# Quality of Service-Aware, Scalable Cache Tuning Algorithm in Consumer-based Embedded Devices

Mohamad Hammam Alsafrjalani and Ann Gordon-Ross
Department of Electrical and Computer Engineering University of Florida (UF), Gainesville, FL, USA
E-mail: mha8@ufl.edu, ann@ece.ufl.edu

**Abstract—To meet energy and quality of service (QoS) constraints in consumer/user-based embedded devices (CEDs), configurable caches can be tuned to a *best* configuration that consumes the least amount of energy while adhering to QoS expectations. However, due to disparate consumer QoS expectations and a myriad of unknown, third-party CED applications, tuning caches in CEDs is very challenging. In this paper, we propose a quality of service-aware, scalable tuning algorithm for configurable caches, which requires no a priori knowledge of applications or design-time efforts.**

## I. INTRODUCTION AND BACKGROUND

Consumer-based embedded devices (CEDs) (e.g., smartphones, tablets, wearable computing, etc.) have quality of service (QoS) expectations, which is the user-expected system performance (e.g., expected response time to user touch-input, global positioning system (GPS) accuracy, etc.). CEDs executing applications, such as drawing, audio compression and recording, video frame decoding, heartrate reading/monitoring, speech-to-text/text-to-speech translation, GPS route tracking, etc. can be viewed as soft real time systems, wherein the application's execution-time constraints are tightly coupled with the QoS (i.e., missed deadlines degrade the user's satisfaction with system performance). Due to increasing consumer demands for these applications, and the proliferation of increasingly performance-capable multicore CEDs with limited battery reserves, reducing energy consumption while meeting QoS expectations is a key design challenge, since reducing the energy consumption trades off performance capabilities and potentially degrades QoS.

QoS degradation occurs when an application executes below an expected performance level—a performance threshold—which may be evaluated as slower-than-expected response to consumer/user touch-input, GPS route updates, choppy video playback, etc. Thus, the key challenge is to reduce the CED's energy consumption by as much as possible without falling below this performance threshold, which is highly user-subjective.

One of the most effective way to regulate energy consumption while maintaining acceptable QoS is optimizing system components that have high impacts on both energy and QoS; the cache/memory hierarchy [3][9][10][11]. Configurable caches [10] enable adherence to an application's unique locality requirements using configurable parameters. Configurable parameters' values can be tuned/adjusted to the *best configuration* that most closely adheres to locality requirements, and thus most closely adheres to the energy and QoS expectations. However, determining the best configuration imposes tuning overhead [6] that degrades QoS or requires static application profiling [9] that may not scale to unknown applications, which limits applicability to the rapidly expanding number of, largely third-party, applications.

To address these challenges, we propose a quality of service-aware, scalable cache tuning algorithm for multicore CEDs with configurable caches that maintains user-expected QoS expectations while reducing energy consumption. We architect a dynamic, general purpose CED cache-tuning algorithm that *requires no a priori knowledge* or *static profiling* of the applications. To accommodate our tuning algorithm's hardware requirement, we build our work based on hardware cache tuners [1] and design a low hardware overhead tuning table to hold the required tuning information used by our algorithm.

Our results reveal disjointed trends for energy savings and tuning-time QoS degradation, which allows our tuning algorithm to prioritize QoS or energy savings. Our algorithm exploited this flexibility using two tuning modes and achieved average energy savings as high as 20.68% and 25.14%, for the data and instruction caches, respectively, and imposed average tuning-time QoS degradation as low as 0.7 occurrences on average, as compared to an off-the-shelf, base configuration commonly found on CEDs. Compared to our results, prior work could reveal additional energy savings, however, prior required a priori knowledge of the applications to avoid a high QoS degradation of 7.7 on average.

## II. HARDWARE LAYOUT AND REQUIREMENTS

Without loss of generality, we illustrate our methods using a quad core system with configurable cache size, line size, and associativity, and a runtime cache tuner, as depicted in Figure 1. To limit runtime intrusion, we use a global hardware cache tuner connected to all cores. For a quad core system, this tuner imposes 322 cycles overhead [1], which negligible considering the number of cycles to execute 1 million instructions. Prior work shows that this tuning interval is long enough to warm up and stabilize the caches [2].

The cache tuner monitors the cores' states (busy, idle, etc.) and reads the application's execution characteristics (e.g., cache miss rate, number of instructions, cycle count) at the end of each *tuning interval*. Tuning intervals are the length of time that the core executes in a particular configuration in order to determine that configuration's execution characteristics. Tuning decisions guide how the design space is explored (i.e., the order and number of configurations evaluated), and the cache tuner makes a tuning decision at the end of each tuning interval using a dynamic tuning algorithm (Section III), the application's execution characteristics, and the application's performance threshold. We assume the performance threshold can be

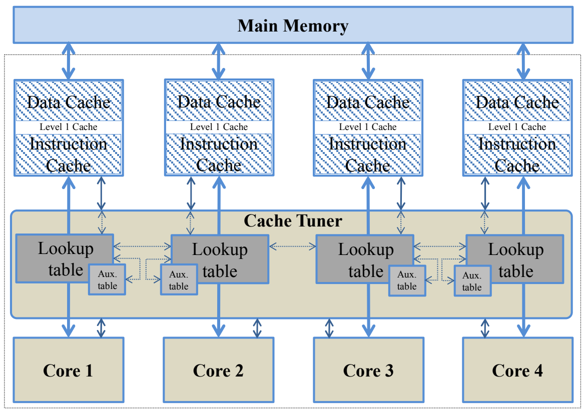


Figure 1: Quad core system with configurable caches and a global cache tuner.

GLSVLSI '16, May 18 - 20, 2016, Boston, MA, USA
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-4274-2/16/05…$15.00
DOI: http://dx.doi.org/10.1145/2902961.2902987

relayed to the tuner by the operating system using memory mapped I/O, special instructions, etc. depending on the implementation details [2].

Our tuner stores the tuning information for tuning decisions in lookup tables, with one entry per application, which includes: the *energy consumption* of the best configuration explored thus far and *configuration characteristics*, which contain per-configuration information. The per-configuration lookup table includes: *explored*, *lowest energy,* and *QoS adherence*, which flag whether or not that configuration has been previously explored, is the lowest energy configuration explored thus far, and adheres to the application's QoS expectations, respectively. The per-application tuning information storage requirement *m* in bits is:

$$m = e + b * \lceil log_2(c) \rceil \quad (1)$$

where $e$ and $b$ are the number of bits required to store the energy consumption and configuration characteristic information, respectively, and $c$ is the number of configurations in the design space. To evaluate the storage requirements as compared to prior work, we assume that *e* is 32 bits, similarly to [1], and *b* is 3 bits, one for each configuration's characteristic flag.

To limit area overhead, this complete tuning information is only stored for an application during tuning. After the best configuration is determined, only that configuration is retained in a smaller auxiliary lookup table. To enable scalability to an arbitrary number of applications we utilize a least recently used replacement policy to govern the auxiliary table's information. Thus, given *a* applications, the total tuning information storage requirement *M* in bits is:

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

## III. DYNAMIC TUNING ALGORITHM

### A. Overview

Algorithm 1 depicts our dynamic tuning algorithm's finite state machine, which contains three states (Section III.B): *information input*, *exploration*, and *evaluation*. For each application, the algorithm explores the design space, one configuration per tuning interval, based on the tuning mode. Tuning is complete when the algorithm meets the tuning mode's stop-tuning condition (e.g., the conservative mode evaluates one cache parameter only) (Section III.B.iii).

### B. Algorithm States

*(i) Information Input State:* When an application is scheduled to a core, either at inception or resumption, the algorithm begins in the *information input state*. First, the algorithm checks the auxiliary table to determine if this application's best configuration has already been determined. If the application is in the auxiliary table, the algorithm tunes the cache directly to the application's best configuration. Otherwise, the algorithm checks if the application is tuned for another core, and if so copies the tuning information from the other core's table to the current core's table, before tuning resumption. Otherwise, the algorithm attempts to the application's *read the tuning information* from the core's lookup table, and begins/resumes design space exploration in the *exploration state*.

*(ii) Exploration state:* The application's lookup table information provides the algorithm with information about all of the previously explored configurations. Using this information, the algorithm determines the next configuration to explore, and the application executes in that configuration for one tuning interval, after which the algorithm transitions to the *evaluation* state.

If there is no tuning information, design space exploration begins in the *base* configuration, which is a configuration that is guaranteed to meet QoS expectations. Even though the base configuration could be any of the CED's configurations, we select the configuration with the largest cache size, line size, and associativity as the base configuration since this configuration has the best potential for being the highest performance configuration. Considering the cache behavior and requirements for our test applications, and to compare our work to prior work [10], our base configuration is an 8KB, 4-way associativity cache with a 32B line size.

If there is tuning information, the algorithm resumes design space exploration using this information. Since the lookup table contains per-configuration information for only configurations that have already been explored, denoted with a set *explored* flag, the algorithm can determine the next parameter and value to explore by traversing the exploration ordering and locating the first configuration that does not have *explored* set. This configuration becomes the *current* configuration to evaluate.

Exploration ordering determines the order in which the configurable parameters and parameter values are explored. To minimize tuning-time QoS degradation, the algorithm traverses the design space in the reverse order of the parameters' impacts on performance [11] (i.e., lower impact to higher impact). This reverse parameter-performance-impact ordering reduces QoS degradation incurred during design space exploration. Since the base configuration is set to the largest parameter values, the algorithm explores configurations similar to the base configuration first.

In order to attain better energy savings, the algorithm will explore configurations with higher energy savings potential, but these configurations also have a higher risk of larger QoS degradation during tuning—extremist configurations that have energy consumptions extremely higher than the base configuration [3]. The algorithm regulates energy savings to QoS degradation using two tuning modes: moderate and conservative modes. Additionally, since larger parameter values have higher performance capabilities as compared to smaller parameter values [11], the algorithm traverses the parameters' values from largest to smallest.

The *exploration state* begins design space exploration with all parameters set to the parameters' largest values. Starting with the associativity, the algorithm evaluates the associativity values from largest to smallest. After the algorithm explores all of the associativity values, the algorithm reduces the line size value, and re-explores all of the associativity values (largest to smallest) for that line size, and repeats this process until all line size values have been explored. Similarly, once all of the line size values have been explored, the algorithm reduces the cache size, and re-explores all of the line size values (largest to smallest), and repeats this process until all cache size values have been explored. After each explored configuration, the algorithm sets the application's configuration's *explored* flag and proceeds to the *evaluation* state.

*(iii) Evaluation state:* In the evaluation state, the algorithm evaluates the current configuration's energy consumption and adherence to QoS expectations. Based on this evaluation's outcome, the tuning mode, and any prior tuning information, the algorithm makes tuning decisions.

If this is the first tuning interval, the algorithm does not have any tuning information in the lookup table. The algorithm adds a new entry to the lookup table for this application, the current configuration's energy consumption is stored in the *energy consumption* (i.e., this is the lowest energy configuration thus far), and the current configuration's *lowest energy* flag is set.

If this is not the first tuning interval, the algorithm compares the current configuration's energy consumption to the lowest energy configuration explored thus far. If the current configuration's energy is lower, the algorithm sets the current configuration's *lowest energy* flag, and proceeds to evaluate the configuration's QoS adherence before invalidating the prior lowest energy configuration's *lowest energy* flag or updating the *energy consumption* field (i.e., the configuration must both reduce energy as compared to the prior lowest energy configuration and adhere to QoS).

The algorithm evaluates QoS adherence by comparing the current configuration's execution characteristics to the application's performance threshold. If the current configuration meets the

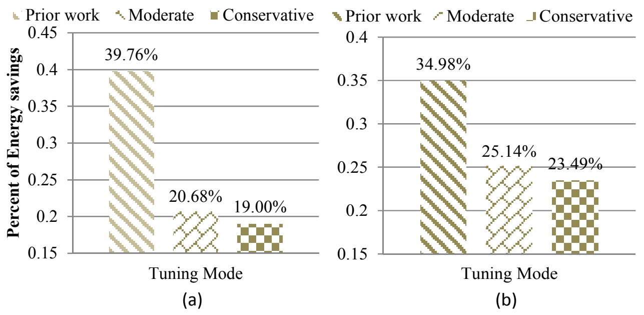


Figure 2. Average best configurations' energy consumptions normalized to the base configuration for all applications and tuning mode for the (a) data and (b) instruction caches

application's performance threshold, then the current configuration adheres to QoS expectations, and the algorithm sets the current configuration's *QoS adherence* flag, invalidates the prior lowest energy configuration's *lowest energy* flag, and updates the *energy consumption* field to the current configuration's energy consumption.

After this comparison, the algorithm makes tuning decisions, which uses the explored configurations' flags' values and the mode's stop-tuning stopping condition. The moderate mode limits tuning-time QoS degradation by only exploring one configuration that degrades QoS. Thus, moderate mode stops tuning as soon as the algorithm explores *one* configuration with the *adherence to QoS* flag unset (i.e., that configuration did not adhere to QoS requirements). To further limit tuning-time QoS degradation, not only does the conservative mode stop tuning as soon as a configuration with an unset *adherence to QoS* flag is explored, but also as soon as the algorithm explores all parameters with low performance impacts—associativity or line size, whichever comes first. Since high-performance-impact parameters have a higher possibility of degrading QoS, this additional stopping condition reduces the *probability* of exploring QoS-degrading configurations.

If the algorithm stops tuning, the best configuration has been determined, which is the configuration with the *lowest energy* and *adherence to QoS* flags set. The algorithm removes the application from the lookup table and stores the application's best configuration in the auxiliary table (Section II).

## IV. EXPERIMENTAL RESULTS

### A. Experimental Setup

To quantify our tuning algorithm's efficacy, we selected ten applications, from MiBench [7] suite, that represent consumer applications typically execute on CED systems. Our applications included audio (adpcm code/decode), image (jpeg code/decode), networking (dijkstra, patricia), security (sha), office (string search), and telecomm (gsm code/decode) applications.

We test our algorithm's ability to save energy while maintaining QoS expectations by designating a minimum performance threshold for our applications. Since these applications do not have/require defined QoS expectations, we determined reasonable performance thresholds based on the execution times for the configurations in the design space (Section II). To ensure that a reasonable number of configurations are able to meet QoS expectations (i.e., our algorithm would not be rigorously tested if *all* configurations could meet QoS expectations), for each application, we calculated the average performance across all of the configurations, and used that average as the application's performance threshold. Given $m = 10$ and $c = 18$ and Equations (1) and (2), our tuner and auxiliary table required 940 bits. For comparison, a prior work's hardware cache tuner with more storage requirements as compared to our design (e.g., per-configuration energy consumption was also stored) imposed only a 4.7% area overhead in a very small MIPS M4K processor [1], thus our tuner's area overhead will be less than 4.7%. Our tuner and auxiliary table imposed less than 0.5% power overhead for our experimental system (Figure 1), which we based on our extrapolations from [1] and the tuner size.

Since our system has private, dedicated L1 data and instruction caches that connect directly to main memory, we can assume that there are no tuning dependencies between the data and instruction caches, and the caches can be tuned simultaneously and evaluated independently. Our algorithm could also explore design spaces with multiple levels of caches in a similar manner, however, since the L1 cache interdependencies introduced through a shared level-two cache vastly increases the design space, multi-cache, multi-pass evaluation heuristics can be used for efficient exploration [6]. We executed our applications using cycle-accurate simulator, gem5 [5] to obtain performance values, and McPAT [8] to calculate the energy values for contemporary 40nm technology.

### B. Evaluation Methodology

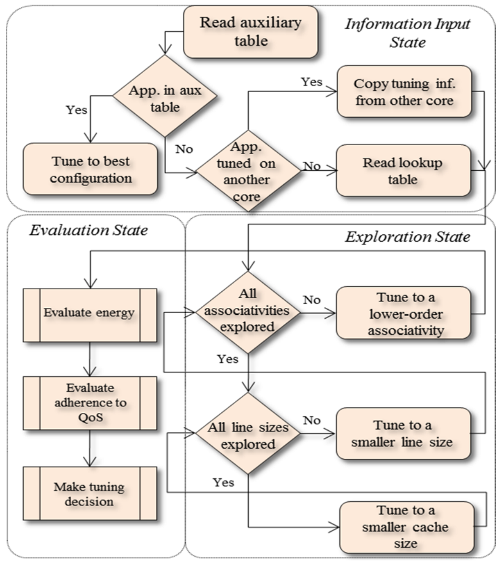
We evaluate our algorithm's energy savings for each tuning mode by normalizing our algorithm's determined configurations' energy consumptions to a base, non-configurable quad-core system's energy consumption (largest parameter values). We quantify the tuning energy overhead by normalizing the application's energy consumption during tuning to the application's energy consumption while executing with the base configuration for the same execution time as tuning required.

We evaluated the adaptability of each tuning mode to meet disparate QoS expectations during tuning by counting the number of times the algorithm explored a QoS-degrading configuration. Since QoS-degrading configurations have disparate execution times, QoS degradation varied in severity based on the configuration's performance. Since the severity of QoS degradation varies between configurations and we plan to quantify this severity in future work.

To compare our algorithm to prior work [11], we modified our algorithm analogously to [11]. We incorporated an *exhaustive* mode to search the complete design space, regardless of QoS degradations. Additionally, the exhaustive mode reveals the energy savings that would be obtained using prior work [9]. However, that prior work required a priori knowledge of the applications to alleviate tuning overhead. Since the exhaustive mode does not require a priori application knowledge, and imposes tuning overhead, our results for the exhaustive mode provides insights on the tradeoffs between energy savings and a priori application knowledge requirements.

### C. Results and Analysis

*(i) Energy Savings:* Figure 2 depicts the percentage of average energy savings achieved by the algorithm's best



Read auxiliary table
Information Input State
App. in aux table
Yes
No
Copy tuning inf. from other core
Tune to best configuration
App. tuned on another core
Read lookup table
Evaluation State
Exploration State
Evaluate energy
All associativities explored
Tune to a lower-order associativity
Evaluate adherence to QoS
All line sizes explored
Tune to a smaller line size
Make tuning decision
Tune to a smaller cache size

Algorithm 1: Dyanamic tuning algorithm's finite state machine

configurations for our ten applications, with respect to the base configuration's energy consumption for each tuning mode for the (a) data and (b) instruction caches.

The tuning modes exhibit similar trends for both the data and instruction caches. As expected, the aggressive mode (as in prior work [9][11]) achieved the highest average energy savings of 39.76% and 34.98% for the data and instruction caches, respectively. Even though the moderate and conservative modes explored fewer configurations, these modes still achieved average energy savings of 20.68% and 25.14%, and 19.00% and 23.49%, respectively.

Our energy overhead analysis revealed that all tuning modes imposed tuning overhead while exploring inferior, non-best configurations. However these overheads are amortized while the algorithm explored configurations that resulted in energy savings, as compared to the base configurations. Furthermore, the aggressive mode imposed more energy overhead, as compared to the moderate and conservative mode. However, the aggressive mode also explored more energy saving configurations, as compared to the moderate and aggressive mode. Since the moderate and conservative modes explored fewer configurations, these modes had a lower probability of exploring extreme configurations, while also exploring energy-saving configurations without exhaustively exploring the design space.

Since our algorithm explores configurations with large parameter values first, the moderate and conservative modes have a lower probability of exploring configurations with small parameter values, which can lead to QoS degradation. The energy and overhead results and our algorithm's parameter and value exploration ordering reveal that high-energy-saving configurations tend to have smaller parameter values. However, extreme configurations also tend to have small parameter values, thus leading to high-risk/high-reward design space exploration methods. Our different tuning modes enable users/designers to moderate this decision.

Since an application's execution time and execution frequency in CED's is subjective to costumer usage, the three tuning modes' beneficence is contingent on consumer usage patterns. Applications that execute frequently, or for extended periods of continuous operation, benefit more from high energy savings over these periods, and thus can tolerate high tuning overhead (i.e., the moderate mode). Alternatively, applications that are expected to execute for a short period of time benefit more from tuning modes that incur lower tuning overhead (i.e., the conservative mode). These short-executing applications do not benefit from the aggressive mode since these applications will not execute for a period of time long enough to amortize the high tuning overhead.

*(ii) Adherence to QoS:* Since prior work explored the entire design space, prior work had the highest number of QoS degradation occurrences, with an average of 7.7 tuning-time QoS degradations. Alternatively, since the moderate mode stopped tuning as soon as the algorithm explored one QoS-degrading configuration, the moderate mode had one QoS degradation occurrence regardless of the cache hierarchy. Since the conservative mode stopped tuning as soon as the algorithm finished exploring all of the low-performance-impact parameters, the conservative mode had the lowest number of QoS degradation occurrences, with an average of 0.7. These averages are low because the conservative mode only explored configurations with large cache sizes, which typically have higher performance capabilities, and thus lower QoS degradation potential.

Even though the conservative mode limits tuning-time QoS degradation, the conservative mode did not guarantee that the best configuration found adheres to QoS expectations. This suggests that not only configuration design space size, but also the number of configurations with low/high performance-impact parameters impact both tuning-time QoS degradation and post-tuning QoS adherence. A design space that has a low percentage of configurations with large parameter values leads to a high number of tuning-time QoS degradation occurrences for the aggressive mode. Alternatively, since high-energy-saving configurations tend to be small parameter value configurations, a design space that has a high percentage of configurations with large parameter values leads to less energy savings.

The difference between the numbers of tuning-time QoS degradation occurrences for the different tuning modes provides the algorithm with the flexibility to adhere to disparate user QoS expectations, which is a necessity for CEDs. Since user expectations vary based on the user's mood, experience level, location (e.g., work, commute, home, etc.), usage (e.g., naval, space, entertainment, etc.), gender, age, environment, time of day, etc. [4], our algorithm is capable of adhering to disparate QoS expectations for different user-defined experiences.

## V. CONCLUSIONS AND FUTURE WORK

In this work, we proposed a tuning algorithm, which determined the best cache configuration while considering tuning-time QoS degradation and energy consumption. Our algorithm requires no a priori application information, profiling information, or design time effort, and tunes the cache during runtime while avoiding tuning-time QoS degradation using two tuning mode options that trade off these competing constraints. Our results revealed average energy savings as high as 20.68% and 25.14%, for the data and instruction caches, respectively, and average tuning-time QoS degradation as low as 0.7. Future work will extend our algorithm to multi-level cache optimization for CEDs, additional tuning heuristics, and disparate QoS and energy savings tradeoffs.

## VI. ACKNOWLEDGEMENT

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.

## VII. REFERENCES

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

[2] Adegbija, T., Gordon-Ross, A. "Energy-efficient phase-based cache tuning for multimedia applications in embedded systems," *IEEE Consumer Communications and Networking Conference*, 2014.

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

[4] Amin, R., Jackson, F., Gilbert, J, Martin, J., Shaw, T. "Assessing the Impact of Latency and Jitter on the Perceived Quality of Call of Duty Modern Warfare 2," *Int. Con. on Human Computer Interaction,* 2013.

[5] Binkert, N.; Beckmann, B.; Black, G.; et al. "The gem5 simulator," SIGARCH Comput. Archit. News 39, 2 (August 2011), 1-7.

[6] Gordon-Ross, A., Viana, P., Vahid, F., Najjar W. Barros, E. "A One-Shot Configurable-Cache Tuner for Improved Energy and Performance," *IEEE/ACM Design, Automation and Test in Europe.* 2007

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

[8] Li, S.; Ahn, Jung Ho; Strong, R.D.; et al. "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," IEEE/ACM Int. Symp. on Microarchitecture, Dec. 2009

[9] Wang, W., Mishra, P., Gordon-Ross, A. "SACR: Scheduling-Aware Cache Reconfiguration for Real-Time Embedded Systems," *Int. Con. on VLSI Design,* 2009.

[10] Zhang, C., Vahid, F., Najjar, W. "A highly configurable cache architecture for embedded systems," *In Proc. of International Symposium on Computer Architecture,* 2003.

[11] Zhang, C., Vahid, F. "Cache configuration exploration on prototyping platforms," *IEEE International Workshop on Rapid Systems Prototyping,* 20