Dynamic Phase-based Tuning for Embedded Systems Using Phase Distance Mapping

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

Arslan Munir
Department of Electrical and Computer Engineering
Rice University
Houston, Texas, USA
arslan@rice.edu

***Abstract*—Phase-based tuning specializes a system's tunable parameters to the varying runtime requirements of an application's different phases of execution to meet optimization goals. Since the design space for tunable systems can be very large, one of the major challenges in phase-based tuning is determining the best configuration for each phase without incurring significant tuning overhead (e.g., energy and/or performance) during design space exploration. In this paper, we propose phase distance mapping, which directly determines the best configuration for a phase, thereby eliminating design space exploration. Phase distance mapping applies the correlation between a known phase's characteristics and best configuration to determine a new phase's best configuration based on the new phase's characteristics. Experimental results verify that our phase distance mapping approach determines configurations within 3% of the optimal configurations on average and yields an energy delay product savings of 26% on average.**

***Index Terms*—Cache tuning, configurable architectures, configurable hardware, dynamic reconfiguration, phase-based tuning, energy savings.**

I. Introduction and Motivation

Due to the pervasiveness of embedded systems, much research has focused on optimizations, such as improved performance and/or reduced energy consumption, to meet stringent design constraints imposed by physical size, battery capacity, cost, real-time deadlines, consumer market competition, etc. However, system optimization is challenging due to numerous tunable parameters (e.g., cache size, associativity and line size [20], replacement policy [22], issue width [2], core voltage and frequency [19], etc.), many of which tradeoff design constraints, such as size versus performance, resulting in very large design spaces with many Pareto optimal systems. The advent of multicore systems further compounds optimization challenges due to a potential exponential increase in the design space when considering dynamic core dependencies and interactions [16], which change during runtime based on the currently co-scheduled tasks. Therefore, in order to meet these increasing challenges for future systems, optimization methodologies must be highly scalable to large design spaces and must be dynamic in nature.

Since applications have varying/dynamic requirements during execution (i.e., different phases of execution) [10][18], configurable/tunable hardware [5][20][22] enables dynamic adaptation to these requirements by specializing tunable parameters to the changing needs of the application. A phase is a length of execution where an application's characteristics, such as cache misses, instructions per cycle (IPC), branch mispredictions, etc., and therefore application requirements, remain relatively stable. To identify phases, the application's execution is broken into fixed or variable length intervals that are typically measured by the number of instructions executed. *Phase classification* [17][18] groups intervals with similar characteristics to form phases, using methods such as K-means clustering [13], Markov predictors [18], etc. *Phase-based tuning* evaluates the application's characteristics and determines the best configuration (specific tunable parameter values) for each phase of execution to best meet design constraints.

The interval length must be carefully defined in a phase-based tuning approach. Intervals that are too long tend to have less stable characteristics, thus making it difficult to determine the phase's best configuration. Intervals that are too short result in too frequent tuning, thus imposing significant accumulated tuning overhead in terms of energy and performance that may intrusively affect system operation/behavior. Since interval length selection is widely researched [6][18], and is thus not a focus of our work, we assume variable length intervals [6], which result in higher optimization potential [6].

A major challenge in phase-based tuning is determining the best configuration for each phase [10] without incurring significant tuning overhead. Most previous methods [20][22] physically explore the design space by executing different configurations, recording the configurations' characteristics, and selecting the best configuration, however, this method incurs a large cumulative tuning overhead while executing inferior (non-optimal) configurations. To reduce tuning

overhead, heuristics significantly prune the design space [7][8][16], however, these heuristics still execute inferior configurations and incur tuning overhead. Analytical methods/models drastically reduce design space exploration by directly determining/calculating/predicting the best configuration based on the design constraints and application characteristics [4][9][15], however, most of these methods are either computationally complex (thus, adversely impacting performance and energy consumption) [4] or not dynamic (i.e., not phase-based) [9][15].

In this paper, we focus on reducing the computational complexity and tuning overhead of dynamic phase-based tuning by directly determining the phases' best configurations, with no design space exploration, using the correlations between a phase's characteristics and the phase's best configuration. We leverage these characteristic-to-configuration correlations to determine the best configurations for new phases based on the new phase's characteristics. We define the *configuration distance* as the difference between two configurations, where the distance is the number of different tunable parameter values between the two configurations, and we define the *phase distance* as the difference between the characteristics of two phases, where the difference is based on how disparate two phases' characteristics are. *Phase distance mapping* uses the phase distance to calculate the configuration distance, and thus directly determines the new phase's best configuration using *configuration estimation*.

We exemplify and evaluate phase distance mapping using cache tuning for separate level one instruction and data caches. Cache tuning determines the best cache configuration in terms of total size, line size, and associativity for reduced energy consumption [8][10][20]. Results reveal that the phase distance correlates closely with the configuration distance and phase distance mapping can determine configurations within 3% of the optimal configurations and achieves an average energy delay product (EDP) savings of 26%.

II. RELATED WORK

Much previous work focuses on tuning configurable hardware to the best configuration for a particular application for reduced energy consumption and/or improved performance. Zhang et al. [20] proposed a configurable cache architecture that determined the Pareto optimal cache configurations trading off energy consumption and performance. Zou et al. [22] introduced a configuration management algorithm that searched the cache design space for the best configuration. Since each of these optimization methods physically explored the design space, these methods incurred tuning overhead.

To reduce tuning overhead, several methods eliminated design space exploration. Gordon-Ross et al. [9] proposed a one-shot approach to cache configuration using a cache tuner that non-intrusively predicted the best cache configuration, without executing inferior configurations, using an oracle-based approach [11]. However, the oracle hardware introduced significant tuning overhead and could only be used judiciously. Ghosh et al. [4] proposed a heuristic that used an analytical model to directly determine the cache configuration based on the designer's performance constraints and application characteristics. However, the computational complexity of the analytical model still incurred overhead in terms of energy consumption and performance. While each of these optimization methods reduced the tuning overhead, these methods did not account for dynamically changing application characteristics (i.e., phases).

To adhere to an application's changing execution requirements, Hajimir et al. [10] proposed intra-task dynamic configuration that tuned a highly configurable cache on a per-phase basis using a detailed cache model. Gordon-Ross et al. [5] investigated the benefits of phase-based tuning over application-based tuning (using one configuration for the entire run of the application) with respect to energy consumption and performance using a detailed cache model, and quantified the tuning overhead in terms of energy and performance due to write backs and cache flushing.

Our work differs from previous phase-based tuning methods by using the correlation between phase distance and configuration distance to directly determine a phase's best configuration, thereby eliminating design space exploration. Our method reduces tuning overhead and computational complexity by extending phase classification hardware to implement phase distance mapping, without incurring any significant hardware overhead. Furthermore, our work can be easily extended to optimize for multiple application characteristics using any tunable hardware.

III. DESIGN SPACE AND PHASE TUNING ARCHITECTURE

The design space contains all of the different configurations/combinations of the tunable parameter values. Our memory hierarchy consists of configurable, private, separate level one (L1) instruction and data caches. The configurable caches are based on Zhang et al.'s [20] highly configurable cache, which provides runtime-configurable total size, associativity, and line size using a small bit-width configuration register. Zhang's configurable cache has served as the basis for several newer architectures [6][8] and can be easily extended to state of the art, more complex architectures, such as heterogeneous multicore systems [16].

To evaluate phase distance mapping, we define a *base cache configuration* for comparison purposes. The base cache configuration, which is an average configuration representing typical embedded microprocessors [20] that might execute our experimental applications (Section 5), is an 8 Kbyte cache composed of four configurable banks, each of which can operate as a separate way (i.e., the base cache is a 4-way set associative cache), and a logical line size of 64 bytes. The configuration register provides configurable associativity by logically concatenating the ways, offering an 8 Kbyte direct-mapped or 2-way set associative cache, and/or shutting down ways, offering a 4 Kbyte direct-mapped or 2-way set associative cache or a 2 Kbyte direct-mapped cache. All cache sizes offer a configurable line size of 16, 32, or 64 bytes by using a base, physical line size of 16 bytes and fetching additional physical cache lines for larger, logical line sizes. We

refer the reader to [20] for additional configurable cache architectural details. Given these parameters, the design space contains eighteen different configurations, however, phase distance mapping can be applied to any design space.

Figure 1 depicts our phase tuning architecture for a sample dual-core system. On-chip components include the processing cores that are connected to private, separate L1 instruction and data caches and the phase characterization hardware. Without loss of generalization, the level one caches are directly connected to off-chip main memory and, since this hierarchy implies that there is no dependence between the caches, the caches can be tuned independently. Phase characterization hardware includes a *tuner*, a *phase classification module* that classifies an application's phases, a *phase distance mapping module*, which includes a lookup table, which stores the inputs to the configuration estimation algorithm, and a *phase history table*. The tuner orchestrates the phase characterization process (Section 4.1), which includes phase distance mapping. The phase distance mapping module implements the configuration estimation algorithm (Section 4.2) to determine a phase's best configuration. After phase distance mapping determines a phase's best configuration, the phase is designated as a *characterized phase* and is added to the *phase history table*, along with the phase's best configuration. We note that in the case of our studied cache hierarchy, the best configuration stored in the *phase history table* represents both the best instruction and data cache configurations. Prior research using similar table structures showed that these structures have very little or no effect on overall system area, performance, and/or energy consumption [18], and the work proposed herein to incorporate phase distance mapping will not significantly increase/impact these overheads, however, in future work we will quantify these overheads.

IV. PHASE DISTANCE MAPPING

Phase distance mapping reduces tuning overhead by directly determining a phase's best configuration by evaluating the correlation between the phase distance and the configuration distance. In this section, we elaborate on how this correlation is leveraged to determine a phase's best configuration and present our algorithm for configuration estimation using phase distance mapping. Even though we exemplify phase distance mapping using cache tuning, we generalize our discussions for any tunable hardware and include cache tuning specifics when necessary.

A. Correlating Phase Distance and Configuration Distance

Phase classification groups intervals that show similar characteristics into phases such that a phase's characteristics are relatively stable during the execution of that phase. As a result of this relative stability, the same configuration can be used for the phase's duration. Therefore, our foundation for phase distance mapping is the hypothesis that the more disparate two phases' characteristics are, the more disparate the phases' best configurations are likely to be, enabling the mapping of the distance between phases to the distance between the best configurations.

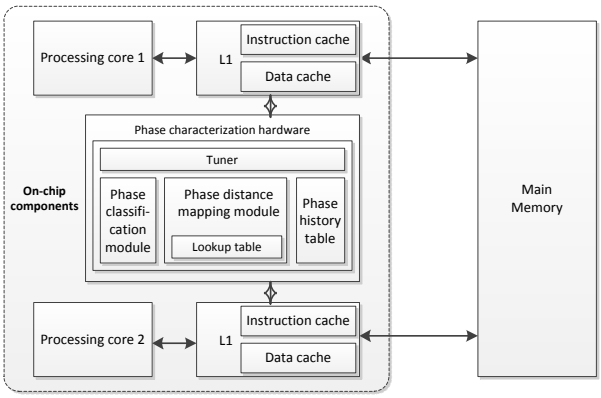


Fig. 1. Phase tuning architecture for a sample dual- cores core system

We calculate the phase distance based on the phase space, which is the set of all of an application's distinct phases. Since phase classification is not the focus of this study, we assume that phase classification has already been applied to the application (using any arbitrary method, such as offline phase classification [17] or online runtime phase tracking and prediction [18]), which produces the application's different phases and the phases' characteristics. Since we study cache tuning and previous work showed that cache miss rates can accurately determine a phase's characteristics [16], we classify the different phases using the phases' cache miss rates. Since comparative cache evaluation is most effective when the caches have the same configuration, we gathered the phases' cache miss rates for the base cache configuration (Section 3).

Figure 2 illustrates phase characterization, which takes as input the classified phases and the phases' characteristics, which are output from phase classification. One phase is designated as the *base phase* P_b. The base phase is the phase to which subsequent phases are compared to calculate the phase distance, and can be designated using one of two methods. Ideally, the base phase and the base phase's best configuration should be determined at design time by the designer using any design space exploration method [17] and are input into the phase characterization process. Alternatively, the first executed phase can be designated as the base phase and the base phase's best configuration can be determined at runtime using any runtime tuning method [8]. This runtime method eliminates designer effort, but may trade off accuracy and introduces additional tuning overhead. For our experiments, we determined the base phase's best configuration by exhaustive search.

When a phase P_i is executed, the first step in phase characterization is to search the *phase history table* for P_i. If P_i is in the *phase history table,* P_i has already been executed and the best configuration C_{Pi} has already been determined. The hardware is configured to C_{Pi} and phase P_i executes in C_{Pi}. If P_i is not in the *phase history table,* P_i is a new phase and the difference between P_i's characteristics and the base phase's characteristics $d(P_b, P_i)$ (i.e., the phase distance) is calculated.

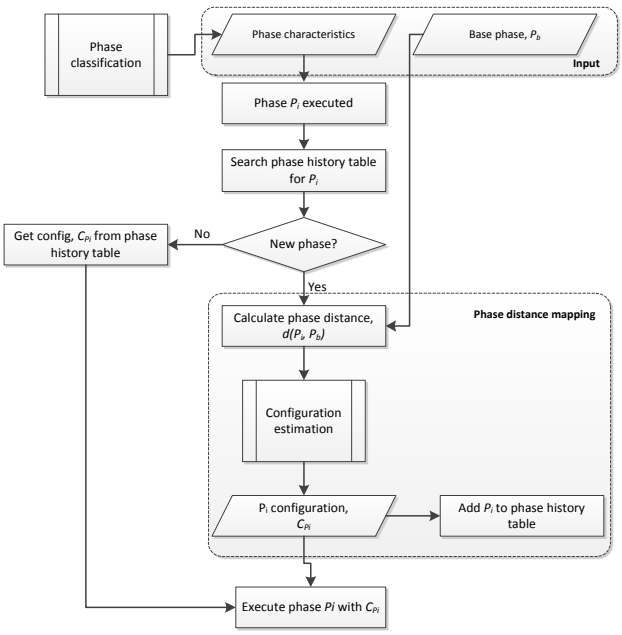


Fig. 2. Phase characterization

The phase distance can be calculated using either a single phase characteristic or multiple phase characteristics. In this work, we use a single phase characteristic, the cache miss rate, to calculate $d(P_b, P_i)$, by normalizing P_i's instruction and data cache miss rates to P_b's instruction and data cache miss rates. This normalization enables quick comparisons of disparate configurations' miss rates. This single-characteristic method is suitable for tuning single components, such as private instruction and data caches that do not have dependencies. In systems with multiple tunable hardware or tunable component dependencies, a multi-characteristic method, such as one that evaluates the cache miss rates and IPC, provides a more holistic view of the phase characteristics and is the focus of our future work

After the phase distance is calculated, the phase distance is used as input to configuration estimation.

B. Configuration Estimation

We empirically developed and refined the configuration estimation algorithm by studying the impact that the different configurations have on the phases' characteristics. Since most embedded systems run single applications or a set of applications within the same domain, configuration estimation can be application domain-specialized with respect to the underlying tunable hardware. However, we point out that even though our configuration estimation is domain-specialized, the algorithm is generalized and can be easily adapted to different domains and tunable hardware. We generalized our configuration estimation algorithm to a variety of common embedded systems application domains, such as networking, image processing, cryptography, and data compression. However, since the majority of our studied applications involved image rotation (application details are presented in Section 5), we specialized the configuration estimation algorithm to an image processing domain by using a base phase from an image rotation application.

Configuration estimation leverages the underlying tunable hardware by considering the impact of the different parameter values on the energy consumption and performance [20]. For example, direct-mapped caches consume less power per access than 4-way set associative caches since only one data array and one tag are read per access, rather than four data arrays and four tags. However, direct-mapped caches can have higher cache miss rates than set associative caches, resulting in more total energy consumption when considering the miss penalties in terms of stall time and power to access the next memory level(s). Even though increasing the cache associativity increases the power per access, the cache miss rate may decrease enough to result in an overall decrease in energy consumption. However, this concept suffers from diminishing returns as increasing the reduction in miss rate (i.e., increasing the set associativity) will eventually not outweigh the increase in power per access. Since this well-known trend is not isolated to cache parameters, configuration estimation must consider diminishing returns for all tunable parameters with similar trends. Our configuration estimation algorithm considers diminishing returns using threshold values for each tunable parameter. A threshold value is the specific parameter value at which further increases in the parameter value may result in increased energy consumption or reduced performance.

Algorithm 1 depicts the configuration estimation algorithm that the phase distance mapping module implements. The algorithm's inputs are: the base phase's best configuration in terms of cache size C_b, associativity A_b, and line size L_b; the configurable cache's minimum and maximum sizes C_{MIN} and C_{MAX}, associativities A_{MIN} and A_{MAX}, and line sizes L_{MIN} and L_{MAX}, respectively; size, associativity, and line size threshold values C_{THR}, A_{THR}, and L_{THR}, respectively; distance windows R_1, through R_7; and the phase distance D. The algorithm outputs phase P_i's determined best cache size, C_i, associativity A_i, and line size L_i.

We empirically determined the threshold cache size, associativity, and line size values as 8 Kbyte, 2-way, and 64 byte, respectively. For example, Figure 3 illustrates how we determine the associativity threshold value in terms of EDP (Joule seconds) for three image rotation phases from our studied applications (Section 5 details the EDP calculation and application phases). In these results, the instruction cache configuration is arbitrarily fixed at the base configuration and the data cache associativity is varied while holding the data cache's size and line size fixed at the base configurations. Since increasing the associativity from 1-way to 2-way results in a decrease in EDP and further increasing the associativity to 4-way results in an increase in EDP, the associativity threshold value is 2. We similarly determined the size and line size threshold values. Even though this is an expected result for a simple trend, this empirical analysis can be used for any tunable parameter with any number of parameter values. Even though the threshold values can be generalized for any application domain, the specific threshold values will vary across different application domains due to different cache

Inputs: C_B, A_B, L_B, C_{MIN}, C_{MAX}, A_{MIN}, A_{MAX}, L_{MIN}, L_{MAX}, C_{THR}, A_{THR}, L_{THR}, R_1, R_2, R_3, R_4, R_5, R_6, R_7
$D = d(P_b, P_i)$

Outputs: C_I, A_I, L_I

```
1   C_I = C_B, A_I = A_B, L_I = L_B
2   Switch (D) {
3       Case R_1, R_2, R_7:
4           C_I ← C_THR
5           break
6       Case R_3:
7           If C_B == C_MIN then
8               C_I ← C_B * 2
9           Else
10              C_I ← C_THR
11          If A_B = A_MIN then
12              A_I ← A_B * 2
13          break
14      Case R_4:
15          C_I ← C_THR
16          If A_B != A_MAX then
17              A_I ← A_B * 2
18          If L_B != L_MIN then
19              L_I ← L_B/2
20          break
21      Case R_5:
22          C_I ← C_THR
23          If A_B = 1 then
24              A_I ← A_THR
25          break
26      Case R_6:
27          If C_B != C_MAX then
28              C_I ← C_MAX/2
29          break
30  }
```

Algorithm 1: Configuration estimation

locality behavior. Therefore, for configuration estimation to be most effective, the threshold values should be application domain-specialized. We note, however, that since our experiments considered phases from diverse application domains, we used generalized threshold values, which underestimate the effectiveness of our configuration estimation algorithm.

Distance windows are phase distance ranges that represent the configuration's distance from the base phase and represent P_i's configuration distance from P_b when changing a parameter's value to another value (e.g., increasing the associativity: A_b * 2). Each distance window has an upper and lower bound and a phase distance D maps to the distance window in which D is bounded by. For our experiments, we created distance windows using a base phase from an image rotation application and evaluated how the parameter values changed for the different phases' optimal configurations (determined by an exhaustive search) with respect to the base phase's configuration. The distance windows relate directly to all of the characteristics used to evaluate D and are applicable to all the tunable parameters represented by D. For example, since we use the cache miss rate to evaluate D, the distance window bounds relate directly to the actual cache miss rate values and are applicable to all of the tunable parameters (cache size, associativity, and line size). We determined that the seven distance windows: $R_1 = [0,0.25]$, $R_2 = (0.25,0.5]$, $R_3 = (0.5,0.75]$, $R_4 = (0.75,1.25]$, $R_5 = (1.25,1.5]$, $R_6 = (1.5,2.5]$, and $R_7 = (2.5,\infty]$), sufficiently cover all the phase distances between the base phase and all of the other phases. The distance windows' bounds represent the normalized difference between P_i's and P_b's cache miss rate. The phase distance D maps to these distance windows such that if $0 \le D < 0.25$, D maps to R_1, if $0.25 \le D < 0.5$, D maps to R_2, etc. In general, the number of distance windows can vary based on a system's intended applications and the applications' phases, the distance windows are specialized based on the evaluated characteristic (e.g., IPC), and if a multi-characteristic method is used for evaluating D, only one set of distance windows is necessary to represent all of the tunable parameters.

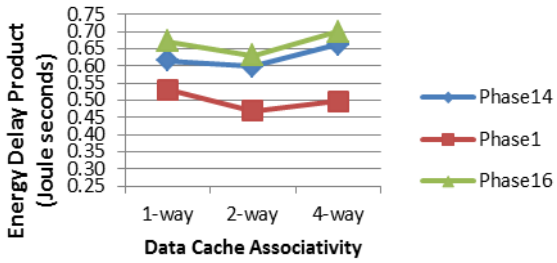


Fig. 3. Associativity threshold value determination using diminishing return effects on the energy delay product for varying data cache associativities.

For each phase P_i, the configuration estimation algorithm is executed twice, once for the instruction cache and once for the data cache. First, the algorithm assigns initial values to C_i, A_i, and L_i as C_b, A_b, and L_b, respectively (line 1), which represent default values for C_i, A_i, and L_i. Default values are used because some configuration distances in some distance windows require no parameter value change for some parameters. Next, the algorithm determines which distance window the phase distance D maps to (line 2) and determines P_i's best configuration based on the configuration distance for the corresponding distance window. If a distance window does not specify a change to a parameter value, then C_i, A_i, and L_i remain as the default values. For example, if phase P_2 is the next phase to be executed and D = 1.08, the algorithm determines that D maps to distance window R_4 (line 14), and determines C_i, A_i, and L_i, based on the configuration distance for R_4 (lines 15 – 19).

V. Experimental Results

We evaluate phase distance mapping by comparing a system that switches to the best configurations, as determined by phase distance mapping, for each phase to a system fixed with the base cache configuration.

A. Experimental Setup

We selected sixteen workloads from the EEMBC Multibench benchmark suite [3], which is an extensive suite of multicore benchmarks that primarily target the embedded market and model a wide variety of realistic applications. Each Multibench workload is a collection of kernels working on a specific dataset. Our selected workloads covered diverse

processing tasks, such as image rotation for different colors/sizes, internet protocol (IP) packet checking, IP packet reassembly, transmission control protocol (TCP) processing, md5 message-digest algorithm checksum calculation, Huffman decoding, etc. Since each workload represents a specific compute kernel, without loss of generality, we assume that each workload represents a different phase, and simulate each phase/workload a single time to completion.

To gather cache miss rates, we use GEM5 [1] to model a homogeneous dual core system with separate, private L1 instruction and data caches. We use McPAT [14] to calculate the system's total power consumption and evaluate the system's energy efficiency using the EDP in Joule seconds:

$$EDP = system\_power * phase\_running\_time^2$$
$$= system\_power * (total\_phase\_cycles/system\_frequency)^2$$

where *system_power* includes the core power and cache power, and *total_phase_cycles* is the total number of cycles to execute a phase to completion. Table I shows some of the system's microarchitectural parameters that contribute to the EDP.

We modeled phase distance mapping and automated our simulations using Perl scripts.

B. Results

Figure 4 (a) shows the EDP savings, as compared to the base configuration for the optimal configuration as determined using an exhaustive search (Optimal) and the best configuration as determined by phase distance mapping (PDM) for a single execution of each of the sixteen phases. Phase 1, which rotates sixteen 4-megapixel greyscale images 90 degrees clockwise, is used as the base phase. On average over all phases, phase distance mapping achieved an EDP savings of 26%, with savings as high as 47% for Phase 5, and was within 3% of the optimal configuration.

To evaluate the effects that a different base phase has on the EDP savings, Figure 4 (b) shows the EDP savings, as compared to the base configuration, using Phase 7 as the base phase. Phase 7 executes Huffman decoding on seven datasets. On average over all phases, phase distance mapping achieved an average EDP savings of 22%, with savings as high as 38% for Phase 15, and was within 7% of the optimal configuration. Phase 8 had the lowest EDP savings (2%), as compared to the optimal (21%), because our algorithm selected a smaller line size than required. However, phase distance mapping still achieved some EDP savings over the base phase. Using Phase 7 instead of Phase 1 as the base phase resulted in a 4% reduction in average EDP savings, while Phase 5's EDP savings dropped by 15%. This reduction in average EDP savings is due to the fact that Phase 7 is the only phase that performs any type of data compression where as six of the phases perform image rotation. To verify this application-domain dependence when designating a base phase, we used Phase 5, another image processing phase, as the base phase. For brevity, we omit the detailed results, but the results revealed that phase distance mapping using Phase 5 as the base phase achieved EDP savings that varied by less than 1% as compared to using Phase 1 as the base phase.

TABLE I. CORE MICROARCHITECTURAL PARAMETERS

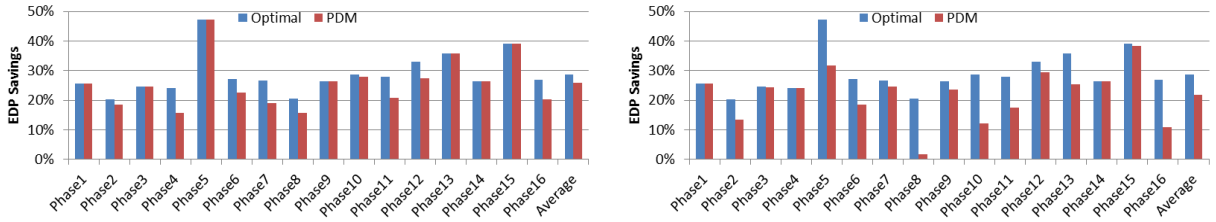
Architectural Configuration	
Processing Cores	2
Clock Rate	2 GHz
Functional Units	2 IntAlu, 1 FPAlu, 1 Mult/DivAlu
Issue Width	1
Physical Registers	32 Int, 32 FP
L1 Instruction and Data Caches	
Cache size	2 Kbyte – 8 Kbyte
Associativity	1-way – 4-way
Line size	16 byte – 64 byte

These analyses reveal that the magnitude of savings is highly application-domain dependent, and that even though good savings can be achieved by using any base phase, carefully considering the application domain when designating the base phase can significantly increase the EDP savings. Designating the base phase for a small, application-domain-specialized system with a small phase space can be done manually during design time, however, this manual designation is infeasible for large, general-purpose systems with a large phase space. For large systems, designers can use cluster analysis (e.g., k-means clustering [12] or graph-based models [21]) to partition the phase space into different domains, and the phase that most closely represents the largest cluster (most prominent domain) can be designated as the base phase.

VI. CONCLUSION AND FUTURE WORK

Phase-based tuning specializes a system's configurations to varying runtime application characteristics to meet design constraints. One of the major challenges of phase-based tuning is determining the phases' best configurations without incurring significant tuning overhead. In this paper, we proposed phase distance mapping, a phase-based tuning method that directly determines the best configuration for a phase with no design space exploration. On average, phase distance mapping determined configurations within 3% of the optimal configuration, with an average energy delay product (EDP) savings of 26%.

Future work includes making the configuration estimation algorithm more adaptable to runtime application execution requirements and a wider variety of application domains, incorporating a feedback mechanism to improve the configuration estimation's accuracy, and modeling more complex systems (e.g., heterogeneous cores with more tunable parameters). Additionally, we will quantify and evaluate the area, energy, and performance overheads of phase distance mapping.



(a) *(b)*

Fig. 4. EDP savings for the optimal configuration (Optimal) and the best configuration determined by phase distance mapping (PDM), as compared with the base configuration, when using (a) Phase 1 and (b) Phase 7 as the base phase. Phase distance mapping is also used to determine the configurations for the base phases, which shows the worst-case scenario for the base phases.

ACKNOWLEDGMENT

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 author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

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

[2] D. Folegnani, "Energy-effective issue logic," 28th Annual International Symposium on Computer Architecture, 2001.

[3] S. Gal-On and M. Levy, "Measuring multicore performance," *Computer,* November, 2008.

[4] A. Ghosh and T. Givargis, "Cache optimization for embedded processor cores: an analytical approach," International Conference on Computer Aided Design, November, 2003.

[5] 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, May, 2008.

[6] A. Gordon-Ross and F. Vahid, "A self-tuning configurable cache," IEEE Design Automation Conference, July, 2007.

[7] A. Gordon-Ross, F. Vahid, and N. Dutt, "Automatic tuning of two-level caches to embedded applications," in Proceedings of the Conference on Design, Automation and Test in Europe, February, 2004.

[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, August, 2005.

[9] A. Gordon-Ross, P. Viana, F. Vahid, W. Najjar, and E. Barros, "A one-shot configurable cache tuner for improved energy and performance," IEEE/ACM Design, Automation and Test in Europe, April 2007.

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

[11] S. Jain, K. Fall, and R. Patra, "Routing in a delay tolerant network," SIGCOMM, 2004.

[12] T. Kanungo, et. al, "An efficient k-means clustering algorithm: Analysis and implementation," IEEE Trans. Pattern Analysis and Machine Intelligence, 2002.

[13] J. Lau, "Structures for phase classification," International Symposium on Performance Analysis of Systems and Software, 2004.

[14] S. Li, et. al, "McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures," 42nd Annual IEEE/ACM International Symposium on Microarchitecture, December, 2009.

[15] A. Munir, A. Gordon-Ross, S. Lysecky, and R. Lysecky, "A one-shot dynamic optimization methodology for wireless sensor networks," International Conference on Mobile Ubiquitous Computing, October, 2010.

[16] M. Rawlins and A. Gordon-Ross, "An application classification guided cache tuning heuristic for multi-core architecture," Asia and South Pacific Design Automation Conference, January, 2012.

[17] T. Sherwood, E. Perelman, and B. Calder, "Basic block distribution analysis to find periodic behavior and simulation points in applications," International Conference on Parallel Architectures and Compilation Techniques, September, 2001.

[18] T. Sherwood, S. Sair, and B. Calder, "Phase tracking and prediction," 30th International Symposium on Computer Architecture, December, 2003.

[19] L. Tee, S. Lee, and C. Tsai, "A scheduling with DVS mechanism for embedded multicore real-time systems," International Journal of Digital Content Technology and its Applications, April, 2011.

[20] C. Zhang, F. Vahid and W. Najjar, "A highly-configurable cache architecture for embedded systems," 30th Annual International Symposium on Computer Architecture, May 2003.

[21] Y. Zhou, H. Cheng, and J. Yu, "Graph clustering based on structural/attribute similarities," VLDB Endowment Journal, August, 2009.

[22] X. Zou, J. Lei, and Z. Liu, "Dynamically reconfigurable cache for low-power embedded system," Third International Conference on Natural Computation, August, 2007.