personal communication.

SKADRON, K., STAN, M. R., HUANG, W., VELUSAMY, S., SANKARANARAYANAN, K., AND TARJAN, D. 2003. Temperature-aware microarchitecture. In *Proceedings of ISCA 30* (Apr.), 2–13.

TULLSEN, D., EGGERS, S., LEVY, H., EMER, J. S., LEVY, H. M., LO, J. L., AND STAMM, R. L. 1996. Exploiting choice: Instruction fetch and issue on an implementable simultaneous multithreading processor. In *Proceedings of ISCA 23* (May), 191–202.

ZHANG, Y., PARIKH, D., SANKARANARAYANAN, K., SKADRON, K., AND STAN, M. 2003. HotLeakage: A temperature-aware model of subthreshold and gate leakage for architects. *The University of Virginia, Department of Computer Science, Technical Report CS-2003-05* (March).

ZYUBAN, V. AND KOGGE, P. 2001. Inherently lower-power high-performance superscalar architectures. *IEEE Trans. Comput. 50*, 3 (Mar.), 268–285.

Received March 2007; revised July 2007; accepted August 2007