Phase-based Cache Reconfiguration For a Highly-Configurable Two-Level Cache Hierarchy

Ann Gordon-Ross
University of Florida
Department of Electrical and Computer Engineering
ann@ece.ufl.edu, http://www.ann.ece.ufl.edu/
Also with the NSF Center for High-Performance Reconfigurable Computing at the University of Florida

Jeremy Lau[1]
Google Inc.
lauj@google.com

Brad Calder[1]
Microsoft Corporation

ABSTRACT
Phase-based tuning methodologies specialize system parameters for each application phase of execution. Parameters are varied during execution, as opposed to remaining fixed as in an application-based tuning methodology. Prior work and logic suggests phase-based tuning may provide significant savings over application-based tuning. We investigate this hypothesis using a detailed cache model and tune a highly-configurable cache on a per-phase basis compared to tuning once per application, and found phase-based tuning to yield improvements of up to 37% in performance and 20% in energy over application-based tuning. Furthermore, we extend previous phase-based tuning of a configurable cache by significantly increasing configurability and show 14% energy improvement compared to previous methods. In addition, we quantify the overhead imposed due to cache reconfiguration.

Categories and Subject Descriptors
B.3.2 [**Design Styles]:** Cache Memories

General Terms
Design.

Keywords
Caches, configurable caches, phase prediction, configurable architecture, phase-based reconfiguration, phase-based tuning.

1. INTRODUCTION
Research shows that applications have vastly different operating requirements [19], thus facilitating optimization or tuning of system parameters to the needs of a particular application to save energy and/or improve performance. Tunable system parameters include bus width and encoding, memory configuration, voltage scaling, peripheral configuration, processor width configuration, etc. An application-based tuning methodology would choose one value for each configurable parameter, representing the greatest optimization potential for the average run of the entire application.

However, applications show varying operating requirements throughout execution [17]. Application execution can be examined and compared on a per interval basis. An interval of execution is a snippet of execution time measured by the number of dynamic instructions executed. Each interval can be classified according to its operating behavior by examining attributes such as cache miss rates, cycles per instruction, and specific pipeline stalls. Different intervals exhibiting similar execution traits can be classified together as belonging to the same phase of execution. Distinct phases tend to be revisited throughout application execution

This predictable phase behavior enables phase-based optimizations. Phase-based tuning allows for system parameters to be specialized for each phase of execution, and thus reconfigured during runtime. By allowing for system parameters to be varied during runtime, there exists greater potential savings due to intense specialization of system parameters to the changing needs of the application.

In this paper, we make three main contributions. First, we investigate the benefits of phase-based tuning over application-based tuning of a highly configurable cache hierarchy with respect to energy consumption and performance using a detailed cache model. Secondly, we extend previous phase-based tuning of configurable caches by significantly increasing cache configurability, resulting in an additional 14% energy savings on average. And lastly, we carefully quantify the overhead imposed by cache reconfiguration in terms of energy and performance due to write backs and cache flushing.

2. RELATED WORK
One method for phase classification and optimizations is done using an offline-profiling step to determine phase boundaries and phase configurations. Specialized hardware then supports runtime reconfiguration of the parameters. Chaver et al. [4] presented a phase-based adaptive fetch mechanism using an offline profiling step to determine necessary system resources and encoded these changes into the binary. Albonesi et al. [1] presented a method to adaptively change cache associativity to the needs of the application. Software analysis was done on the code to determine

GLSVLSI'08, May 4-6, 2008, Orlando, Fl, USA.

[1] *This works was done while the author was affiliated with the University of California, San Diego.*

associativity requirements and special instructions were used to signal hardware changes during runtime.

Phase classification and optimizations can also be done in a strictly online approach to both characterize phases and determine the best configuration for each phase. Dhodapkar et al. [5] discussed a method to determine phase changes based on examination of the current working set of the application. Balasubramonian et al. [3] used miss rates, CPI, and branch frequency information to detect changes in application behavior for cache specialization. Hallnor et al. [9] used counters within the cache to count way accesses to predict miss rates for all cache sizes during an interval for bank level shutdown. Shen et al. [14] used locality profiling and runtime prediction to predict locality phases within the program.

Our work differs from previous phase-based cache tuning methodologies by exploring a significantly larger set of cache configurations due to a more highly configurable cache, resulting in greater energy and performance benefits on average. We also compare phased-based tuning of a configurable cache with application-based tuning of a configurable cache.

3. CONFIGURABLE CACHES AND PHASE CLASSIFICATION

Configurable caches allow for cache parameters such as total size, line size, and associativity to be specialized to the needs of an application. Core-based processors allow a designer to choose a particular cache configuration during system design time [2] and methods exist to assist designers in choosing the best configuration [7][8][19]. There are even hard processors that allow their caches to be configured during system reset or even during runtime [1][3][12][19].

3.1 Configurable Architecture

To provide intense specialization of the cache hierarchy to the changing needs of an application and thus greater potential energy savings, we examine a very highly configurable two level hierarchy with separate level one instruction and data caches and a unified level two cache.

For the level one cache, we utilized the highly configurable cache designed by Zhang et al. [19] offering configurable total size, line size, and associativity. Special configuration bits enabled reconfiguration of all parameters during runtime. Furthermore, hardware layout of the cache showed that reconfigurability did not impact cache access time.

For the level two cache, we used the architecture offered in the Motorola M*CORE processor featuring way management [12]. With way management, configuration bits specified each way to cache instructions only, data only, instructions and data (unified), or be shutdown. Each way in a way management cache could be designated as one of those four possibilities. In addition to way management, we included the same configurable line size as the level one caches. Furthermore, fabrication [13] of the cache showed that reconfigurability does not impact cache access time.

Given the configurability of each level, the configurable cache hierarchy offered nearly 18,000 different cache configurations.

3.2 Cache Exploration Heuristic

Since exhaustive exploration of 18,000 different cache configurations is infeasible, we developed a highly effective heuristic (ACE-AWT) [8] for the configurable cache architecture discussed in section 3.1. ACE-AWT tuned for maximum energy savings and found a near optimal solution achieving 62% energy savings on average while searching on average only 30 out of the 18,000 configurations – merely 0.2% of the search space. We developed ACE-AWT for an application-based tuning environment and, given ACE-AWT's effectiveness, we sought to explore additional savings revealed by phase-based tuning. Further details can be found in [8][19].

3.3 Phase Classification

To perform phase classification, application execution must first be broken into fixed sized intervals. An interval is measured by the number of dynamic instructions executed and is the granularity at which phases will be identified. Phase classification is the process of grouping intervals showing similar behavior patterns together. By grouping intervals together into phases, optimizations applied to one phase interval will apply equally as well to all intervals classified as that phase.

For this study, we utilized our offline phase classification methodology [15][16], which was extended for online runtime phase tracking and prediction [18] using a small amount of hardware. Phase classification gathers information about basic block execution without compiler support. Specialized hardware monitors execution and traps branch instructions and tallies the number of instructions executed between subsequent branch instructions. The phase classification technique aggregates basic block information in a small table and collectively represents a footprint of execution. The phase classification technique compares current footprints to previous footprints to determine new phases of execution and reoccurrences of previously identified phases. This tracking method is independent of the underlying system architecture.

After phase classification, phase prediction can predict when a phase transition will occur and what phase will be entered using two important predictors. The first predictor is the set of phases leading up to the prediction and the second predictor is the duration of execution spent in those phases.

We further extended phase classification by not attempting to classify transition phases with steady phases [11]. With fixed length intervals, there is rarely a clean transition from one phase to the next, leaving a period of time with erratic behavior. In our methodology, we used this extension to classify transition phases for what they are and do not compare them with steady phases and thus, we do not attempt to optimize for the transition phases.

4. RESULTS

4.1 Experimental Setup

We examined a large selection of both floating point and integer benchmarks from the SPEC2000 benchmark suite because SPEC applications show a greater variation during execution than typical embedded system kernel benchmarks. However, the general

methodology presented in this paper and the results are applicable to embedded application devices as well as desktop environments. We ran each benchmark using the full reference input sets and used multiple input sets per application. Benchmarks will be listed as the benchmark name followed by the input set.

We implemented the cache tuning heuristic using a Perl script to drive simulation. We simulated each configuration explored using SimpleScalar to gather cache hit and miss ratios. We utilized our energy estimation model for the highly-configurable cache [8]. This model took into consideration all energy associated with the cache hierarchy including miss latency and bandwidth, dynamic cache access energy, static energy, main memory energy, and CPU stall energy. We did not include leakage energy of system components during reconfiguration because the granularity chosen results in little reconfiguration overhead. We discuss this further in section 4.3.

In order to calculate accurate energy consumption of an application with a phase-based configurable cache, we modified SimpleScalar to model the reconfigurable cache hierarchy allowing variation of the level one and level two cache parameters during a single run of the application. We applied ACE-AWT to one occurrence of each phase to determine the best cache configuration for each phase. We supplied a set of cache configurations representing the best configuration for each phase to SimpleScalar and at the end of each interval, the phase identification of the next interval could be used to look up the appropriate cache configuration to switch to. We precisely modeled the reconfiguration behavior for each cache, taking great care to accurately model runtime behavior so that we could accurately quantify flushing due to reconfiguration.

4.2 Energy and Performance

To determine energy and performance, we compared the energy consumption of the application executing with a tuned cache to the energy consumption of the application executing with a base cache configuration. For the base cache configuration, we chose a cache hierarchy that both reflected the needs of the applications and represented a configuration that may be found on a platform running these applications. The level one base caches were 32 Kbyte 4-way set associative caches with a 32 byte line size and the level two cache was a 128 Kbyte, fully unified 4-way set associative cache with a 64 byte line size.

Figure 1 shows the energy consumption normalized to the base cache configuration for each application for both application-based tuning and phase-based tuning. On average across all benchmarks, application-based tuning resulted in 34% energy savings while phase-based tuning resulted in 37% energy savings. Phase-based tuning gave an additional 3% energy savings (*avg*) over application-based tuning with additional savings as high as 11% and 20% for *gzip_random* and *bzip_graphic*, respectively. However, close inspection of the results showed that some benchmarks actually consumed *more* energy after being phase-tuned. We investigated these situations and discovered that for these benchmarks, a very large percentage of phases were classified as transition phases and were thus unclassifiable and not tuned for. Unclassified phases ran as high as 80-90% for *art_110* and *art_470* and 32% for *gcc_200*. Taking into consideration that such a large portion for the application is unclassified making these application unsuitable candidates for phase-based tuning, phase-based tuning performed surprisingly well. We point out that in these situations, application-based tuning would be used instead of phase-based tuning and therefore, we reevaluated the results and considered only those benchmarks with positive energy savings for phase-based tuning compared to application-based tuning. The average energy savings (*avg_modified*) of application-based tuning changed to 35% and phase-based rose to 40%.

Figure 2 shows the execution time normalized to the base cache configuration for each application for both application-based tuning and phase-based tuning. On average across all benchmarks (*avg*), application-based tuning resulted in a 16% *increase* in execution time while phase-based tuning resulted in only a 10% *increase* in execution time. Phase-based tuning reduced the negative performance impact by 6%. A few benchmarks – *gzip_random, bzip_graphic,* and *vpr_place* – showed performance savings as high as 37% for phase-based tuning over application-based tuning. In addition, when we removed the benchmarks where phase-based tuning did not show positive energy savings compared to application-based tuning, phase-based tuning reduced the negative performance impact (*avg_modified)* by 10%.

We also compared our phase-based tuning of a highly configurable cache to previous phase-based cache tuning of less configurable caches. One previous method offered either a large or a small cache and the others offered a small subset of caches trading off different requirements [3][6][14]. We examined energy consumption for phased-based tuning offering 2 configurations per phase (either small or large caches), 5 configurations per phase chosen to trade off instruction and data requirements, and

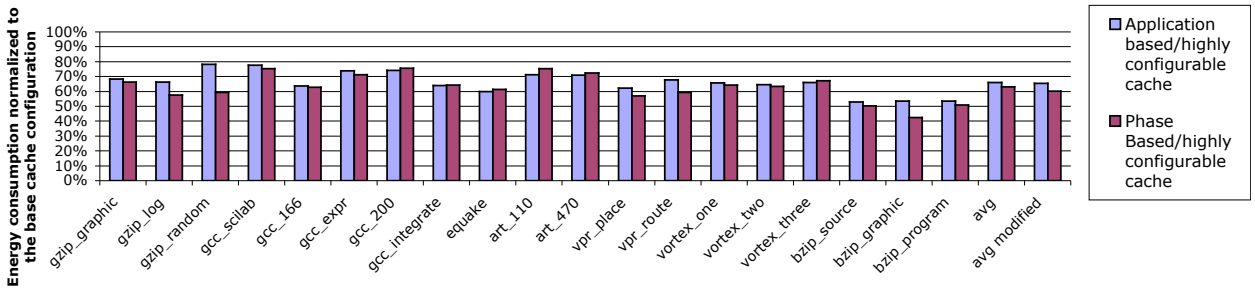

Figure 1: Energy consumption normalized to the energy consumption of the base cache configuration. *Avg* is the average over all benchmarks. *Avg_modified* is the average over all benchmarks where phase-based tuning offers positive energy savings over application-based tuning.

the highly-configurable cache discussed in this paper offering 18,000 different configurations per phase. On average, energy savings for these methods were 23%, 27%, and 39%, respectively. Whereas offering only 2 configurations gave a large initial reduction in energy considering the small size of the configuration space, increasing the configuration space to 5 configurations only netted an additional 4% energy savings. For most benchmarks, significant energy savings were only obtained by searching a highly configurable cache.

4.3 Cache Reconfiguration Overhead

When analyzing the effectiveness of phase-based cache tuning compared to application-based cache tuning, it is important to consider the overhead incurred while switching between different cache configurations. Changing a cache parameter is likely to cause flushing of dirty entries. During cache reconfiguration, we used SimpleScalar to count the number of additional writebacks incurred due to changing cache parameters. On average over all benchmarks, the total number of writebacks increased by less than 1% resulting in a 0.88% increase in energy and a 0.65% increase in execution time.

5. CONCLUSIONS

We analyzed the benefits of phase-based cache tuning over application-based cache tuning using a detailed cache model. We conclude that whereas application-based tuning provides the largest initial reduction in energy consumption, phase-based tuning offers up to 20% additional energy savings. In addition, we examined performance and observed that phase-based tuning improves the negative impact on performance that is incurred in application-based tuning. Application-based tuning incurred a 19% increase in execution time while phase-based tuning only incurred a 9% increase in execution time. In addition, we show the need for utilizing a highly configurable cache in phase-based cache tuning by comparing our phase-based tuning method with previous methods that utilize designs offering only a few possible cache configurations. Our method offers 14% additional energy savings over previous methods.

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

[2] Arc International, www.arccores.com

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

[4] Chaver, D., Rojas, M, Pinuel, L., Prieto, M., Tirado, F., Huang, M. Energy-aware fetch mechanism: trace cache and BTB customization. International Symposium on Low Power Electronics and Design, 2005.

[5] Dhodapkar, A, Smith, J, Comparing program phase detection techniques. MICRO 2003

[6] Dhodapkar, A., Smith. Managing multi-configuration hardware via dynamic working set analysis. 29th Annual International Symposium on Computer Architecture.

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

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

[9] Hallnor, E., Reinhardt, S. Dynamic leakage energy management of secondary caches. Technical Report CSE-TR-459-03, University of Michigan, 2002.

[10] Lau, J., Perelman, E., Calder, B. Selecting software phase markers with code structure analysis. International Symposium on Code Generation and Optimizatoin, 2006.

[11] Lau, J., Schoenmakers, S., Calder, B., Transition phase classification and prediction. HPCA 2005

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

[13] Personal communication with M*Core developers

[14] Shen, X., Zhong, Y., Ding, C. Locality phase prediction. 11th International Conference on Architectural Support for Programming Languages and Operating Systens, 2004.

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

[16] Sherwood, T., Perelman, E., Hamerly, G., Calder, B. Automatically characterizing large scale program behavior. 10th International Conference on Architectural Support for Programming Languages and Operating Systems, 2002.

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

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

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

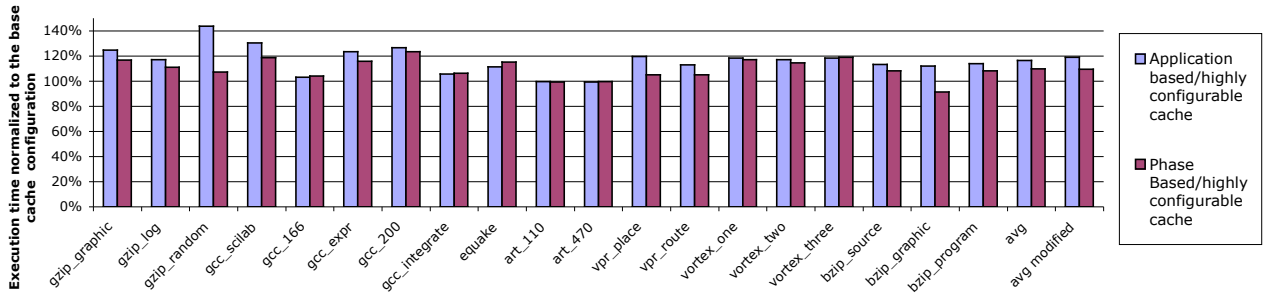

Figure 2: Execution time normalized to the execution time of the base cache configuration. *Avg* is the average over all benchmarks. *Avg_modified* is the average over all benchmarks where phase-based tuning offers positive energy savings over application-based tuning.