Dimetrodon: Processor-level Preventive Thermal Management via Idle Cycle Injection*

Peter Bailis*, Vijay Janapa Reddi*‡, Sanjay Gandhi, David Brooks*, and Margo Seltzer*
*Harvard University, ‡AMD Research

Abstract

Processor-level dynamic thermal management techniques have long targeted worst-case thermal margins. We examine the thermal-performance trade-offs in average-case, preventive thermal management by actively degrading application performance to achieve long-term thermal control. We propose Dimetrodon, the use of idle cycle injection, a flexible, per-thread technique, as a preventive thermal management mechanism and demonstrate its efficiency compared to hardware techniques in a commodity operating system on real hardware under throughput and latency-sensitive real-world workloads. Compared to hardware techniques that also lack flexibility, Dimetrodon achieves favorable trade-offs for temperature reductions up to 30% due to rapid heat dissipation during short idle intervals.

Categories and Subject Descriptors

C.0 [**Computer Systems Organization**]:
General—*Hardware/Software interfaces and System architectures*

General Terms

Performance, Reliability

Keywords

Thermal management, Average-case design, Idle injection

1. INTRODUCTION

Thermal management is increasingly important across several domains. Increased operating temperatures can result in exponentially reduced mean-time-to-failure (MTTF) values [25], while processor leakage power increases exponentially with temperature [23, 24]. Power costs have begun to eclipse the cost of physical hardware [6], and the power required to cool a processor is nearly equivalent to the electricity required to power it [17]. Up to 80% of data center construction cost is attributable to power and cooling infrastructure [5], and chiller power, a historically dominant data center energy overhead, scales quadratically with the amount of heat extracted [18], . Processor cooling is also a significant problem for mobile devices as thermal conditions can affect user experience through both heat dissipation and potentially intrusive cooling solutions (e.g., noisy fans) [22].

Traditional dynamic thermal management (DTM) techniques focus on reducing worst-case thermal *emergencies* but do not contribute to lowering overall temperatures While these techniques have many benefits such as increased reliability [23] and decreased chip packaging requirements [24], they are not designed to operate under normal thermal conditions. In practice, these DTM mechanisms are not activated except under extreme thermal conditions that are likely caused by some other catastrophic failure (e.g., cooling system problems).

This work focuses on reducing average-case processor operating temperatures, exploring the trade-offs between application performance and long-term thermal behavior through *preventive* thermal management. Our focus is on thread-level thermal management; once a thread is executing on a particular core, we want to control its thermal impact. Multicore-aware strategies, such as core migration [11] as well as more complex thermal-aware thread schedule placement [9], are orthogonal to the problem we consider here but are potentially complementary to our goals. We focus solely on reducing temperature but also ensure that additional energy is not consumed by the CPU as a result.

Dimetrodon[1] is a software-level thermal management technique designed to assist in application-level proactive thermal management. We employ *idle cycle injection*, a scheduler-level mechanism to inject idle cycles of variable length into process execution, providing responsive, fine-grained control, allowing individual threads to absorb substantial portions of the burden of cooling, carefully mitigating performance reductions. Per-thread policy control allows us to target only key heat-producing workloads as opposed to system-wide policies such as current dynamic voltage and frequency scaling (DVFS) mechanisms, which may unfairly penalize heterogeneous workloads [12].

We implemented Dimetrodon in a commodity operating system and evaluated its efficiency in reducing temperatures while providing predictive throughput and latency models. We evaluate our techniques on modern commodity hardware using a mix of industry-standardized throughput and latency-sensitive workloads and derive quantitative models for the trade-off between application performance and temperature reduction (over the idle temperature). Dimetrodon achieves favorable performance for many of our workloads (over 16:1 for small temperature reductions and close to 1:1 for larger reductions) and outperforms similar techniques such as voltage and frequency scaling for temperature reductions up to 30%. Dimetrodon's strength comes in injecting short idle periods during which the processor is able to cool quickly; however, larger idle periods exhibit diminishing marginal benefits, decreasing its efficiency for large temperature reductions.

The main contributions of this work are:

DAC '11 Jun 05-10 2011, San Diego, CA, USA

[1]The *Dimetrodon* genus of large prehistoric reptiles were the dominant carnivores of their time. *Dimetrodon* possessed a large sail attached to its back, which enabled efficient thermal regulation and was likely used to control its body temperature [7].

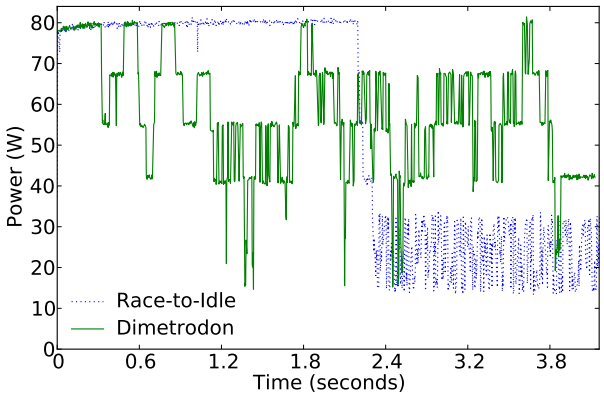

Figure 1: Race-to-idle versus Dimetrodon power consumption. The scheduler injected idle cycles into a multi-threaded CPU-bound process, lowering average power consumption during execution; the four power levels correspond to periods during which a varying number of the four processor cores idled.

- We propose the use of **idle cycle injection**, a flexible software technique, for per-thread preventive thermal control.
- We present an **implementation and evaluation** of several preventive thermal management techniques on real hardware in a commodity operating system.
- We **characterize the application-level impact** across worst-case thermal load and real-world workloads.

2. DESIGN AND MODEL

Typical *race-to-idle* scheduling, under which jobs are run to completion and the processor subsequently idles, incurs twofold energy costs: it requires energy to power the CPU and energy to power the cooling system to dissipate the heat generated by this unrestricted execution. If, however, we can run the job more slowly, incorporating some of the idle cycles into the job execution, the processor will produce less heat on average during execution. Figure 1 shows how we can achieve lower power on average during computation by slowing execution. We can redistribute computation in order to lower the average power dissipated while the processor is active.

Our goal is to maintain a lower average temperature over time. To target this average-case execution, Dimetrodon periodically idles the processor, injecting idle cycles at the scheduler level. By idling the processor for periods of time in-between regular program execution (at the scheduler *quantum*—timeslice—level), it briefly enters low-power states and cools. Idle cycle injection can be implemented as a scheduler policy that can be adjusted online according to the thermal profile and performance constraints of the application. The timescale of quanta is measured in milliseconds, which allows the processor long-term cooling opportunities.

From the perspective of a single thread, Dimetrodon moves idle cycles from idle periods after it completes to periodic intervals during process execution. We can vary both the proportion and length of these idle periods. Assuming that we can enter similar idle states during these injected periods as after execution completes, then we consume the same total energy while using less average power (§2.2). We express the proportion of idle periods as a probability; this is not the only possible injection model, however it simplifies our analysis and implementation.

2.1 Per-thread Software-level Control

Dimetrodon provides high flexibility. At the software level (particularly at the operating system or hypervisor level), one can control which threads are affected with arbitrary precision. Based on software system-specific information such as a process's user-granted priority level, thermal characteristics, voluntary and involuntary preemption patterns, and overall system condition (temperature, power consumption, etc.), the thermal management system can make more informed decisions about when to idle the processor and which threads to slow. Similarly, one can override policy control for high-priority threads or in times of high system load.

Software-level control allows fine-grained policies independent of the hardware platform. DVFS is not yet available for individual cores on commodity hardware [14] and may be limited by minimum voltage constraints, while it and other hardware techniques such as clock throttling typically allow only coarse, system-level policies with limited configurability [8]. Operating system control provides a wide range of possibilities: one can easily select the desired trade-off between throughput and heat with high resolution, allowing changes in throughput on the magnitude of fractions of a percent, as opposed to tens of percent. On processors that do not support low power idle states or clock gating, Dimetrodon is still useful as executing an idle loop of `nop` equivalents allows many functional units within the processor to cool. The decision whether to idle can be made efficiently and effectively at the scheduler level.

2.2 Analytical Model: Throughput and Power

Under Dimetrodon, each time the scheduler is about to schedule a thread, with user-defined probability p, it instead runs the idle thread for a quantum of length L. By varying p and L, we achieve different trade-offs between cooling and latency. For example, if p is 50% and L is the same length as a scheduling quantum, then we double the length of time for the job to run, but we lower the average heat produced by the job. Overall, the processor will use the same total amount of energy to complete the computation, provided it can enter similar idle states in-between computation intervals as following it. By increasing p, we increase the job's latency, but can reduce temperature by more. Decreasing L can gain back some of the latency loss at a possibly reduced cooling benefit.

Understanding the impact of the p and L parameters and selecting appropriate configurations is subtle. The analysis presented here and the validation and evaluation in Section 3 provide insight as to how Dimetrodon can be used in practice.

Throughput and runtime. We can consider a CPU-bound thread t that spends its entire real-world runtime of R seconds on the CPU with an average quanta length of q milliseconds. Suppose t is scheduled S times before it completes. If we idle the processor for a period of time L with probability p at each time t is scheduled, then we can predict t's runtime under Dimetrodon, $D(t)$.

$$D(t) = R + S\,\frac{p}{1-p}\,L$$

The thread must run for at least R seconds in order to complete, but each time t is scheduled it may be preempted. $\frac{p}{1-p}$ is the number of idle quanta per each execution quanta of t. For example, if we idle with probability 75%, then 3 out of 4 times t is scheduled we will idle instead, so there will be 3 idle quanta for every 1 executed quanta. Therefore, there are 3 S idle quanta. In practice we can determine S by dividing R by the average quanta length of t, q. We do not affect the fixed, non-CPU-related runtime overheads of thread execution.

Power. Varying the number of idle cycles injected results in lower average processor power consumption and therefore less heat dissipation by the processor while maintaining the same total energy, provided we can enter the same idle states. Transition times in the tens of μs [15] are negligible at quanta lengths measured in ms, however microarchitectural state may play a larger role (e.g., if a low power state flushes cache lines).

We analytically compare Dimetrodon's power consumption to a typical race-to-idle scenario. As above, consider a thread t with runtime R, average quantum length q, and idle cycle length L. For race-to-idle, we consider a window of time of length $D(t)$ such that the processor idles for time $t_{idle} = D(t) - R$. Assuming an active processor power consumption u watts and idle power consumption of m watts, then under race-to-idle the processor will consume

$$uR + t_{idle}\, m$$

joules. Under Dimetrodon, the processor will consume u watts during the R seconds that t is running and m watts for the $\frac{L}{q}\ \frac{p}{1-p}\ R$ seconds it is not idling. Therefore, with Dimetrodon, the processor will consume

$$uR + \frac{L}{q}\ \frac{p}{1-p}\ m\ R$$

joules. The two policies consume the same amount of total energy. Intuitively, this is the case because we simply shift the idle cycles from after the computation ends to between compute quanta. The average power while executing t is lower for $p < 1$.

3. EVALUATION

We evaluated Dimetrodon in a commodity operating system on a modern server in order to characterize its effects on processor heat and the role of parameters p and L. We first validate our analytical model presented in Section 2.2, then evaluate Dimetrodon using a worst-case heat generation stress test, `cpuburn`. We next examine real-world workloads from the SPEC CPU2006 benchmark suite and demonstrate per-thread control. We also characterize the effect on latency-sensitive workloads using SPEC Web.

3.1 Implementation

We implemented Dimetrodon in the FreeBSD 7.2 kernel[2]. When the scheduler selects the next thread to run, we decide whether to run the thread or whether to run the idle thread. If we decide to idle, we pin the thread that would have run on the runqueue (so it is not run by another processor) and schedule the kernel idle thread instead, which causes the processor to enter the idle state. Once the idle quantum is over, the preempted thread is unpinned and is made runnable again. While we could have avoided context switching overheads by trapping the thread in the kernel and issuing `hlt` instructions, choosing to run the idle thread greatly simplified our implementation. We control Dimetrodon using system calls.

We always schedule kernel-level threads. This is a policy decision that could easily be changed, however care should be taken to avoid preempting certain critical kernel threads. For example, when servicing an interrupt from the network as in a web server, a kernel thread will first run to handle the interrupt, and then notify a user thread. If we preempt kernel threads, then the processing of the network event may be delayed twice — once in the kernel and again in the user thread.

3.2 Experimental Setup

We tested Dimetrodon on a representative 1U rackmount server. Our server had an Intel Nehalem-based Xeon E5520 quad-core processor running at 2.26 GHz rated at 80 watts within a Supermicro SYS-6016T-MTLF chassis. It had 4 GB of DDR3 ECC RAM, a 500 GB 7200 RPM hard drive, and four five-watt case fans. The processor supported the C1E low power state (which does not flush the processor cache) and had a DVFS scaling settings every 133 MHz with a minimum of frequency of 1.6 GHz (71% of maximum). To measure processor power consumption, we connected a Fluke i410 current clamp to the processor power leads and used a Keithley 2701 ethernet-enabled multimeter to collect measurements. We maintained a thermostat setpoint at 25.2°C and fixed the system fan speed at full using an external controller. We used the FreeBSD `coretemp` module to measure temperature for each core (reported results) and used external temperature sensors at the server's rear air vents.

In order to simplify our analysis, we disabled simultaneous multithreading (SMT) on our processor, which allows multiple thread contexts to execute on a single core. In order to cause the entire core to enter the C1E low power state we need to halt all thread contexts on the core. This is feasible but requires additional care in co-scheduling idle quanta.

For each configuration, we executed four instances of each benchmark in parallel (one per core). To evaluate the trade-offs, we compare the reduction in temperature (over the idle temperature) with the reduction in application performance. We refer to 1:1 trade-offs solely as a baseline for comparison,

3.3 Model Validation

We validated both our power and throughput models using the `cpuburn` package [20] (specifically `burnP6`), which contains a single-threaded infinite loop containing a compact sequence of x86 instructions designed to thermally stress test processors.

We compared our analytical model for throughput to our implementation. We ran our finite `cpuburn` under a variety of configurations and measured nominal deviation from the predictive model. For 100 trials per configuration ($p \in \{25, .5, .75\}$, $L \in \{25, 50, 75, 100\}$ ms), our implementation resulted in throughputs that were on average 1.0% lower than expected. This throughput reduction is due mostly to configurations with higher p; we believe the deviation from our model increased as p increased largely due to context switching and state monitoring overheads.

We measured the energy consumed by Dimetrodon versus race-to-idle for equivalent periods of time (for $p \in \{.25,.5,.75\}$, $L \in \{50,100\}$ ms). We recorded the power consumed by the processor three times per millisecond throughout the execution of a finite loop of `cpuburn` instructions with a runtime of 7 seconds. Over an average of five trials for each benchmark Dimetrodon consumed between 97.6% and 103.7% of the energy of race-to-idle, with an average deviation of -0.37% and an average absolute deviation of 1.67%. Given the clamp accuracy (approximately 3.5%), these measurements validate our power model.

3.4 System Characterization

We next characterized Dimetrodon behavior for static policies under a worst-case thermal load. We executed many instances of `cpuburn` concurrently, fully burdening the processor. Core temperatures stabilized after approximately 300 seconds of `cpuburn`, and the average relative rise above the idle temperature was approximately equivalent across fan speed configurations. As shown in Figure 2, the temperature regularly fluctuates but is significantly reduced from the configuration where no injection occurs. These fluctuations are due to our probabilistic implementation; a more deterministic model would likely result in smoother curves but with similar overall temperature trends.

We measured the average temperature over the last 30 seconds of a 300 second execution and calculated the reduction in system temperature compared to the idle temperature relative to the temperature produced by unconstrained operation. For example, an

[2]FreeBSD 7.2 supports two schedulers, the 4.4 BSD scheduler, a traditional multi-level feedback queue with a fixed timeslice of 100ms, and the ULE scheduler, a modern scheduler designed to better support multiprocessor and low latency systems, however the 4.4 BSD scheduler was the default scheduler through FreeBSD 7.0 [26]. For simplicity of implementation, we modified the 4.4 BSD scheduler, however the mechanism generalizes to ULE and other schedulers.

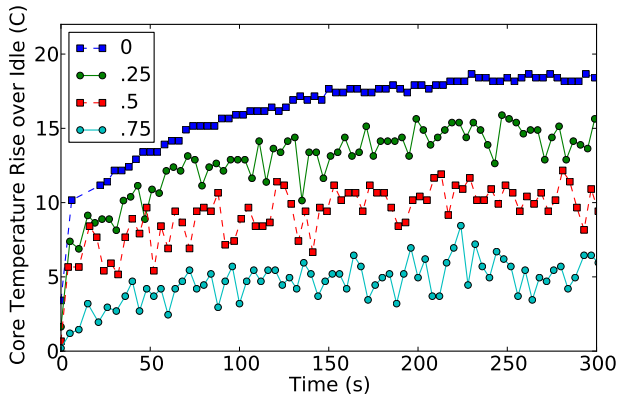

Figure 2: Average core temperature increase over the idle temperature during five minutes of `cpuburn` execution for different idle proportions (p). L=100 ms.

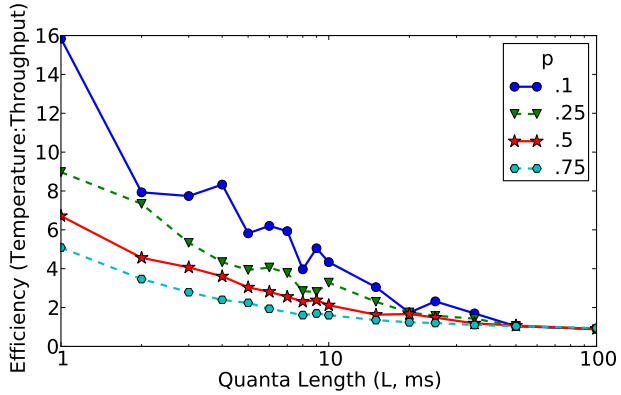

Figure 3: Efficiency of Dimetrodon for `cpuburn` varying idle quanta length (L) and proportion (p). There are diminishing marginal benefits to increasing the quanta length. Higher p curves are smoother due to the probablistic implementation.

idle temperature of 40°C, an unconstrained temperature 60°C, and a resulting temperature of 50°C would constitute a 50% reduction in temperature over idle.

Dimetrodon achieved at least a 1:1 trade-off between `cpuburn` throughput and temperature decrease compared to the idle temperature but typically achieved better. Efficiency was correlated with cycle length; as shown in Figure 3, short idle quanta lengths are particularly efficient, but there are diminishing marginal returns for longer quanta lengths. Fundamentally, each core was able to cool (exponentially) quickly within a short time window, but this efficiency decreased in longer time windows. Accordingly, we obtained smaller temperature decreases more efficiently than large temperature decreases. For a particular reduction in throughput, preempting `cpuburn` for a shorter idle cycle duration (decreasing L) but a more frequent interval (increasing p) allowed for a better temperature to throughput trade-off than using a longer idle quanta length. $\frac{100p}{L} > 1$ holds for pareto boundary configurations. We achieved a 16:1 temperature to throughput trade-off at a temperature reduction of 4.4%, but only a 1:1 trade-off at a temperature reduction of 90%. For large reduction targets, we could not use the more efficient short idle quanta and efficiency dropped.

We developed a quantitative metric in order to better characterize the trade-off between throughput and temperature. By curve-fitting the pareto boundary between temperature and throughput, we quantify the trade-off between desired temperature reduction r and throughput reduction $T(r)$ as

$$T(r) = \alpha\, r^{\beta}$$

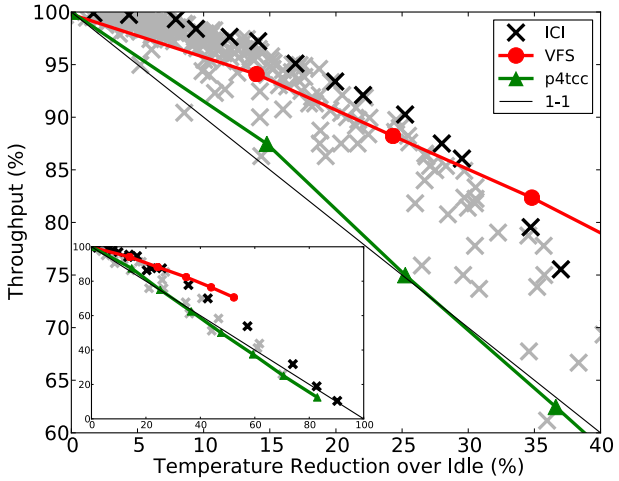

Figure 4: Wide-range parameter sweeps of Dimetrodon compared to other thermal management techniques. The pareto boundary is darkened.

where α and β are constants for $r \in [0, .75]$. For `cpuburn`, $\alpha = 1.092$ and $\beta = 1.541$. For $r > .75$, $T(r) \approx r$.

Finally, we compared Dimetrodon to several other comparable techniques We ran `cpuburn` under static policies using voltage and frequency scaling (VFS) [3] and the FreeBSD `p4tcc` driver, which controls the processor's thermal control circuit as a fine-grained clock gating technique [4], exhaustively sweeping set points for each mechanism. As shown in Figure 4, VFS allowed good trade-offs between throughput and temperature (for example, a 30% throughput reduction produced a 50% temperature reduction) due in large part to its quadratic reduction in power utilization as voltage scales down. Additionally, unlike Dimetrodon, VFS actually reduces total power consumed by the system.

However, Dimetrodon outperformed VFS for temperature reductions up to 30%. These data points correspond to the very short, efficient idle quanta lengths previously shown in Figure 3. However, the diminishing marginal utility of idle cycle lengths limited idle cycle injection's effectiveness for large temperature reductions, at which point the quadratic power benefits of VFS became more effective. In cases where system-wide policies are tolerable, it is advantageous to use VFS for preventive thermal management when temperature reductions greater than 30% are necessary.

While small idle quanta allowed the best trade-offs, `p4tcc`, which activated fine-grained, clock-level duty cycling, performed significantly worse, failing to achieve even 1:1 performance to throughput trade-offs at high temperature reductions. This suggests that the optimal idle cycle length is longer than the length of several clock signals and that the benefit of reducing cycle length decreases at extremely short time periods. Based on these results, the optimal idle period appears closer to the order of one ms, but may be shorter.

3.5 CPU-Bound Workloads

We subsequently evaluated Dimetrodon impact on several real-world workloads from the SPEC CPU2006 benchmarking suite [1], an industry-standard benchmark designed to test system processor and memory performance.

We first determined the thermal profiles of each of the SPEC

[3] Because FreeBSD did not support DVFS for our motherboard and processor, we ran our VFS tests under Linux 2.6.32 (Ubuntu 10.04 LTS) using the same binary and number of processes. We verified that the `cpuburn` temperature increases are equivalent on both Linux and FreeBSD with default processor settings.

Workload	Rise (%)	α	β
cpuburn	100	1.092	1.541
calculix	99.3	1.282	1.697
namd	87.2	1.248	1.546
dealII	84.4	1.324	1.688
bzip2	84.4	1.529	1.811
gcc	80.3	1.425	1.848
astar	71.7	1.351	1.416

Table 1: Real workload results. Average per-core temperature increase over the idle temperature for benchmarks from SPEC CPU2006 expressed as a percentage of the temperature increase for `cpuburn` when run unmodified (race-to-idle). Best-fit parameter estimation is shown for throughput reduction $T(r)$ for $r \in [0, .5]$.

CPU benchmarks. Based on this characterization, we selected six benchmarks that spanned a range of thermal profiles to examine in further detail. We tracked the average quantum length for each of the benchmarks (L), and found that the workloads were entirely CPU-bound. Subsequently, our throughput model is also applicable to these workloads.

We then examined Dimetrodon's effectiveness across various idle quantum lengths and probabilities as in our characterizations and developed predictive models for each workload, shown in Table 1. Despite the observed differences in absolute temperature increases, the differences in pareto optimal trade-offs between throughput and temperature were negligible, except at low throughput reductions. The absolute amount of heat being dissipated was different across workloads, but the relative efficiency curves did not substantially change. All workloads achieved better than 1:1 temperature to throughput trade-offs until at least 50% temperature reductions. Most benchmarks behaved similarly to `cpuburn` except `astar`, which was less effectively modulated; this is because it was significantly cooler-running than the other benchmarks and therefore benefited less from aggressive thermal modulation.

3.6 Thread-Specific Control

We now demonstrate Dimetrodon's usefulness in per-thread control. Degrading total system performance to limit a single process's heat output is inefficient; instead, as discussed in Section 2.1, per-thread control is desirable for thermally heterogeneous workload combinations. In this demonstration, we consider a periodic, short-running process, the "cool" process (a loop that executed `cpuburn` for six seconds, slept for one minute, and repeated), executing concurrently with a CPU-bound application, the "hot" process (four instances of `calculix`). As shown in Figure 5, degrading the cool process's performance because it is co-located with the hot process is undesirable if we want to optimize for per-process throughput while lowering temperature. Under global, non-thread-specific (system-wide) thermal actuation, the cool process is unfairly penalized for the "hot" process's heat generation. With per-thread control, the "cool" process can run interrupted while system-level temperatures are lowered. For per-chip techniques such as DVFS, the only solution to this problem is to intelligently schedule jobs across machines in the datacenter [16], which is possibly expensive if performed online, or to migrate threads between cores [11], which may be ineffective on fully-burdened machines [9]. Instead, we can use per-thread control.

3.7 Web Server Workload

In order to demonstrate Dimetrodon's impact on quality of service (QoS) sensitive applications, in addition to our throughput-based benchmarking, we also considered a latency-sensitive workload: web serving. While the impact on a QoS-sensitive workload

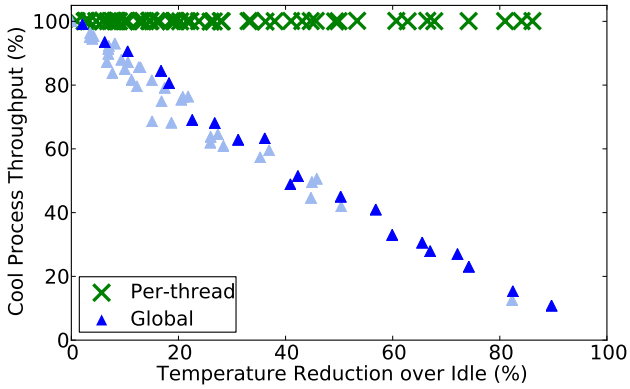

Figure 5: Global versus thread-specific control in Dimetrodon using idle quanta injection. With thread-specific control, the lower-heat "cool" process can execute without interruption while the system temperature is lowered by degrading "hot" process performance. With system-wide policies, cool processes are unfairly penalized. The pareto boundary is darkened.

is dependent on the particular QoS metric, we ran SPECWeb[4] [2], an industry standard web benchmark testing web server loads such as banking and eCommerce applications. This workload is significantly different from the prior workloads we have considered: latency is a critical factor and the load generated by a request is much quicker to fulfill.

We ran SPECWeb with 440 simultaneous connections split across two physical clients on a private network. Our server experienced approximately 15-25% load per core throughout the benchmark (the highest load possible for our configuration), and the overall temperature rise was approximately 6°C with no thermal actuation.

Several competing factors influence Dimetrodon's efficiency for SPECWeb. The workload allows for idle periods in-between uninterrupted processor execution due to time between requests, providing natural thermal modulation. Dimetrodon alters the distribution of these idle periods, and, in delaying execution, can lead to increases in the total number of outstanding requests. Because SPECWeb continually issues requests, if we defer a request, then when it is eventually processed there may be higher load on the system, possibly leading to increased heat generation. Therefore, injection efficiency depends on balancing the heat-reducing idle cycle injection and deferring idle cycles, which increases processor load and heat. Under heavy load, the processor is closer to saturation; fewer natural idle periods exist, and injecting idle cycles does not induce the same load-increasing behavior.

Overall, however, Dimetrodon is useful in reducing heat generation in a web serving context. SpecWEB performance is determined by three QoS thresholds: "good" (three second response or less), "tolerable" (five second response time or less), and "fail" (longer than five seconds) [2], each providing a range for allowable performance degradation. We show Dimetrodon's efficiency across a range of idle cycle amounts and lengths in Figure 6. At the lower, "tolerable" QoS threshold, we allowed up to 20% temperature reductions with virtually no drop-off in performance, and temperature reductions up to 50% incurred correspondingly smaller costs to performance. Even under tighter requirements ("good" metric), we allowed at least 1:1 and often better trade-offs until temperature reductions of 30% or more, at which point performance quickly falls below the acceptable range. Again, shorter quanta lengths were more efficient in reducing temperature than longer quanta lengths.

[4]Particularly, we used the SpecWeb2005 eCommerce workload also found in SPECWeb2009 [3].

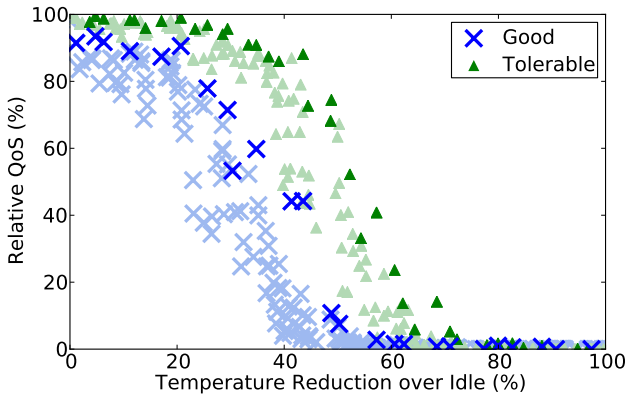

Figure 6: QoS and temperature reductions for web workload according to both "good" and "tolerable" metrics. Dimetrodon efficiency depends on the QoS threshold placement. The pareto boundary is darkened.

These results suggest that under latency-sensitive workloads, Dimetrodon's effect on performance will largely determined by both QoS metric and workload distribution.

4. RELATED WORK

Many dynamic thermal management techniques target worst-case thermal bounds. There is a wide spectrum of methods [8] ranging from the microarchitectural level [24] to the operating system level [21]. Techniques such as DVFS, fetch throttling, and clock gating are standard features on many existing microprocessors and reduce the burden of worst-case temperature management [8]. Several software-level scheduling schemes [9, 11] explicitly consider proactive thermal management and the temperature-application performance trade-offs involved. The key distinction between Dimetrodon and previous work is its focus on preventive, *average-case* cooling; our success metric is based on limiting overall temperature instead of bounding temperatures below a critical threshold, which leads to a different approach to thermal management.

Dimetrodon's idle cycle injection is similar to other techniques. Rohou and Smith [21] targeted thermal reductions by restricting the allowed CPU utilization of hot processes. Choi et al. [9] proposed several reactive techniques for operating system level dynamic thermal management, including a "cool loop" that would run when the system became overburdened. We explore the effects of both idle cycle length and proportion in a proactive thermal management context. Gandhi et al. [10] proposed the use of a similar scheduler-level idling technique for power-capping in data centers; Google recently introduced this mechanism into the Linux kernel [19]. Dimetrodon and this final technique target different domains (heat and power), but rearchitecting the power-capping mechanism to use shorter idle quanta would provide thermally-beneficial side-effects.

Dimetrodon is complementary to several existing thermal management techniques. While we have focused on idle cycle injection due to its flexibility, many hardware methods such as DVFS and throttling are applicable in a preventive thermal management context. Dimetrodon can be used in conjunction with multi-server thermal management solutions such as multi-core thermal management [11] and thermal-aware job placement [16]. Dimetrodon acts on a per-thread level, but can be combined with these techniques, especially when temperature predictions are inaccurate, when a data center experiences uniform temperature increases, or when there are no "cool" machines available. The substantial amount of literature targeting power reductions [13] may also prove useful in preventive thermal management as power reductions can translate to heat reductions.

5. CONCLUSIONS

We have presented a novel approach to preventively reducing average-case processor temperatures, Dimetrodon, which injects brief periods of inactivity into application execution, allowing the processor to cool while entering a low-power idle state. Using a prototype implementation and a real-world server-class platform, we examined the trade-offs between application performance and temperature reduction across both worst-case thermal load and a range of real-world throughput and latency-sensitive benchmarks. Dimetrodon is particularly effective for short idle periods but exhibits diminishing benefits with longer idle periods. Software idle cycle injection can be applied on a per-thread basis and outperforms voltage and frequency scaling (which allows quadratic reductions in power) for temperature reductions up to 30%. We conclude that Dimetrodon provides predictable performance trade-offs while allowing flexible operation.

6. REFERENCES

[1] SPEC CPU2006. `http://www.spec.org/cpu2006/`.

[2] SPECWeb2005. `http://www.spec.org/web2005/`.

[3] SPECWeb2009. `http://www.spec.org/web2009/`.

[4] *Intel 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide*, March 2010.

[5] BARROSO, L. A., AND HÖLZLE, U. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan and Claypool Publishers, 2009.

[6] BELADY, C. In the data center, power and cooling costs more than the it equipment it supports. *Electronics Cooling 23*, 1 (February 2007).

[7] BRAMWELL, C. D., AND FELLGETT, P. B. Thermal regulation in sail lizards. *Nature 242* (1973), 203–205.

[8] BROOKS, D., AND MARTONOSI, M. Dynamic thermal management for high-performance microprocessors. In *HPCA '01*.

[9] CHOI, J., CHER, C.-Y., FRANKE, H., HAMANN, H., WEGER, A., AND BOSE, P. Thermal-aware task scheduling at the system software level. In *ISLPED '07*.

[10] GANDHI, A., HARCHOL-BALTER, M., DAS, R., KEPHART, J., AND LEFURGY, C. Power capping via forced idleness. In *WEED '09*.

[11] GOMAA, M., POWELL, M. D., AND VIJAYKUMAR, T. N. Heat-and-run: leveraging SMT and CMP to manage power density through the operating system. In *ASPLOS '04*.

[12] HASAN, J., JALOTE, A., VIJAYKUMAR, T. N., AND BRODLEY, C. E. Heat stroke: Power-density-based denial of service in smt. In *HPCA 2005*.

[13] ISCI, C., BUYUKTOSUNOGLU, A., CHER, C.-Y., BOSE, P., AND MARTONOSI, M. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *MICRO '06*.

[14] KIM, W., GUPTA, M., WEI, G. Y., AND BROOKS, D. System level analysis of fast, per-core DVFS using on-chip switching regulators. In *ISCA '08*.

[15] MEISNER, D., GOLD, B. T., AND WENISCH, T. F. Powernap: eliminating server idle power. In *ASPLOS '09*.

[16] MOORE, J., CHASE, J., RANGANATHAN, P., AND SHARMA, R. Making scheduling "cool": temperature-aware workload placement in data centers. In *USENIX ATC '05*.

[17] PATEL, C. D., AND SHAH, A. J. Cost model for planning, development and operation of a data center. Hewlett Packard Technical Report HPL-2005-107R1.

[18] PELLEY, S., MEISNER, D., WENISCH, T. F., AND VANGILDER, J. W. Understanding and abstracting total data center power. In *WEED '09*.

[19] QAZI, S. Idle cycle injection in Linux. Presented at the 2010 Linux Collaboration Summit.

[20] REDELMEIER, R. cpuburn. `http://pages.sbcglobal.net/redelm/`.

[21] ROHOU, E., AND SMITH, M. D. Dynamically managing processor temperature and power. In *In 2nd Workshop on Feedback-Directed Optimization* (1999).

[22] ROTEM, E., COHEN, A., AND CAIN, H. Temperature measurement in the Intel Core Duo processor. In *THERMINIC '06*.

[23] SANTARINI, M. Thermal integrity: A must for low-power IC digital design. *EDN* (September 2005), 37–42.

[24] SKADRON, K., STAN, M. R., HUANG, W., VELUSAMY, S., SANKARANARAYANAN, K., AND TARJAN, D. Temperature-aware microarchitecture. In *ISCA '03*.

[25] SRINIVASAN, J., ADVE, S. V., BOSE, P., AND RIVERS, J. A. The case for lifetime reliability-aware microprocessors. In *ISCA '04*.

[26] THE FREEBSD PROJECT. Generic kernel configuration file for FreeBSD/i386 in FreeBSD 7.0. `http://fxr.watson.org/fxr/source/i386/conf/GENERIC?v=FREEBSD70`.