Energy-efficient Phase-based Cache Tuning for Multimedia Applications in Embedded Systems

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 University of Florida
e-mail: tosironkbd@ufl.edu, ann@ece.ufl.edu

***Abstract*—The proliferation of multimedia applications in embedded systems has led to a research focus on optimizing the energy consumption of these applications without significantly degrading the execution time and adhering to data processing deadline constraints. To maximize optimization potential, phase-based tuning methodologies specialize system configurations to different phases of application execution with respect to design constraints. Multimedia applications are ideal candidates for phase-based tuning since these applications exhibit variable execution characteristics. In this paper, we propose a phase-based tuning methodology for multimedia applications that leverages application characteristics to determine the best cache configurations for different phases of execution. Results reveal that phase-based tuning for multimedia applications determines cache configurations within 1% of the optimal on average and yields an average energy delay product savings of 29%.**

***Keywords*—Cache tuning, dynamic reconfiguration, phase-based tuning, multimedia applications, energy delay product (EDP) reduction.**

I. Introduction and Motivation

Multimedia applications are becoming ubiquitous in embedded systems, and further increasing predominance is anticipated due to these applications appeal to embedded system consumers. This predominance, coupled with consumer demands for advanced functionality and performance, increases the size and complexity of multimedia applications, thus demanding additional energy and computational capabilities. These pressures present a major design challenge for embedded system designers since most embedded systems are battery operated and have many diverse, stringent design constraints (e.g., area, energy, temperature, etc.).

Several optimization methods have been proposed to address increased energy consumption, many of which involve dynamically tuning/specializing the system configuration to the executing applications' requirements and design constraints. Configurable systems contain tunable parameters/hardware, such as cache size, clock frequency, issue width, voltage, etc., with values that can be changed during runtime. Tuning determines the best combination of these parameter values (i.e., configuration) to meet application requirements.

Our work focuses on cache tuning since cache tuning is an ideal option for energy reduction due to the memory hierarchy's significant contribution to the microprocessor's total energy [13]. Cache tuning determines the best cache configuration, such as cache size, associativity, and line size, that best adheres to application requirements and design constraints. Since multimedia applications have data-intensive bandwidth requirements, caches are widely used to bridge the processor-memory performance gap [18], and cache tuning can significantly reduce the energy consumption while still meeting performance constraints.

To realize the benefits of tuning, application-based tuning uses a single, best configuration that represents the greatest average optimization potential for the entire application based on the design constraints. However, since applications show varying operating requirements throughout execution, previous work proposed phase-based tuning [5]. Phase-based tuning further increases the optimization potential by exploiting variations in the application's characteristics (e.g., instructions per cycle (IPC), cache misses, branch mispredictions, etc.) and uses the best configuration for each different application execution phase. An application execution phase is a length of execution where the application's characteristics, and therefore best configuration that adheres to the design constraints, remain relatively stable.

Multimedia applications are ideal candidates for phase-based tuning since multimedia applications exhibit variable application characteristics [8] during runtime. Most multimedia applications periodically process units of data—frames—and this processing typically has a defined execution deadline. Previous work [9] analyzed multimedia application characteristic variability during runtime at the frame granularity and observed that same-typed frames exhibited stable frame-to-frame application characteristics, and similar energy consumption and execution time characteristics, throughout execution. These observations suggest that multimedia application frame types are analogous to phases, and can benefit from phase-based tuning.

To employ phase-based tuning in multimedia applications, a major challenge is determining the best configuration for each phase without incurring significant tuning overhead in terms of execution time and/or energy. This challenge arises from potentially large design spaces consisting of all possible configurations in systems with many tunable parameters and parameter values. Additionally, phase-based tuning has typically emphasized energy optimization [4][5][14], which may adversely impact the execution time and potentially manifest as missed deadlines.

Previous work proposed analytical methods [4] to minimize tuning overhead by directly determining, calculating, or predicting the best configurations based on the design constraints and application characteristics. Therefore, to optimize energy consumption for multimedia applications while limiting execution time degradation and/or missed deadlines, we leverage a prior analytical phase-based tuning methodology, *phase distance mapping* (PDM) [1]. PDM used a computationally simple, dynamic analytical model/algorithm that leveraged the difference between two phases' characteristics to directly determine/predict a phase's best configuration with minimal tuning overhead. However, since PDM used a configuration estimation algorithm that was specialized to the specific studied application domains, PDM's effectiveness was contingent on a priori application analysis, resulting in considerable design time overhead. Even though PDM achieved significant energy delay product (EDP) savings for a variety of applications, PDM did not achieve significant EDP savings when used specifically for multimedia applications since the configuration estimation algorithm did not consider specific multimedia application characteristics (e.g., high spatial locality).

In this work, we propose a phase-based cache tuning methodology for multimedia applications, which leverages fundamental PDM concepts combined with multimedia application characteristics to dynamically determine the best cache configurations with no design time overhead and minimal execution time tuning overhead. Results reveal that our methodology meets execution deadlines, achieves 29% system EDP savings and increases PDM's EDP savings by 18%, with an execution time overhead of only 1%.

II. Background and Related Work

Much previous work studied dynamic voltage and frequency scaling (DVFS) techniques to optimize temperature, energy consumption, and/or execution time in multimedia applications [8][10][17][19]. However, little prior work exploits the energy optimization potential for phase-based cache tuning in multimedia applications. In this section, we present general related work and background on phase-based cache tuning, PDM, and the multimedia application characteristics considered in our work.

A. Phase-based Cache Tuning

Phase-based tuning requires hardware- or software-based phase classification. Phase classification breaks an application's execution into fixed or variable length intervals measured by the number of instructions executed and intervals with similar characteristics are grouped to form phases. Shen et al. [15] showed that data locality was ideal for phase classification by using a method that combined data locality profiling and runtime prediction to predict application phases. Sherwood et al. [16] showed that phase classification using basic block distribution analysis was highly correlated with architecture-dependent application characteristics, such as cache miss rates, branch mispredictions, IPC, address prediction miss rates, value prediction miss rates, and register update unit occupancy percentage. Balasubramonian et al. [2] determined that cache miss rates, IPC, and branch frequency were effective for phase classification. Since our work evaluates cache tuning and cache miss rates are effective for phase classification, we use cache miss rates to classify phases.

To determine the best cache configurations for classified phases, phase-based cache tuning requires a configurable cache architecture, such as the configurable cache proposed by Zhang et al. [20], which provided dynamically configurable total cache size, associativity, and line size using a small bit-width configuration register. The proposed architecture consisted of separate configurable level one instruction and data caches. The authors' proposed base configurable cache had four physical ways (i.e., 4-way set associative) implemented as individual cache banks. The ways could be shutdown to reduce the cache size or concatenated to form a direct-mapped or 2-way set associative cache. Given a base, physical line size, multiple lines could be fetched and logically concatenated to configure larger line sizes.

A specialized hardware cache tuner, connected to the cache hierarchy/busses, orchestrates cache tuning and explores the configurable cache design space. The cache tuner can use any design space exploration method/heuristic [4][5][14] to minimize tuning overhead incurred from executing inferior, non-optimal configurations that do not adhere to the design constraints. If the level one caches are separate, the cache tuner can successively and independently tune the instruction and data caches, thus simplifying the exploration heuristic, however, systems with unified second level caches require more complex exploration heuristics [5]. The cache tuner monitors each explored configuration's cache statistics, such as number of accesses, misses, etc., while executing the phase for one tuning interval. For accurate configuration evaluation, the tuning interval must be long enough to warm up the cache and stabilize the cache statistics. The cache tuner evaluates the phase's energy consumption and EDP using the statistics, an energy model, and the execution time to determine the next configuration to explore, or stops exploration if the best configuration has been determined. The cache tuner can store the characterized phase and associated best configurations in a data structure, such as a phase history table [1], which eliminates redundant tuning overhead for the phases' subsequent executions.

B. Phase Distance Mapping (PDM)

The phase distance is the difference between the characteristics of a characterized phase—a phase with a previously determined best configuration—and an uncharacterized phase. PDM compared a single, previously characterized phase—a base phase—with a newly executed uncharacterized phase, and used the phase distance to calculate the configuration distance between these phases. The configuration distance is the difference between (i.e., change in) the two configurations' tunable parameter values, such as increasing the associativity by a factor of two. To determine the uncharacterized phase's best configuration, PDM used a configuration estimation algorithm that defined distance windows, which represented phase distance ranges and corresponding configuration distances from the base phase.

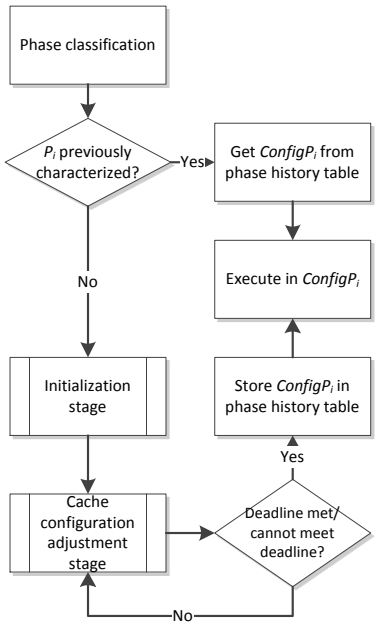

Fig. 1. Multimedia application phase-based tuning algorithm overview

PDM's configuration estimation algorithm used seven a priori designer-defined distance windows, which sufficiently covered the phase distances between the base phase and all of the other phases, and were defined with respect to the base phase P_b's best configuration's parameter values. Each distance window defined a phase distance range with a minimum Win_L and maximum Win_U value, and a phase distance D mapped to the distance window that D was bounded by (i.e., $Win_L < D < Win_U$). The distance windows related directly to the cache miss rates since the cache miss rates were used to evaluate the phase distance. These distance windows were applicable to all of the tunable parameters represented by the phase distance, which were the cache size, associativity, and line size. When an uncharacterized phase P_i was executed, PDM calculated the phase distance $D = d(P_b, P_i)$ between P_i and P_b. PDM used D to determine which distance window P_i mapped to and used the configuration distances specified for each parameter with respect to P_b's parameter values to determine P_i's best cache configuration (e.g., $A_b * 2$, where A_b is P_b's associativity). We refer the reader to [1] for additional PDM details.

C. Multimedia Application Characteristics

Prior works evaluated multimedia application characteristic variability and predictability, which we leverage for phase-based cache tuning. Xu et al. [18] studied multimedia application memory access characteristics and showed that multimedia applications typically access input data using block-partitioning algorithms. In these algorithms, data is broken into smaller data blocks and accessed one data block at a time. The authors showed that these data blocks contained significant data reuse and spatial locality, implying that multimedia applications generally benefit from large line sizes. Also, a large number of data memory references were to the applications' internal data structures, such as input data arrays, which were typically relatively small and could easily fit into relatively small caches (e.g., 8 Kbyte caches [18]).

Hughes et al. [8] observed that multimedia applications exhibited variable characteristics during execution, and average IPC and power consumption during frame processing were almost constant across same-typed frames. The authors also observed that different frame types exhibited little variation in execution time, implying that the future frames' execution times could be predicted based on the previously executed frames' execution times. Srinivasan et al. [17] exploited variable execution characteristics for dynamic thermal management (DTM) using DVFS and configurable instruction issue width and number of active functional units. Lee et al [10] and Yeo et al. [19] also used DVFS for DTM in MPEG-2 and MPEG-4 decoding, respectively. Hughes et al. [9] used DVFS and configurable instruction issue width, instruction window size, and number of functional units to reduce energy consumption. However, none of these works explored the increased optimization potential that phase-based cache tuning affords.

III. Multimedia Application Phase-based Tuning Algorithm

Since much prior work already focused on phase classification and our algorithm is independent of the particular phase classification method used, we assume that the phases have already been classified.

Fig. 1 depicts an overview of our multimedia application phase-based tuning algorithm. To consider both energy consumption and execution time, our algorithm uses EDP as the evaluation metric. Our algorithm explores each configuration for one tuning interval, and calculates the energy consumption and execution time for the tuning interval to determine the next configuration to explore or halts exploration if the best configuration has been determined.

For each phase, our algorithm executes for the instruction cache and then the data cache, evaluating each cache separately and in succession. When a phase P_i is executed, if P_i has been previously characterized, P_i is executed in the previously determined best cache configuration $ConfigP_i$ that is stored in the phase history table. If P_i has not been previously characterized (i.e., P_i is a new phase), the algorithm starts the *initialization stage*, which leverages PDM's configuration estimation algorithm to determine an initial cache configuration (Section III.A). This configuration serves as a starting configuration for the *configuration adjustment stage* and reduces the tuning overhead from executing inferior, non-optimal configurations (Section III.B).

The cache configuration adjustment stage executes/explores different configurations to determine $ConfigP_i$. If P_i completes execution before our algorithm determines the best configuration, our algorithm stores the lowest EDP configuration explored thus far as $ConfigP_i$ in the phase history table, and continues configuration exploration on P_i's subsequent executions starting with the stored $ConfigP_i$. If $ConfigP_i$ meets P_i's deadline, our algorithm stores $ConfigP_i$ in the phase history table, and P_i executes in $ConfigP_i$. If $ConfigP_i$

does not meet the deadline, our algorithm returns to the cache configuration adjustment stage, and iterates in that stage until a configuration that meets the deadline is determined. If no configuration meets the deadline, $ConfigP_i$ defaults to the configuration with an execution time closest to the deadline.

A. Initialization Stage

During the initialization stage, our algorithm determines P_i's energy consumption and execution time when executing in the base cache configuration for one tuning interval. The base configuration represents a non-configurable system that maximizes performance when the cache is not specialized to any specific application/phase and can be defined by the configurable cache's maximum parameter values, since the maximum parameter values typically maximize performance. Our algorithm also determines P_i's deadline, which may be application designer-specified or implicitly specified by the executing application [8] due to input data stream processing requirements.

To minimize the number of configurations explored and the tuning overhead during the cache configuration adjustment stage, our algorithm determines an initial cache configuration $ConfigP_{i\_init}$ that is presumably closer to the best configuration than the base configuration. To determine *ConfigPi_init*, our algorithm uses either P_i's PDM configuration or the most similar phase's best cache configuration $ConfigP_{msp}$, where the most similar phase is the previously executed phase with the smallest phase distance from P_i. To determine P_i's PDM cache configuration, our algorithm uses PDM's configuration estimation algorithm. Since PDM requires a characterized base phase, our algorithm uses an arbitrary base phase (e.g., the first executed phase) and can use any efficient tuning/exploration heuristic (e.g., [14]) to minimize design space exploration and determine the base phase's best cache configuration without incurring significant tuning overhead. This tuning heuristic is only used during the first phase's execution and subsequent phases' PDM configurations are determined directly using PDM.

To determine the most similar phase's best cache configuration, our algorithm calculates the phase distance D between the currently executing phase P_i and all previously characterized phases P_{i-1}... P_{i-n}, where n is the number of previously characterized phases. P_i's most similar phase P_{msp} is the phase with the minimum D from P_i. Our algorithm executes P_i in P_{msp}'s best configuration and compares this configuration's EDP to P_i's PDM best configuration's EDP. Our algorithm then sets $ConfigP_{i\_init}$ to the lowest EDP configuration and stores this configuration in the phase history table. If the base configuration achieves lower EDP than P_i's PDM and P_{msp} configurations, our algorithm uses the base configuration as $ConfigP_{i\_init}$. After determining $ConfigP_{i\_init}$, our algorithm starts the configuration adjustment stage to determine P_i's best configuration.

B. Cache Configuration Adjustment Stage

Fig. 2 depicts the cache configuration adjustment stage, which improves on $ConfigP_{i\_init}$'s EDP by iteratively tuning

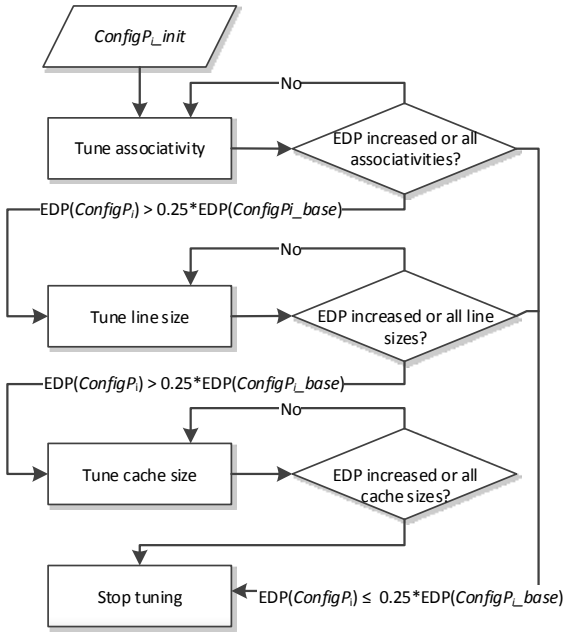

Fig. 2. Cache configuration adjustment stage

ConfigPi_init's parameter values. Iterative tuning executes each potential configuration for one tuning interval to evaluate the configuration's EDP and adherence to design constraints.

Our tuning algorithm exploits multimedia application characteristics for efficient parameter tuning. Since multimedia applications typically partition data into smaller blocks that contain significant data reuse and spatial locality, our algorithm iteratively tunes the cache sizes and the line sizes from the largest to smallest consecutive/unexplored values, and associativities from smallest to largest. Additionally, to prevent explored configurations from significantly increasing $ConfigP_{i\_init}$'s EDP, our algorithm tunes the cache parameters in increasing order of the parameter's impact on the EDP: associativity, followed by the line size, and finally the cache size [20]. Each cache parameter is incremented/decremented in powers of two starting from $ConfigP_{i\_init}$, while holding the other parameters fixed until there is no EDP reduction or all of the parameter values have been explored, wherein the parameter value with the lowest EDP is that parameter's final determined value in $ConfigP_i$.

To minimize tuning overhead from exploring too many inferior cache configurations, our algorithm compares the executing configuration's EDP to the base configuration $ConfigP_{i\_base}$'s EDP and only iterates in the cache configuration adjustment stage if subsequent explored configurations are expected to significantly reduce the EDP. Based on empirical analysis, we assumed 25% to be a significant EDP reduction (Section IV.B verifies that this EDP reduction produced near-optimal EDP savings). If the executing configuration's EDP is not 25% less than $ConfigP_{i\_base}$'s EDP, our algorithm iteratively tunes the cache parameters until there is no EDP reduction in order to determine $ConfigP_i$.

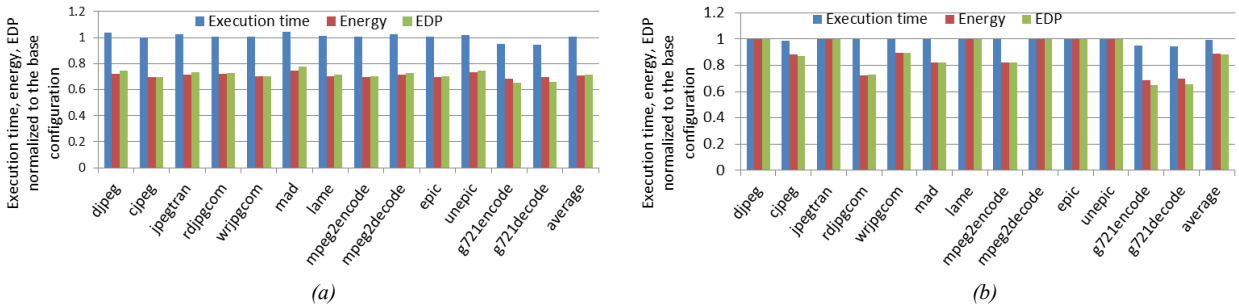

Fig.3. Execution time, energy, and EDP normalized to the base cache configuration using (a) Deadline-1 and (b) Deadline-2

IV. Experimental Results

A. Experimental Setup

We quantified our phase-based cache tuning algorithm's EDP savings using thirteen benchmarks representative of common consumer multimedia applications from the Mediabench [11] and Mibench [7] benchmark suites. The benchmarks were specific compute kernels performing single tasks, such as JPEG compression and decompression, MPEG-2 decode and encode, audio encoding and decoding, etc. Since each benchmark was a small compute kernel, without loss of generality, we assumed that each benchmark represented a different phase [6] and used different benchmarks to simulate different execution phases.

To generate cache miss rates and core statistics, we used GEM5 [3] to model a processor with separate, private level one instruction and data caches, and modeled our algorithm and automated the simulations using Perl scripts. For each phase and each configuration explored, we simulated GEM5 using the configuration's parameter values. We calculated the total system energy consumed, execution time, and EDP with McPAT [12] using the statistics generated from GEM5. We calculated the EDP in Joule seconds as:

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

where *system_power* included the core and cache powers and *total_phase_cycles* was the number of cycles to execute a phase to completion.

To quantify EDP savings as compared to a non-configurable cache, we compared to an 8 Kbytes, 4-way set associativity base cache with a 64 byte line size, which represented an embedded microprocessor suitable for our experimental applications [20]. Given this base cache, the configurable cache sizes ranged from 2 to 8 Kbyte, associativities ranged from direct-mapped to 4-way, and line sizes ranged from 16 to 64 bytes, all in power-of-two increments. We also quantified the EDP savings as compared to the optimal, lowest EDP cache configuration, which we determined by exhaustively simulating all of the configurations in the design space for each phase.

We used a tuning interval of one million cycles, since our experiments showed that this tuning interval was long enough to stabilize the cache statistics. The phases were looped to allow our algorithm to complete cache tuning.

To evaluate various real-world application requirements, we defined four deadlines that represented different timing constraints and provided insights into our algorithm's ability to adhere to these timing constraints. Deadline-1 represented loose timing constraints with no specified deadlines. Deadline-2 represented stringent timing constraints with the phases' deadlines set to the phase's minimum execution time, which was defined by the configuration with the shortest execution time. Deadline-3 and -4 represented less stringent timing constraints with the deadlines set to 5% and 10% longer than the minimum execution time, respectively.

B. Results

Fig. 3 (a) and (b) depict the execution time, energy, and EDP of the best configuration as determined by our phase-based cache tuning algorithm normalized to the base cache configuration (baseline of one) for Deadline-1 and -2, respectively. Fig. 3 (a) shows that our algorithm achieved 29% energy and EDP savings on average over all of the phases, with energy and EDP savings as high as 32% and 35%, respectively, for *g721encode*. The average energy and EDP savings were within 1% of the optimal. Our algorithm determined the optimal EDP configurations for ten of the thirteen phases, while *mad*, *mpeg2encode*, and *epic*'s EDP were within 1% of the optimal. On average over all of the phases, our algorithm degraded the execution time by only 1%, and up to 4% for *djpeg* and *mad*. However, our algorithm reduced the execution time for *g721encode* and *g721decode* by 5% and 6%, respectively. Since Deadline-1 had no specified deadlines, our algorithm was able to minimize the EDP while imposing minimal execution time degradation.

Fig. 3 (b) shows our algorithm's ability to reduce the energy consumption and determine configurations that met the stringent, minimum execution deadlines for all phases. Due to these deadlines' stringent timing constraints, the energy and EDP savings reduced by 11% and 12%, respectively, as compared to Deadline-1's EDP savings, with savings as high as 32% and 35%, respectively, for *g721encode*. For six of the phases, the base configuration defined the minimum execution time, thus our algorithm determined the base cache configuration as the best cache configuration and these phases showed no energy and EDP savings.

Results showed (details omitted for brevity) that Deadline-3 and -4's timing constraints did not degrade our algorithm's ability to achieve significant EDP savings. Our algorithm determined cache configurations with execution times that

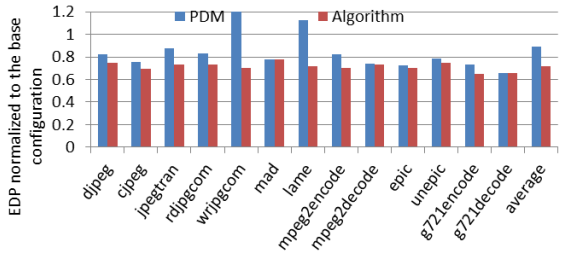

Fig. 4. PDM's and our algorithm's configurations' EDPs normalized to the base configurations (baseline of one).

varied from the minimum execution time by only 1% as compared to Deadline-1, determined configurations that met all of the phases' deadlines, and achieved similar EDP and energy savings as Deadline-1.

Fig. 4 depicts PDM's and our algorithm's configurations' EDP savings normalized to the base cache configuration for each phase for Deadline-1. We evaluated Deadline-1 only since PDM provided no mechanism for adhering to specified deadlines. Even though PDM's configurations improved over the base configurations for most phases, on average over all of the phases, PDM achieved only 11% EDP savings while our algorithm's EDP savings was 29%, an 18% increase over PDM, with increased savings as high as 62% for *wrjpgcom*. Additionally, our algorithm reduced PDM's execution time degradation by 7% (detailed results omitted for brevity). Our algorithm achieved more EDP savings than PDM in all of the phases except *mad* and *g721decode*, where the EDP savings were similar because PDM also determined the optimal configuration.

V. Conclusions

Multimedia applications are becoming pervasive in embedded systems, and much research focuses on optimizing these applications' energy consumptions without significantly degrading the execution time and adhering to timing constraints. In this work, we presented a phase-based cache tuning algorithm for multimedia applications that leverages multimedia application characteristics, such as high spatial locality, to determine the best cache configuration for each execution phase with no design time overhead. Results showed our algorithm's effectiveness in determining the best cache configurations that maximized energy delay product (EDP) savings while adhering to stringent execution deadlines. On average, our algorithm determined cache configurations within 1% of the optimal configurations, with an average EDP savings of 29% as compared to using a default base configuration, and an execution time degradation of only 1%. Future work includes evaluating our algorithm in systems with multiple cache levels and extending our algorithm to tune other configurable hardware that have significant impacts on multimedia applications (e.g., clock frequency, instruction issue width, etc.).

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.

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

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

[4] A. Ghosh and T. Givargis, "Cache optimization for embedded processor cores: an analytical approach," International Conference on Computer-Aided Design, 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, 2008.

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

[7] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown, "MiBench: a free, commercially representative embedded benchmark suite," International Workshop on Workload Characterization, 2001.

[8] C. J. Hughes, P. Kaul, S. Adve, R. Jain, C. Park, and J. Srinivasan, "Variability in the execution of multimedia applications and implications for architecture," ISCA, 2001.

[9] C. J. Hughes, J. Srinivasan, and S. Adve, "Saving energy with architectural and frequency adaptations for multimedia applications," MICRO, 2001.

[10] W. Lee, K. Patel, and M. Pedram, "Dynamic thermal management for MPEG-2 decoding," International Symposium on Low Power Electronics and Design, 2006.

[11] C. Lee, M. Potkanjak, and W. Mangione-Smith, "MediaBench: a tool for evaluating and synthesizing multimedia and communication systems," International Symposium on Microarchitecture, 1997.

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

[13] 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.

[14] M. Rawlins and A. Gordon-Ross, "CPACT – The conditional parameter adjustment cache tuner for dual-core architectures," International Conference on Computer Design, 2011.

[15] X. Shen Y. Zhong, and C. Ding, "Locality phase prediction," International Conference on Architectural Support for Programming Languages and Operating Systems, 2004.

[16] 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, 2001.

[17] J. Srinivasan and S. Adve, "Predictive dynamic thermal management for multimedia applications," International Conference on Supercomputing, 2003.

[18] Z. Xu, S. Sohoni, M. Rui, and Y. Hui, "An analysis of cache performance of multimedia applications," IEEE Transactions on Computers, January 2004.

[19] I. Yeo, H. Lee, E. Kim, K. Yum, "Effective dynamic thermal management for MPEG-4 decoding," International Conference on Computer Design, 2007.

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