A Framework for Thermal and Performance Management

Davide B. Bartolini[1], Filippo Sironi[1,2], Martina Maggio[3], Riccardo Cattaneo[1],
Donatella Sciuto[1], Marco D. Santambrogio[1,2]

[1]Politecnico di Milano, [2]Massachusetts Institute of Technology, [3]Lund university

{bartolini, sironi, cattaneo, sciuto, santambrogio}@elet.polimi.it, martina.maggio@control.lth.se

Abstract

In modern computing facilities, higher and higher operating temperatures are due to the employment of power-hungry devices, hence the need for cost-effective heat dissipation solutions to guarantee proper operating temperatures. Within this context, dynamic thermal management techniques (DTM) can be highly beneficial in proactively control heat dissipation, avoiding overheating. The large-scale adoption of DTM may eventually allow the use of more cost-effective heat dissipation system, with great power consumption advantages for large datacenters.

Preventive thermal management is a technique to achieve long-term thermal control via performance degradation. However, this may result in impaired Quality of Service (QoS) and Service Level Agreements (SLAs) breaking. We address this problem by proposing a self-adaptive framework combining performance and thermal management targeting Chip Multi-Processors (CMPs). The proposed methodology harnesses control-theoretical controllers for driving idle-cycle injection and threads priority adjustment, in order to provide control over the processor temperature, while taking applications' QoS (in terms of performance) into account. We implemented our framework in the FreeBSD operating system and evaluated it on real hardware, also comparing it with a previous framework for preventive DPTM.

1 Introduction

Recently, VLSI design – in particular for processors – has been heavily influenced by the approachment of the power wall. The shift from single to multi-core processors was a tentative response to power constraints, passing from one very complex processor to a set of simpler on-chip cores. However, even multi-core processors are hitting new barriers, originating the phenomenon of dark silicon [5]. Packing more and more transistors on a single die of the same size, with energy efficiency not scaling any more, results in an ever increasing power density, which requires more efficient cooling systems to keep proper operational thermal conditions. Maintaining processors cool is crucial for reliability and efficiency: higher average operating temperatures lead to a drastic reduction of the Mean Time To Failure (MTTF) [18] and highly increased leakage power [17]. Moreover, processors are acknowledged to be one of the most power-hungry and heat-producing components in a computing system and, for plenty of server workloads, power consumption is reported to increase almost linearly with processor utilization [4]. These considerations are of utter importance for data centers; up to 80% of the Total Cost of Ownership (TCO) – including both building and maintenance costs – is due to the cooling infrastructure and the overall power consumption [8].

Recently, processor-level Dynamic Thermal Management (DTM) techniques received quite a lot of attention and the trend is to move from runtime guards for emergency situations to always-on proactive control systems. Preventive DTM is a technique leveraging modern processors' features such as idling power states (C-states) in order to actively degrade performance for achieving thermal control [2]; however, doing so may cause problems when SLAs exist on the provided QoS.

We tackle this problem with a **Dynamic Performance and Temperature Manager** (**DPTM**) framework based on self-adaptive computing [15], realizing preventive DTM while accounting for desired applications performance in terms of QoS specified by user-signed SLAs. The DPTM framework is an extension of a commodity operating system and it is able to inject idle cycles for cooling down cores and to vary threads' priority for affecting applications' performance. The main contributions of this work are: (i) harnessing idle-cycle injection to drive the processor temperature towards a desired set point; (ii) accounting for desired QoS and controlling threads' priorities for respecting SLAs on performance; (iii) coupling the thermal and the performance-aware mechanisms for both preventive DTM and SLAs enforcement. We implemented DPTM extending FreeBSD 7.2 [1] and the evaluation leverages applications from the PARSEC 2.1 Benchmark Suite [3].

2 State of the Art

Dynamic Thermal Management (DTM) has been an active research topic in the recent years; some relevant works found in literature are surveyed in this section.

Rohou and Smith [14] presented an evaluation of the idle-cycle injection. Since then, many researchers have exploited their early findings. Kumar and Thiele [9] proposed a DTM technique for real-time systems with the objective of minimizing the system peak temperature while meeting deadlines. The authors implemented a scheduler based on leaky-bucket shapers able to dynamically inject idle cycles and delay execution without violating deadlines. Experimental results show that the peak temperatures observed are 8.8 K lower with respect to those observed using a standard real-time scheduler. This technique is simple, effective, and able to manage any task arrival pattern; however, it is tailored to processors lacking Dynamic Voltage and Frequency Scaling (DVFS).

Gupta and Mahapatra [7] developed a DTM technique for real-time systems able to reduce violations of temperature constraints through DVFS and still meeting task deadlines. They implemented an on-line technique relying on a feedback controller that sets voltage and frequency so as to reduce thermal dissipation and meet deadlines. This can be done considering as input an upper-bound system temperature, the past system temperatures, and the jobs slacks. Simulation results show that constraint violations are reduced by 18.9% on average.

Recently, Bailis et al. [2] proposed *Dimetrodon*, a framework for desktop and server systems realizing preventive DTM through idle-cycle injection [14]. They argue that DVFS is an invasive technique and should be used only when relevant temperature variations are needed. Idle-cycle injection, instead, guarantees fine-grained control over the thermal dissipation of each running thread. Dimetrodon was implemented within the FreeBSD 7.2 kernel and evaluated on CPU-intensive workloads. The gathered data demonstrate that Dimetrodon obtains a better temperature/performance trade-off with respect to DVFS for temperature reductions of up to 30% of the idle temperature. A drawback of Dimetrodon lies in the logic employed to inject idle cycles, which are inserted randomly without considering possible SLAs on the delivered QoS. Moreover, the system does not allow to define a set point for the processor temperature, but just the idle-cycle injection probability.

The DPTM framework we propose builds upon the methodology harnessed by Dimetrodon and addresses its drawbacks, allowing to specify a temperature set point and coupling thermal-awareness with performance-awareness in order to avoid breaking SLAs.

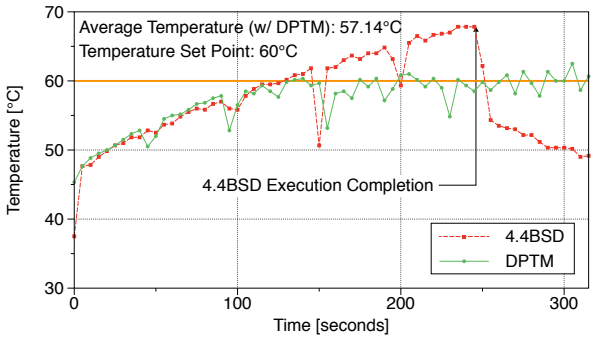


Figure 1: Average processor temperature with a race-to-idle (4.4BSD) scheduler and the DPTM framework when executing the same compute-bound workload.

3 Methodology

Typical scheduling infrastructures in commodity operating systems adhere to the race-to-idle approach: applications are run to completion in order to idle the system as soon as possible, thus increasing the throughput of applications. This approach is generally energy-efficient overall [6], but it leads to higher peak power draw and thus to higher maximum chip temperatures. To contrast this – sometimes undesirable – property, whenever applications can afford a decrease in their throughput the scheduling infrastructure may exploit policies to reduce the peak power, thus reducing the maximum processor temperature and consequently requiring less power to operate the cooling infrastructure, the costs of which greatly impacts the management expenses, especially in the context of data centers and server farms [13, 8]. Our proposed approach exploits this idea and Figure 1 shows an example execution highlighting the difference between a common race-to-idle approach and that proposed in this paper. Temperature reduction can be achieved by injecting idle cycles during the execution of a workload, letting the processor briefly go to a low power state, thus reducing the instantaneous absorbed power and, subsequently, the chip temperature. Randomly injecting idle cycles (as done in state-of-the-art approaches [2]), however, cannot be affordable when SLAs exist on the performance of some applications. Our methodology applies to this scenario, where processor-bound applications are assigned high-level QoS goals in terms of desired throughput, similarly to what was proposed in different contexts [16]. Exploiting this knowledge, we can choose the victim tasks of the idle-cycle injection process among applications not bound to an SLA, while not penalizing those with specified performance goals. Moreover, our framework also drives the scheduling priorities of SLA-bound applications in order to achieve finer performance control.

3.1 Control-Theoretical Models

To achieve control over peak and average processor temperature and processor-bound applications performance, we extend the scheduling infrastructure of a commodity operating system with a thermal-aware and a performance-aware policy. The former monitors the current processor temperature and drives idle-cycle injection to trigger the low-power mode in the cores, reducing the instantaneous absorbed power and, subsequently, the processor temperature. The latter observes user-defined throughput goals (i.e., SLAs on the QoS of specific applications) and the current throughput, driving the scheduling priority of the threads of SLA-bound applications in order to increase or decrease the processor time they are ensured, thus affecting their performance.

When designing the models, we considered a trade-off between accuracy and complexity. Systems usually show non-linear behavior, but accounting for non-linearity in the model often reduces the provable properties and can impair the possibility of building a controller. We approximate non-linearity with linear models by devising a strategy to estimate parameters and to reduce the imprecision due to non-linearities.

Bailis et al. [2] already discussed the basics of idle-cycle injection. Conversely to that work, which is based on a thermal-unaware probabilistic policy, we adopt a control-theoretical policy with a feedback loop, able to automatically drive the average processor temperature towards a specified set point, with the aim of cutting down the thermal peak typical of a race-to-idle approach. Let $T_i(k)$ be the temperature of the i-th core measured at time k; the goal of the controller is to stabilize $T_i(k)$ close to the set point $\tilde{T}$. We assume the model shown in Equation (1) represents the processor's thermal characteristics.

$$T_i(k+1) = T_i(k) + \mu_i \times idle_i(k) \tag{1}$$

The value $idle_i(k) \in [0\%, 100\%]$ represents the fraction of idle time injected in the i-th core during the time interval between the k-th and $k+1$-th sampling instants, while μ_i is an unknown parameter. The control-theoretical system, designed as an adaptive deadbeat controller [10], computes $idle_i(k)$ per core at each sampling instant. A deadbeat controller is synthesized so as the closed-loop transfer function equals a pure delay, i.e., after one control step the set point $\tilde{T}$ should be transferred to the output temperature. It is possible to analytically demonstrate that, if μ_i is known, the set point will be attained [10] and the temperature will be kept at the reference level. Intuitively, idle cycles will be injected when the temperature gets too high, while no action will be taken while temperature remains lower than the set point. Since μ_i cannot be given a priori, we estimate it based on the last temperature measurements through an Exponential Weighted Average (EWA) adaptive filter.

The performance-aware policy features another control-theoretical mechanism devoted to driving threads priorities; the policy requires user-defined goals to define SLAs and instrumented applications to provide throughput measures, similarly to what was proposed by Sironi et al. [16]. SLA-bound applications are assigned a throughput goal stated as an interval $[thr_{min}, thr_{max}]$, where thr_{min} is intended as the minimum performance satisfying the SLA and thr_{max} is a level over which no significant QoS improvements are achieved. Legacy (i.e., non instrumented) applications are assumed not to be bound to any SLA. Let $r_i(k)$ be the throughput of the i-th application at time k; the goal of the controller is to keep the performance of each instrumented application close to its set point $\tilde{r} = \frac{thr_{min}+thr_{max}}{2}$. We assume the performance model in Equation (2).

$$r_i(k+1) = r_i(k) + \eta_{i,j} \times \Delta priority_{i,j}(k) \tag{2}$$

The value $\Delta priority_{i,j}(k) \in [-50, 50]$ represents the priority of the j-th thread of the i-th application, while $\eta_{i,j}$ is an unknown parameter. At each sampling instant, the control-theoretical system computes $\Delta priority_{i,j}(k)$ per thread per application. Also in this case, the closed-loop system is designed to be a pure delay. As for the thermal-aware policy, if η_i is known then the set point signal will be attained. The $\Delta priority$ will be decreased whenever performance is getting higher than needed, while higher priority will be given when the desired QoS is not being met. Also in this case, the parameter $\eta_{i,j}$ is periodically estimated with an EWA adaptive filter.

3.2 Performance-Temperature Trade Off

Within the DPTM framework, a synergy is created between the thermal and performance-aware policies by means of a heuristic, which couples them for stronger performance control and finer thermal control. Since the performance-aware policy is in place to avoid breaking SLAs, this policy has precedence over the thermal-aware one. More in detail, when an application bound to an SLA is running below its thr_{min}, the performance-aware policy is able to mark its threads to *prevent idle*. Conversely, when a controlled application is over performing (i.e., its throughput is above thr_{max}), its threads are marked to *force idle*. The thermal-aware policy uses these two flags to never charge idle time to threads of applications not respecting their SLA, while always charging idle time to threads of applications running faster than needed. In practice, performance is traded for temperature reduction only if no SLAs get broken.

4 Implementation

We implemented the proposed DPTM framework as an extension of FreeBSD 7.2, modifying the 4.4BSD scheduler[1]. The thermal-aware policy consists of a set of high-priority kernel threads measuring the temperature of cores and a high-priority kernel thread implementing the control-theoretical system described in Section 3.1, Equation (1). Temperature measurements are gathered by reading the appropriate Model Specific Register (MSR) for each core, with an accuracy of 1°C. Temperature constraints are defined through the `sysctl` utility. Similarly, the performance-aware policy is made up of a set of timers computing the throughput of applications and a high-priority kernel thread implementing the control-theoretical system described in Section 3.1, Equation (2). The infrastructure for throughput measurements is a complete port of the Heart Rate Monitor (HRM), proposed by Sironi et al. [16], to FreeBSD 7.2 kernel.

The 4.4BSD scheduler is based on a multilevel feedback queues infrastructure: all runnable threads are assigned a priority determining the run queue they are placed on and threads are migrated according to their changing priority. In selecting the new thread to run, the scheduling infrastructure scans the run queues from the highest to the lowest priority and it chooses the first thread found on the first non-empty run queue. Multiple threads on the same run queue are managed in a round robin fashion and are assigned a fixed timeslice of 100 ms [11]. In extending the 4.4BSD scheduling infrastructure, we were careful to preserve its desirable properties, like non-starvation and priority decay.

The performance-aware policy is an extension of the scheduling infrastructure acting in a decoupled fashion. The priority of threads is adjusted using an additive term ($\Delta priority_{i,j}$) for each thread j of an SLA-bound application i. This operation induces the migration of the threads from a run queue to another, according to goals and performance. The 4.4BSD scheduler subdivides threads in five scheduling classes according to their assigned priority. The performance-aware policy works on threads in the *time-sharing user* class (i.e., regular applications' threads) and further checks are applied in order to avoid the additive term to modify the scheduling class of a thread (e.g., to the *real-time user* or to *idle* classes). The policy also sets an additional per-thread flag, allowing to realize the synergy with the thermal-aware policy; each thread j of a performance-controlled application i is marked either *force idle* or *prevent idle* as explained in Section 3.2.

[1]FreeBSD 7.2 supports two different schedulers: 4.4BSD and ULE. We chose 4.4BSD as a base for fair comparison with Dimetrodon [2].

The thermal-aware policy acts in coordination with the 4.4BSD scheduler: when the scheduler chooses the next thread to run, the control-theoretical system decides whether to actually execute it or to schedule the idle thread instead. The preemption of system critical threads (i.e., bottom-half kernel threads, such as interrupt handlers, or top-half kernel threads) and of threads marked *prevent idle* is automatically impeded, while the idle thread is always set to be executed if the next chosen thread by the 4.4BSD algorithm is marked *force idle*.

5 Experimental Evaluation

The experimental evaluation of the DPTM framework was carried out in two parts. The first part concerns the evaluation of the thermal-aware policy alone and a comparison with Dimetrodon, a state-of-the-art framework for preventive DTM with available source code [12]; this part is covered in Section 5.1. The second part, covered in Section 5.2, evaluates the complete DPTM framework.

5.1 Thermal-Aware Policy

To experimentally characterize the thermal-aware policy, we used a workstation equipped with an Intel Core i7-990X six-core processor, 6 GB of DDR3-1066 non-ECC RAM, and FreeBSD 7.2. Intel Hyper-Threading (HT) and Turbo Boost were disabled while Enhanced Intel SpeedStep was enabled to allow the processor to enter low-power modes. The operating system was configured to boot either the Dimetrodon kernel or the DPTM kernel. The high-priority kernel threads for measuring the cores temperatures and implementing the idle-cycle injection controller in the DPTM framework were set to run with a period equal to the 4.4BSD scheduler timeslice (i.e., 100ms). We evaluated the thermal-aware policy by running applications from the PARSEC 2.1 Benchmark Suite, each run consisting of ten consecutive executions of the same application; the same experiments were repeated with Dimetrodon and with the DPTM framework. The idle-cycle injection probability parameter of Dimetrodon (which uses an open-loop probabilistic idle-cycle injection policy and provides no possibility of choosing a set point) was empirically chosen for each application, with the duration of each idle period always set to 50 ms; the temperature set point for DPTM framework was set to a value close to the average temperature measured with Dimetrodon (arbitrarily chosen as the closest smaller multiple of five, e.g., 55 °C when Dimetronon yields 59 °C).

Table 1 reports the data coming from this first experiments. Each row shows the results for one of the applications showing, for the two frameworks, the configuration parameter and the average and standard deviation of

Table 1: Data from the evaluation of the DPTM framework's thermal-aware policy and comparison with Dimetrodon. The configuration parameters are shown along with the resulting average and standard deviation of the measured processor temperature. The performance speedup of the DPTM framework over Dimetrodon is also reported.

Application	*Dimetrodon*			*DPTM framework*			$\frac{Dimetrodon}{DPTM}$
	Idle Inj. Prob.	Avg [° C]	Std. Dev. [° C]	Set Point [° C]	Avg [° C]	Std. Dev. [° C]	*Perf. Speedup*
blackscholes	50%	51.16	3.45	50	49.96	2.98	1.27×
ferret	20%	58.02	5.01	55	56.36	2.97	1.12×
fluidanimate	10%	60.48	2.36	60	58.86	3.20	1.32×
swaptions	50%	59.00	4.60	55	54.03	3.33	1.57×

the measured processor temperature. The last column reports the performance speedup obtained using the DPTM framework with respect to using Dimetrodon. These data are also represented in Figure 2. From the plots, it is immediate to see how the proposed thermal-aware policy was able to achieve at least the same temperature reduction measured with Dimetrodon (by attaining the chosen set points), with the advantage of also achieving a runtime reduction (i.e., performance speedup) between 1.12× and 1.57×. This speedup is due to the feedback loop-based policy of the DPTM framework, which avoids to inject unnecessary idle cycles, conversely to the static probabilistic injection employed by Dimetrodon.

5.2 Complete DPTM framework

The second part of the experimental section evaluates the whole DPTM framework. To do so, we used a workstation equipped with an Intel Core i7-870 quad-core processor running at 2.93 GHz, 4 GB of DDR3-1066 non-ECC RAM, and FreeBSD 7.2 configured to boot with the DPTM kernel. The overall experimental setup is analogue to that used for the first part of the evaluation. The workload for this test combines four 4-threaded instances of the *swaptions* benchmark to simulate a resource-limited scenario. Three of the four instances are used to simulate non SLA-bound applications, while one is bound to a high-level throughput goal expressed in number Monte Carlo simulations per second. On the employed workstation, a single 4-threaded instance of *swaptions* run alone achieves a throughput of about 90000 sims/s, while four instances run at the same time with the base 4.4BSD scheduler yield a throughput of about 22500 sims/s each; in this latter scenario, the processor temperature reaches 80° C after a few seconds. The experiment consists in setting a higher performance goal for one of the instances (i.e., between 37000 and 43000 sims/s) and, at the same time, specifying a processor temperature set point at 60° C. Figure 3 shows both the throughput achieved by the SLA-bound application and the processor temperature

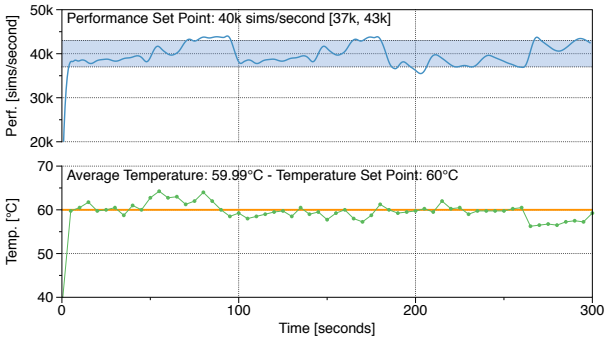


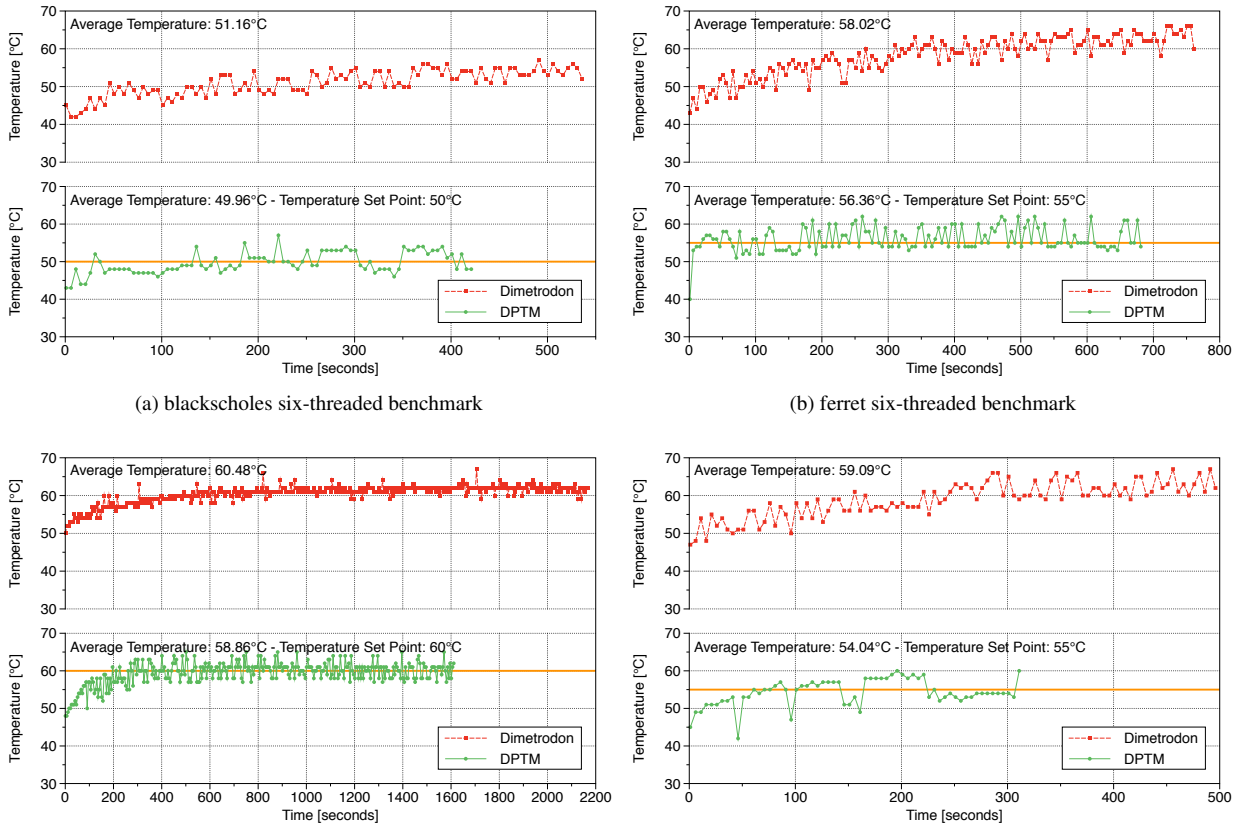
Figure 3: Results for four 4-threaded instances of swaptions. One of the instances is bound to an SLA with a throughput goal of 40000 sims/s and a range between 37000 and 43000 sims/s. The thermal set point is 60° C.

trace during the experiment. This experiment provides evidence of the full capabilities of the DPTM framework which, beyond applying preventive DTM to cap thermal peaks, is also able to let SLA-bound processor intensive applications meet their QoS goal in terms of throughput.

6 Discussion and Future Work

The experimental evaluation of the DPTM framework, which we presented in this paper describing its base methodology and its current implementation on FreeBSD 7.2, shows its efficacy in realizing preventive DTM while accounting for existing SLAs on processor-bound applications' QoS in terms of throughput. The results show that the DPTM framework advances the state-of-the-art – represented by Dimetrodon – in terms of preventive DTM, while also adding the novelty of a coupled performance and thermal-aware policy.

What is presented in this paper is a preliminary incarnation of the DPTM framework, aimed at demonstrating the soundness of the methodology; some simplifications were made in order to produce a proof-of-concept, opening ways for future works further improving the frame-



(a) blackscholes six-threaded benchmark

(b) ferret six-threaded benchmark

(c) fluidanimate eight-threaded benchmark

(d) swaptions six-threaded benchmark

Figure 2: Results from the evaluation of the thermal-aware policy of the DPTM framework. The Figures represent the processor temperature traces; the respective aggregate data and used parameters are also reported in Table 1.

work. For instance, the thermal-aware policy is currently based on per-core controllers considering only the measured core temperature for driving idle-cycle injection; while this approach works in practice, it ignores thermal coupling among cores, which can be relevant on modern Chip Multi-Processors (CMPs) and could be considered in future works. Another improvement could be refining the idle-cycle injection mechanism to equally charge idle cycles to all the threads of a parallel application, avoiding artificial critical paths slowing down only one thread in case of synchronization points.

On the performance-aware policy side, an improvement would be also considering performance goals in terms of latency or response time, which could allow characterizing applications such as web servers, making the DPTM framework more widely applicable. Moreover, the assumed performance model is not the most accurate possible and, despite working quite well in practice, the system could benefit from a more accurate modeling.

A topic non directly covered in this paper is the situation where SLAs cannot be met with available system resources. For instance, the experiment presented in Section 5.2 could be repeated by setting SLAs on all the four applications with performance goals too high to be attained at the same time on the reference workstation. According to the current implementation of the DPTM framework, in this situation the thermal-aware policy would be disabled (all the threads would be marked *prevent idle*) and the available resources would be fairly distributed by the performance-aware policy to the SLA-bound applications weighted with respect to their goal. Hence, no application would respect its SLA, but the performance of each would be proportional to the respective goals. This way, the system realizes sort of different QoS levels in case the system resources are not enough to fully respect the SLAs. Improvements in this scenario could come from refinements to the heuristic coupling the thermal and the performance-aware policies for more flexibility, allowing configuring different degrees of priority between the two.

References

[1] The FreeBSD Project. `http://www.freebsd.org/`.

[2] P. Bailis, V. J. Reddi, S. Gandhi, D. Brooks, and M. Seltzer. Dimetrodon: Processor-level Preventive Thermal Management via Idle Cycle Injection. In *Proceedings of the 48th Annual Design Automation Conference*, 2011.

[3] C. Bienia. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, 2011.

[4] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao. Energy-Aware Server Provisioning and Load Dispatching for Connection-Intensive Internet Services. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, 2008.

[5] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger. Dark Silicon and the End of Multicore Scaling. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, 2011.

[6] M. Garrett. Powering Down. *Queue*, 5(7), 2007.

[7] N. Gupta and R. Mahapatra. Temperature Aware Energy Management for Real-Time Scheduling. In *12th International Symposium on Quality Electronic Design*, 2011.

[8] U. Hoelzle and L. A. Barroso. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan and Claypool Publishers, 1st edition, 2009.

[9] P. Kumar and L. Thiele. Cool Shapers: Shaping Real-Time Tasks for Improved Thermal Guarantees. In *Proceedings of the 48th Annual Design Automation Conference*, 2011.

[10] W. S. Levine. *The Control Handbook*. CRC Press, 2nd edition, 2010.

[11] M. K. McKusick and G. V. Neville-Neil. *The Design and Implementation of the FreeBSD Operating System*. Addison-Wesley Professional, 1st edition, 2004.

[12] Peter Bailis and Vijay Janapa Reddi and Sanjay Gandhi and David Brooks and and Margo Seltzer. Dimetrodon Source Code. `http://www.bailis.org/projects/dimetrodon/`.

[13] P. Ranganathan and J. Chang. Saving the World, One Server at a Time, Together. *Computer*, 44(5), 2011.

[14] E. Rohou and M. D. Smith. Dynamically Managing Processor Temperature and Power. In *Proceedings of the Second Workshop on Feedback-Directed Optimization*, 1999.

[15] M. Salehie and L. Tahvildari. Self-Adaptive Software: Landscape and Research Challenges. *ACM Trans. Auton. Adapt. Syst.*, 4(2):14:1–4:42, 2009.

[16] F. Sironi, D. B. Bartolini, S. Campanoni, F. Cancare, H. Hoffmann, D. Sciuto, and M. D. Santambrogio. Metronome: operating system level performance management via self-adaptive computing. In *Proceedings of the 49th Annual Design Automation Conference*, DAC '12, pages 856–865, New York, NY, USA, 2012. ACM.

[17] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-Aware Microarchitecture. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, 2003.

[18] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. The Case for Lifetime Reliability-Aware Microprocessors. In *Proceedings of the 31st Annual International Symposium on Computer Architecture*, 2004.