Techniques for Multicore Thermal Management: Classification and New Exploration

James Donald and Margaret Martonosi

Department of Electrical Engineering
Princeton University
{jdonald,mrm}@princeton.edu

Abstract

Power density continues to increase exponentially with each new technology generation, posing a major challenge for thermal management in modern processors. Much past work has examined microarchitectural policies for reducing total chip power, but these techniques alone are insufficient if not aimed at mitigating individual hotspots. The industry's current trend has been toward multicore architectures, which provide additional opportunities for dynamic thermal management.

This paper explores various thermal management techniques that exploit the distributed nature of multicore processors. We classify these techniques in terms of core throttling policy, whether that policy is applied locally to a core or to the processor as a whole, and process migration policies. We use Turandot and a HotSpot-based thermal simulator to simulate a variety of workloads under thermal duress on a 4-core PowerPC^{TM}processor. Using benchmarks from the SPEC 2000 suite we characterize workloads in terms of instruction throughput as well as their effective duty cycles. Among a variety of options we find that distributed control-theoretic DVFS alone improves throughput by 2.5X under our test conditions. Our final design involves a PI-based core thermal controller and an outer control loop to decide process migrations. This policy avoids all thermal emergencies and yields an average of 2.6X speedup over the baseline across all workloads.

1. Introduction

As power density has increased exponentially with Moore's Law, thermal cooling challenges have become a prominent and vexing aspect of computer systems design [3, 13]. While mechanical cooling solutions (heatsinks, fans, and so forth) remain the front-line mechanisms for dealing with the thermal wall, these approaches are costly, unwieldy, and do not represent a complete solution to the problem.

Thermal-aware techniques at the architecture level have gained momentum over the past five years as a means for optimizing processor performance while also abiding by rapidly-worsening thermal constraints [33]. Thermal-aware architecture techniques are related to power-aware techniques [5, 19, 20, 21, 22, 24, 36, 37, 38] but are a distinct area because of thermal-aware design's concern both with local hotspot constraints as well as with aggregate thermal limits.

This paper begins by dividing the CMP thermal design space into a taxonomy of orthogonal design choices. This taxonomy allows us to systematically and quantitatively explore the thermal design space. In some parts of the space, we quantify the benefits of useful combinations of previously-proposed approaches. In other parts of the space, however, we propose novel thermal control techniques and quantify their value. For example, one of the key novelties of the paper lies in our use of formal control theory techniques to propose, design, and evaluate a multi-loop control mechanism that allows the operating system and the processor hardware to collaborate on a robust, stable, and effective thermal management policy. We know of no other architecture work exploiting multi-loop formal control.

The contributions of this paper are as follows:

- Distributed DVFS provides considerable performance improvement under thermal duress, on average improving throughput by 2.5X relative to our baseline. While the design complexity cost of multiple clock domains is considerable, we show that the performance potential is significant as well.
- When independent per-core DVFS controls are unavailable, we find that other options perform well. In particular, a thread migration policy without per-core DVFS can still improve performance by as much as 2X.
- These methods can be elegantly combined through our sensor-based migration policy involving multi-loop control. The operating system engages in coarse-grained control and migration, while the hardware level engages in a finer-grained level of formal control based on DVFS. We find that this method offers up to 2.6X improvements over baseline.

Overall, this paper offers insights on the combined leverage of DTM methods, on the value of distributed DVFS for thermal management, and on the robustness and feasibility of formal multi-loop control via OS-processor collaborations. Given the importance of thermal design in current and future processors, these contributions represent useful next steps for the field of thermal-aware architecture.

The remainder of this paper is structured as follows. Section 2 presents our motivation and approach on thermal control. Section 3 describes our simulation environment and experiment methodology to quantify the properties of these systems. Section 4 examines implementation issues for our formal control methods. Sections 5 through 7 show our experimental results. Section 8 discusses related work. Section 9 offers our conclusions.

2. Motivation and Design Options

Since temperature is largely dependent on power output over time, general power-reduction techniques are typically good first steps for temperature-aware design. In addition to aggregate heat production, however, there can be significant temperature variance across different regions of the die, and thus one also must worry about more localized hot spots at particular portions of the chip.

benchmark	category	steady-state temperature (º C)
gzip	SPECint	70
mcf	SPECint	59
parser	SPECint	67
twolf	SPECint	67
mesa	SPECfp	65
swim	SPECfp	62
lucas	SPECfp	63
sixtrack	SPECfp	71

(a) Temperatures of stable benchmarks.

benchmark	category	temperature range (º C)
bzip2	SPECint	67-72
ammp	SPECfp	58-64
facerec	SPECfp	65-71
fma3d	SPECfp	61-67

(b) Temperature ranges for benchmarks without a steady temperature.

Table 1: Measured processor temperatures on a Pentium M Banias notebook.

2.1 Application Thermal Variation

Our work in large part aims to quantify the benefits of combining heat *balancing* methods with heat reduction. We do this because thermal variations are so important, and this section demonstrates this importance using real-system measurements. We present measurements taken from real hardware to demonstrate the range of thermal characteristics across different benchmarks selected from the SPEC 2000 suite [16]. We perform measurements at room temperature on a notebook utilizing a Pentium M Banias 1.5 GHz processor and running Red Hat Linux 7.3 with its kernel upgraded to version 2.6.11. Using the Advanced Configuration and Power Interface (ACPI) we read the temperature off a single thermal diode at the edge of the processor [1]. Due to interface restrictions all measurements are rounded to the nearest degree Celsius.

We first compile all benchmarks with base settings using gcc version 2.96 for C programs and Intel Fortran Compiler version 9.0 for Fortran programs. Before running any benchmark we allow the computer to sit idle briefly and confirm that it has reached its idle temperature. Once we launch a benchmark run, we wait one minute, and then poll the processor temperature repeatedly. Most programs reach a relatively-stable steady-state temperature, and these per-program, steady-state temperatures are shown in Table 1 (a). Not all benchmarks gravitate towards a single-steady temperature, however, and Table 1 (b) lists the programs where temperatures continually rise and fall throughout execution. As shown, processor steady-state temperatures for such benchmarks can differ by as much as 12°.

Our real-system findings are consistent with simulation work by other sources. For example, `gzip` and `bzip2` are two of the hottest integer benchmarks [9] and `sixtrack` is one of the hottest floating point benchmarks [15, 29]. Also, `mcf` is by far the coolest due to its memory-bound execution. Both its overall IPC and temperature are relatively kept low when a limited L2 cache is provided [23], as in this case where the Banias processor provides only 1 MB.

Overall, these real-system measurements show first that applications have quite distinct thermal profiles, and second, that the time-varying nature of applications and workloads warrants truly dynamic approaches to thermal management. While we cannot physically measure the spatial thermal variations within a core, we can surmise that CMPs running multiprogrammed combinations of these applications will show spatial variations at the core level at least, and likely within the core as well.

	No migration		Counter-based migration		Sensor-based migration	
	Stop-go	DVFS	Stop-go	DVFS	Stop-go	DVFS
Global	Stop-go	Global DVFS	Stop-go + counter-based migration	Global DVFS + counter-based migration	Stop-go + sensor-based migration	Global DVFS + sensor-based migration
Distributed	Dist. stop-go	Dist. DVFS	Dist. stop-go + counter-based migration	Dist. DVFS + counter-based migration	Dist. stop-go + sensor-based migration	Dist. DVFS + sensor-based migration

Table 2: Thermal control taxonomy, forming 12 possible thermal management schemes.

2.2 Thermal Control Taxonomy

For controlling hot spots, one can either (a) reduce heat production, or (b) balance heat production. Under this reasoning, our work seeks to classify DTM schemes in a systematic manner so that we can characterize and quantify their design tradeoffs taken both individually and in combinations.

Our taxonomy is depicted in Table 2. We regard our policy decisions as a set of orthogonal axes. One axis refers to the type of low-level control employed. Among the choices we explore are that of a stop-go policy (turn off a core or the whole chip when thermal management indicates the temperature should be reduced) and DVFS (apply voltage-frequency scaling to reduce temperature). Our second axis is that of deciding whether to use a global controller for all cores, or whether to use distributed per-core approaches. Per-core decisions may require more complex hardware, but in turn will let us respond more individually to the needs of different applications running on each core. Our final axis regards a process migration policy, which acts on a more coarse-grain time scale. Options here are to never migrate threads (the base case) or to migrate threads in response to either thermal-sensor readings, or counter-based thermal proxies. We explore these 12 options in different combinations in the sections that follow. Inevitably, there are always further axes one might consider. (For example, simultaneous multithreading and heterogeneous cores are two other axes which impact thermal issues). Nonetheless, we feel that this taxonomy helps guide us through many interesting and useful combinations of thermal design features for CMPs.

2.3 Stop-go vs. DVFS

One of the most basic forms of dynamic thermal management is known as global clock gating [5] or "stop-go". This involves freezing all dynamic operations and turning off clock signals to freeze progress until the thermal emergency is over. When dynamic operations are frozen, processor state including registers, branch predictor tables, and local caches are maintained, so much less dynamic power is wasted during the wait period. Thus stop-go is more like a suspend or sleep switch rather than an off-switch.

Our stop-go mechanism is a coarse-grain operation signaled by the processor and carried out by the operating system. Once a thermal sensor reaches the designated threshold, a thermal trap is signaled and processes are frozen for 30 milliseconds. After lowering the temperature a few degrees through stalling, the processor can resume. We choose this interval to be coarse-grain in part because it reflects the slow heating and cooling time constants of thermal variations (milliseconds [12]), and in part because it leads to a relatively simpler implementation.

The DVFS policy involves more of a continuous adaptive scheme. By enabling a continuous range of frequency and voltage combinations we can predictively use these to reduce power consumption. Thus our DVFS policy is not as simple as the stop-go mechanism, but we leverage on past control-theoretic work to systematically obtain suitable parameters. We use a setpoint slightly below the thermal threshold and use a PI controller to adaptively control the frequency and voltage levels to aim towards this target threshold. Our DVFS mechanism has a higher design cost than the rudimentary stop-go mechanism due to the complexity of implementing a flexible phase-lock loop (PLL) and voltage scaling capabilities.

2.4 Distributed Policies vs. Global Control

While our first axis of classification in Table 2 focuses on the decision of stop-go vs DVFS policies, our second axis designates the scale on which these policies are applied. One possibility ("global") is to implement a stop-go or DVFS policy regarding the entire chip as a single unit. This has been the method used primarily in the first generation of commercial multicore processors, due to its reduced design complexity. In the case of DVFS, this avoids communication difficulties that would arise with multiple clock domains. Furthermore, if all cores are likely to heat at the same rate, a policy which cools all cores in sync could be sufficient.

While global control policies work well for workloads that heat the chip uniformly, our real-system measurements in the following section show that this uniformity is relatively unlikely. "Performance asymmetry" is the characteristic of workloads to show very different performance characteristics depending on the choice of applications, and it is a clear characteristic of emerging multicore applications [2]. If a global policy is used, a single hotspot on *one* of the cores could result to unnecessary stalling or slowdown on *all* cores. The more cores on the chip, the more potential performance is lost due to the single hotspot. Distributed policies, such as "Dist. DVFS" and "Dist. stop-go" as labeled in Table 2, instead allow each core to independently handle its own thermal management to a good extent. This paper shows that for thermal purposes, choosing a distributed policy may be well worth the necessary added design complexity.

2.5 OS-based Migration Controllers

The final axis we consider regards the migration policy. Migration can help balance heat production across all cores. All the previously-mentioned policies—and combinations of them—can still have remaining thermal imbalances which can be further remedied through migration. Consider for example a common case: in a 2-core system managed by a DVFS policy, the integer register file could be the limiting hotspot on one core, while the floating-point register file might be the limiter on the other core. For example, we see this case when running the `gzip-twolf-ammp-lucas` workload in our tests. Judiciously migrating threads can allow the system to achieve better performance than DVFS-based methods alone.

We consider migration policies managed by the operating system. Timer interrupts from a typical OS happen on the order of a millisecond apart, and this is actually more than enough to get sufficient potential from migration. Particularly in our best cases of distributed DVFS combined with migrations, the hotspot "drift" is much slower than a typical

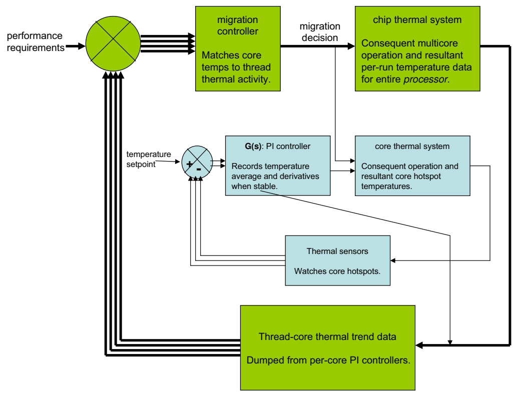

performance requirements
migration controller
Matches core temps to thread thermal activity.
migration decision
chip thermal system
Consequent multicore operation and resultant per-run temperature data for entire *processor*.
temperature setpoint
+
-
G(s): PI controller
Records temperature average and derivatives when stable.
core thermal system
Consequent operation and resultant core hotspot temperatures.
Thermal sensors
Watches core hotspots.
Thread-core thermal trend data
Dumped from per-core PI controllers.

Figure 1: Feedback control system involving inner loop for local control and outer loop for coarse-grain migration decisions. The sets of four thick lines represent sets of data from each core.

temperature gradient in the more thermally-chaotic stop-go policy. Given this time scale, when migration is used on top of control-theoretic DVFS, we can model our overall system as a two-loop structure as shown in Figure 1. The inner loop is the DVFS policy, while the outer loop is the migration policy. The migration policy depends to a great extent on data gathered by the PI controller in the inner loop.

We study two migration mechanisms in this paper. The first one is based on performance counters used to determine the resource intensities of various threads. We draw some of these ideas from [10] and [29], which describe concepts of mixing complementary resources based on profiled information. Our second mechanism is known as sensor-based migration. The purpose of this mechanism is to avoid reliance on performance counter proxy data which may be too abstract at times. Although both migration mechanisms, as well as all DTM policies in our study, rely on thermal sensors to make proper decisions at the correct times, the difference between the counter-based and sensor-based migration policies is that the latter uses sensors over time to track thermal properties of all processes. An elegant property of the sensor-based approach is that it depends directly on data gathered from the inner control loop as depicted in Figure 1.

3. Experimental Methodology

This section details our architectural, power estimation, and temperature modeling infrastructure. Figure 2 shows the overall flow. This section describes each of the illustrated levels of modeling, our modifications for thermal control, our test benchmarks, and metrics used.

3.1 Turandot and PowerTimer Processor Model

Using Turandot [27] we model a 4-core processor as detailed in Table 3. Most internal core parameters are similar to those used in [10] and [23], although for example, we use a larger 4 MB L2 cache.

PowerTimer [4] is a parameterizable power estimation tool which operates in conjunction with Turandot. Its hierarchical power models are derived through empirical circuit-level simulations and calibrated according to technology parameters. Component power across simulation intervals is then calculated by scaling according to the counts of various ar-

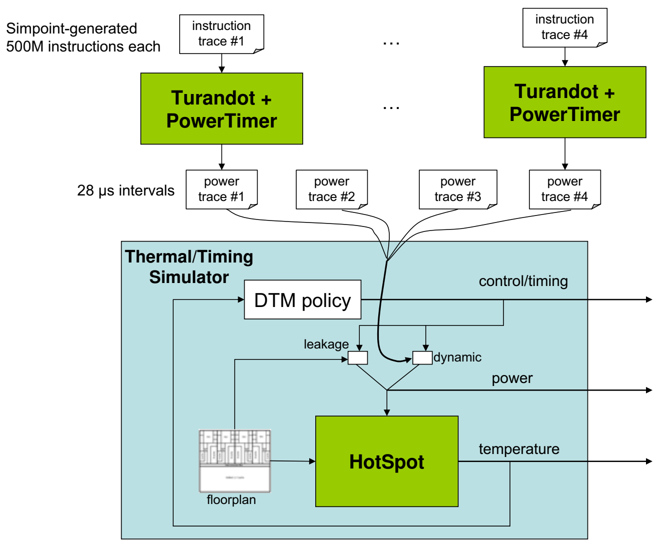


Figure 2: Power trace-based simulation method involving interdependence between Turandot, PowerTimer, leakage modeling, HotSpot, and thermal management policies.

Global Design Parameters	
Process Technology	90 *nm*
Supply Voltage	1.0 V
Clock Rate	3.6 GHz
Organization	4-core + shared L2 cache
Core Configuration	
Reservation Stations	Mem/Int queue (2x20), FP queue (2x5)
Functional Units	2 FXU, 2 FPU, 2 LSU, 1 BXU
Physical Registers	120 GPR, 108 FPR, 90 SPR
Branch Predictor	16K-entry bimodal, 16K-entry gshare, 16K-entry selector
Memory Hierarchy	
L1 Dcache	32 KB, 2-way, 128 byte blocks, 1-cycle latency
L1 Icache	64 KB, 2-way, 128 byte blocks, 1-cycle latency
L2 cache	4 MB, 4-way LRU, 128 byte blocks, 9-cycle latency
Main Memory	100-cycle latency
DVFS Parameters	
Transition penalty	10 μs
Minimum freq scale	20% (720 MHz)
Minimum transition	2% of range
Migration Parameters	
Migration penalty	100 μs

Table 3: Design parameters for modeled CPU and its four cores.

chitectural events. As shown in Figure 2, we use Turandot and PowerTimer to generate power *traces* to be used as inputs to our thermal simulator. Using SimPoint [31], we simulate representative traces of 500 million instructions from each program. These produce long (hundreds of milliseconds) output traces of power behavior containing data samples every 100,000 cycles (28 μs).

Leakage power is becoming a significant component of total power, especially with more aggressively-scaled technologies. We cannot rely on PowerTimer, however, for leakage values since these numbers are dependent on temperature, whose calculation comes later in our toolflow. Therefore, we describe our leakage power modeling approach below, as part of the thermal/timing approach.

3.2 HotSpot Thermal Model

As a component of our thermal/timing simulator, we use HotSpot version 2.0 [18, 33] which uses parameters that have been calibrated against a real chip as well as a power-trace-driven simulation capability. HotSpot calculates temperatures by modeling physical traits in a thermal system using a method analogous to calculating voltages in a circuit made up of resistors and capacitors. Required input to our thermal simulator includes a floorplan designating the locations and adjacencies of various processor components. The model also includes the heatsink, fan (convection), and thermal interface material.

We use a floorplan similar to that used in [23], except we have extended our layout for 4 cores and reduced the core size accordingly. Each of these cores has its various components necessary for an out-of-order pipeline and the four cores are connected through a shared L2 cache and interconnect.

HotSpot supports calculating transient temperatures as well as estimating steady-state temperatures. A number of past works [9, 11, 14, 33] have focused on steady-state. One advantage of steady-state temperatures is the ability to estimate long term temperatures from only a short simulation interval. Our experiments in adaptive control, however, require our simulator to know how temperatures change across time. Hence we focus on transient temperatures.

3.3 Thermal/Timing Simulator for DTM

As depicted in Figure 2, our thermal/timing simulator tests our thermal management policies in order to collect timing, power, and temperature data for any of our multiprogrammed workloads. The thermal/timing simulator uses power traces as inputs to its thermal control simulations. The simulator uses these recorded power values, controls the rate of progression through the trace, and scales power values in response to thermal control decisions. With DVFS, for example, it adjusts the time and energy calculations for that core (or for the whole chip) to account for the new voltage/frequency setting. Because DVFS dynamically changes the length of a cycle, and because in some of our methods each core may be operating with a different cycle time, the thermal/timing simulator framework tracks progress in terms of "absolute time" rather than cycles.

When a power trace for a particular benchmark is completed before the end of the simulation, that trace is restarted at the beginning and this process is continued until total of 0.5 seconds of silicon time has elapsed. For calculating leakage dynamically, we use the temperatures reported by HotSpot as input to a leakage model based on an empirical equation from [17].

In order to model shared structures such as the L2 cache with this trace-based method, the single-threaded simulations with Turandot actually are capacity-limited to use only one quarter of the L2 cache, while retaining pessimistic power costs of the full-size cache. This likely overestimates cache power, but the cache is never a hotspot. Another pessimistic approximation result of our methods is that for a DVFS mechanism. A memory-bound application can exploit CPU-memory slack [37], which our traces will not show. Thus, it is likely to gain more energy savings from DVFS compared to performance loss than a CPU bound application.

workload name	benchmarks	properties (integer / floating point)
workload1	gcc, gzip, mcf, vpr	int, int, int, int
workload2	crafty, eon, parser, perlbmk	int, int, int, int
workload3	bzip2, gzip, twolf, swim	int, int, int, fp
workload4	crafty, perlbmk, vpr, mgrid	int, int, int, fp
workload5	gcc, parser, applu, mesa	int, int, fp, fp
workload6	bzip2, eon, art, facerec	int, int, fp, fp
workload7	gzip, twolf, ammp, lucas	int, int, fp, fp
workload8	parser, vpr, fma3d, sixtrack	int, int, fp, fp
workload9	gcc, applu, mgrid, swim	int, fp, fp, fp
workload10	mcf, ammp, art, mesa	int, fp, fp, fp
workload11	ammp, facerec, fma3d, swim	fp, fp, fp, fp
workload12	art, lucas, mgrid, sixtrack	fp, fp, fp, fp

Table 4: Four-process workloads of interest and respective mix types of SPEC benchmarks.

Overall, the benefits of this coarse-grain power trace method are (i) that it simulates dynamic thermal variations and (ii) that it allows us to apply our control methods on the long time scales appropriate for observing temperature changes.

3.4 Workloads

Our simulated workloads are formed by selecting among 22 benchmarks including 11 SPECint benchmarks and 11 SPECfp benchmarks to mix into designated four-process workloads as shown in Table 4. The benchmark type is especially relevant to our study on overheating specific resources. Integer benchmarks are most likely to have their prime hotspot in the integer register file unit and its associated logic while floating point benchmarks are likely to stress the floating point register unit. Thus, we list the corresponding benchmark suite categories for all elements of each workload.

The type of benchmark alone—whether integer or floating point—does not completely categorize its resource intensities. Integer benchmarks in the SPEC suite have varying degrees of IPC. Furthermore, all floating point benchmarks make use of integer registers to some extent. It is thus possible that some floating point benchmarks even have higher integer register intensities than various integer benchmarks. Nonetheless the general categorization is a helpful guide in understanding how such workloads might behave subject to varying forms of thermal control.

3.5 Metrics

Our goal in these thermal control applications is to maximize performance subject to a fixed temperature constraint, in our case not allowing any part of the chip to go above 84.2° C. Therefore one of the most natural performance metrics is the raw instruction throughput for each workload (Billions of instructions per second, or BIPS).

While BIPS is a good basic performance metric, it can, however, be difficult to interpret in multiprogrammed workloads. This is because fairly running a low-IPC application can lead to worse-appearing performance than unfairly skewing execution toward the high-IPC applications in the workload. For this reason, we also provide results on each run's achieved percentage of "duty cycle". Duty cycle describes the ratio of time that useful work is being done, relative to the total time including work period and the rest (stop) period. For example, if a processor is in a globally-stalled mode four times as often as it runs in active mode, it has a duty cycle of 20%.

While duty cycle is very straightforward for stop-go situations, we also adapt it to DVFS as well. Although the processor may attempt to do useful work at each cycle, the varying frequencies in DVFS approaches change the effective duty cycle. For this reason, we use an adjusted duty cycle metric as follows. When summing up the total duty cycle, we scale the contributions accordingly by the dynamic frequency. For example, if all cores run at 30% of maximum speed for an entire execution this amounts to a duty cycle of 30%. If all cores run half the time at 30% speed and the other half of the time at 40%, this results in a duty cycle of 35%. This adjusted duty cycle is a good indicator of the ratio of the total work done relative to the total possible work that could be done if all cores were run at their maximum clock frequency not subject to any thermal constraint. Under this reasoning, overhead delays (such as that for adjusting the PLL under a DVFS policy) or for the context switch penalty under a migration policy are not counted as useful total work and thus also lower the adjusted duty cycle.

4. Applying Formal Control to Thermal DVFS

The DVFS portion of our thermal-management mechanisms use a control theoretic approach to determine appropriate voltage and frequency settings. Here we present some background information required to understand this approach, before introducing the other policies and our comparative results.

4.1 Background: Closed-loop DVFS Control

When designing our DVFS controller we apply closed-loop control theory. Formal feedback control has recently found numerous applications in architecture and systems [33, 34, 36, 37, 38]. Our work is novel, however, in composing together these formal control methods along with other control techniques in a multi-loop system.

Closed-loop control is a robust means to control complex systems so that the controlled value rapidly converges to the desired target output value. In our case, the measured variables are the thermal sensor values at various hotspots, and the output actuator is the mechanism to scale the voltage and frequency. This control loop is represented in the inner loop of Figure 1. Some of the arrows in a typical closed-loop diagram have been drawn as several arrows in Figure1. This reflects that multiple temperatures are fed into a single PI controller. Since an individual controller governs an entire core or processor, it typically selects the hottest of the input temperatures.

The standard PI controller equation, written in its Laplace form, is as follows.

$$G(s) = K_p + \frac{K_i}{s}$$

As shown, the two components are the proportional and integral terms. The proportional component defined by K_p reflects the basic gain of the controller responding to error, while the integral term containing K_i is there to compensate for any offsets and reduce the settling time. (A proportional-integral-derivative (PID) controller is another option, but we found that the derivative term has little benefit for this type of thermal control.)

MATLAB tests similar to [32] allow us to determine settling time and stability for typical thermal fluctuations. We use constants of $K_p = 0.0107$ and $K_i = 248.5$ in all of our tests. Owing to the robustness of PI systems and the inherent stability of the thermal system under study, these constants can actually deviate significantly while still achieving the intended goals. In fact, our proportional constant is set two orders of magnitude smaller than that used in [32], in order to maintain control with smoother transitions. Another issue is that of the system's sensor delay. Fortunately, the primary delay (thermal sensors) is quite small [8] compared to the time scales on which temperature varies.

A benefit of formal feedback control is the ability to prove stability. We use a root locus plot with the stability criterion that all the poles (the frequencies at which the characteristic function blows up to infinity) must lie to the left of the y-axis in the Laplace space. We verified this for our controller using MATLAB.

4.2 Thermal Control Mechanism for DVFS

The mathematical description given above proposes a controller that can be applied continuously, not limited by physical constraints. In our actual experiment this controller must be given appropriate limits and be discretized in time. In order to convert the Laplace transform into its corresponding discrete-time z-transform, we use the MATLAB function `c2d`, specifying a time interval of 28 μs to match the frequency of our thermal measurements. We then arrange the transform to directly specify our discrete online equation:

$$u[n] = u[n-1] - 0.0107e[n] + 0.003796e[n-1]$$

The error function $e[n]$ is simply the difference between the measured temperature and the target temperature. The target temperature is just below the thermal threshold. Through this, the system maximizes performance subject to the allowable temperature constraints. Convenient aspects of this controller are that despite involving an integral term, it is relatively simple to implement in hardware as it depends only on the previous controller output, previous error value, and current error value.

On a real system, PI controllers are subject to certain limits which stray from a purely linear design. One of the most basic limits is that of clipping on the output. The output, which is the specified frequency scaling factor, cannot extend to infinity since the cores cannot run beyond their maximum frequency as limited by the speed of transistor gates. When a core or processor is not in thermal danger but rather acting in a cool period, the controller will output its maximum value which reports a frequency scaling factor of 1.0. On the lower end, we restrict the minimum frequency scaling factor to 0.2. In the discrete model described in the above equation, this can be implemented fairly simply in hardware.

Another non-ideal characteristic of real PI controllers is that of integral windup, which describes when the integrator component continually integrates only because the input error remains unmitigated for an extended period due to physical limits. For instance, when a core is above its target temperature, the controller will try to cool it by lowering the frequency. However, chip components cannot cool any faster than the physical limitations allow. If integral windup occurs, when the condition is finally satisfied it can take a long time for the controller to "wind down". Fortunately, our discrete implementation with clipping quickly takes care of this. The simple discrete implementation in the above equation combined with clipping prevents a hidden integral component from building up.

Finally, in real systems, voltage changes are not instantaneous. A penalty of 10 μs is assumed for each frequency and voltage change.

5. Exploring Stop-go and DVFS in both Global and Distributed Policies

This section covers a portion of the policy combinations in our spectrum. In particular we examine issues of using stop-go and DVFS, and we also consider whether to apply each mechanism in a local or distributed fashion. Section 6 then explores migration policies largely with the intention of comparing to the original policy combinations in this section.

5.1 Stop-Go Policy Implementations

Compared to the formal control approaches used for DVFS management as described in the preceding section, our stop-go mechanism is quite simple. Each core is run at full blast as long as it does not exceed a particular thermal trippoint. Thermal sensors at the two register file units on each core sense the hotspot temperatures. When one is found to be just below the thermal threshold of 84.2° C, a thermal interrupt is issued. The core which caused this interrupt is then stalled for 30 ms. At this point, the hotspot will have cooled below the threshold and the core can continue running.

5.2 Distributed versus Global Policy Implementations

In the distributed policies, stop-go and DVFS techniques are applied to individual cores. Each DVFS controller takes in at least two inputs since it watches two hotspots, but the mathematical implementation goes by whichever sensor reports to be hotter. In the distributed cases, each core operates independently, without any coordination with other cores. For the cases of global stop-go and global DVFS, a single decision is made for all the cores on the chip. Thus, there is effectively only a single PI controller which calculates based on the hottest of all sensors across all cores.

5.3 Results

Results for all 12 workloads are shown in Figure 3. We report results relative to a baseline policy of distributed stop-go. Global stop-go has the worst performance of the 12. Its duty cycle is less than 20% and for this reason, we focus our attention on the other 11 possibilities for the remainder of the paper. Although our baseline characteristics are heavily dependent on our processor configuration and cooling system model, we have examined other scenarios. For example, we found that raising the temperature threshold to 100° C increased the duty cycles of these results and others presented in our paper to raise by 10 to 15%. Nonetheless, the relative performance tradeoffs remain as presented.

We present instruction throughput for the three non-migratory configurations—global stop-go, synchronous DVFS, and distributed DVFS—all normalized to the distributed stop-go results. As seen in the graph, the distributed DVFS policy does the best overall, on average more than double the instruction throughput of the distributed stop-go policy and four times that of global stop-go. The largest gain is due to voltage scaling. In addition, however, the distributed DVFS policy still gives significantly better throughput than the global. Tabulated results showing the average throughput and duty cycle for these policies are given in Table 5.

The overall improvement by distributed DVFS over a distributed stop-go policy is a performance improvement of 2.5X. The duty cycle numbers also reflect these improvements. In particular the active time reported for distributed stop-go is about 30% while distributed DVFS achieves more than 80%. We also have performed other experiments confirming the validity of our duty cycle metric. We ran simula-

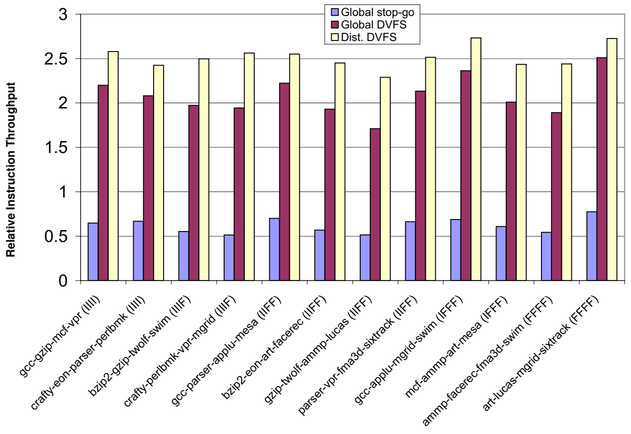


Figure 3: Normalized instruction throughput of all workloads relative to baseline distributed stop-go policy.

	BIPS	duty cycle	relative throughput
Stop-go	2.79	19.77%	0.62
Dist. stop-go	4.53	32.57%	1.00
Global DVFS	9.36	66.49%	2.07
Dist. DVFS	11.36	81.02%	2.51

Table 5: Average instruction throughput, effective duty cycle, and performance relative to dist. stop-go across all workloads for various policies.

tions with unrestricted maximum temperatures, and found that the proportion of the achieved BIPS relative to the non-controlled case was accurately predicted by the measured duty cycle.

The benefits seen here are consistent with past work which has shown much benefit from DVFS policies as opposed to primitive throttling policies. Likewise, allowing multiple voltage levels is greatly beneficial for power [19] and hence thermal control. The drawback of these design choices is of course mainly in design complexity. As our results show, for high-performance processors, the design costs of the distributed DVFS are nicely rewarded by the clear benefits of improved performance.

6. Migration Policies for Thermal Control

The third axis in our spectrum of thermal control options is determined by whether or not to use migration and the choice of migration mechanism used. We explore migration mechanisms implemented via OS control for two main reasons. First, benefits from migration policies happen on a relatively long time scale if migration is implemented on top of quicker policies such as stop-go and DVFS. Second, process control and context switches are traditionally something for which the operating system has final jurisdiction. Thus our migration mechanisms are called upon no more than once every 10 milliseconds, which is the typical timer interrupt setting for a Linux kernel. Both of our migration mechanisms, counter-based and sensor-based, are affected by the DVFS or other policies previously explored and thus a feedback relation exists when both DVFS and a migration policy are implemented. In this relation, the operating system needs to keep track of timing data for all processes. Although not explored in our experiments which are restricted to four-program workloads, in any system there can easily be a greater number of processes than cores.

When the OS decides to migrate threads for the purpose of thermal control, the relevant tracking information is flushed and stored and in our simulations, each core involved takes a penalty of 100 μs. Once this is completed, the overriding thermal policy is the primary mechanism of thermal protection, until 10 milliseconds have passed and the involved threads become eligible for migration again. The actual decision algorithms for which threads to migrate are described in more detail below.

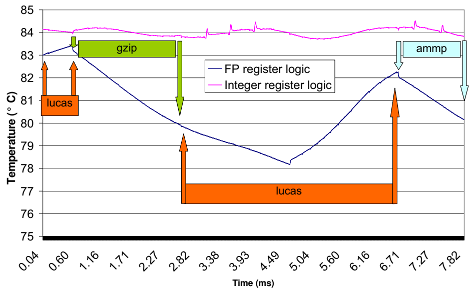
```
(1) remaining_processes = processes[];
(2) sort(cores[], most_hotspot_imbalance)
        where hotspot_imbalance =
                critical_hotspot.temperature - secondary_hotspot.temperature for core[i];
(3) foreach (cores[1..n]) { // find best matchings
        matching_process = least_intense(remaining_processes, cores[i].critical_hotspot);
        cores[i].assigned_process = matching_process;
        remaining_processes -= matching_process;
    }
(4) foreach (cores[1..n]) { // migrate if beneficial
        if (cores[i].current_process != cores[i].assigned_process)
                migrate(cores[i].assigned_process, cores[i]);
    }
```

Figure 4: Pseudocode algorithm for migration decisions in counter-based migration.

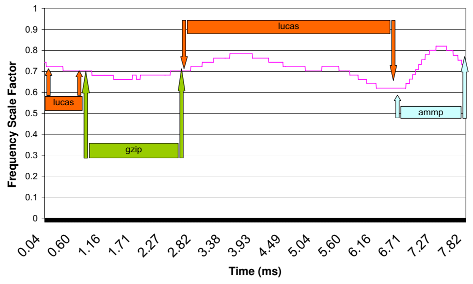
6.1 Counter-based Migration: Method

We examine first a performance counter-based migration policy, which is the simpler of the two discussed. The counters are used here to estimate the thermal intensity of a particular resource. For example, for the integer register file, performance counters are used to keep track of the number of accesses for individual threads. The performance counter information used here includes cycle counts, the number of integer register file accesses, the number of floating point register accesses, and instructions executed. Much of the reasoning for our mechanism is borrowed from [29], which proposes mixing resource heat intensities with SMT. With no frequency scaling, we are interested in the ratio of register file accesses per cycle. When frequency scaling is used, it becomes necessary to know the number of accesses per adjusted cycle instead.

Our overall algorithm is shown in pseudocode in Figure 4. Every thread's various performance counters are recorded throughout execution. This way the operating system is aware of the various resource intensities of all running programs. Migration decisions are actuated when the local thermal control of at least two individual cores signals that their critical hotspots have changed. If this happens more often than 10 milliseconds, extra requests are simply ignored since there is little reason to enact a thermal migration on such a short time scale. When it is time to test for eligible migrations, the cores with the most critical hotspot imbalance are considered first. Hotspot imbalance is defined as the difference in temperature between the core's critical hotspot and the temperature of the core's second hottest distinct hotspot. In order of most need for migration, the decision algorithm searches for a suitable candidate. Each decision is done by seeing which thread would be most able to reduce heating of the critical hotspot on each core. In some cases, the best candidate for a thread to migrate will be itself, in which case a migration is not done. A set of migrations can be as simple as a single swap, or as complex as a four-way rotation.



(a) Temperature of hotspots on first core.



(b) Time-dependent frequency control output across this same interval.

Figure 5: Temperatures and DVFS control across several migration intervals for a sample workload.

6.2 Counter-Based Migration: Results

Figure 5 shows the effects and intentions of several migrations on a single core. Initially, `lucas` is running on this core. As the floating-point register file heats up, `gzip` is chosen to migrate in. This allows the FP register file to cool off considerably. Although `lucas` does not immediately counteract the cooling, it eventually does raise the temperature again before being evicted by `ammp`. The critical hotspot which determines the DVFS speed is the integer register file throughout the entire run. The other hotspot tends to "drift" depending on the imbalance in the benchmarks used.

Further complications are involved with migration when all cores are allowed to operate at different frequencies. Since DVFS naturally affects performance, a core prone to a different scaling factor has effectively different heating patterns for its hotspots. Figure 5 (b) shows the corresponding frequency changes on the single core to result in temperatures shown in Figure 5 (a). To account for this, when DVFS is involved the scaling factor is recorded on runs and stored by the operating system. In this way, this combined policy takes feedback information from the inner loop DVFS control loop. This is then used to scale the power estimations from performance counters by a cubic relation.

Our overall results from this counter-based migration policy are shown in Table 6. When used in conjunction with a distributed local stop-go policy, counter-based migration provides a 2X performance improvement and sees about the same increase in duty cycle. This means that in cases where DVFS is not available, stop-go can be used for basic heat reduction, and migration provides better performance through heat balancing. In the next section, we discuss our alternative policy that does not depend on performance counters.

6.3 Sensor-based Migration: Method

The counter-based approach is appealing because it relies on easily-accessed hardware counters of microarchitectural activities that have intuitive meaning to hardware and software designers. Furthermore, their values can be directly attributed to threads and code. They are not, however, a direct representation of thermal behaviors. Instead, they are at best a proxy.

Here we explore instead a second migration method based directly on reading on-chip thermal sensors. With sensor-based policies, the mechanism is complicated significantly by external factors. Although vertical heat conduction typically matters more than lateral heat conduction [9, 11, 33], the lateral effects are not small enough to ignore. Our sensor-based mechanism seeks to know the slopes of temperature transitions, and these may depend on hotspots from a neighboring unit or core. For example, a certain thread will appear to have different temperature gradients when running on different cores due to different external factors, such as being located closer to the edge of the chip. Furthermore, if a DVFS or stop-go policy is applied, the trend sensing calculations must appropriately time-scale the measured temperature changes to account for this. Depending on which core and at what time measurements are taken, this could give different results. Because of these issues, our design requires recording the scaling factors (as seen by the PI controller) and using the average to scale the measured thermal trends appropriately.

Our algorithm for the sensor-based migration is more complex than the counter-based policy. Although our decision algorithm is almost the same as that presented in Figure 4, determining individual threads' hotspot intensities through thermal sensors is more complex than direct counter information. The apparent intensity on various cores for a single thread will appear different as each core has different thermal situations. For instance, a core next to the cache may have less thermal intensity due to the cache's relatively cool temperature. We therefore need to profile threads in a systematic manner so that relative temperature gradients can be used to estimate the thermal intensity of all possible thread-core combinations. The flow diagram in Figure 6 describes our steps to accomplish this. There is a grid maintained by the operating system so that the migration decision can be made to estimate a thread's hotspot behavior on a particular core. To estimate thread intensity, each core needs to be run and dynamically tested with at least two threads, and each thread needs to have recorded sensor data from running on at least one core.

A benefit from this approach is that much of the feedback information can be recorded in the PI hardware which does arithmetic operations with the temperatures on a time scale appropriate for recording the trends. As with counter-based migration, for the distributed DVFS case, each recorded temperature trend must be scaled down by a cubic relation according to the recorded frequency scaling factor.

	BIPS	duty cycle	relative throughput	speedup over non-migration
Stop-go, counter-based migration	5.34	37.93%	1.18	1.91
Dist. stop-go,counter-based migration	9.15	65.12%	2.02	2.02
Global DVFS, counter-based migration	9.88	70.05%	2.18	1.06
Dist. DVFS, counter-based migration	11.62	82.42%	2.57	1.02

Table 6: Average instruction throughput and duty cycle for performance counter-based migration policies.

6.4 Sensor-based Migration: Results

Tabulated results comparing the sensor-based migration policy with the non-migration and counter-based migration

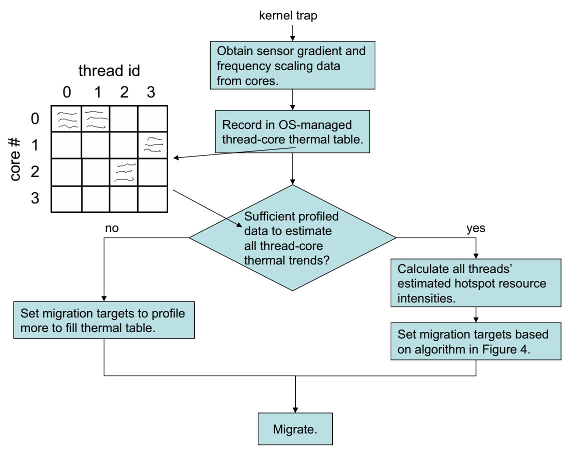

kernel trap
Obtain sensor gradient and frequency scaling data from cores.
Record in OS-managed thread-core thermal table.
thread id
0 1 2 3
core #
0 1 2 3
Sufficient profiled data to estimate all thread-core thermal trends?
no
yes
Set migration targets to profile more to fill thermal table.
Calculate all threads' estimated hotspot resource intensities.
Set migration targets based on algorithm in Figure 4.
Migrate.

Figure 6: Flow chart demonstrating the steps taken upon an OS interrupt to decide on sensor-based migrations.

	BIPS	duty cycle	relative throughput	speedup over non-migration	speedup over counter-based migration
Stop-go, sensor-based migration	5.43	38.64%	1.20	1.95	1.02
Dist. stop-go, sensor-based migration	9.27	66.61%	2.05	2.05	1.01
Global DVFS, sensor-based migration	9.63	68.37%	2.13	1.03	0.97
Dist. DVFS, sensor-based migration	11.70	82.64%	2.59	1.03	1.01

Table 7: Average instruction throughput and duty cycle for sensor-based migration policies.

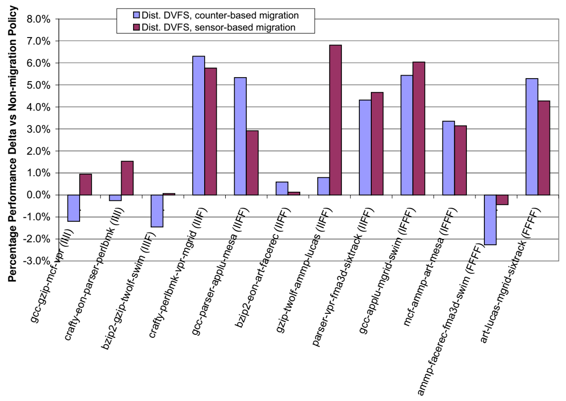


Figure 7: Individual gains/losses of various workloads due to either migration policy in conjunction with distributed DVFS (best-performing practical policy of the original four).

policies are compared in Table 7. We find that the sensor-based mechanism performs slightly better overall. For more detail, we have plotted the individual workload performances across migration policies in Figure 7.

Our overall results show that both migration policies are beneficial and feasible. Sensor-based migration mechanisms perform overall with a 2.1X speedup over the baseline stop-go policy and an overall speedup of 2.6X with proportionally the same increase in duty cycle when combined with distributed DVFS. The best performance comes when both mechanisms do not improve performance for every workload, and this can be explained in the fact that they are both

	No migration		Counter-based migration		Sensor-based migration	
	Stop-go	DVFS	Stop-go	DVFS	Stop-go	DVFS
Global	0.62X	2.1X	1.2X	2.2X	1.2X	2.1X
Distributed	baseline	2.5X	2X	2.6X	2.1X	2.6X

Table 8: Summary of all policy combinations and their respective multiplicative increases in instruction throughput relative to distributed stop-go.

doing approximation algorithms to best estimate migration decisions. Errors due to the algorithm assumptions can lead to decisions that not always optimal, but on average do give a significant performance benefit.

The options presented in this section and the previous section present designers with several viable choices for a total thermal management policy. In simpler designs such as global stop-go, migration makes up for much of the benefit that would be found in a system invoking DVFS. Furthermore, the migration controls are mostly OS-controlled and hence can be reconsidered and reprogrammed after chip production.

7. Summary and Recommendations

Instead of viewing some conventional mechanisms as competing alternatives, this paper has explored a combination of orthogonal methods to determine where the most gains are seen and which policies work best together. To summarize this here, we recall our table from Section 2, and present a similar organization except filled with the overall relative instruction throughput for all policy combinations in Table 8.

Our basic results fortify distributed DVFS as a strong foundation for thermal control, reflecting on average more than 2.5X increase in throughput over our base policy of distributed stop-go.

Both counter-based and thermal trend-based migration policies are able to significantly increase performance through hotspot balancing, respectively reporting 2X and 2.1X improvements of the baseline stop-go policy. When implementing migration on top of other policies we see diminishing returns but a net benefit on the most aggressive distributed DVFS policy with a 2.6X speedup over baseline.

Duty-cycle measurements offer a good view of how close we have come to full-speed execution. For example, while simple stop-go techniques result in duty cycles below 20%, the best multi-loop combination of migration and DVFS improves the duty cycle to an average over 82%. Given the threat of thermal emergencies, 100% duty cycle is not possible for these workloads, but values in excess of 80% are quite close.

8. Related Work

Since reducing power density has the effect of reducing temperature, temperature-aware approaches benefit much from the same techniques as in power-aware design. One key difference is that temperature-aware approaches seek not necessarily to reduce the average temperature but also focus on the thermal constraints of individual hot spots, as our work does. Other differences arise in the metrics that are most relevant. Our work uses many of these techniques demonstrated in prior studies on processor power, but applies these mechanisms directly to the problem of thermal control.

With temperature control as a key limitation to processor performance, many recent works in computer architecture focus on issues of thermal control [9, 10, 15, 18, 22, 23, 29, 33]. In particular, some of the more closely related works explore temperature-aware design issues in multithreaded architectures similar to ours. For example, our prior work [9] explored temperature issues in simultaneous multithreaded (SMT) and multicore designs and found common characteristics of thermal stress. We did not, however, delve into thermal control techniques to alleviate these problems. Ghiasi and Grunwald examine thermal properties of dual core designs in particular [11]. However, their work also focuses on steady-state temperatures for fixed configurations rather than dynamic control. Li et al. examine general issues of performance, power, and thermal characteristics of both multicore and SMT processors [23]. Although they do explore a number of thermal management policies, they view these as competing alternatives rather than taking a broad approach like ours that includes combinations of techniques. Li and Martinez attempt to explore methods to model performance and power efficiency for multicore processors subject to thermal constraints [21], but they do not delve into the various options of thermal management policies. Chaparro et al. have examined issues on designing clustered processors and the potential for temperature improvement through clustering methods [6, 7]. While covering many microarchitectural details for such multicore designs, this work does not focus on the thermal control policy and uses a routine core-hopping mechanism.

Related more to our study are some other works which focus more on the design of control policies for thermal management. For example, Shang et al. have proposed adaptive mechanisms for thermal management by focusing primarily on interconnect power control, but they use techniques of a primarily power-aware nature rather than focusing on mitigating localized hotspots [30]. Other prior work of ours has examined methods for thread-sensitive fetching that rely on performance counters to adaptively control temperature for SMT architectures [10]. Related to this is work by Hasan et al. that attempts to protect SMT processors against the threat of thermally constrained resources used for a denial-of-service attack [15]. This, however, protects against a very specific case of malicious attacks and does not generalize to adequate control of normal processes.

Powell et al. [29] describe techniques for thread assignment and migration and the intuitive nature of migrating computation appropriately to balance temperatures [29]. Their work uses performance counter-based information in a similar manner to our counter-based migration policy. Although they compare their technique directly to stop-go and DVFS policies, they do not consider the possibility of combining such techniques as we have done.

9. Conclusions

Our work here presents a framework and methodology for evaluating a variety of thermal control options. We have performed real hardware measurements to quantitatively demonstrate aspects of the challenges in adaptive thermal control. Through simulation of architectural and thermal models we have examined a range of thermal management options. We have characterized all twelve policy combinations both in terms of instruction throughput and effective duty cycle.

Our best performing thermal control combination includes both control-theoretic distributed DVFS and a sensor-based migration policy. This design represents an elegant two-loop system allowing the migration policy to utilize feedback information from the core controllers. It also demonstrates the value of hardware-software collaboration on the thermal problem. Hardware performs fine-grained adjustments and ensures that thermal emergencies are avoided, while software uses migration to perform heat balancing and seeks to optimize the workload's performance.

Taken together, these studies create an overall picture regarding the issues and benefits of various thermal control policies and their combinations. DVFS mechanisms require added on-chip flexibility in the PLL and voltage modulation, but we show them to be robust and effective. Our migration schemes are fairly lightweight in implementation; they are designed to operate either with hardware performance counters (available on essentially all current processors) or feedback-involved core control.

Although we have examined our spectrum of thermal control policies as a mixture among three axes, these are not the only possible dimensions. SMT and asymmetric cores are two possible extensions.

Given the increasing challenges of thermal design in current and future processors, creative combinations of effective DTM policies are likely to be the only way to truly gain leverage on the problem. With that in mind, this paper has offered a taxonomy of DTM techniques and has used the taxonomy to propose and explore interesting and novel DTM methods spanning from OS software down to control-theoretic hardware.

Acknowledgements

We thank Gilberto Contreras, Qiang Wu, and the anonymous reviewers for their helpful comments. We would also like to thank Eugene Otto for his help in obtaining hardware thermal measurements. This work is supported in part by grants from NSF, Intel, SRC, and the GSRC/C2S2 joint microarchitecture thrust.

References

[1] ACPI - Advanced Configuration and Power Interface. www.acpi.info, 2005.

[2] S. Balakrishnan et al. The Impact of Performance Asymmetry in Emerging Multicore Architectures. In *Proc. of the 32nd Intl. Symp. on Computer Architecture*, June 2005.

[3] S. Borkar. Design Challenges of Technology Scaling. *IEEE Micro*, pages 23–29, July/Aug. 1999.

[4] D. Brooks et al. Power-Aware Microarchitecture: Design and Modeling Challenges for Next-Generation Microprocessors. *IEEE Micro*, 20(6):26–44, Nov/Dec. 2000.

[5] D. Brooks and M. Martonosi. Dynamic Thermal Management For High-Performance Microprocessors. In *Proc. of the 7th Intl. Symp. on High-Performance Computer Architecture*, Jan. 2001.

[6] P. Chaparro et al. Distributing the Frontend for Temperature Reduction. In *Proc. of the 11th Intl. Symp. on High-Performance Computer Architecture*, Feb. 2005.

[7] P. Chaparro, J. González, and A. González. Thermal-Effective Clustered Microarchitectures. In *Proc. of the First Workshop on Temperature-Aware Computer Systems*, June 2004.

[8] J. Clabes et al. Design and Implementation of the POWER5^{TM}Microprocessor. In *Proc. of the 2004 Intl. Solid-State Circuits Conf.*, Feb. 2004.

[9] J. Donald and M. Martonosi. Temperature-Aware Design Issues for SMT and CMP Architectures. In *Proc. of the 5th Workshop on Complexity-Effective Design*, June 2004.

[10] J. Donald and M. Martonosi. Leveraging Simultaneous Multithreading for Adaptive Thermal Control. In *Proc. of the Second Workshop on Temperature-Aware Computer Systems*, June 2005.

[11] S. Ghiasi and D. Grunwald. Design Choices for Thermal Control in Dual-Core Processors. In *Proc. of the 5th Workshop on Complexity-Effective Design*, June 2004.

[12] A. González. Research Challenges on Temperature-Aware Computer Systems (panel). In *Second Workshop on Temperature-Aware Computer Systems*. Intel Corp., June 2005.

[13] S. Gunther et al. Managing the Impact of Increasing Microprocessor Power Consumption. *Intel Technology Journal*, Q1(5), June 2001.

[14] Y. Han, I. Koren, and C. A. Moritz. Temperature Aware Floorplanning. In *Proc. of the Second Workshop on Temperature-Aware Computer Systems*, June 2005.

[15] J. Hasan et al. Heat Stroke: Power-Density-Based Denial of Service in SMT. In *Proc. of the 11th Intl. Symp. on High-Performance Computer Architecture*, Feb. 2005.

[16] J. L. Henning. SPEC CPU2000: Measuring CPU Performance in the New Millennium. *IEEE Computer*, 33(7):28–35, July 2000.

[17] S. Heo, K. Barr, and K. Asanovic. Reducing Power Density through Activity Migration. In *Proc. of the Intl. Symp. on Low Power Electronics and Design*, Aug. 2003.

[18] W. Huang et al. Compact Thermal Modeling for Temperature-Aware Design. In *Proc. of the 41st Design Automation Conf.*, June 2004.

[19] A. Iyer and D. Marculescu. Power Efficiency of Multiple Clock Multiple Voltage Cores. In *Proc. of the IEEE/ACM Conf. on Computer-Aided Design*, Nov. 2002.

[20] S. Kaxiras et al. Comparing Power Consumption of an SMT and a CMP DSP for Mobile Phone Workloads. In *Proc. of the 2001 Intl. Conf. on Compilers, Architecture, and Synthesis for Embedded Systems*, Nov. 2001.

[21] J. Li and J. Martinez. Power-Performance Implications of Thread-level Parallelism on Chip Multiprocessors. In *$P{=}AC^2$: IBM Conf. on Architectures, Compilers, and Circuits*, Sept. 2005.

[22] Y. Li et al. Understanding the Energy Efficiency of Simultaneous Multithreading. In *Proc. of the Intl. Symp. on Low Power Electronics and Design*, Aug. 2004.

[23] Y. Li et al. Performance, Energy, and Thermal Considerations for SMT and CMP Architectures. In *Proc. of the 11th Intl. Symp. on High-Performance Computer Architecture*, Feb. 2005.

[24] Z. Lu et al. Reducing Multimedia Decode Power Using Feedback Control. In *Proc. of the Intl. Conf. on Computer Design*, Oct. 2003.

[25] J. McGregor. x86 Power and Thermal Management. *Microprocessor Report*, Dec. 2004.

[26] A. Merkel, A. Weissel, and F. Bellosa. Event-Driven Thermal Management in SMP Systems. In *Proc. of the Second Workshop on Temperature-Aware Computer Systems*, June 2005.

[27] M. Moudgill, J.-D. Wellman, and J. H. Moreno. Environment for PowerPC Microarchitecture Exploration. *IEEE Micro*, 19(3):15–25, May/June 1999.

[28] C. D. Patel. Smart Chip, System and Data Center: Dynamic Provisioning of Power and Cooling from Chip Core to the Cooling Tower (keynote). In *Second Workshop on Temperature-Aware Computer Systems*. HP Labs, June 2005.

[29] M. D. Powell, M. Gomaa, and T. N. Vijaykumar. Heat-and-Run: Leveraging SMT and CMP to Manage Power Density Through the Operating System. In *Proc. of the 11th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, Oct. 2004.

[30] L. Shang et al. Thermal Modeling, Characterization and Management of On-Chip Networks. In *Proc. of the 37th Intl. Symp. on Microarchitecture*, Dec. 2004.

[31] T. Sherwood et al. Automatically Characterizing Large Scale Program Behavior. In *Proc. of the 10th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, Oct. 2002.

[32] K. Skadron, T. Abdelzaher, and M. R. Stan. Control-Theoretic Techniques and Thermal-RC Modeling for Accurate and Localized Dynamic Thermal Management. In *Proc. of the 8th Intl. Symp. on High-Performance Computer Architecture*, Feb. 2002.

[33] K. Skadron et al. Temperature-Aware Microarchitecture. In *Proc. of the 30th Intl. Symp. on Computer Architecture*, Apr. 2003.

[34] D. C. Steere et al. A Feedback-driven Proportion Allocator for Real-rate Scheduling. In *Proc. of the Third Symp. on Operating System Design and Implementation*, Feb. 1999.

[35] A. Weissel and F. Bellosa. Dynamic Thermal Management for Distributed Systems. In *Proc. of the First Workshop on Temperature-Aware Computer Systems*, June 2004.

[36] Q. Wu et al. Formal Online Methods for Voltage/Frequency Control in Multiple Clock Domain Microprocessors. In *Proc. of the 11th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, Oct. 2004.

[37] Q. Wu et al. A Dynamic Compilation Framework for Controlling Microprocessor Energy and Performance. In *Proc. of the 38th Intl. Symp. on High-Performance Computer Architecture*, Feb. 2005.

[38] Q. Wu et al. Voltage and Frequency Control with Adaptive Reaction Time in Multiple-Clock-Domain Processors. In *Proc. of the 11th Intl. Symp. on High-Performance Computer Architecture*, Nov. 2005.