Realizing Closed-loop, Online Tuning and Control for Configurable-cache Embedded Systems: Progress and Challenges

Islam S. Badreldin*, Ann Gordon-Ross*, Tosiron Adegbija†, and Mohamad Hammam Alsafrjalani‡
*University of Florida, †University of Arizona, ‡University of Miami
*{ibadreldin,anngordonross}@ufl.edu, †tosiron@email.arizona.edu, ‡alsafrjalani@miami.edu

***Abstract*—The cache subsystem is a major contributor to energy consumption in commercial microprocessors used in embedded systems. To reduce energy, designers can perform design space exploration (DSE) to determine a suitable cache configuration that matches system constraints and goals while minimizing energy consumption. Traditionally, this cache tuning step has been a static process where heuristics or analytical models are used to determine an optimal or near-optimal cache configuration prior to runtime given a known application, application set, or application domain. Even though the configuration may change during runtime for different phases of execution, the specific configuration for each phase remains fixed. This static nature is too restrictive for modern, complex embedded systems that are expected to operate under diverse, unknown operating environments, run unknown applications, and with vastly different user quality of experience (QoE) expectations (e.g., smart phones). Therefore, cache tuning must change from a static optimization process to a dynamic optimization process that adapts online during runtime transparently to the user/system needs. The key challenge is determining the configuration that adheres to QoE expectations while minimizing energy consumption without degrading the user experience during DSE. Despite the wealth of progress that has been made, the realization of a closed-loop, fully adaptive, online-tunable cache subsystem still faces many challenges. In this paper, we review the progress made in the area of static and dynamic cache tuning, discuss the challenges that still exist in this area, and propose a prediction-assisted control-theoretic framework to address these challenges.**

I. Introduction and Motivation

Heterogeneous multicore processors are becoming pervasive in embedded systems as a way to enable the efficient execution of increasingly complex applications while minimizing energy consumption. Using a system with different core configurations enables applications to execute on a core that is suitable given the application's performance/energy goals. Different core configurations range from coarse-grained optimization options, such as cores with different instruction set architectures (ISAs), voltages, frequencies, specialized hardware (graphic processing units (GPUs) and field-programmable gate arrays (FPGAs)), etc. to fine-grained optimization options, such as different pipeline depths, reorder buffer size, instruction issue width, etc. Since the cache subsystem consumes 16% to 50% [1], [2] of the total system power, this fine-grained option offers potentially significant energy savings, thus making the cache subsystem an ideal optimization candidate.

The cache subsystem provides optimized performance and reduced energy consumption by exploiting application-specific resource requirements using the *best* (optimal or near-optimal) cache configuration that most closely adheres to system constraints and goals. This configuration designates the best values for the cache's tunable parameters, such as cache size, line size, and associativity. Since prior work showed that tuning these particular cache parameters enable the largest energy savings potential [3], without loss of generality, we assume that these parameters are tunable in our work.

To provide flexible optimization options, configurable caches (e.g., [3]) must be suitable for modern complex embedded systems. These caches must operate in unknown and changing environments, run a wide variety of potentially unknown applications, and adhere to system constraints, goals, and runtime-defined time-varying user-defined quality of experience (QoE) expectations. Given the diversity of subjective end-user QoE expectations, the tunable cache parameters must now also be able to change during runtime in response to user-specific feedback as well as application requirements while adhering to system constraints.

Achieving this goal is extremely challenging due to many factors since QoE expectations vary substantially during runtime for individual users and between different users. For example, based on a user's battery lifetime or performance quality expectancies, the best configuration can range from lowest energy to best performance, respectively. These expectancies may differ for different types of applications and environmental conditions, such as time-of-day (e.g., less frequent notifications during work hours), ambient lighting (e.g., slower frame rates in low lighting), accelerometer readings (e.g., fewer GPS updates while walking as compared to driving). Since different applications or application *phases* (i.e., execution intervals with stable performance characteristics, such as instructions per cycle (IPC), cache miss rate (CMR), etc.) have varying runtime memory requirements [4], [5], [6], [7], the best cache configuration must correlate these differences with the user QoE expectations, resulting in an extremely large and complex design space.

Design space exploration (DSE) is an optimization process that dynamically evaluates an *optimization objective function* to dynamically determine the best cache configuration at runtime. The objective function includes *optimization metrics* [8],

such as application performance and/or energy consumption. Efficient and effective DSE is a challenging process, and prior work has extensively explored static and dynamic solutions (e.g., [5], [6], [9], [10], [11], [12], [13], [14], [15], [16], [17]) using heuristic searches, subsetting methods, or analytical models.

Complex heterogeneous architectures exacerbate many cache tuning challenges, and introduce new challenges that necessitate novel dynamic cache tuning methods. To the best of our knowledge, few prior works have begun to address these challenges, some of which include: determining when to reconfigure the cache–the *tuning interval*–with no prior knowledge of the running applications [5]; considering runtime-defined time-varying user-defined QoE expectations and optimization metric objectives; considering the general complexity of multicore architectures (e.g., cache coherence, data sharing among cores, synchronization, etc.).

Another critical challenge is addressing DSE's impact on the user experience. Since exploring configurations far from the best configuration incurs significant tuning overhead (e.g., significantly increased power or reduced performance) [5], DSE must ensure that while determining the best configuration, the user experience is not degraded beyond a dissatisfaction threshold (e.g., the point at which the user experiences anger or frustration with the device). One recent work by Alsafrjalani et al. [6] studied online tuning while considering DSE-related degradation and showed that best configurations could be determined while limiting this degradation.

Despite all of the advancements in cache tuning methods, these outstanding challenges leave a void in adaptive, online, closed-loop cache tuning systems that consider runtime-defined time-varying user-defined QoE expectations while limiting DSE-related degradation. In this work, we review the progress made toward static and dynamic cache tuning, detail some of the challenges that face the progress in the area of online dynamic cache tuning, and propose a system architecture that addresses some of these challenges.

II. General Problem Formulation

In this section, we formally define the general problem of cache tuning for configurable-cache embedded systems, define optimization metrics, and review general DSE solutions.

A. Formal Notations

Following the notation in [11], we denote a set of n applications $A = \{a_1, a_2, \ldots, a_n\}$ to be executed on an embedded system with a configurable cache offering a set of m possible cache configurations $C = \{c_1, c_2, \ldots, c_m\}$. Each application has a set of k_i phases $P_{a_i} = \{p_1, p_2, \ldots, p_{k_i}\}$, wherein each phase can require a different cache configuration. Without loss of generality, we assume an optimization process in which a Pareto-optimal configuration (i.e., best configuration) that trades off the energy and performance optimization metrics must be determined [9].

We denote the energy and performance optimization metrics for application a_i executing in phase p_k with cache configuration c_j as $e(a_i, p_k, c_j)$ and $u(a_i, p_k, c_j)$, respectively. The optimization objective is to determine a Pareto-optimal configuration $c_0 \in C$ such that for application a_i, $e(a_i, p_k, c_0) \leq e(a_i, p_k, c_j)$ where with strict inequality for some c_j , and a similar condition on $u(a_i, p_k, c_0)$. The set of all Pareto-optimal configurations is known as the Pareto-optimal set [18], which includes the best configurations in terms of both metrics [9]. At runtime, the dynamic optimization process selects a configuration from the Pareto-optimal set given additional constraints such as QoE [6], resulting in a QoE-aware optimization objective.

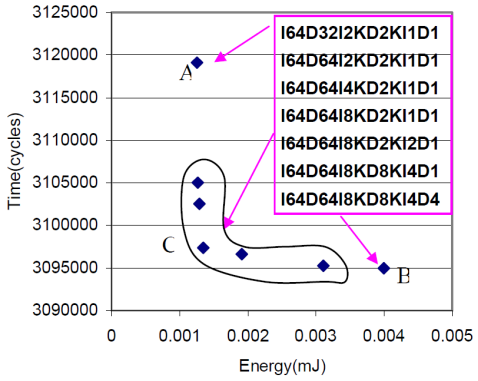

Fig. 1. Example Pareto-optimal set of cache configurations as presented in [9] where points A, B, and C belong to the Pareto-optimal set. The legend denotes tunable parameter values for each configuration.

Besides determining Pareto-optimal configurations, a simpler optimization problem that is often employed in previous work focuses on minimizing only the energy optimization metric [5], [6], [10], [11], [12], [14], [16]. In this case, an optimal solution is one such that $e(a_i, p_k, c_0) \leq e(a_i, p_k, c_j)$ with strict inequality for some c_j, and similarly for $u(.)$. An additional simplification that is sometimes used involves determining a single configuration that is best, on average, for an application's complete execution (application-based cache tuning). In this case, the dependency of the optimization metrics on the application phase is removed (i.e., $e(a_i, p_k, c_j)$ becomes $e(a_i, c_j)$ and $u(a_i, p_k, c_j)$ becomes $u(a_i, c_j)$).

B. Optimization Metrics

Pareto-optimal-based optimization involves a trade off analysis between two or more metrics. Given N optimization metrics, the Pareto-optimal set will always contain at least N required configurations. As an example, Fig. 1 illustrates this principle for a sample Pareto-optimal set for cache configurations (parameter values are denoted in the legend) that trades off total energy consumption in mJ and performance in clock cycles. The two required configurations are the minimum energy consumption, point A, and the best performance, point B, configurations. Point C also belongs to the Pareto-optimal set and represents a configuration that trades off performance and energy consumption.

In the context of configurable cache tuning, several metrics have been previously used. Common performance optimiza-

tion metrics (i.e., $u(.)$) include the CMR [9], [16], [19], or the execution time in cycles [9]. The energy optimization metric (i.e., $e(.)$) can be estimated using cache performance statistics and modeling tools [10], [14], [15], [16] such as CACTI [20], or a set of equations that represent an energy model [9], [11], [21]. The estimation can be done during runtime using cache performance hardware counters and the energy equation model can be implemented in a custom co-processor or dedicated hardware to provide a hardware-based energy calculation module [5].

C. Design Space Exploration (DSE)

There are many different methods for exploring the design space to determine the best cache configuration. Exhaustive DSE would evaluate the optimization metrics for every cache configuration $c_i \in C$. Even though this method accurately determines the optimal configuration, as the number of configurable cache parameters, parameter values, and hierarchy depth grows, exhaustive DSE is computationally and runtime prohibitive given increasingly large design spaces. Several approaches have attempted to reduce DSE complexity using heuristics and subsetting methods that reduce the number of configurations explored. These methods, while inexact, determine near-best configurations while avoiding a large penalty in terms of the optimization objective [10], [6], [9], [11], [14], [15], [16], [17].

Other approaches circumvent the complexity of DSE by using analytical or statistical models that attempt to directly determine the best cache configuration given application memory access patterns and characteristics [12], [22], or application phase information [7], [13]. Since phase-based methods provide greater benefits than application-based methods, Sections IV and V focus on phase-based approaches.

Regardless of the exploration method, DSE can be divided into two categories. In offline DSE, the design space is statically searched before runtime using full or partial knowledge of the applications or application domains that are expected to run on the embedded system [6], [9], [10], [11], [14], [15], [16], [17]. A key feature of offline exploration is that these approaches can batch-evaluate the optimization objective for each cache configuration in software (e.g., [11], [16]) or in hardware (e.g., [9]) before runtime. Online DSE profiles the applications dynamically at runtime using real input stimuli (e.g., [5], [6], [12], [13]), thus more accurately determining the best configuration at the expense of DSE overhead. We further refer the reader to [23] for a complete review of the different DSE methods. In the next two sections, we review and discuss the progress in offline and online design space exploration for configurable caches.

III. Offline Design Space Exploration (DSE)

Offline DSE can be done either in software simulation using trace-based methods or instruction set simulators, or using hardware emulation methods. Regardless of the method used, technological and system complexity advances have resulted in an exponential growth of the design space resulting in prohibitively large file sizes (for trace-based) and DSE time. To reduce DSE time, methods that approximate the design space using a subset that contains near-best solutions have been developed to prune the design space. This section discusses several solutions for offline DSE and design space subsetting.

A. Simulation- and Hardware-based Exploration

Given full or partial knowledge of the application(s) or the application domains that are expected to run on the embedded system, cycle-accurate instruction set simulators (e.g., [24], [25]) can be used to estimate the energy and performance associated with each possible cache configuration. Approaches using this method were extensively reviewed in [23]. Even though a powerful host machine can be used to run the simulator and reduce the time needed to explore each configuration, this method is only feasible for optimization problems that involve a relatively small number of possible cache configurations. For larger design spaces on more complex systems where a single configuration may take several hours or days to simulate, this time needs to be further reduced [11], [16].

DSE time can be considerably reduced using hardware-based emulation. Zhang et al. [9] developed a runtime reconfigurable hardware cache architecture that enabled the cache size, line size, and associativity to be changed using a small set of configuration registers. This architecture was intended for prototype-oriented platforms to rapidly explore the configuration design space for a set of known applications to determine the Pareto-optimal set. Considering the example Pareto-optimal set illustrated in Fig. 1, the authors implemented a hardware-based heuristic that rapidly searched for points A and B, and then searched for point C.

B. Subsetting Methods

The majority of the subsetting approaches [11], [15], [16], [17] rely on the principle of near-optimality. The authors observed that many cache configurations in the design space provide approximately similar values of the optimization objective function. Therefore, at the expense of a small penalty to the optimization objective, the design space can be reduced from C to C'—a smaller subset of the original design space (i.e., $|C'| \ll |C|$). While C' is not guaranteed to contain a globally optimal configuration, the authors showed that C' contains many near-optimal configurations with the benefit of a much smaller design space to explore in C' as compared to C. In order to compute C', the authors used a clustering approach that was originally applied to online segmentation of time series data [26]. Some subsetting methods [10], [14], [21] were geared toward online DSE and are discussed in Section IV. We refer the reader to [17], where these approaches were recently reviewed, for further details.

IV. Online Design Space Exploration (DSE)

For embedded systems that are expected to run applications that are unknown a priori, the cache configuration design space must be explored online at runtime. This is typically the case for consumer embedded devices [6], such as smartphones and

tablets. Online exploration in its most basic form involves the execution of a given application on the target architecture using different cache configurations in order to capture the values of the optimization objective function under each configuration. DSE is followed by selecting the best configuration that most closely meets optimization requirements and constraints from the configurations that were explored. Collectively, these two steps are referred to as cache tuning [23]. This section briefly overviews several online cache tuning methods.

A. Heuristic-based Methods

Heuristic methods selectively explore/search only a portion of the design space by using information about the current and past explored configurations to choose the next configuration to explore. Zhang et al. [21] developed a greedy heuristic search to determine the lowest energy configuration (i.e., point A in Fig. 1). The parameters were explored in increasing order of the parameters' impacts on the energy consumption for single-level caches. Gordon-Ross et al. [10], [14] extended this approach to two-level caches and demonstrated that this heuristic search explored only a small fraction of the original design space without significantly compromising energy consumption. Alsafrjalani et al. [6] proposed an online QoE-aware approach [11] for multi-core cache configurations using a heuristic-based search to perform online DSE that minimized DSE-time degradation. Zhao et al. [27] analyzed the behavior of a large pool of applications offline and used the learned information to heuristically select a best cache configuration online.

B. Phase-based Methods

During execution, an application's characteristics, such as CMR, IPC, branch misprediction rate, etc., change over the application's different execution phases. Prior work demonstrated that the best cache configurations differ even within the same application according to these application phases [4], [7]. Therefore, approaches have emerged that take phase behavior into account during online cache tuning [13], [28], [7].

Peng et al. [13] proposed multiple state machines that monitor the program execution to make cache reconfiguration decisions while attempting to avoid unnecessary reconfigurations. Hajimir et al. [28] proposed an intra-task dynamic cache reconfiguration method that used a dynamic programming-based algorithm to determine the best cache configuration, while reducing DSE overhead. However, these methods still required DSE whenever the cache configuration needed to be changed. More recently, Adegbija et al. [7] proposed a phase distance mapping method that eliminated the need for DSE by directly estimating the best configuration for an application phase given the phase's performance characteristics. The proposed technique employed correlations between a known phase's best configuration and characteristics, and a new phases' characteristics.

C. Analytical/Predictive Modeling Techniques

Since online DSE can often execute an application in inferior sub-optimal cache configurations that result in severe performance and energy consumption degradation, reducing or eliminating DSE is an attractive solution. Phase distance mapping, is one such solution and is considered as a predictive modeling approach. Another predictive modeling approach was presented in [29] where the authors constructed IPC curves for different sets of applications that allowed the application performance to be predicted for all level two cache way values by observing the application's performance for only one level two cache way value. Such cross-configuration prediction of application performance can significantly speed up online DSE by examining different configurations simultaneously while executing a single configuration. Prior works have shown that analytical models that rely on the application execution trace can be successfully used [12], [22].

V. Online Cache Tuning and Control

The ultimate goal of a configurable cache system is to realize self-tuning that is transparent to the end user and the running applications [5], [13]. To achieve this goal, this self-tuning cache system can be formulated as a feedback control system that self-monitors performance and adjusts the cache parameters in order to meet runtime-defined time-varying user-defined QoE expectations. For the best closed-loop control performance, the application phases' characteristics, the phases' cache requirements, and the tuning interval must be taken into consideration.

There has been some prior work related to these goals. Peng et al. [13] developed multiple state machines to monitor application execution, detect performance changes that accompany phase changes (i.e., tuning interval), and tune the cache accordingly. However, determining the tuning interval is extremely challenging. Gordon-Ross et al. [5] showed that the tuning interval must never overshoot a change in application phase in order to avoid severe energy penalties that result from executing an inappropriate cache configuration through an undetected application phase change. When overshoot happens, the new application phase operates using the cache configuration that was determined to be the best configuration for a previous phase but may no longer be the best configuration for the new phase, imposing *tremendous* energy penalties.

A self-tuning cache system must provide the following functionalities: 1) be capable of autonomously monitoring and collecting information about the optimization metrics; 2) be able to dynamically reconfigure the tunable parameter values; 3) be capable of determining when and how to change the cache parameters in response to varying application phase requirements without introducing severe penalties [5], [13]; and 4) adapt to user-defined QoE expectations; and limit user experience degradation during DSE.

These requirements echo the notion that a self-tuning cache system is a variation of a feedback control system, an example of which was proposed in [5] and is reproduced in Fig. 2. The purpose of that control system was to dynamically adjust the tuning interval such that the interval matched as closely

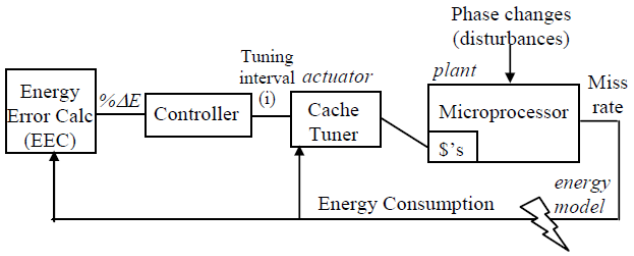

Fig. 2. Proposed feedback control system architecture for a self-tuning cache [5].

as possible to the application's phase changes without overshooting. The CMR was used as the performance optimization metric to determine when cache reconfiguration needed to occur, and an energy model was implemented as an online energy calculator to provide the energy optimization metric that guided the closed-loop operation of the control system. In reference to the general problem formulation in Section II, we also note the correspondence between these two performance and energy optimization metrics, which we denoted as $u(.)$ and $e(.)$, respectively.

Adegbija et al. [7] exploited application phase information online to directly estimate the best cache configuration using a statistical model that correlated known and new phase characteristics to the best known cache configuration for a base phase. The advantage of this approach was that the model essentially eliminated DSE by inferring a best cache configuration. This approach can be classified as a statistical machine learning-based approach where a large dataset of known application phases and associated best cache configurations are used offline and comprises the training dataset. A statistical model is learned from this training dataset, and subsequently used online to make predictions about new data [30]. Since it is possible to summarize the statistics of a huge training dataset using a lightweight statistical model that is later used online, we argue that machine learning-based approaches represent a key component in online cache tuning and control. However, since these prediction-based methods are error-prone, online adaptation and control must be used in order to correct for the modeling-related errors and to update the statistical models at runtime. Therefore, in our proposed approach, we use online adaptation in the statistical model as presented in the next subsection.

A. Proposed Architecture

Fig. 3 depicts our proposed architecture for closed-loop, online cache tuning and control. Similar to a closed-loop control system, our proposed architecture comprises a controller, which is the cache tuner module, and a plant that is to be controlled, which is the microprocessor's cache subsystem.

The cache tuner module determines the best cache configuration for the executing application phase. This module takes into account the dynamic user QoE expectations as part of the optimization objective function. The cache tuner module also

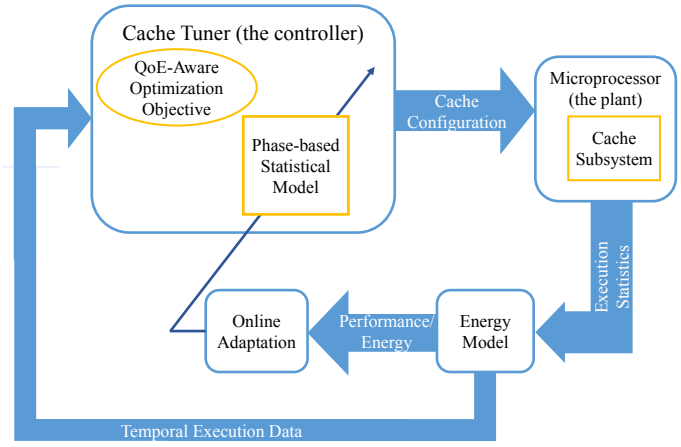

Fig. 3. Our proposed architecture for closed-loop, online cache tuning and control.

uses an adaptive phase-based statistical model to predict the best cache configuration that satisfies the current optimization objective. This statistical model assists DSE by simultaneously predicting the performance and energy metrics' values for multiple cache configurations or even by directly predicting the best cache configuration based on execution statistics.

We propose that the statistical model incorporates linear/non-linear adaptive filters—a class of statistical models that convolve a time series with filter weights to perform online prediction—and can adapt the weights of this model using an online adaptation method [31]. While prior work used instant-by-instant phase information for statistical prediction [7], we propose to consider the temporal history of the execution characteristics at a fine-grained temporal resolution for statistical prediction. In other words, we consider the application execution statistics as well as the performance and energy metrics as time series data, and collectively refer to this data as temporal execution data. This machine learning-based approach is poised to simultaneously improve the prediction performance, and to operate at a higher temporal resolution while providing an end-to-end prediction model that does not explicitly require classification of the executing application phase. Our proposed architecture makes use of three major elements to improve upon previous approaches: 1) considers the temporal history of the application's execution characteristics and the application's phase information using online filtering; 2) monitors the energy and performance of the running applications' phases and uses this information to update and adapt the online filter; and 3) adapts to runtime-defined time-varying user-defined QoE expectations.

B. Unexplored Challenges

One of the major challenges for online cache tuners is to avoid user experience degradation due to DSE overheads, such as power/energy and performance [7], [17]. To regulate these overheads, we combine our proposed adaptive statistical model with design space subsetting and consider the temporal execution data in order to improve the prediction performance while

minimizing the computational overhead. Another challenge is that online cache tuners are expected to perform generally well while executing a large variety of applications from vastly different application domains. In the most challenging situation, none of this information is available during design time, thus no efforts can be made to enable more accurate runtime predictions based on the particular application or domain characteristics (e.g., data streaming applications typically have high CMRs). In our proposed architecture, the online adaptation module is responsible for changing the weights of the filters in the statistical model based on ground-truth data observed in real-time as different application phases execute. By adapting the embedded statistical model online, our proposed architecture can continue to improve the prediction accuracy whenever degraded performance for some application/phase is observed.

Predictions can also be improved by tracking best configurations and correlating those configurations to device sensor readings. For example, ambient lighting conditions could be used to regulate video frame rate, thereby reducing the frame rate in low lighting. Predictions can also be done using time-of-day correlations. Users likely use their devices differently during working hours, thus QoE expectations may be lower in general than during non-working hours. For any type of correlation, historical data can be kept that enables direct lookups of best configurations given different sensor value combinations to eliminate DSE for known scenarios. This historical data could also be mined to determine user patterns and characteristics that can enable best configurations to be predicted for unknown scenarios. Whereas this challenge is not a make-or-break point for self-tuning systems, an effective solution would greatly improve QoE.

A challenge that all online cache tuners must address is the introduced power/energy and performance overheads during DSE, and general area overheads due to the architectural structures required to enable and orchestrate cache tuning. To have the most accurate tuning interval estimation, the tuner should frequently evaluate execution characteristics to avoid severe penalties for missed phase changes. To address this challenge, we construct our architecture to operate at two different intervals. Since the statistical model uses an online filter, this model can operate at a relatively high sampling frequency such that the model is almost guaranteed to never miss a phase change. The filter implementation can use a recursive formulation such that the filter can cover a long temporal span at a high sampling resolution while using few filter weights. The reduced number of filter weights results in significantly fewer computations, and, therefore, power and area savings for the on-chip hardware filter implementation. To further reduce the power overhead, the controller can limit cache reconfigurations to cases where the expected optimization metric gains are beyond a predefined threshold. In other words, while the online statistical filter runs at a high sampling resolution, the cache reconfiguration decisions occur at a much lower rate that is based on statistical thresholding.

Another challenge is incorporating user feedback to identify what the user considers a best configuration, and to ascertain how much DSE-related degradation is tolerable. Identifying the user-defined best configuration can be done using existing operational settings that allow the user to enter energy-savings or performance-enhanced modes. However, this does not provide the fine level of granularity necessary to determine Pareto-optimal best configurations for diverse operating environments and applications. An intrusive approach would be to periodically poll the user to swipe left or right based on their level of satisfaction. However, given the granularity of feedback needed to gather information during DSE and that the system may only execute for a fraction of a second or several seconds in each configuration, this method is infeasible. A non-intrusive approach would be to use the device's internal sensors to infer user satisfaction, such as using the accelerometer to detect user frustration that physically agitates their usage of the device. However, this method does not provide detailed enough feedback and would not work for all users. An ideal situation will likely combine both intrusive and non-intrusive techniques. Finding a solution to this challenge is critical to realizing self-tuning systems.

VI. Conclusions

Efficient and effective dynamic configurable cache tuning is critical for modern embedded systems that are required to operate in unknown environments, and execute a myriad of applications that are typically unknown at design time while dynamically responding to user quality of experience (QoE) expectations. Realizing a self-tuning cache that is transparent to both the user and the executing applications is hindered by many challenges. Critical challenges include limiting the user's experience degradation during design space exploration (DSE), adapting to unknown application requirements, minimizing the area and power/energy overheads introduced by the self-tuning circuitry, and determining how to incorporate user feedback and device sensor readings. In addition, a self-tuning cache subsystem should ideally adapt at a fine-grained, application phase-based level.

In this paper, we reviewed current progress in cache tuning and provided an overview of our proposed online, adaptive, closed-loop control system architecture for dynamic cache tuning. Our proposed architecture attempts to address some of the challenges by: 1) utilizing application phase information and temporal execution data at a high sampling frequency in order to accurately predict the best cache configuration; 2) implementing the prediction model using computationally efficient statistical filters; 3) adapting the prediction model online to account for changing application execution statistics; 4) using a QoE-aware optimization objective function; and 5) operating at two different sampling frequencies in order to minimize missed application phase changes while simultaneously saving dynamic power/energy consumption and architecture area. Finally, we identified numerous unexplored challenges, many of which are critical to realizing self-tuning systems. Our future work involves implementing the proposed

architecture in order to quantify and evaluate our architecture and solutions' effectiveness for dynamic cache tuning.

ACKNOWLEDGMENTS

This work was supported by the National Science Foundation (CNS-0953447 and CNS-1718033). 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] A. Malik, B. Moyer, and D. Cermak, "A low power unified cache architecture providing power and performance flexibility (poster session)," in *Proceedings of the 2000 international symposium on Low power electronics and design*. ACM, 2000, pp. 241–243.

[2] S. Mittal, "A survey of architectural techniques for improving cache power efficiency," *Sustainable Computing: Informatics and Systems*, vol. 4, no. 1, pp. 33–43, 2014.

[3] C. Zhang, F. Vahid, and W. Najjar, "A highly configurable cache for low energy embedded systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 4, no. 2, pp. 363–387, 2005.

[4] T. Sherwood, E. Perelman, G. Hamerly, S. Sair, and B. Calder, "Discovering and exploiting program phases," *IEEE micro*, vol. 23, no. 6, pp. 84–93, 2003.

[5] A. Gordon-Ross and F. Vahid, "A self-tuning configurable cache," in *Proceedings of the 44th annual Design Automation Conference*. ACM, 2007, pp. 234–237.

[6] M. H. Alsafrjalani and A. Gordon-Ross, "Quality of service-aware, scalable cache tuning algorithm in consumer-based embedded devices," in *Proceedings of the 26th edition on Great Lakes Symposium on VLSI*. ACM, 2016, pp. 357–360.

[7] T. Adegbija, A. Gordon-Ross, and A. Munir, "Phase distance mapping: a phase-based cache tuning methodology for embedded systems," *Design Automation for Embedded Systems*, vol. 18, no. 3-4, pp. 251–278, 2014.

[8] D. E. Kirk, *Optimal Control Theory: An Introduction*. Courier Corporation, 2004.

[9] C. Zhang and F. Vahid, "Cache configuration exploration on prototyping platforms," in *Rapid Systems Prototyping, 2003. Proceedings. 14th IEEE International Workshop on*. IEEE, 2003, pp. 164–170.

[10] A. Gordon-Ross, F. Vahid, and N. Dutt, "Fast configurable-cache tuning with a unified second-level cache," in *Low Power Electronics and Design, 2005. ISLPED'05. Proceedings of the 2005 International Symposium on*. IEEE, 2005, pp. 323–326.

[11] P. Viana, A. Gordon-Ross, E. Keogh, E. Barros, and F. Vahid, "Configurable cache subsetting for fast cache tuning," in *Proceedings of the 43rd annual Design Automation Conference*. ACM, 2006, pp. 695–700.

[12] A. Gordon-Ross, P. Viana, F. Vahid, W. Najjar, and E. Barros, "A one-shot configurable-cache tuner for improved energy and performance," in *Proceedings of the conference on Design, automation and test in Europe*. EDA Consortium, 2007, pp. 755–760.

[13] M. Peng, J. Sun, and Y. Wang, "A phase-based self-tuning algorithm for reconfigurable cache," in *Digital Society, 2007. ICDS'07. First International Conference on the*. IEEE, 2007, pp. 27–27.

[14] A. Gordon-Ross, F. Vahid, and N. D. Dutt, "Fast configurable-cache tuning with a unified second-level cache," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 1, pp. 80–91, 2009.

[15] M. H. Alsafrjalani and A. G. Ross, "Dynamic scheduling for reduced energy in configuration-subsetted heterogeneous multicore systems," in *Embedded and Ubiquitous Computing (EUC), 2014 12th IEEE International Conference on*. IEEE, 2014, pp. 17–24.

[16] M. H. Alsafrjalani, A. G. Ross, and P. Viana, "Minimum effort design space subsetting for configurable caches," in *Embedded and Ubiquitous Computing (EUC), 2014 12th IEEE International Conference on*. IEEE, 2014, pp. 65–72.

[17] M. H. Alsafrjalani and A. Gordon-Ross, "Low effort design space exploration methodology for configurable caches," *Computers*, vol. 7, no. 2, p. 21, 2018.

[18] T. Givargis and F. Vahid, "Platune: a tuning framework for system-on-a-chip platforms," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 11, pp. 1317–1327, 2002.

[19] T. Adegbija and A. Gordon-Ross, "PhLock: A cache energy saving technique using phase-based cache locking," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 1, pp. 110–121, 2018.

[20] G. Reinman and N. P. Jouppi, "CACTI 2.0: An integrated cache timing and power model," *Western Research Lab Research Report*, vol. 7, 2000.

[21] C. Zhang, F. Vahid, and R. Lysecky, "A self-tuning cache architecture for embedded systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 3, no. 2, pp. 407–425, 2004.

[22] A. Ghosh and T. Givargis, "Cache optimization for embedded processor cores: An analytical approach," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 9, no. 4, pp. 419–440, 2004.

[23] W. Zang and A. Gordon-Ross, "A survey on cache tuning from a power/energy perspective," *ACM Computing Surveys (CSUR)*, vol. 45, no. 3, p. 32, 2013.

[24] D. Burger and T. M. Austin, "The simplescalar tool set, version 2.0," *ACM SIGARCH computer architecture news*, vol. 25, no. 3, pp. 13–25, 1997.

[25] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.

[26] E. Keogh, S. Chu, D. Hart, and M. Pazzani, "An online algorithm for segmenting time series," in *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*. IEEE, 2001, pp. 289–296.

[27] H. Zhao, X. Luo, C. Zhu, T. Watanabe, and T. Zhu, "Behavior-aware cache hierarchy optimization for low-power multi-core embedded systems," *Modern Physics Letters B*, vol. 31, no. 19-21, p. 1740067, 2017.

[28] H. Hajimiri and P. Mishra, "Intra-task dynamic cache reconfiguration," in *VLSI Design (VLSID), 2012 25th International Conference on*. IEEE, 2012, pp. 430–435.

[29] M. Moreto, F. J. Cazorla, A. Ramirez, and M. Valero, "Online prediction of applications cache utility," in *Embedded Computer Systems: Architectures, Modeling and Simulation, 2007. IC-SAMOS 2007. International Conference on*. IEEE, 2007, pp. 169–177.

[30] M. B. Christopher, *Pattern recognition and machine learning*. Springer-Verlag New York, 2016.

[31] S. Haykin, *Adaptive filter theory*. Prentice-Hall, Inc., 1986.