TaSaT: Thermal-Aware Scheduling and Tuning Algorithm for Heterogeneous and Configurable Embedded Systems

Mohamad Hammam Alsafrjalani
Department of Electrical and Computer Engineering,
University of Miami, Coral Gables, FL, USA
alsafrjalani@miami.edu

Tosiron Adegbija
Department of Electrical and Computer Engineering,
The University of Arizona, Tucson, AZ, USA
tosiron@email.arizona.edu

ABSTRACT

Heterogeneous and configurable systems (HaCS) have been widely used to meet stringent runtime performance and energy constraints in embedded systems. However, no prior work has addressed the emerging runtime thermal constraints in these systems. To leverage HaCS' capabilities to meet thermal constraints, in addition to performance and energy constraints, we propose *TaSaT, a Thermal-aware Scheduling and Tuning* algorithm for HaCS. TaSaT reduces HaCS temperature while meeting performance and energy constraints during runtime, without a priori knowledge of applications.

ACM Reference Format:

Mohamad Hammam Alsafrjalani and Tosiron Adegbija. 2018. TaSaT: Thermal-Aware Scheduling and Tuning Algorithm for Heterogeneous and Configurable Embedded Systems In *GLSVLSI '18: 2018 Great Lakes Symposium on VLSI*, May 23–25, 2018, Chicago, IL, USA. ACM, NY, NY, USA, 6 pages. https://doi.org/10.1145/3194554.3194576

1 INTRODUCTION

The ubiquitous nature of embedded systems has placed stringent and disparate constraints, including energy, performance, and temperature on these systems. Whereas energy and performance have traditionally been the predominant constraints, temperature is a growing concern in embedded systems due to the systems stringent resource constraints. Due to the small size, limited availability of cooling mechanisms (e.g., on-board fans), and cost of system real estate, keeping an embedded system's temperature within thermal constraints, in addition to necessary performance and energy optimizations, is a major challenge for system designers.

Several microprocessor optimizations exist to enable adherence to potentially competing performance and energy constraints. For example, heterogeneous multiprocessors—such as the ARM big.LITTLE [5], ARM Juno [6]—offer fixed, disparate configurations to provide coarse-grained adaptation to different applications' execution requirements. To enhance this adaptation, configurable systems with tunable components (e.g., cache, pipeline depth, issue window, etc.) can be tuned/adjusted to meet specific, fine-grained application requirements [10][19][23]. Furthermore, heterogeneous and configurable systems (HaCS) [3][4] provide larger design spaces that enable both coarse-grained and fine-grained adaptation to applications' runtime needs.

To address increasingly critical thermal challenges, several thermal-aware optimizations have been employed to reduce the core temperature. Dynamic thermal management (DTM) [8] leverages the temperature-performance tradeoff of dynamic voltage and frequency scaling (DVFS), decode throttling, speculation control and instruction cache toggling to reduce the core temperature. Complementing DTM, activity migration [13] and temperature-aware scheduling [22] reduce core temperature by executing the applications on the core(s) that will result in the lowest average system temperature. Furthermore, phase-based tuning [1] has been used to determine Pareto optimal configurations in configurable systems for fine-grained execution time, energy, and temperature.

Whereas prior thermal-aware optimizations addressed thermal issues in homogenous and configurable systems, to the best of our knowledge, no work has addressed thermal constraints in HaCS, which is particularly challenging due to the coarse- and fine-grained adherence to constraints in HaCS. We have identified three main challenges of thermal management in HaCS: (1) considering HaCS' typically large design spaces, the heterogeneous and configurable parameters that offer the best coarse- and fine-grained performance and temperature tradeoffs must be determined; (2) HaCS thermal optimization must be scalable to applications that are unknown at design time; and (3) given a multicore HaCS, a thermal-aware low-overhead scheduling and tuning algorithm that leverages HaCS' flexibility to satisfy performance and temperature constraints must be designed.

In this paper, we address the aforementioned challenges by evaluating design choices for a multicore heterogeneous and configurable system (HaCS) where the executing applications are unknown a priori. To enable thermal management in HaCS, while simultaneously satisfying performance and energy needs, we propose a novel thermal-aware scheduling and tuning (TaSaT) algorithm for HaCS that requires no a priori knowledge, or offline profiling of the applications. Due to the cache's significant impact on overall system performance and energy, we focus on cache optimization in this work, and show that a heterogeneous, configurable cache offers substantial advantages for thermal-aware runtime optimizations. We design our TaSaT algorithm as part of a hardware cache tuner that can hold scheduling and tuning information of an arbitrary number of applications using a least recently used (LRU) replacement policy.

Our results reveal that TaSaT reduces the system's temperature, execution time, and energy by 16.53%, 10.74%, and 22.87% respectively, as compared to a base commercial off-the-shelf (COTS) system. Additionally, TaSaT reduced the system temperature by 4.70%, as compared to prior work, and imposed a tuning-time performance overhead of 7.77% as compared to an

GLSVLSI '18, May 16–18, 2018, Chicago, IL, USA

ACM ISBN 978-1-4503-5724-1/18/05…$15.00
https://doi.org/10.1145/3194554.3194576

ideal system with a priori knowledge of applications' best performance configurations.

2 RELATED WORK

Much prior work contributed novel heterogeneous systems (e.g., [5][16][17]), configurable systems (e.g., [10][11][19][23]), heterogeneous and configurable (e.g., [3][4]systems), and associated scheduling (e.g., [14][15][21]) , tuning (e.g., [1][11]), and scheduling and tuning algorithms (e.g., [3][4]). Given these expansive research areas, for brevity, we only discuss the most related works on heterogeneous and configurable systems (HaCS) and associated runtime scheduling and tuning algorithms.

2.1 Heterogeneous and Configurable Systems (HaCS)

To achieve coarse- and fine-grained adherence to energy, Alsafrjalani et al. [4] designed a quad-core heterogeneous and configurable system (HaCS). The cores featured fixed, heterogeneous cache sizes, and enabled coarse-grained adherence to performance constraints. The cores also featured configurable cache line size and associativities, and enabled fine-grained adherence to performance constraints. The authors designed a scheduling and tuning algorithm that dynamically profiled and mapped the applications to cores that consumed the lowest energy. Once the applications were scheduled, a tuner adjusted the core's cache line size and associativity to best match the application's cache requirements. However, the authors did not evaluate the thermal impacts of their proposed work.

Adegbija et al. [1] considered the thermal impact while optimizing the performance and energy of configurable systems. The authors used an evolutionary algorithm to determine Pareto optimal cache configurations and clock frequencies for different application phases. However, the proposed technique required the executing applications to be profiled and classified offline, at design time, which limited the technique from being applied to general purpose applications where the executing applications are unknown a priori. Furthermore, the proposed technique only considered a single-core system, and did not take into account application-to-core scheduling.

2.2 Scheduling and Tuning Algorithms

Yeo et al. [22] contributed a thermal-aware scheduler based on a thermal prediction model. The prediction model used the applications' runtime execution characteristics and classification derived from offline thermal profiling to predict the applications' thermal behaviors. Using this prediction information and a core-based thermal model, a thermal-aware scheduler migrates the application to a core that would take the longest to reach the temperature threshold. Although this approach is applicable to multicore systems, the work was evaluated on homogeneous multicore systems, which have much less variation in resource constraints as compared to heterogeneous systems. Furthermore, since this approach required offline application profiling, it is not scalable to applications that are unknown during design time.

To extend scheduling to unknown applications, Alsafrjalani et al. used a dynamic scheduling and tuning algorithm for a HaCS [4]. The algorithm profiled the applications dynamically, enabling the system to scale to an arbitrary number of applications. When the applications best (lowest) energy core was determined, the algorithm scheduled the application to that core and tuned the core's cache line and associativity. However, if the best core was not available the algorithm attempted to utilize a not-best, however idle, core, such that the total static and dynamic energy was minimized. However, the algorithm did not account for performance loss if applications needed to halt until the best, or near-best core was available.

To address performance bottleneck, Alsafrjalani et al. [3], developed a performance/quality aware scheduling and tuning algorithm for HaCS. To discover low energy core and configuration while maintaining performance, the algorithm profiled the application on the cores, and explored tunable configurations, using performance ordering. The algorithm explored cores/configurations that had the highest performance expectation and continued until a core/configuration degraded the application's performance below a predefined performance threshold. Even though the proposed approach was applicable to any HaCS and arbitrary number of applications, the approach did not consider temperature gain/loss as a byproduct of determining the best configurations for energy or performance.

In this paper, we present the first—to the best of our knowledge—dynamic thermal-aware scheduling and tuning (TaSaT) algorithm for heterogeneous and configurable systems (HaCS). TaSaT requires no a priori knowledge of executing applications, and determines system configurations that can optimize temperature, energy, or performance, while incurring minimal, tuning-time overhead.

3 Thermal-Aware HaCS Hardware Design

Given the expansive design space (available configuration options) to construct a thermal-aware HaCS, this section details the steps of determining the heterogeneous parameters, configurable parameters, and our hardware architecture and the tuner hardware requirements.

3.1 Heterogeneous and Configurable Parameters

To design a HaCS system that adheres to performance and energy constraints, and to evaluate and compare our system to prior work [4], we used configurable caches with configurations that have high impact on performance and energy [23]. To gain insight into coarse- and fine-grained performance-energy tradeoff, we used a design space of 729 configurations, comprising different cache sizes, line sizes, and associativities. The design space comprises the cross product of instruction and data caches with the following parameters: size of 32, 16, and 8KB, line sizes of 64, 32, and 16B, and associativities of 4, 2, and 1.

Since our goal is to reduce temperature while adhering to

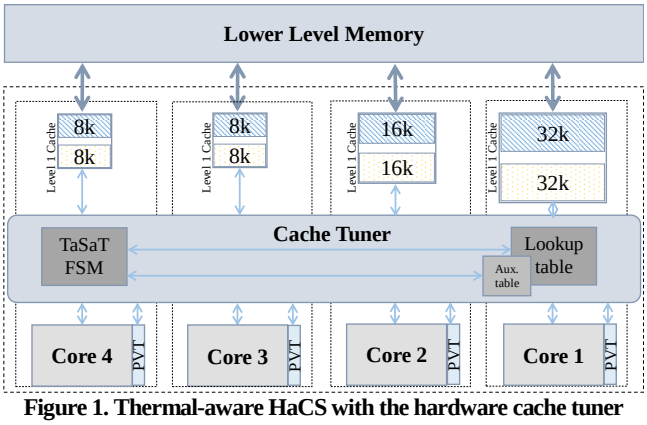
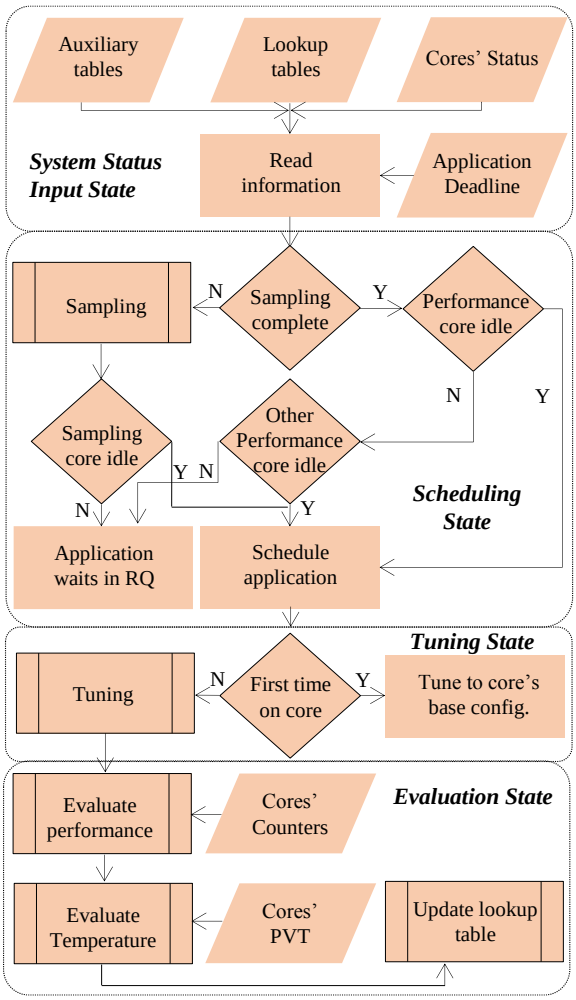


Figure 1. Thermal-aware HaCS with the hardware cache tuner

performance constraints, our system must feature configurations that simultaneously achieve both temperature and performance optimization goals. For coarse-grained performance optimization, we designed each core in our HaCS system with a fixed, different cache size. To complement coarse-grained optimization with fine-grained optimization, each core features configurable line size and associativity. As a first step, we explored the cache parameters that provided the best performance-temperature tradeoffs by exhaustively searching our design space, using a set of embedded applications [9].

Our exhaustive evaluation revealed that the different configurations had substantially different, and potentially conflicting, impacts on performance, energy, and temperature. However, since our approach features an online scheduling and tuning algorithm, exploring 729 configurations can impose significant performance and energy overhead, due to extreme configurations that significantly degrade performance, energy, and/or temperature. For example, our exhaustive search showed 48% and 49% performance and energy overheads, respectively.

Furthermore, using the exhaustive evaluation, we were able to narrow down the best cache configurations for performance, energy, and temperature. We observed that a wide range of configurations with sizes from 8KB to 32KB achieved the best performance and energy for the different applications in our experiments (the experimental setup is detailed in Section 6). However, the best configurations for temperature optimization only included the 8KB cache. Based on these observations, our HaCS system features cores with 8, 16, and 32KB cache sizes, as shown in Figure 1. We empirically determined that the 8KB was the most likely to minimize temperature, at the expense of performance or energy. Thus, since our work targets temperature optimization, we designated two cores (out of four) as 8KB to provide more opportunities for scheduling to the lowest-temperature core. The 16KB and 32KB caches allow temperature optimization to be traded off, when necessary, in favor of performance or energy.



Algorithm 1 Thermal-aware Scheduling and Tuning (TaSaT) Algorithm

3.2 Hardware Tuner Requirements

To enable dynamic tuning, we use a low-overhead hardware cache tuner featuring dedicated buses connected to the cores' power, voltage, and thermal (PVT) sensors [6]. The tuner also implements TaSaT algorithm's finite state machine (Section 4), and a lookup table to store tuning information (e.g., explored configurations, applications' best configurations). The tuner imposes 322 cycles time overhead [2], an insignificant overhead compared to the number of cycles required for our tuning interval of 1 million instructions [3]. During each tuning interval, the tuner measures/calculates an application's energy, performance, and/or temperature, in order to determine which configuration to explore during the next interval. This interval is long enough to warm up and stabilize the caches [3].

The number of bits m required to store scheduling and tuning information per application is given by:

$$m = s + (e + p + t + f) * \lceil log_2(c) \rceil \quad (1)$$

where e, p, and t are 32-bit values that store the energy, clock cycles, and temperature information, s is a 1-bit *stop-tuning* flag, f is a 2-bit *performance and temperature* flag, and c is the number of configurations in the design space. Furthermore, the number of bits n required for an auxiliary table to store the applications' best configurations is given by:

$$n = a * \lceil log_2(c) \rceil \quad (2)$$

where a is an 8-bit value that stores the best configuration information for up to 256 simultaneously applications. If more space is needed for additional applications, the tuner uses LRU policy to update the lookup and/or auxiliary table(s).

Thus, the lookup and auxiliary tables require $m * n$ bits (728 bits in our case), which is insignificant, as compared to the level 1 cache sizes of 8KB to 32KB. Prior work [2] revealed that a hardware cache tuner with more storage requirements than our design (e.g., per-configuration energy consumption was also stored) imposed only a 4.7% area overhead in a very small MIPS M4K processor; our tuner's area overhead will be significantly less.

4 TaSaT FOR THERMAL AWARE HaCS

4.1 Overview

Algorithm 1 depicts our thermal-aware scheduling and tuning (TaSaT) algorithm's flow chart. The algorithm has four states: System Status Input, Scheduling, Tuning, and Evaluation states. For each application in the ready queue, the algorithm determines the core with the best cache size to schedule the application to, tunes the core's cache line size and associativity, evaluates the performance and temperature impact of this execution, and finally updates the lookup tables.

4.2 TaSaT Algorithm States

4.2.1 System Status Input State. When an application is placed in the ready queue to be executed, TaSaT starts/resumes the scheduling and tuning process based on the information in the system status input state. TaSaT checks the auxiliary table for information about the application's best core, cache size, and associativity. If the auxiliary table contains this information, TaSaT attempts to schedule the application to the best core and directly tune the core cache line size and associativity to the best configuration. Otherwise, TaSaT uses the cores' status information (e.g., idle, busy), application deadline information, which may be designer-specified or implicitly specified by the executing application due to input data stream processing requirements, and the application's tuning information from the lookup table, and transitions to the scheduling state.

4.2.2 Scheduling State. In the scheduling state, the algorithm determines which performance core to schedule the application to. A performance core is a core that does not violate an application's deadline constraint. If sampling is complete, the performance core(s)—there may be more than one for each application—are known, and the algorithm attempts to schedule the application to an idle performance cores. If no performance core is available, the application remains in the ready queue. If sampling is not complete, the algorithm uses a sampling sub-process to sample the application on all/remaining cores.

Since cache size has the highest impact on performance [3][23], the sampling sub-process starts with the core with the largest cache size. The application is executed for one tuning interval on each core in descending order of cache sizes and with the base line size and associativity. If the core does not violate the deadline, the algorithm flags the core as a performance core. When the algorithm schedules an application to a core, the algorithm transitions to the tuning state.

4.2.3 Tuning State. Using the information from the lookup table, the algorithm tunes the core's cache line size and associativity. If there is no tuning information in the lookup table (i.e., the application is executing on the core for the first time), the algorithm starts tuning from the largest line size and associativity, since these values are likely to not degrade performance [3]. For subsequent tuning intervals, the algorithm calls a tuning process that explores other cache line sizes and associativities, in a descending performance ordering (i.e., explores all line sizes first).

The tuning process employs a pseudo hill-climbing heuristic to explore the cache line size and associativity values. To avoid local performance degradation, the process uses a predetermined threshold value, *w*, that tracks the number of times the tuning resulted in degraded performance. The process tunes to a lower cache size and associativity for one tuning interval. Each time the application is scheduled to a core, the tuning process explores a lower cache line size and associativity, as long as the number of performance degradations has not exceeded *w*. Once the cache line size and associativity are determined, the algorithm begins the evaluation state.

4.2.4 Evaluation State. To evaluate the core/configuration performance, in the evaluation state, the algorithm uses an evaluate performance sub-process. The process reads in the core counters to measure the time elapsed for the application to execute and compares that time to the application deadline. If the application exceeded the deadline, the algorithm updates the performance flag bit, f_0, of that configuration to 1 (marked as a sub-performance configuration). Using the information from the lookup table, the process also determines the number of sub-performance configurations explored thus far. If this number exceeds *w* then the process sets the stop tuning flag, *s*, to 1 for the application.

Similarly, to evaluate the temperature reduction, the algorithm calls an evaluate temperature sub-process. The process reads in the cores' PVT and compares these values to the values of previously explored configurations. If the temperature value is lower than of the previously explored configuration(s), the process sets the temperature-saving flag, f_1, of that configuration for the application.

Using the results from the performance and temperature evaluation, the algorithm determines if the currently explored configuration results in acceptable performance and temperature reduction. The algorithm then update/store this information in the lookup table for the algorithm to use in subsequent executions.

5 EXPERIMENT SETUP

To evaluate our system, we used a quad-core HaCS architecture (Section 3), and ran seventeen applications that represent common embedded systems applications [9]. The quad-core HaCS features cores with separate level one (L1) instruction and data caches of fixed sizes 8, 16, and 32KB. The caches' line sizes and associativities are tunable to any combination of 8, 16, and 32B, and 1-, 2- and 4-way, respectively. We modeled our quad-core system similar to the ARM Cortex A9, consisting of a 4-width out-of-order issue processor with 8 pipeline stages, 45 nm technology, and a clock frequency of 2GHz. The PVT are modeled after ARM's Juno Development SoC [6].

We used the EEMBC [9] Automotive benchmarks and six MiBench [12] benchmarks selected to represent different application domains. The benchmarks were specific compute kernels performing specific tasks in different application domains, including networking, image processing, security, etc. To obtain performance results, we executed the application on gem5 simulator [7], to obtain energy results, we fed the simulation results output McPAT [18], and we used Hotspot 5.0 [20] as the thermal modeling tool. To model fan-less systems, we designed our system with a heat sink of 1mm and spreader 0.1 mm thickness, and we set the convection resistance to 4K/W.

To simulate a load environment, we generated a queue of 5000 applications from our benchmark applications with a designated ready/arrival time and deadline. To generate a realistic arrival rate, we used a Gamma distribution centered around a predetermined execution time

$$X = \frac{\sum_{i=1}^{A}\sum_{j=1}^{c} x(i,j)}{A*C*c} \quad (3)$$

where *A*, *C*, and *c*, are the numbers of applications, cores, and configurations, respectively, and *x(i,j)* is the execution time of application *i* using configuration *j*. To generate a realistic deadline for an application, we designated the application's average execution time using all configuration as the deadline.

6 EVALUATION METHODOLOGY

We evaluated the performance, temperature, and energy optimizations while executing a queue of 5000 applications (Section 5) using six systems, all of which featured quad-core processors. However, to evaluate the system tradeoffs, and scheduling and tuning overhead, the systems differed in the number of cache configurations, scheduling algorithms, and a priori knowledge of applications.

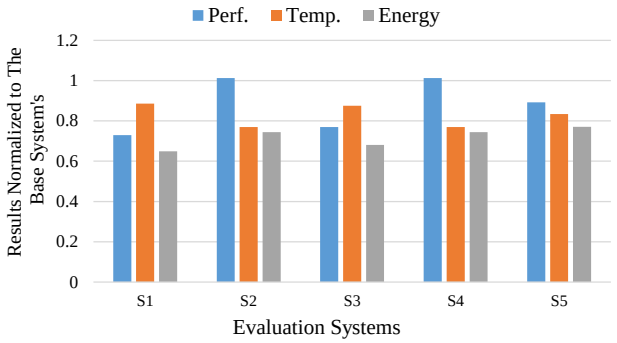


Figure 2. Performance, Temperature, and Energy of all systems, normalized to the Base System's

Table 1 depicts the six systems, including the cache configurations, a priori knowledge of applications (and the applications' deadlines), and the scheduling and tuning algorithms used in each system. The base system represents a commercial of-the-shelf (COTS) system with fixed 32KB-4W-64B cache and a first-come first-served (FCFS) scheduling. The configuration column represents the cache configurations featured in the system. S1 and S2 caches comprised the complete design space of 729 configurations, while S3-S5 comprised the same subset of 81 cache configurations (Section 3.1). All systems, except S5, had a priori knowledge of the applications' best performance and/or temperature configuration; i.e., we assumed there is no scheduling and tuning overhead. S1—S4 scheduled the applications using FCFS to the core with the best performance (P) or temperature (T) configuration, whereas S5 used our TaSaT algorithm to make scheduling and tuning decisions.

We compared systems S1, S2, S3, S4, and S5 to the base system for performance, temperature, and energy optimizations. We also compared S1 to S3 and S2 to S4 to measure performance and temperature degradation, respectively, when a configuration subset is used instead of the complete design space. Since S1—S4 assume a priori knowledge of executing applications, there is no scheduling and tuning overhead; the overhead results from using configuration subsets rather than the full design space.

To obtain S5's performance and temperature tradeoffs with respect to an optimal system with a priori knowledge of application we compared S5 to S1 and S2, respectively. We also evaluated the S5's scheduling and tuning overhead by comparing S5's optimization potential to S3 and S4. Furthermore, since S3 comprised a subset of configurations, the systems represent prior work [3][4] in which performance is prioritized. We compared the temperature and energy savings of S5 to S3 to obtain tradeoffs as compared to prior work, when temperature is prioritized.

7 RESULTS AND ANALYSIS

This section presents the performance, temperature, and energy results and analysis based on our experiment setup and evaluation methodology.

Table 1: Evaluation Systems

System\Setup	$Config.	A Priori	Sched. Al.
Base	32-4-64	yes	FCFS
S1	complete	yes	FCFS-P
S2	complete	yes	FCFS-T
S3	subset	yes	FCFS-P
S4	subset	yes	FCFS-T
S5	subset	no	TaSaT

7.1 Performance

Figure 2 depicts the average performance, temperature, and energy improvements of S1, S2, S3, S4, and S5 normalized to the base system. Values below one correspond to less time, temperature, and energy, as compared to the base system. S1, S3, and S5 resulted in 27.09%, 23.07%, and 10.74% better performance, as compared to the base system, since these systems prioritized performance optimization during scheduling. However, S2, and S4, degraded the performance by 1.33% and 1.30%, respectively, since these systems used FCFS-T, which prioritized lower temperature over better performance. Since S2 and S4 had a priori knowledge of the applications' best temperature configurations, the applications were scheduled to the best-temperature cores, which resulted in performance degradation. Furthermore, even though S2 had all the cache configurations, while S4 had a subset of configurations, both systems achieved similar performance for the applications; thus, the eliminated configurations in S4 did not affect the system's performance.

To extend our performance analysis, we recorded how much time the systems' scheduling and tuning decisions took. Figure 3 depicts the average number of deadline misses that occurred while executing the applications. The applications missed the deadline 25% of the time in the base system, since the base system scheduled the applications to first available cores, and all cores offered the same configuration. Our analysis revealed that although the base system's caches featured the largest cache size, and the best average case performance, the base configuration was not the best for all of the applications. Executing some of the applications (e.g., basefp) on the base configuration degraded performance; as a result, other applications were prevented from meeting their deadlines. In S1 and S3, no applications missed their deadlines, since both systems offered the complete design space and prior knowledge of the best configurations.

Since S1 and S3 offered the complete and subsetted design space, respectively, the subsetted design space contained enough best and near-best performance configurations. However, S3, S4 and S5 incurred deadline misses while executing the applications, since S3 and S4 prioritized temperature and S5 explored the design space during runtime and accrued runtime overhead. Since S2 and S4 offered the complete and subsetted design space, respectively, the subsetted design space contained the same best and near best performance configurations, while providing best temperature configurations. This result also suggests that even though execution times on S2 and S4 were similar (Figure 2), S2 and S4 may impact the performance of a hard real-time performance system differently (Figure 3).

Finally, the applications in S5, which represent our HaCS with TaSaT algorithm, missed the deadline on average of 7.77%. Since S5 had no a priori knowledge of the applications, the scheduling and tuning overhead accounted for the deadline misses of some of the applications. The 7.77% tuning time overhead will amortize over time and is acceptable in soft real-time systems [3]. Furthermore, as compared to prior work that prioritized performance *and* had prior knowledge of the applications' best configurations (S3), S5 decreased performance by 16.03%, since S5 used multiple performance cores for an application. Although S5 degraded the performance, that did not necessarily translate to a substantial increase in applications missing deadlines and is then

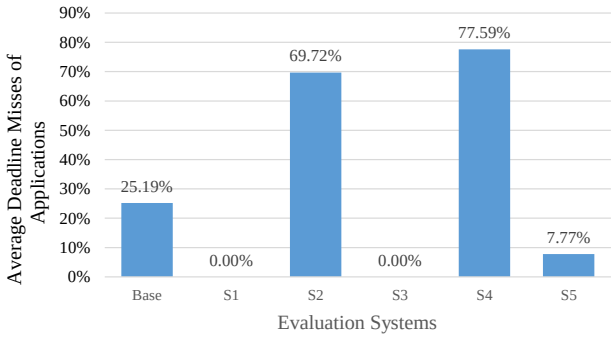


Figure 3. The average number of applications' executions that exceeded the deadline for all the application's in the queue

within 7.77% of an ideal system while saving 4.70% temperature (Section 7.2), as compared to S3.

7.2 Temperature

Systems S1, S2, S3, S4, and S5 reduced the temperature by 11.34%, 23.01%, 12.42%, 23.07%, and 16.53%, respectively, as compared to the base system, which had an average temperature of 89.55° C. Since S2 and S4 prioritized temperature, these two systems resulted in the highest temperature savings among the five systems. Also, S1 and S3 resulted in less temperature savings than S2 and S4, since S1 and S3 systems used FCFS-P, which prioritized higher performance over better temperature. Since S1 and S3 had a priori knowledge of the application's best performance configurations, the applications were scheduled to the best-performance core, which resulted in increased average temperatures as compared to S2 and S4. Moreover, we observed that even though S2 and S4 offered the complete and subsetted design space, respectively, there was no impact on the temperature; both systems' temperature savings were similar.

Furthermore, we observed that S5's temperature savings of 16.53% was less than that of S2 and S4, since S5's balanced between temperature reduction and performance, while S2 and S4 prioritized temperature only. Additionally, compared to an ideal system that prioritizes temperature (S4), S5, traded off 8.64% in temperature savings for 11.90% performance improvement, and 90.65% fewer missed deadlines.

7.3 Energy

Systems S1, S2, S3, S4, and S5 resulted in 35.10%, 25.52%, 31.90%, 25.53%, and 22.87% energy savings, respectively, as compared to the base system. Although S1 and S3 prioritized performance, the systems reduced the execution time substantially, with a less substantial power increase, which resulted in an overall energy reduction. Overall, our approach did not incur any energy overheads as compared to the base system.

8 CONCLUSION AND FUTURE WORK

In this work we presented a thermal-aware scheduling and tuning (TaSaT) algorithm for reduced temperature in heterogeneous and configurable systems (HaCS). Our algorithm required no a priori knowledge of applications and can be scaled to any HaCS and applications. TaSaT explored the design space using performance order configurations and obtained 16.53% and 22.87% temperature and energy reduction, respectively, as compared to a base system. The algorithm imposed tuning-time performance overhead of 7.77%, as compared to an ideal system with a priori knowledge of applications. Furthermore, our algorithm reduced temperature by 4.67% compared to prior work. Future work will extend our algorithm to multi-level cache optimization and multi-objective optimization of temperature, performance, energy, and energy-delay product (EDP).

REFERENCES

[1] Adegbija, T.; Gordon-Ross, A., "*Thermal-aware phase-based tuning of embedded systems*," Great Lakes Symposium on VLSI (GLSVLSI), 2014

[2] Adegbija, T.; Gordon-Ross, A.; Rawlins, M., "*Analysis of cache tuner architectural layouts for multicore embedded systems*," Int. Con. on Performance Computing and Communications, 2014.

[3] Alsafrjalani, M., H.; Gordon-Ross, A., "*Quality of service-aware, scalable cache tuning algorithm in consumer-based embedded devices*," International Great Lakes Symposium on VLSI (GLSVLSI), 2016

[4] Alsafrjalani, M., H.; Gordon-Ross, A., "*Dynamic Scheduling for Reduced Energy in Configuration-Subsetted Heterogeneous Multicore Systems*," Int. Con. on Embedded and Ubiquitous Computing, 2014

[5] ARM Ltd., big.LITTLE Technology, White Paper: http://www.arm.com/files/pdf/big_LITTLE_Technology_the_Futue_of_Mobile.pdf

[6] ARM Ltd., https://developer.arm.com/products/architecture/a-profile/docs/100113/latest/hardware-description/juno-arm-development-platform-soc

[7] Binkert, N.; et al., "*The gem5 simulator*," Computer Architecture News, 2011

[8] Donald, J.; Martonosi, M., "*Techniques for Multicore Thermal Management: Classification and New Exploration*," International Symposium in Computer Architecture (ISCA), 2006.

[9] EEMBC. The Embedded Microprocessor Benchmark Consortium http://www.eembc.org/benchmark/automotive _sl.php, Sept. 2013

[10] Folegnani, D.; Gonzalez, A., "*Energy-effective issue logic*," Int. Symp. on Computer Architecture (ISCA), 2001

[11] Gordon-Ross, A.; Vahid, F., "*A Self-Tuning Configurable Cache*," IEEE Design Automation Conference (DAC), 2007

[12] Guthausch, M., R.; et al., "*Mibench: a free, commercially representative embedded benchmark suite*," IEEE Workshop on Workload Characterization, 2001.

[13] Heo, S.; Barr, K.; Asanovic, K., "*Reducing Power Density through Activity Migration*," Int. Symp. on Low Power Electronics & Design (ISLPED), 2003

[14] Joao, José A., Aater Suleman, M., Mutlu, O., Patt, N., "*Utility-based acceleration of multithreaded applications on asymmetric CMPs. SIGARCH,"* Computer Architecture News, 2013

[15] Kim, K.; Kim, D.; Park, C.; "*Real-time scheduling in heterogeneous dual-core architectures,"* International Conference on Parallel and Distributed Systems, 2006

[16] Kumar, R.; et al., "*Single-ISA heterogeneous multi-core architectures: the potential for processor power reduction*," 36th Int. Symp. on Microarchitecture, 2003

[17] Kumar, R.; Tullsen, D.; N. Jouppi, N.; Ranganathan, P., "*Heterogeneous chip multiprocessors*," IEEE Computer, vol. 38, Nov. 2005

[18] Li, S.; et al., "*McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures*," Int. Symp. on Microarchitecture, 2009

[19] Malik, A.; Moyer, B.; Cermak, D., "*A low power unified cache architecture providing power and performance flexibility,"* Int. Symp. on Low Power Electronics and Design (ISLPED), 2000

[20] Skadron, K.; et al., "*Temperature-aware microarchitecture: modeling and implementation,"* Transactions on Architecture and Code Optimization, 2004.

[21] Van Craeynest, K.; Jaleel, A.; Eeckhout, L.; Narvaez, P.; Emer, J.; "*Scheduling heterogeneous multi-cores through performance impact estimation (PIE),*" 39th Int. Symp. on Computer Architecture (ISCA), 2012

[22] Yeo, I.; Kim, E.; J., "*Temperature-aware scheduler based on thermal behavior grouping in multicore systems*," Conference on Design, Automation and Test in Europe (DATE), 2009

[23] Zhang, C.; Vahid, F.; Najjar, W.; "*A highly configurable cache architecture for embedded systems*," Int. Sym. on Computer Architecture (ISCA), 2003