A Self-Tuning Configurable Cache

Ann Gordon-Ross and Frank Vahid*
University of California, Riverside – Department of Computer Science and Engineering
http://www.cs.ucr.edu/{~ann}{~vahid}, {ann,vahid}@cs.ucr.edu
**Also with the Center for Embedded Computing Systems at University of California, Irvine*

ABSTRACT
The memory hierarchy of a system can consume up to 50% of microprocessor system power. Previous work has shown that tuning a configurable cache to a particular application can reduce memory subsystem energy by 62% on average. We introduce a self-tuning cache that performs transparent runtime cache tuning, thus relieving the application designer and/or compiler from pre-determining an application's cache configuration. The self-tuning cache applies tuning at a determined tuning interval. A good interval balances tuning process energy overhead against the energy overhead of running in a sub-optimal cache configuration, which we show wastes much energy. We present a self-tuning cache that dynamically varies the tuning interval, resulting in average energy reduction of as much as 29%, and within 11% of the energy savings of an optimal self-tuner tuning at ½ of the phase interval and within 13% of the oracle.

Categories and Subject Descriptors
B.3 **[Memory Structures]**: Performance Analysis and Design Aids.

General Terms
Algorithms and design.

Keywords
Configurable cache, reconfigurable cache, phase-based, cache tuning, reconfigurable architecture, low energy, low power, architecture tuning, embedded systems.

1. INTRODUCTION
Configurable caches allow for cache parameters, such as total size, associativity, and line size, to be varied. Configurable caches enable architectural specialization to a particular application, for improved power, energy, and/or performance.

Cache tuning is the process of determining the best cache parameter values for a given application. For example, an application with a large working set benefits from a large cache; a small cache yields excess energy due to thrashing (the working set being swapped in and out). Conversely, an application with a small working set benefits from a small cache; a large cache would waste energy due to high energy cost per fetch and unnecessary static power. Similarly, specific spatial and temporal localities suggest best line size and associativity settings.

Cache requirements vary greatly across applications [14]. Tuning a configurable cache to a particular application can reduce average memory access energy by 62%, along with performance improvements in most cases [5]. Cache requirements can even vary within an application [11], and tuning the cache to phases can yield additional improvement.

Software-reconfigurable caches [1][2][14] enable cache parameter values to be adjusted dynamically. One dynamic approach requires that cache configurations have been pre-determined statically, with reconfiguration taking place by adding instructions to an application that updates the cache configuration register during phase changes. Another dynamic approach is fully transparent, in which the cache itself automatically adjusts its parameters to the executing application, i.e., the cache is *self-tuning*.

Self-tuning enables the energy savings of cache tuning without any designer effort, simplifying the design process, maintaining standard tool flows, and keeping binaries portable; a self-tuning cache can be incorporated transparently into any cache-based architecture. Furthermore, a self-tuning cache can tune to runtime changes in environment, such as changes to input patterns or software updates.

One self-tuning cache design dynamically invokes a self-tuning mode, at a specified *tuning interval*, that efficiently explores the configuration space, or examines special counters, to predict the best cache configuration for the currently-running application or phase [2][13][15]. A challenge in self-tuning is determining a good tuning interval. If phase changes are periodic, then for maximum savings, the tuning interval should closely match the ***phase interval***, or the length of time a system executes between phase changes. A ***phase change*** is any change in the executing application such that a different cache configuration would be better than the previous one. Because the tuning process consumes extra energy, if the tuning interval is too short, excess tuning energy is expended. If the tuning interval is too long, energy is wasted running in sub-optimal configurations.

The best tuning interval is highly dependent on runtime factors and thus is hard to pre-determine. Previous methods use feedback control to determine *if* tuning should occur at the fixed tuning interval, but they do not analyze the chosen interval nor attempt to *adjust* the tuning interval. We show that significant energy is wasted if the tuning interval does not closely match the changing phases of the system.

We present an in-depth study of a dynamically-adjusting tuning interval. Whereas most previous methods use fixed tuning intervals, we introduce an effective online algorithm to adjust the tuning interval to match the phase interval of a system. We design a feedback controlled self-tuner that is system independent, which examines execution and determines the tuning interval regardless of the number of applications and/or presence of an operating system. Our methodology is widely applicable to embedded, desktop, and even super-computing environments. Furthermore, this methodology can easily be incorporated into many existing periodic dynamic tuning methodologies for improved results.

DAC 2007, June 4–8, 2007, San Diego, California, USA

2. ONLINE ALGORITHMS

We use the framework of online algorithms, which process data as data arrives and are unable to view the entire data set, for our approach. A self-tuner tunes the tuning interval based on current and past, but not future, system behavior.

We design a system to monitor the effects of the current tuning interval to evaluate the tuning interval. Our methodology is similar to a feedback control system [9]. A basic control system consists of a plant (system under control), a goal (set-point), and a method to manipulate the plant (actuator). Changes made via the actuator modify the plant and thus modify the output of the system to meet the set-point. A feedback control system examines the output of the system and determines the error in relation to the set-point, thus allowing the actuator to adjust based on observed error. We view self-tuning as a feedback control system by considering energy change as the observed output of the system, and treating the cache tuner as the actuator used to reduce the energy consumption of the memory subsystem plant. However, set-points are typically fixed values, but in the self-tuner, the set-point is the minimization of energy consumption, which makes modeling as a control system difficult. In the next section, we adapt feedback control system theory to our methodology.

3. SELF-TUNING METHODOLOGY

We describe the design of a feedback control system capable of monitoring the tuning interval and transparently optimizing the self-tuner for reduced energy consumption.

3.1 Motivating example

A strictly periodic tuning approach blindly tunes at a fixed interval, which may be either too long or too short. Event-driven techniques poll event counters at some interval, comparing counters (such as a cache miss rate counter) to thresholds, to determine if tuning should occur. However, even if a stable system is observed (e.g., cache miss rate is stable) and thus tuning skipped, a phase change may occur between polling intervals, meaning the interval is too long.

Figure 1 illustrates the impact of a tuning interval being too long or too short, assuming a fixed periodic phase interval of 10 million cycles, where at each phase change, the system chooses a random benchmark from a given benchmark suite (further details of the system setup are described in section 4.1). The figure shows energy consumption of a self-tuning instruction cache subsystem normalized to a system running with a fixed base cache. The self-tuning energy consists of the energy consumed during tuning plus the energy consumed by the system running with the configuration determined during tuning as it executes for the remainder of the tuning interval. The optimal system would be an oracle that immediately recognizes phase changes and runs the tuner.

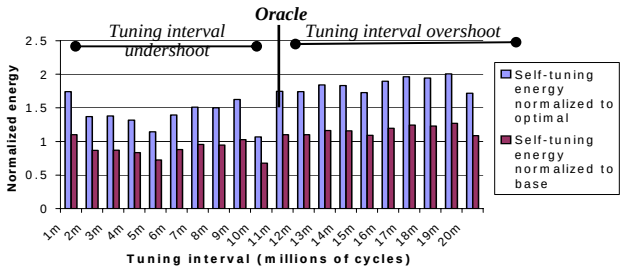

Figure 1: Self-tuning energy normalized to base system for too-long and too-short tuning intervals for a fixed phase interval of 10 million cycles.

A tuning interval that matches the phase interval (in this case, 10 million cycles) yields optimal results. However, without omniscient information during runtime, it impossible to accurately predict a phase change 100% of the time (Sherwood [12] presents an efficient and effective method to predict phase changes, but due to the difficulty of predicting the future, it is not perfect). With the optimal tuning interval, the self-tuning system reduces 32% of energy over a base cache system, which includes 7% overhead due to the tuning process itself. However, if the phase interval is not tracked precisely, the penalty is severe and there is no potential energy savings compared to the base system.

Too short (undershoot) or too long (overshoot) of fixed tuning intervals reduces energy savings. Overshoot means a phase change is missed and the new phase is executed using the current cache configuration. We observed that overshoot results in a tremendous energy penalty. A cache tuned to a particular phase is best for that phase but is likely very poor for another phase. We refer to these configurations as extremist configurations. Overshoot results in the system consuming *more* energy than a base cache system, meaning it is better to not perform self-tuning than have overshoot. Overshoot energy details appear in [6].

In contrast to overshoot, undershoot yields energy savings compared to a base cache. The best results are at a tuning interval of 5 million cycles, which is half the phase interval, yielding 28% energy savings over a base system. Furthermore, if ½ of the phase interval is not tracked precisely, we see that the penalty is acceptable and the system can still save energy compared to the base system.

Thus, interval tuning is important, and since it is better to undershoot the tuning interval than overshoot the interval, we sought to develop a methodology for tracking ½ of the phase interval.

3.2 Feedback control system

Figure 2 shows our feedback controlled self-tuner, modeled after a control system. The goal of our system is to control the tuning interval to minimize over all energy consumption.

Since our system has no set-point, but rather seeks to minimize energy, our system is similar but not exactly a feedback control system. The plant is a microprocessor. Changes to the plant occur as side effects of changes made to the cache system by the actuator, or cache tuner. Plant changes reflect in the system miss rate, after application of the energy model (discussed in section 4.1), serves as the feedback to the system. Phase changes represent disturbances to the plant causing changes in the miss rate.

After a tuning process, the energy error is calculated by the Energy Error Calculator (EEC). A change in energy (ΔE) before tuning compared to after tuning signifies a phase change. If the energy before tuning is the same as the energy after tuning, we assume no phase change occurred during the last interval. The

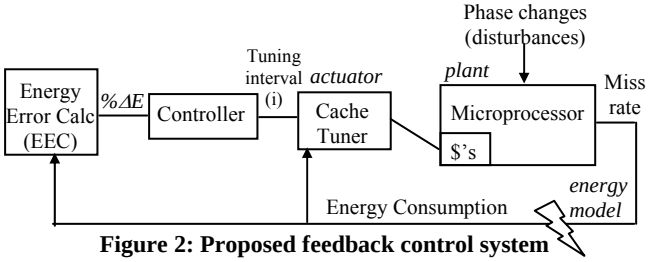

Figure 2: Proposed feedback control system

EEC outputs the percent change in energy (%ΔE). To prevent erratic behavior due to artifacts in disturbances, the %ΔE is calculated over a window of past tuning intervals. We determine the best window size (***W***) through experimentation in section 4.2.

The controller uses %ΔE to vary the tuning interval. If %ΔE is small, then the system does not benefit from tuning, thus the system should be tuned less often. Conversely, if %ΔE is large, then the system does benefit from tuning, thus the system should be tuned more often.

3.3 Controller logic

As earlier mentioned, overshoot should be avoided. If the phase change is overshot, the tuning interval should back off rapidly to avoid excess energy wasted in a suboptimal cache configuration. Thus, the tuning interval should increase slowly but decrease quickly. This is similar to an attack/decay online algorithm [9].

We initially developed a fixed equation to control the interval length using a fixed percentage to increase/decrease the tuning interval. Through experimentation, we observed that fixed increments/decrements of the tuning interval did not offer enough granularity of change for the tuning interval, and we were unable to stabilize the system.

To stabilize the tuning interval, we developed a variable, two-part equation similar to a fuzzy logic system, in which changes neither need to be absolute nor do changes need to use fixed value logic. With fuzzy logic, the tuning interval can be changed based on how close or far away the system is from being stable as is observed by the %ΔE of the system.

Figure 3 graphically depicts the equations to calculate the tuning interval. The x-axis plots %ΔE and the y-axis is the percent change applied to the tuning interval based on %ΔE. Variables are denoted in bold italics. The graph assumes a Point of Stability (***PoS***), which is the %ΔE we would expect to see in a stable system with good a tuning interval, and thus there should be no change to the tuning interval (y-axis point is 1). For %ΔE values less that the ***PoS***, the system tunes too frequently. At the extreme where %ΔE = 0 (no change in energy after tuning), the tuning interval should be increased by the maximum amount, which we call ***U***. For 0 <= %ΔE < ***PoS***, the change in tuning interval ***U*** is calculated using the equation passing through points (***PoS***, 1) and (0, ***U***).

For %ΔE > ***PoS***, we assume a maximum %ΔE of 100%. In reality, %ΔE could be greater than 100% but we assume that for values greater than 100% we are only interested in decreasing the tuning interval by the maximum amount ***D***. For ***PoS*** < %ΔE < 100%, the percentage change ***D*** applied to the tuning interval is calculated using the equation passing through points (100%, ***D***) and (***PoS***, 1).

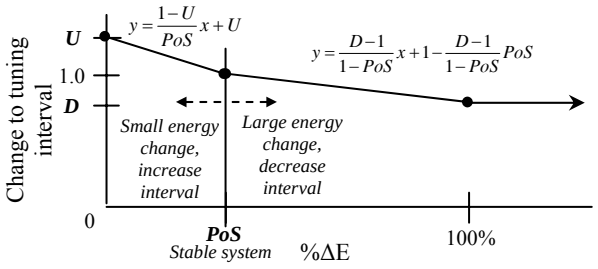

Figure 3: Two part equation to map change in energy (x-axis) to a change in tuning interval (y-axis).

We determine appropriate values for ***U, D,*** and ***PoS*** in Section 4.2 through experimentation.

4. RESULTS

4.1 Experimental setup

We use the highly configurable cache and tuning heuristic developed by Zhang [14]. We configure the level one instruction cache only. Possible cache configurations include cache sizes of 2, 4, and 8 KBytes, line sizes of 16, 32, and 64 bytes, and direct-mapped, 2-way, and 4-way set associativities. We compare to a base cache configuration of an 8KB, 4-way set associative cache with 32-byte line size [14], which performs reasonably across all the examined benchmarks.

We use 20 benchmarks from the Powerstone [10] and MediaBench [8] benchmark suites. We execute each benchmark on SimpleScalar for every cache configuration to gather hit and miss rates. We apply the tuning heuristic and the energy model in [14] to determine the best cache configuration per application. As tuning energy is application-dependent, we calculate tuning energy per application. We estimate the tuning time as 1 million cycles and use actual cycles-per-instruction values for each application.

We model the overall system in C++ and, without loss of generality, switch applications to simulate phase changes. For each experiment, we simulate 10,000 phase changes.

4.2 Tuning interval function evaluation

We first determined the best constant values for the variables in Figure 3 and the window size from Section 3.2 using a periodic system. We assumed a training set of 2 applications switching at a fixed phase interval. We varied one parameter while holding other parameters fixed to determine the best value for that parameter. The best value now becomes the fixed value for that parameter, and we varied the next parameter. Using knowledge of cache and application behavior and observations gathered during initial testing, we chose the fixed values conservatively (small) to eliminate erratic behavior. We note that the best values are highly dependent on the actual system, the applications running, and the phase switching frequency, however we will show in section 4.3 that the values determined with our training system work well for a variety of different test systems.

Graphs depicting fluctuation in energy consumption can be found in [6]. For each variable, we chose the value that resulted in greatest energy savings. We varied ***PoS***, followed by ***U***, ***D***, and finally ***W***. The final values chosen were: ***PoS*** = 0.17, ***U*** = 1.11, ***D*** = 0.82, and ***W*** = 2. We observe that these values cause the tuning interval to converge to ½ of the phase interval quickly with little oscillation after convergence [6].

To determine values for a more diverse system, we applied the same technique to a system with a fixed phase interval, but where a random benchmark was chosen from our set of 20 benchmarks at each phase change. The final values chosen for the system were: ***PoS*** = 0.16, ***U*** = 1.12, ***D*** = 0.85, and ***W*** = 5.

4.3 Energy savings

To evaluate the feedback controlled self-tuner and the values determined for ***PoS***, ***U***, ***D***, and ***W*** for test systems, we first looked at two-phase systems switching at a fixed phase interval for random pairs of benchmarks. Figure 4 shows energy consumption normalized to the base cache system for the optimal oracle self-

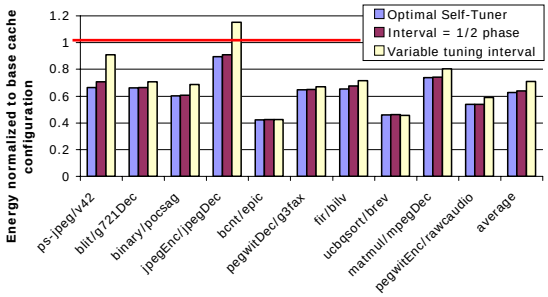

Figure 4: Energy consumption normalized to the base system for a 2-phase system switching at a fixed period of 62.5 ms.

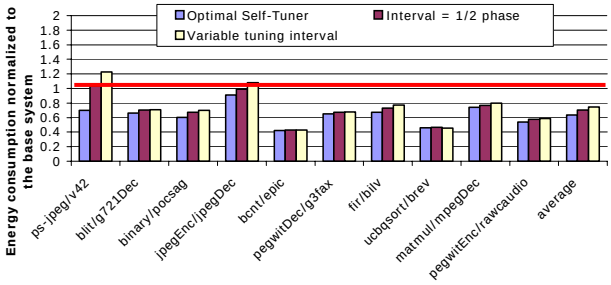

Figure 5: Energy consumption normalized to the base system for a 2-phase with varying phase intervals

tuner (tunes precisely at the phase change), a self-tuner that tunes at exactly ½ the phase interval, and our self-tuner with a variable tuning interval. A y value of 1 represents the energy consumption for the base system, and all bars below that line denote energy savings. If we compared our variable interval self-tuner with an actual system, our method would outperform most fixed tuning intervals. However, we compare to the best possible tuning intervals in order to show how close our methodology comes to obtaining optimal results.

Assuming a 400 MHz system, Figure 4 shows a system with a phase interval of 62.5 ms ([6] shows results for different systems). The self-tuner with variable tuning interval comes within 13% of the optimal self-tuner, and reduces energy by 29% compared to a base system.

Figure 5 evaluates a less periodic 2-phase system. For this system, the base phase interval is 62.5 ms and at each phase change, there is a 50% chance of the next phase interval being increased or decreased by 50%. Again, the self-tuner with a variable tuning interval performs very well, coming within 17% of the optimal self-tuner, within only 6% of the self-tuner at a fixed rate of ½ the phase interval, and with 26% energy savings compared to the base system.

4.4 Initial tuning intervals, untunable systems

Choosing an appropriate initial value for the tuning interval is important. We determined through experiments that if a very large initial tuning interval was chosen relative to the phase interval, aliasing could lead to interval values converging at rates greater than the phase interval. For example, in a two phase periodic system, if the tuning interval is so long that is skips over the execution of one of the phases, the energy before and after tuning will be the same because the system missed the phase change that occurred in between tuning times. We observed no problem starting with a small interval. Thus, it is better to start the tuning interval at a very small value.

We assumed that the phases did not switch faster than the tuning time, but a real system may violate that assumption. If an application switches faster than the system can tune for, the system will always be tuning and hence incur energy overhead. Thus, a minimum interval value should be specified. If the tuning interval remains at the minimum value for too long, then the system should be deemed untunable and the cache system set to a reasonable base configuration. To react to future system changes, the self-tuner could be re-invoked periodically to see if the system has changed to a tunable state.

5. CONCLUSIONS

We presented a novel dynamic self-tuning cache for reduced energy. Previous methods use a fixed tuning interval that we observed could have significant energy overhead. We developed a feedback control system for adjusting the tuning interval to match system needs. The self-tuner reduces average energy consumption by 29%, and comes within 11% of the energy savings of the best possible non-oracle tuner that tunes at half of the phase interval, and within 13% of the oracle. Future work includes improving our self-tuner to be more robust in highly diverse environments.

6. ACKNOWLEDGEMENTS

This work was supported in part by the National Science Foundation (CNS-0614957) and the Semiconductor Research Corporation (2005-HJ-1331).

7. REFERENCES

[1] D. Albonesi. Selective cache ways: on-demand cache resource allocation. MICRO 1999

[2] R. Balasubramonian, D. Albonesi, A. Byuktosunoglu, S. Dwarkada. Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures. MICRO 2000.

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

[4] T. Givargis, F. Vahid. Platune: a tuning framework for system-on-a-chip platforms. IEEE Trans. on Computer Aided Design, Nov. 2002.

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

[6] A. Gordon-Ross, F. Vahid. A self-tuning configurable cache. University of California, Riverside. Technical Report UCR-CS-2007-03001.

[7] S. Kaxiras, Z. Hu, M. Martonosi. Cache decay: exploiting generational behavior to reduce cache leakage power. ICCD July 2001

[8] C. Lee, M. Potkonjak, W.H. Mangione-Smith. MediaBench: A tool for evaluating and synthesizing multimedia and communication systems. MICRO 1997.

[9] G. Magklis, G. Semeraro, D.H. Albonesi, S.G. Dropsho, S. Dwarkadas, M.L. Scott. Dynamic frequency and voltage scaling for a multiple-clock domain microprocessor. IEEE Micro, Nov-Dec 2003.

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

[11] T. Sherwood, E. Perelman, G. Hamerly, S. Sair, B. Calder. Discovering and exploiting program phases. IEEE Micro: Micro's Top Picks from Computer Architecture Conferences, December 2003.

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

[13] S. Yang, M. Powell, B. Falsafi, K. Roy, T. Vijaykumar. An integrated circuit/architecture approach to reducing leakage in deep-submicron high-performance i-caches. HPCA, January 2001.

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

[15] H. Zhou, M.C. Toburen, E. Rotenberg, T.M. Conte. Adaptive mode control: a static power efficient cache design. PACT, September 2001.