Thermal-aware Phase-based Tuning of Embedded Systems

Tosiron Adegbija and Ann Gordon-Ross*
Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL 32611, USA
tosironkbd@ufl.edu & ann@ece.ufl.edu
Also affiliated with the NSF Center for High-Performance Reconfigurable Computing (CHREC) at UF

ABSTRACT
Due to embedded systems' stringent design constraints, much prior work focused on optimizing energy consumption and/or performance. However, since embedded systems have fewer cooling options, rising temperature, and thus temperature optimization, is an emergent concern. We present thermal-aware phase-based tuning—TaPT—that determines Pareto optimal configurations for fine-grained execution time, energy, and temperature tradeoffs. Results show that TaPT reduces execution time, energy, and temperature by as much as 5%, 30%, and 25%, respectively, while adhering to designer-specified design constraints.

Categories and Subject Descriptors
B.3.2 [**Hardware**]: Memory Structures: Design Styles – *cache memories.*

Keywords
Dynamic thermal management, phase-based tuning, thermal-aware tuning, energy savings, configurable caches, dynamic optimization.

1. INTRODUCTION AND MOTIVATION
Embedded systems have been the focus of much optimization research due to these systems' pervasiveness and intrinsic design constraints. Since most embedded systems are battery operated, reducing energy consumption without significantly degrading system performance is a key design optimization. However, temperature is also a growing issue in embedded systems optimization research since most embedded systems have fewer cooling options as compared to general purpose computers due to area/size, cost, and energy constraints. Most embedded systems only dissipate heat by passive convection, thus necessitating efficient thermal management methodologies.

Increased chip temperature in an embedded system can result in increased cooling costs, reduced mean time to failure (MTTF), reduced reliability, etc. Increased temperature can also lead to thermal emergencies, which can result in an exponential increase in leakage power and compounding temperature increases (i.e., thermal runaway), leading to permanent chip damage. To address these issues, several dynamic thermal management (DTM) techniques have been proposed for managing chip temperature, most of which leverage clock gating [5], dynamic voltage scaling (DVS), dynamic frequency scaling (DFS), dynamic voltage and frequency scaling (DVFS) [19], and task migration [11].

DTM techniques have been successful in reducing chip temperature, however, some DTM techniques consider temperature in isolation [16], which may adversely affect other design objectives, such as execution time and/or energy consumption. Furthermore, the applications' execution characteristics (e.g., cache misses, instructions per cycle (IPC), branch mispredictions, etc.) can affect the temperature [21], and even though some DTM techniques consider inter-application characteristic variations [12], these techniques do not consider intra-application characteristic variations, which can significantly limit optimization potential [7].

To increase optimization potential using fine-grained system tuning (i.e., specialization) to varying application characteristics without incurring significant tuning overhead in terms of energy, area, and/or performance, previous work has proposed phase-based tuning [6]. A phase is a length of execution where an application's characteristics remain relatively stable, and therefore the best system configuration, or specific parameter values (e.g., cache size, associativity, line size, clock frequency, etc.), that adhere to the design constraints also remain relatively stable. Phase-based tuning requires configurable hardware with tunable parameters where the parameter's values can be specified/changed during runtime. Phase-based tuning also requires a mechanism to evaluate the application's characteristics to determine the best system configuration for each phase of execution to best satisfy design objectives (e.g., minimize energy, execution time, etc.) and design constraints (e.g., temperature thresholds). Previous work showed that phase-based tuning significantly reduced energy consumption in embedded systems [6]. For example, Gordon-Ross et al. [8] showed that phase-based cache tuning saved as much as 62% of the memory access energy. However, little work studied the combination of phase based-tuning and DTM.

Since prior work showed that phase-based cache tuning significantly impacts energy consumption and execution time, and DTM techniques can significantly impact temperature, energy consumption, and execution time, we combine phase-based cache tuning and DFS for fine-grained and efficient temperature, energy, and execution time optimization. However, since optimizing one design objective may adversely impact the other design objectives, combining these techniques presents a multi-objective optimization problem. The solution to a multi-objective optimization problem is the Pareto optimal configuration set, which enables designers to choose the system configuration that best meets the design constraints.

We present thermal-aware phase-based tuning (TaPT), which dynamically determines the Pareto optimal system configurations trading off execution time, energy, and temperature design objectives. TaPT is based on the strength Pareto evolutionary algorithm II (SPEA2) [23], which is a well-known and effective evolutionary algorithm for solving multi-objective optimization problems. We modify SPEA2 to implement phase-based tuning and consider designer-selected priority settings. These priority settings

GLSVLSI '14, May 21 - 23 2014, Houston, TX, USA

http://dx.doi.org/10.1145/2591513.2591586

allow designers to prioritize a design objective, thus trading off/degrading non-prioritized design objectives to increase the prioritized design objective based on design constraints. TaPT's runtime automation aids designers in adhering to design constraints with no design time effort. TaPT leverages previously proposed/existing configurable hardware, thus minimizing the additional hardware overhead with respect to these prior techniques. Experimental results show that compared to using the same system configuration throughout an application's execution, TaPT reduces execution time, energy consumption, and temperature by as much as 5%, 30%, and 25%, while adhering to designer-specified design constraints.

2. BACKGROUND AND RELATED WORK

Since much previous work focuses on phase-based tuning [1][6][8] and DTM [5][11][19] separately, and to the best of our knowledge, our work is the first to combine phase-based tuning and DTM, we present related work and background in these two areas. We also present background and key concepts for SPEA2, which serves as the basis for TaPT.

2.1 Phase-based Tuning and DTM

To facilitate phase-based tuning, hardware- or software-based phase classification partitions an application's execution into intervals, measured by the number of instructions executed. Intervals showing similar characteristics can be clustered into phases. Balasubramonian et al. [3] used cache miss rates, cycles per instruction (CPI), and branch frequency characteristics to detect changes in application characteristics for cache tuning, and found that these characteristics were effective for phase classification. Since we utilize cache tuning in this work, for brevity, we limit our review to phase-based cache tuning.

Phase-based tuning can leverage any configurable cache architecture (e.g., [7]) and tuning method to search the configuration design space, which consists of all the different system configurations/combinations of tunable parameter values. Zhang et al. [22] proposed a low energy and area overhead configurable cache architecture that provided runtime-configurable total cache size, associativity, and line size using a small, hardware-settable bit-width configuration register. Motorola's M*CORE processor [14] provided per-way configuration using way management, which allowed ways to be shut down or designated as instruction only, data only, or unified.

A major challenge of phase-based tuning is tuning the configurable hardware to the best configuration for each phase without incurring significant tuning overhead. Gordon-Ross et al. [6] presented cache design space exploration heuristics that when used for phased-based tuning, realized as much as 39% energy savings on average as compared to non-phase-based tuning (i.e., using a single configuration for the entire application). Hajimir et al. [10] presented a dynamic programming-based algorithm to find the best cache configuration for each phase. However, these methods only focused on energy savings and did not consider thermal issues.

To reduce chip temperature dissipation, several DTM techniques have been proposed. Brooks et al. [5] investigated clock gating, which turns off the clock signals during thermal emergencies. Heo et al. [11] proposed task migration, which migrated tasks from a hot core to a cooler core to avoid a thermal emergency. However, these works did not explicitly consider the tradeoffs between energy, temperature, and execution time, thus increasing the possibility of significantly degrading one design objective while optimizing other

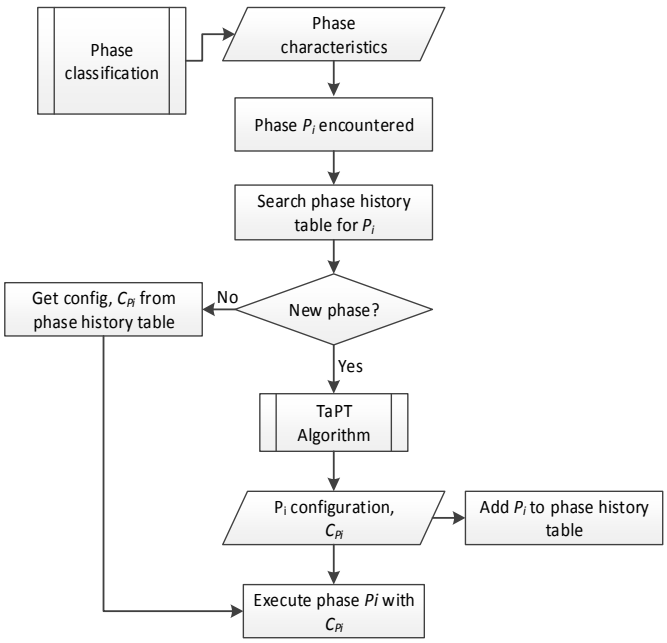

Figure 1. TaPT overview

design objectives. Furthermore, these methods were not phase-based and did not consider intra-application variations.

Our work differs from previous works by combining phase-based cache tuning and DFS to achieve Pareto optimal configurations that trade off execution time, energy, and temperature, thus achieving fine-grained multi-objective optimization.

2.2 SPEA2 Algorithm

Evolutionary algorithms leverage biological evolutionary concepts, such as population, reproduction, mutation, selection, etc., for efficiently determining Pareto optimal solutions to multi-objective optimization problems. The solution space consists of all of the possible solutions to the optimization problem, the population is a subset of the solution space, and the population's solutions are referred to as individuals. A solution's fitness dictates the solution's quality and represents how well the solution adheres to design constraints. Evolution iterates over successive generations of populations, where each evolution considers the population's individuals' finesses and replaces the least fit individuals with new solutions from the solution space, and interjects random solution mutations to create the successive generation.

Prior work shows that SPEA2 outperforms most other evolutionary algorithms for solving multi-objective optimization problems [23]. SPEA2 uses *elitism,* which maintains an external set of non-dominated solutions, called an *archive*. A solution is non-dominated (or Pareto optimal) if none of the design objectives can be improved without degrading another design objective. For example, given two configurations C_x and C_y, C_x dominates C_y (written as $C_x \succ C_y$) if and only if:

$$\forall i \in \{1, 2, \ldots, k\} : f_i(C_x) \geq f_i(C_y) \quad \exists j \in \{1, 2, \ldots, k\} : f_j(C_x) > f_j(C_y) \tag{1}$$

where k is the number of objectives and f_k represents the design objectives' objective functions, and $fk(C_x)$ characterizes how well C_x achieves the design objectives.

For brevity, we present an overview of SPEA2, and refer the reader to [23] for additional details. SPEA2 takes the solution space as input and outputs the Pareto optimal solution set. SPEA2 generates

Input: n, s, A_{size}, G, Q
Output: Pi's best configuration

```
0    t ← 0
1    for i ← 1 to s do
2        C_i ← rand() / s + 1
3    end
4    population is {C_1, C_2, …, C_s}
5    for j ← 1 to n do
6        D_j ← d(P_i, P_j)
7    end
8    A_msp ← archive(P_j) | D = min(D_j)
9    if n == 0 && t == 0 then
10       archive ← ∅
11   else if k > 0 && t == 0 then
12           archive ← A_msp
13       end
14   else
15       archive ← archive(t-1)
16   end
17   U ← population + archive
18   for (C_i ∈ U) do
19       fit(C_i) ← calculateFitness(C_i)
20   end
21   archive ← getNonDominated(U)
22   size(archive) ← A_size
23   if t == (G – 1) then
24       bestConfiguration(P_i) ← min(f(Q))
25       exit
```

an initial population and creates an empty archive and populates the first generation's archive with the population's non-dominated individuals. For subsequent generations, SPEA2 calculates the population's and archive's individuals' finesses, and populates the next generation's archive with the population's and archive's non-dominated individuals. When the maximum number of generations has been reached and/or number of solutions that satisfy the design objectives have been determined, the current archive contains the Pareto optimal set.

3. THERMAL-AWARE PHASE-BASED TUNING (TAPT)

TaPT leverages several fundamental assumptions based on mechanisms that have been widely studied and implemented in embedded systems [15][22]. Since our work is independent from the specific phase classification technique leveraged and prior work presents many phase classification techniques, we assume phase classification has already been performed and the applications' phases and the phases' instruction and data cache miss rates and IPC characteristics are input into phase-based tuning. We also assume that DFS is enabled, and the system has a temperature sensor, a hardware tuner [7] to orchestrate phase classification and implement TaPT, and a hardware-tunable cache with tunable size, associativity, and line size. In this section, we present an overview of TaPT and details of the TaPT algorithm.

3.1 Overview of TaPT

Figure 1 depicts an overview of TaPT. TaPT takes as input the classified phases' characteristics, which are output from phase classification. To minimize tuning overhead, a phase history table stores information about previously executed phases and the phases' best system configurations. When a phase P_i is executed, if P_i is in the phase history table, P_i has been previously executed (i.e., P_i is a not new phase) and the stored best system configuration C_{Pi} is used to execute P_i. If P_i is not in the phase history table (i.e., P_i is a new phase), TaPT determines P_i's best system configuration C_{Pi}, P_i is executed with C_{Pi} and C_{Pi} is stored in the phase history table for subsequent executions of P_i.

3.2 The TaPT Algorithm

TaPT contains three designer-specified priority settings, X, N, and T, which prioritize execution time, energy, or temperature minimization, respectively. These priority settings enable TaPT to efficiently determine the best system configuration C_{Pi} for a phase P_i while adhering to designer-specified constraints. The priority settings trade off the non-prioritized design objectives in favor of the prioritized design objective. For example, X trades off increased execution time and increased temperature for minimized energy. If the designer does not specify a priority, the priority setting defaults to S, which prioritizes energy delay product (EDP) minimization to account for both energy consumption and execution time while also reducing temperature and/or preventing a significant temperature increase. TaPT also allows the designer to associate a peak temperature threshold with each priority setting, such that TaPT determines Pareto optimal configurations that do not exceed the temperature threshold.

To ensure equal probability of selection for all configurations when generating the population, TaPT uses random uniform distribution, and on system startup, the initial archive is an empty set since there are no previously executed phases. TaPT generates P_i's archive from P_i's population's and archive's non-dominated configurations (Equation (1)) using the configurations' fitness and stores P_i's final archive in the phase history table. A configuration C_i's fitness is the sum of C_i's dominators' strengths, and a configuration's C_i's strength $S(C_i)$ is the number of configurations dominated by that configuration such that:

$$S(C_i) = |\{C_j \mid C_j \in P \cup A \;\forall\; C_i \succ C_j\}| \quad (2)$$

where P and A are P_i's population and archive, respectively. C_i's fitness $R(C_i)$ is:

$$R(C_i) = \sum S(C_j) \;\forall\; C_j \in P \cup A,\; C_j \succ C_i \quad (3)$$

where $R(C_i) = 0$ indicates that Ci is non-dominated.

To implement phase-based tuning, TaPT calculates the phase distances [1] between the currently executing phase P_i and all of the previously executed phases P_{i-1}, P_{i-1}, ..., P_{i-n}. The phase distance is the difference between two phases' characteristics, which the authors in [1] calculated using the normalized difference between the two phases' cache miss rates. However, since TaPT tunes multiple hardware parameters (instruction and data cache configurations and clock frequency), TaPT calculates the phase distance using the Euclidean distance between the instruction cache miss rate (iMR), data cache miss rate (dMR), and the instructions per cycle (IPC). The phase distance D between two phases P_i and P_j is:

$$D = \sqrt{(iMR_{Pi} - iMR_{Pj})^2 + (dMR_{Pi} - dMR_{Pj})^2 + (IPC_{Pi} - IPC_{Pj})^2} \quad (4)$$

TaPT uses the most similar phase's archive as the currently executing phase's initial archive, where the most similar phase has

the minimum D from P_i. Since phases with stable characteristics require similar configurations, using the most similar phase's archive as P_i's initial archive starts the TaPT algorithm with solutions that are presumably closer to P_i's Pareto optimal solutions, as compared to an archive from the randomly-generated initial population.

Algorithm 1 depicts the TaPT algorithm, which executes for each new phase P_i. The algorithm takes as input the number of previously executed phases n and a designer-specified population size s, archive size A_{size}, number of generations G, and priority setting Q. The algorithm outputs P_i's best system configuration. The product of s and G defines the maximum number of configurations explored/executed during tuning, which limits the tuning overhead, and A_{size} specifies the size of the archive and ensures that only the most fit configurations (Equations (2) and (3)) are stored in the archive. Given the nature of evolutionary algorithms, the archive does not necessarily contain the actual Pareto optimal solutions. In general, larger s and G values determine solutions that are closer to the Pareto optimal solutions, but also increase tuning overhead. Alternatively, smaller s and G values reduce tuning overhead, but may also determine configurations that are farther from the Pareto optimal solutions. We extensively evaluated different values of s, G, and A_{size} and observed that s and G values that explored 4% of the design space and A_{size} = 5 yielded an efficient balance between determining Pareto optimal solutions and reduced tuning overhead.

First, TaPT generates an initial population from the configuration space and calculates the phase distance D between the currently executing phase and all of the previously executed phases (lines 1 – 7). Next, TaPT initializes P_i's archive to P_i's most similar phase's archive (i.e., the phase with the minimum distance D from P_i) (lines 8 and 16). At system startup (n = 0), there are no previously executed phases (D = null), and the archive is initialized to an empty set (lines 9 – 10). For each generation, TaPT uses the previous generation's Pareto optimal set as the current generation's initial archive (line 15). TaPT calculates each population's and archive's configuration's fitness using Equations (2) and (3), and updates the current generation's archive with the non-dominated configurations (lines 17 – 21). TaPT maintains P_i's archive's size at A_{size} by discarding the least fit configurations or adding the most fit configurations from the population (line 22).

On the final generation, TaPT selects the *best configuration* from the archive that optimizes the specified priority setting (line 24). Finally, TaPT stores C_{Pi} in the phase history table (Figure 1) for P_i's subsequent executions.

3.3 Computational Complexity and Hardware Overhead

TaPT calculates $S(C_i)$ and $R(C_i)$ with worst-case time complexity $O(m^2)$, where m is the sum of the population and archive sizes, and calculates D with worst-case time complexity $O(n)$, where n is the number of previously executed phases. Thus, since these calculations dominate TaPT, TaPT results in minimal computation overhead. Furthermore, since TaPT utilizes previously proposed and implemented hardware, such as a DFS mechanism, phase history table, and configurable caches, TaPT imposes no additional hardware overhead as compared to prior work.

4. EXPERIMENTAL RESULTS

4.1 Experimental Setup

We evaluated TaPT's execution time, energy, EDP, and temperature savings by comparing a system that switches to the best configuration, as determined by TaPT, for each phase to a base system with a fixed system configuration. The base system had 32 Kbyte, 4-way private level one (L1) instruction and data caches with 64 byte line sizes, and a processor clock frequency of 2 GHz . This configuration is similar to current embedded systems (e.g., Motorola RAZR XT890 [15]), and thus serves as a good base comparison to a commercial off-the-shelf (COTS) system.

We modeled an embedded processor architecture, similar to the ARM Cortex A9 [2], consisting of a 4-width out-of-order issue processor with 8 pipeline stages and 45 nm technology. Our experiments represent state-of-the-art embedded systems, and our results and analyses extend to future and/or more complex systems (e.g., n-core processors, heterogeneous systems, etc.) because TaPT is independent of these system characteristics. The processor's

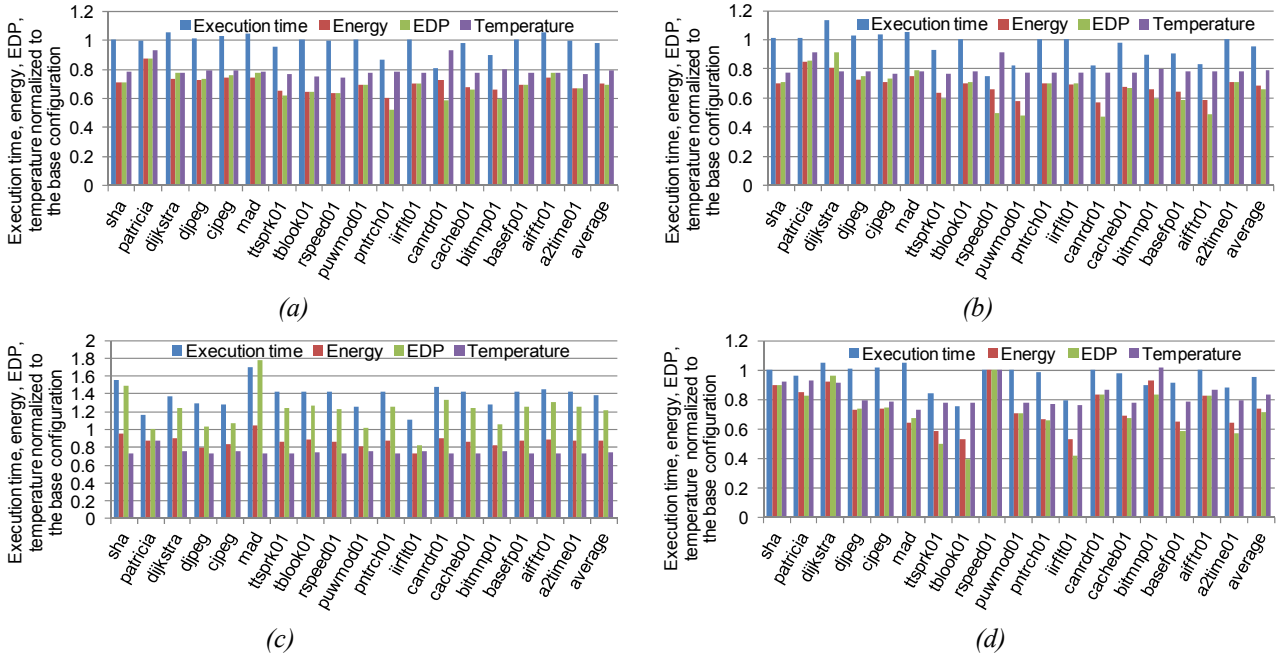

Figure 2. Execution time, energy, EDP, and temperature normalized to the base configuration for priority settings (a) *S*, (b) *N*, (c) *T*, and (d) *X*.

configurable L1 instruction and data cache sizes ranged from 8 to 32 Kbyte, line sizes ranged from 16 to 64 byte, and associativities ranged from 1- to 4-way, all in power-of-two increments. The processor offered seven clock frequencies ranging from 800 MHz to 2 GHz in 200 MHz increments. Given these parameter values, the design space contains 1,701 configurations.

We modeled the processor using GEM5 [4] and generated cache miss rates and core statistics, which we used to calculate the execution time. We also used these statistics to calculate the system's total energy consumption and EDP with McPAT [13]. We used Hotspot 5.0 [20] as the thermal modeling tool to measure the temperature using a floorplan and silicon chip area similar to the ARM Cortex A9 processor. We ran thermal simulations and sampled the application's power consumption at 10 ms intervals, similar to modern operating systems (e.g., Linux) [18]. Previous work [18] showed that this fine-grained sampling accurately depicted the application's temperature characteristics during execution. To simulate an embedded system without cooling mechanisms, such as a heat sink and/or spreader, we set the convection resistance to 4K/W and the heat sink and spreader thickness to 1 mm and 0.1 mm, respectively, which are considered negligible in Hotspot.

To model a variety of real-world embedded system applications, we used eighteen benchmarks: twelve EEMBC [17] Automotive benchmarks (the full suite could not be evaluated due to compilation errors) and six MiBench [9] benchmarks selected to represent different application domains. The benchmarks were specific compute kernels performing specific tasks in different application domains, such as networking, image processing, security, etc.

We implemented TaPT using Perl scripts to drive simulations and executed each phase once to completion. To implement phase classification, we ran execution trace simulations on each benchmark using GEM5 to generate cache miss rates and IPC statistics, and grouped intervals with similar characteristics as phases using variable-length intervals [7], which previous work found to be effective for phase classification. Since the benchmarks were specific compute kernels, our experiments revealed that the benchmarks exhibited relatively stable characteristics throughout execution. Without loss of generality, this characteristic stability enabled us to consider each kernel/benchmark as a different phase of execution.

To determine appropriate values for s, G, and A_{size}, we ran extensive experiments with different values and observed that s = 20, G = 3, and A_{size} = 5 achieved a good balance between Pareto optimal solutions and tuning overhead. These values explored only 4% of the design space, while larger values increased tuning overhead without significantly improving the Pareto optimal solutions and smaller values reduced tuning overhead, but achieved sub-Pareto-optimal solutions. s and G are system dependent and can be scaled appropriately for different design spaces.

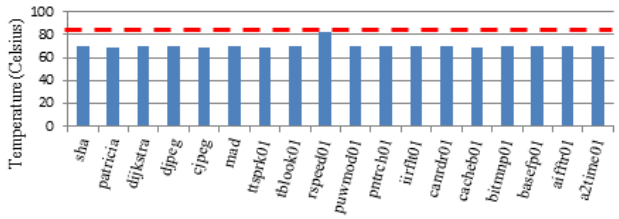

Figure 3. Peak temperatures with respect to a temperature threshold of 82°C (broken horizontal line).

To explore several diverse design objectives, we modeled all of TaPT's priority settings using these values of s, G, and A_{size}. To evaluate the impact of designer-specified temperature thresholds lower than the base configuration's average peak temperature of 89°C (determined by simulation), we evaluated empirically-determined high and low temperature thresholds set at 82°C and 65°C, based on the range of temperatures observed during simulation. The high 82°C threshold illustrates a system where the primary concern is for the temperature to be maintained below 82°C to prevent overheating damage, while the low 65°C threshold represents a strict temperature-constrained system to illustrate how maintaining a low temperature impacts the other objective functions.

4.2 Results

Figure 2 depicts the execution time, energy, EDP and temperature of the best configurations as determined by TaPT normalized to the base system configuration for a single execution of each benchmark/phase for each priority setting. Figure 2 (a) depicts a zero-designer-effort system, with a default priority setting S (EDP prioritization) and no temperature threshold. The results show average EDP, energy, execution time, and temperature reductions of 31%, 30%, 2%, and 21%, respectively, with maximum reductions of 48%, 35%, 19%, and 5%, respectively. For some phases, prioritizing EDP minimization only slightly reduced the temperature. For example, *candr01*'s EDP, energy, and execution time reduced by 40%, 27%, and 18%, respectively, while reducing the temperature by only 8%. However, other phases suffered increased execution time, up to 6%, to prioritize EDP minimization, but gained significant reductions in energy and temperature. For example, *mad*'s EDP, energy, and temperature reduced by 23%, 26%, and 21%, respectively, while increasing the execution time by 4%. In general, priority setting S minimizes EDP, and reduces the energy consumption and temperature for all phases, with only minor increases in execution time for some phases.

Figure 2 (b) shows that priority setting N (energy prioritization) and a temperature threshold of 82°C resulted in average execution time, energy, EDP, and temperature reductions of 4%, 31%, 34%, and 20%, respectively. Figure 3 illustrates the impact of a high temperature threshold, and depicts the phases' peak temperatures with respect to the threshold temperature 82°C. For all of the phases, the temperature never exceeded 82°C, because rather than minimizing temperature, TaPT maintained the temperature at or below 82°C, which allowed for further execution time, energy, and EDP reduction since the temperature threshold was relatively high.

Figure 2 (c) depicts a much lower temperature threshold set at 65°C and priority setting T (temperature prioritization). On average over all of the phases, the energy and temperature decreased by 13% and 25%, respectively. However, the execution time and EDP significantly increased by 39% and 22%, respectively. TaPT maintained a peak temperature for all the phases within 65°C to 68°C, however, to maintain this low peak temperature, TaPT traded off execution time and energy consumption. Increasing the temperature threshold to 70°C (results not shown for brevity) decreased the energy, EDP, and temperature by 27%, 26%, and 21%, respectively, while the execution time only increased by 2%. These results show TaPT's ability to trade off objective functions in order to adhere to design constraints. The results also show the extent to which some objective functions may be adversely affected in a multi-objective optimization problem where one of the objective functions is significantly constrained.

Figure 2 (d) shows that using priority setting X (execution time prioritization) with no temperature threshold decreased execution

time, energy, EDP, and temperature by 5%, 26%, 29%, and 16%, respectively. For example, TaPT significantly decreased *tblook*'s execution time, energy, EDP, and temperature by 24%, 47%, 60%, and 22%, respectively. However, for some phases there was no execution time decrease, such as *mad*, which increased by 5% while the energy, EDP, and temperature decreased by 36%, 32%, and 27%, respectively. Even though TaPT achieved significant execution time improvement for some phases, the base configuration performed well in terms of execution time for most phases. Thus, for those phases, TaPT determined configurations that did not significantly increase the execution time, while also reducing the energy and temperature. Therefore, even though TaPT emphasized execution time minimization in this experiment, since the base configuration was a high performing configuration on average, the relatively low execution time reduction was expected. In general, TaPT successfully achieved significant savings in terms of execution time, energy, and temperature while adhering to specified design constraints.

4.3 TaPT Exploration Time

We evaluated TaPT's exploration time by comparing how much time TaPT required to determine a phase's best configuration with how much time was required to determine the phase's best configuration using an exhaustive search of the design space. On average over all the phases, TaPT reduced the exploration time from 142 seconds to 6 seconds, with the longest and shortest exploration reductions being from 321 seconds to 13 seconds and from 25 seconds to 1 second, respectively. Thus, on average, TaPT reduced the exploration time by 96%, translating to a 25 times speedup in exploration time.

5. CONCLUSIONS

In this paper, we presented thermal-aware phase-based tuning, TaPT, which combines phase-based cache tuning and dynamic frequency scaling (DFS) to determine Pareto optimal configurations for different application phases of execution. We show TaPT's effectiveness in determining Pareto optimal configurations that significantly reduce execution time, energy, energy delay product (EDP), and temperature, with minimal computational complexity, while adhering to specified design constraints. Future work includes incorporating runtime phase classification into TaPT to provide optimization for systems where the executing applications are not known and/or classified a priori. Additionally, we plan to verify TaPT's scalability to more complex systems with much larger design spaces (e.g., heterogeneous multi-/many core systems).

6. ACKNOWLEDGMENTS

This work was supported by the National Science Foundation (CNS-0953447). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

7. REFERENCES

[1] T. Adegbija, A. Gordon-Ross, and A. Munir, "Dynamic Phase-based Tuning for Embedded Systems Using Phase Distance Mapping," International Conference on Computer Design, 2012.

[2] ARM, http://www.arm.com/products/processors/cortex-a/cortex-a9.php.

[3] R. Balasubramonian, D. Albonesi, A. Byoktosunoglu, and S. Dwarkada, "Memory hierarchy reconfiguration for energy and performance in general-purpose architectures," International Symposium on Microarchitecture, 2000.

[4] N. Binkert, et al, "The gem5 simulator," Computer Architecture News, May 2011.

[5] D. Brooks and M. Martonosi, "Dynamic thermal management for high performance microprocessors," International Symposium on High-Performance Computer Architecture, 2001.

[6] A. Gordon-Ross, J. Lau, and B. Calder, "Phase-based cache reconfiguration for a highly-configurable two-level cache hierarchy," ACM Great Lakes Symposium on VLSI, 2008

[7] A. Gordon-Ross and F. Vahid, "A self-tuning configurable cache," IEEE Design Automation Conference, 2003.

[8] A. Gordon-Ross, F. Vahid, and N. Dutt, "Fast configurable-cache tuning with a unified second level cache," International Symposium on Low Power Electronics and Design, 2005.

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

[10] H. Hajimir and P. Mishra, "Intra-task dynamic cache reconfiguration," International Conference on VLSI Design, 2012.

[11] S. Heo, K. Barr, and K. Asanovic, "Reducing power density through activity migration," International Symposium on Low Power Electronics and Design, 2003.

[12] R. Jayaseelan and T. Mitra, "Temperature aware task sequencing and voltage scaling," International Conference on Computer-Aided Design, 2008.

[13] S. Li, et al, "McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures," International Symposium on Microarchitecture, 2009.

[14] A. Malik, W. Moyer, and D. Cermak, "A low power unified cache architecture providing power and performance flexibility," International Symposium on Low Power Electronics and Design, 2000.

[15] Motorola RAZR i XT890 - http://www.gsmarena.com/motorola_razr_i_xt890-4998.php.

[16] M. Pedram and S. Narian, "Thermal modeling, analysis, and management in VLSI circuits: principles and methods," Special Issue on Thermal Analysis of ULSI, Vol. 94, No. 8, pp. 1487-1501, 2006.

[17] J. Poovey, M. Levy, and S. Gal-On, "A benchmark characterization of the EEMBC benchmark suite," International Symposium on Microarchitecture, 2009.

[18] S. Sharifi, A. Coskun, and T. Rosing, "Hybrid dynamic energy and thermal management in heterogeneous embedded multiprocessor SoCs," Asia and South Pacific Design Automation Conference, 2010.

[19] K Skadron, "Hybrid architectural dynamic thermal management," Design Automation and Test in Europe, 2004.

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

[21] I. Yeo and E. Kim, "Temperature-aware scheduler based on thermal behavior grouping in multicore systems," Design Automation and Test in Europe, 2009.

[22] C. Zhang, F. Vahid, and W. Najjar, "A highly configurable cache architecture for embedded systems," International Symposium on Computer Architecture, 2003.

[23] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the strength pareto evolutionary algorithm," Swiss Federal Institute of Technology, Dept. of Electrical Engineering, Technical Report 103, 2001.