On the Interplay of Loop Caching, Code Compression, and Cache Configuration

Marisha Rawlins and Ann Gordon-Ross*
Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL 32611, USA
mrawlins@ufl.edu & ann@ece.ufl.edu
**Also with the NSF Center for High-Performance Reconfigurable Computing*

Abstract – Even though much previous work explores varying instruction cache optimization techniques individually, little work explores the combined effects of these techniques (i.e., do they complement or obviate each other). In this paper we explore the interaction of three optimizations: loop caching, cache tuning, and code compression. Results show that loop caching increases energy savings by as much as 26% compared to cache tuning alone and reduces decompression energy by as much as 73%.

I. Introduction

Since an embedded system's memory hierarchy can consume as much as 50% of the total system power [12], there exists much previous work on individual instruction cache energy optimization techniques. Since many of these techniques target different cache aspects and provide large energy savings (as much as 82% [11]), it is unclear how these techniques would interact if applied together and there exists little research exploring these interactions [5][6][13]. Since a system designer may choose to apply several different optimization techniques, it is important to evaluate how dependent optimization techniques interact (i.e., do these techniques complement each other, degrade each other, or does one technique obviate the other). In this work, we focus on the interactions between three popular cache optimization techniques: loop caching, cache configuration, and code compression.

Loop caches are small devices that provide an effective method for decreasing memory hierarchy energy consumption by storing frequently executed code (critical regions) in a more energy efficient structure than the level one (L1) cache [8][14]. The main purpose of a loop cache is to provide the processor with as many instructions as possible while the larger, more power hungry L1 instruction cache remains idle. The Preloaded Loop Cache (PLC) [8] requires designer-applied static pre-analysis to store complex code regions (code with jumps) where as the Adaptive Loop Cache (ALC) [14] performs this analysis during runtime and requires no designer effort.

Off the shelf microprocessors typically fix the cache configuration to a configuration that performs well on average across all applications. However, this average configuration is rarely an application's optimal configuration since different applications exhibit different runtime behaviors. Instruction cache tuning analyzes the instruction stream and configures the cache to the lowest energy (or highest performance) configuration by configuring the cache size, block size, and associativity. Cache tuning therefore enables application-specific energy/performance optimizations [5][6].

Code compression techniques were initially developed to reduce the static code size in embedded systems. However, recent code compression work [2][11] investigated the effects of code compression on instruction fetch energy in embedded systems. In these systems, energy is saved by storing compressed instructions in the L1 instruction cache and decompressing these instructions (during runtime) with a low energy/performance overhead decompression unit.

Studying the interaction of existing techniques reveals the practicality of combining optimization techniques. For example, if combining certain techniques provides additional energy savings but the combination process is non-trivial (e.g., circular dependencies for highly dependent techniques [5]), new design techniques must be developed to maximize savings. On the other hand, less dependent techniques may be easier to combine but may reveal little additional savings. Finally, some combined techniques may even degrade each other. These studies provide designers with valuable insights for determining if the combined savings is worth the additional design effort.

In this paper, we explore additional energy savings revealed by combining loop caching with two other state-of-the-art optimization techniques: cache tuning and code compression. We have observed that, although cache tuning dominates energy savings, loop caching can provide an additional 26% energy savings. Also, our experiments on loop caching and code compression revealed that the loop cache can effectively reduce the decompression overhead of a system while providing up to 73% overall energy savings.

II. Related Work

A. Loop Caching

The ALC is the most flexible loop cache (loop cache contents are dynamically loaded/changed during runtime) and can store complex loops (i.e., loops with control of flow (cof) changes such as taken branches and forward jumps). Fig. 1 (a) shows the loop cache's architectural placement. The ALC [14] identifies and caches loops during runtime using lightweight control flow analysis. The ALC identifies loops when the loop's last instruction (a short backward branch (sbb) instruction) is taken. The ALC fills the loop cache with the loop instructions on the loop's second iteration and from the third iteration onwards, the ALC supplies the processor with the loop instructions (i.e., the L1 cache is idle). Since loop caches require 100% hit rates, the ALC stores valid bits (exit bits) to determine whether the next instruction should be fetched from the ALC or the L1 cache (thus this transition incurs no additional cycle penalty).

PLC [8] operation is similar to the ALC's (the PLC can store complex loops), however PLC contents are statically profiled and pre-analyzed during design time and loaded during system startup.

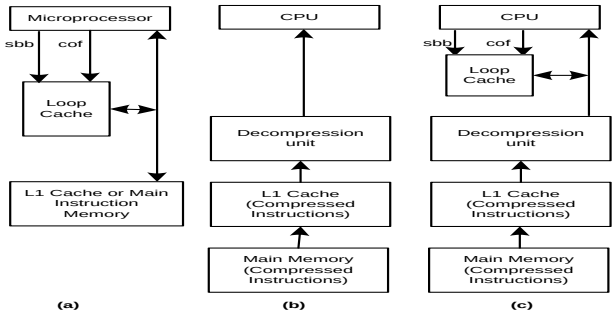

Fig. 1. (a) Architectural placement of the loop cache, (b) the Decompression on Fetch (DF) architecture, and (c) the Decompression on Fetch (DF) architecture with a Loop Cache to store Decompressed Instructions

Both the ALC and PLC can reduce instruction memory hierarchy energy by as much as 65% [8][14]. Since instructions stored in the PLC never enter the L1 cache, using a PLC affects the locality of the L1 cache. Eliminating instructions from the L1 cache could affect the overall energy savings of the system since the L1 cache (an important source of energy savings) takes advantage of an application's locality for improved performance and energy consumption.

B. Cache Tuning

Since energy is wasted when the cache configuration (size, line size, and associativity) does not reflect the needs of the specific application, much previous work focuses on cache configuration specialization (*cache tuning*). Motorola's M*Core M3 processor [12] and Albonesi [1] tune the cache size and associativity using way designation or way shutdown. Zhang et al. [17][18] developed way concatenation, a method that logically concatenated ways to adjust associativity. Line size can be adjusted using line concatenation [17], which logically implements larger line sizes as multiples of physical smaller line sizes. Previous work on cache tuning has shown that a single level highly configurable cache (configurable size, line size, and associativity) can achieve more than 40% average energy savings [17][18].

C. The Combined Effects of Cache Tuning and Other Optimization Techniques

Nacul et al. [13] investigated the effects of combining dynamic voltage scaling (DVS) with dynamic cache reconfiguration (DCR). Results showed that, when applied individually, DVS and DCR reduced energy consumption by similar amounts on average. However, combining DVS and DCR resulted in up to 27% additional energy savings, versus using either technique individually, for tasks with longer deadlines.

Previous work evaluated the effects of combining hardware/software partitioning with cache tuning [6]. Hardware/software partitioning removes the critical regions from the software and implements these critical regions in smaller, more energy efficient custom hardware, such as a field programmable gate array (FPGA). Results showed that a non-partitioned system achieved average instruction cache energy savings of 53% while a partitioned system achieved average instruction energy savings of 55% with improved performance showing that cache tuning is still beneficial even after hardware/software partitioning is applied.

Other previous work evaluated the effects of combining code reordering and cache tuning [5]. Code reordering attempts to improve system performance by placing frequently executed instructions contiguously in memory, thus improving spatial locality and cache utilization (it is well known that code reordering does not always improve performance). Combining code reordering with cache tuning resulted in only a 2% increase in energy savings compared to cache tuning individually. However, cache tuning eliminated the performance degradation for applications that did not benefit from code reordering alone. Finally, for certain applications, code reordering resulted in cache configurations that reduced the area overhead, since the increased spatial locality provided by code reordering resulted in smaller, more efficient cache configurations.

D. Code Compression

Several code compression techniques are based on well-known lossless data compression mechanisms. Wolfe and Chanin [16] used Huffman coding to compress/decompress code for RISC processors. They also introduced Line Address Tables (LATs), which mapped program instruction addresses to their corresponding compressed code instruction addresses.

Lekatsas et al. [11] incorporated different data compression mechanisms by separating instructions into groups. Codes appended to the beginning of an instruction group identified the group's compression mechanism. This approach achieved system (cache, processor, and busses) energy savings between 22% and 82%.

Benini et al. [2] proposed a low overhead Decompression on Fetch (DF) (Fig. 1 (b)) technique based on fast dictionary instructions. The authors noted that since in the DF architecture the decompression unit was on the critical path (since the decompression unit was invoked for every instruction executed), the unit must have a low decompression (performance) overhead. In their approach, the authors profiled the executable to identify the 256 most frequently executed instructions (denoted as S_N) and replaced those instructions with an 8-bit code if that instruction and its neighboring instructions could be compressed into a single cache line. Results showed average system energy savings of 30%.

III. Loop Cache and Level One Cache Tuning

A. Experimental Setup

To determine the combined effects of loop caching and cache tuning, we determined the optimal (lowest energy) loop cache and L1 configurations for systems using the ALC and the PLC for 31 benchmarks from the EEMBC [4], MiBench [9], and Powerstone [15] benchmark suites (all benchmarks were run to completion, however, due to incorrect execution not related to the loop caches, we could not evaluate the complete suites).

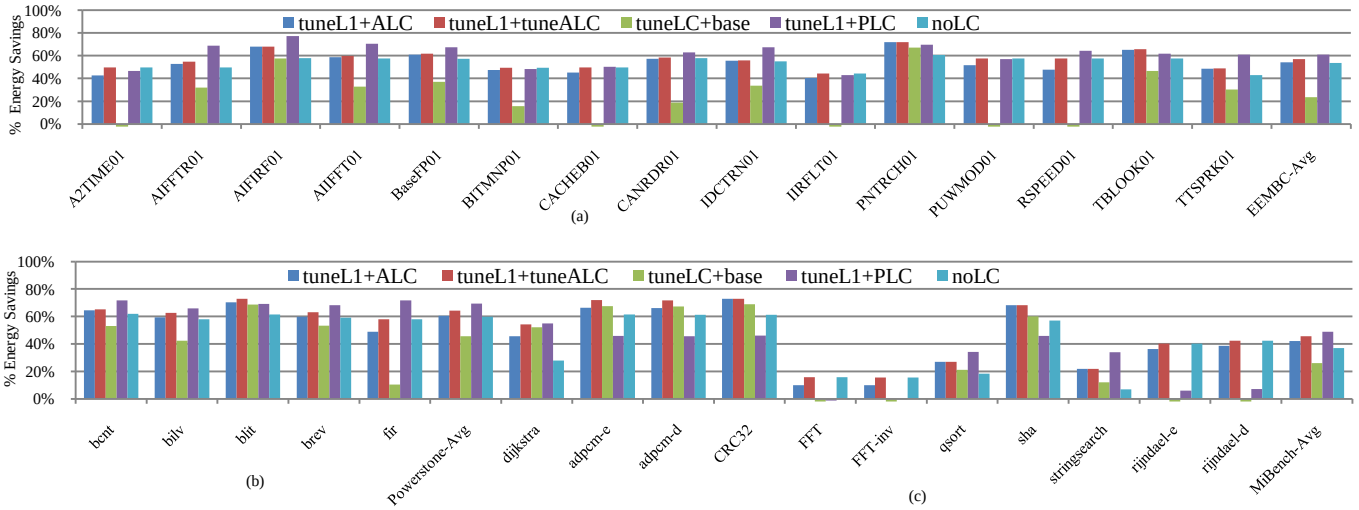

Fig. 2. Energy savings (compared with the base system with no loop cache) for loop caching and cache tuning for the (a) EEMBC, (b) Powerstone, and (c) MiBench benchmark suites.

We used the energy model and methods in [14] to calculate energy consumption for each configuration. For comparison purposes, we normalize energy consumption to a ***base system*** configuration with an 8 KB, 4-way set associative L1 instruction cache with a 32-byte line size (a configuration shown in [18] to perform well for a variety of benchmarks on several embedded microprocessors) and with no loop cache. We implemented each loop cache design in SimpleScalar [3]. We varied the L1 instruction cache size from 2KB to 8KB, the line size from 16 bytes to 64 bytes, and the associativity from direct-mapped to 4-way [17][18], and varied the loop cache size from 4 to 256 entries [14]. In our experiments we searched all possible configurations to find the optimal (lowest energy) configuration, however, heuristics (such as in [6][7][17]) can also be applied for dynamic configuration.

Our experiments evaluated three different system configurations. In the first experiment, we tuned the L1 cache with a fixed 32-entry ALC for the EEMBC and MiBench and a fixed 128-entry ALC for Powerstone (denoted as tuneL1+ALC) ([14] showed that these sizes performed well on average for the respective benchmark suites). In the second experiment, we quantified additional energy savings gained by tuning both the L1 instruction cache and the ALC (denoted as tuneL1+tuneALC). In our final experiment, we tuned the L1 cache while using a fixed 128-entry PLC (denoted as tuneL1+PLC). For thorough comparison purposes, we also report energy savings obtained by tuning the ALC using a fixed L1 base cache configuration (denoted as tuneLC+base) and tuning the L1 cache in a system with no loop cache (denoted as noLC).

B. Analysis

Fig. 2 depicts energy savings for all experiments described in Section III.A normalized to the base system. In summary, these results compare the energy savings for combining loop caching and L1 cache tuning with the energy savings for applying loop caching and cache tuning individually.

First, we evaluated energy savings for each technique individually. L1 cache tuning alone achieved average energy savings of 53.62%, 59.61%, and 37.04% for the EEMBC, Powerstone, and MiBench benchmark suites, respectively. ALC tuning in a system with a base L1 cache achieved average energy savings of 23.41%, 45.55%, and 26.04% for the EEMBC, Powerstone, and MiBench benchmark suites, respectively. These results revealed that in general, ALC tuning alone did not match the energy savings of L1 cache tuning alone. In this case a smaller optimal L1 cache saved more energy than the ALC combined with the (much larger) base cache. For example, tuning the ALC with a fixed base L1 cache achieved 33.57% energy savings for EEMBC's *IDCTRN01*. However, when L1 cache tuning was applied, the 8 KB, 4-way, 32-byte line size base L1 cache is replaced with a much smaller 2 KB, direct-mapped, 64-byte line size L1 cache, resulting in energy savings of 55.07%.

However, loop cache tuning alone can save more energy than L1 cache tuning without a loop cache when the optimal L1 cache configuration is already similar to the base cache (such as *dijkstra* in Fig. 2). Also, when ALC loop cache access rates are high, ALC cache tuning alone is sufficient such as with EEMBC's *PNTRCH01*, Powerstone's *blit*, and MiBench's *CRC32*, which all have loop cache access rates greater than 90%.

Next, we evaluated the combined effects of a fixed sized ALC with L1 cache tuning (tuneL1+ALC in Fig. 2). Additional energy savings were minor as compared to L1 cache tuning alone (average energy savings are 54.24%, 60.59%, and 42.06% for the EEMBC, Powerstone, and MiBench benchmark suites, respectively). Although the average improvement in energy savings across the benchmark suites was approximately 1%, adding a fixed sized ALC improved energy savings by as much as 14.91% for MiBench's *stringsearch* benchmark. Also, in cases where loop caching alone resulted in negative energy savings (benchmarks with less than a 10% loop cache access rate), this negative impact was offset using L1 cache tuning. For example, even when the ALC caused a 9% increase in energy consumption, the overall energy savings was still 47.83% for EEMBC's *RSPEED01* benchmark since L1 cache tuning dominated the overall energy savings.

Our next experiment investigated the effects of tuning both the L1 and the loop cache (tuneL1+tuneALC in Fig. 2). Although there were improvements in energy savings, the resulting average energy savings of 56.88%, 64.36%, and 45.56% for the EEMBC, Powerstone, and MiBench benchmark suites, respectively, were not significantly better than the energy savings achieved by L1 cache tuning alone. Even though the average energy savings improvement across all benchmark suites was approximately 4%, additional energy savings reached as high as 26.30% for MiBench's *dijkstra* benchmark.

Additionally, when comparing a system with a tuned L1 cache and a fixed sized ALC, the improvement in average energy savings was minor, averaging only 2% over all benchmark suites, however, improvements reached as high as 10% for EEMBC's *RSPEED01* benchmark. The reason for the minor additional energy savings was because the energy savings for the optimal ALC size was very close to the savings for the fixed sized ALC for two reasons: 1) the optimal ALC size was typically similar to the fixed sized ALC size (the ALC's size was chosen because it performed well on average for each particular suite); and 2) loop cache access rates leveled off as the loop cache size increased [14]. This finding is significant in that it reveals that L1 cache tuning obviates ALC tuning. If a system designer wishes to incorporate an ALC, simply tuning the L1 cache and adding an appropriately sized ALC is sufficient. This finding's significance is also important for dynamic cache tuning since using a fixed sized ALC decreases design exploration space by a factor of seven since we eliminate the need to combine each L1 configuration with seven ALC sizes.

The results presented thus far suggest that, in general, in a system optimized using L1 cache tuning, an ALC can improve energy savings, but it is not necessary to tune the ALC since L1 cache tuning dominates the energy savings. We observed that, since the optimal ALC configuration does not change the optimal L1 cache configuration, there is no need to consider the ALC during L1 cache tuning. The L1 cache configuration remains the same regardless of the presence of the ALC because using an ALC does not remove any instructions from the instruction stream, nor does the ALC prevent those instructions from being cached in the L1 cache and therefore, does not affect the locality. In fact, the L1 cache supplies the processor with instructions during the first two loop iterations to fill the ALC [14]. The additional energy savings achieved by adding an ALC to the optimal L1 cache configuration results from fetching instructions from the smaller, lower energy ALC [14]. The tradeoff for adding the ALC is an increase in area, which can be as high as 12%. However, this area increase is only a concern in highly area-constrained systems, in which case the system designer should choose to apply L1 cache tuning with no ALC.

Since the ALC does not change the actual instructions stored in the L1 cache (the ALC only changes the number of times each instruction is fetched from the L1 cache), our final experiment involved combining the L1 cache tuning with a fixed sized PLC, since the PLC actually eliminates instructions from the L1 cache. Tuning the L1 cache and using a fixed sized PLC resulted in average energy savings of 61.04%, 69.33%, and 48.91% for the EEMBC, Powerstone, and MiBench benchmark suites, respectively. On average, adding the PLC to L1 cache tuning revealed an additional energy savings of 9.64% as compared to L1 cache tuning alone (with no loop cache) with individual additional savings ranging from 10% to 27% for 12 of the 31 benchmarks. Furthermore, since the PLC is preloaded and the preloaded instructions never enter the L1 instruction cache, using a PLC can change the optimal L1 cache configuration, especially when PLC access rates are very high. Adding the PLC changed the optimal L1 cache configuration for 14 benchmarks, which resulted in area savings as high as 33%. Whereas these additional savings may be attractive, we reiterate that these additional savings come at the expense of the PLC pre-analysis step and requires a stable application.

IV. Code Compression, Loop Caching, and Cache Tuning

Using a loop cache can decrease decompression overheads (performance and energy) for DF techniques by storing/caching uncompressed instructions (Fig. 1 (c)) in a smaller, more energy efficient loop cache. The magnitude of this overhead reduction is dependent on an application's temporal and spatial locality. In addition, code compression reduces the L1 cache requirements. In this section, we quantify the overhead reduction afforded by introducing a loop cache as an instruction decompression buffer, in addition to cache tuning for both the L1 and loop caches.

A. Experimental Setup

To determine the combined effects of code compression with cache tuning, we determined the optimal (lowest energy) L1 cache configuration for a system using a modified DF architecture (Fig. 1 (c)) for the same 31 benchmarks and experimental setup as described in Section III.A. For comparison purposes, energy consumption and performance was normalized to a base system configuration with an 8 KB, 4-way set associative L1 base cache with a 32-byte line size (with no loop cache). Based on [14] we used a 32-entry ALC and a 64-entry PLC for our experiments.

We used Huffman encoding [10] for instruction compression/decompression. Branch targets were byte aligned to enable random access decompression and a LAT translated uncompressed addresses to corresponding compressed addresses for branch and jump targets.

We modified SimpleScalar [3] to include the decompression unit, LAT, and loop cache. The energy model used in Section III.A was modified to include decompression energy. We also measured the performance (total number of clock cycles needed for execution). The performance measured was normalized to the performance of the base system with uncompressed instructions and no loop cache.

B. Analysis

Fig. 3 depicts the (a) energy and (b) performance of the optimal (lowest energy) L1 cache configuration for a system that stores compressed instructions in the L1 cache and uncompressed instructions in a loop cache (ALC or PLC) normalized to the base system with no loop cache. For

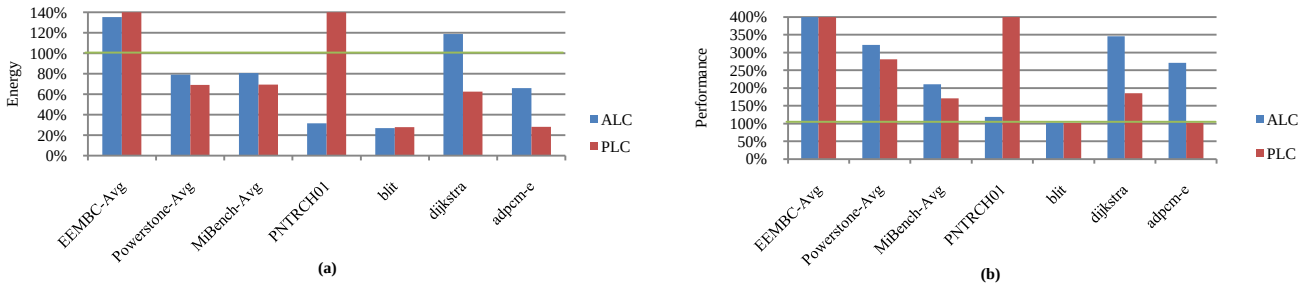

Fig. 3. (a) Energy and (b) performance (energy and performance normalized to the base system with no loop cache) for the lowest energy cache configuration averaged across the EEMBC benchmarks (EEMBC-Avg), Powerstone benchmarks (Powerstone-Avg), MiBench benchmarks (MiBench-Avg), and for selected individual benchmarks (*PNTRCH01*, *blit*, *dijkstra*, and *adpcm-e*)

brevity, Fig. 3 shows average energy and performance for each benchmark suite and selected individual benchmarks that revealed interesting results.

Fig. 3 (a) shows that, on average, for the EEMBC benchmarks, the optimal L1 cache configuration combined with the 32-entry ALC did not result in energy savings. However, on average, the Powerstone and MiBench benchmarks achieved energy savings of 20% and 19%, respectively (Fig. 3 (a)) for the system with an ALC.

Analysis of the benchmark structure revealed that both Powerstone and MiBench benchmarks contain only a few loops that iterate several times (several Powerstone and MiBench benchmarks stay in the same loop for hundreds of consecutive iterations) resulting in energy savings and a lower performance overhead. EEMBC benchmarks however, contain many loops that iterate fewer times than the Powerstone and MiBench benchmarks (several EEMBC benchmarks stay in the same loop for less than 20 consecutive iterations). EEMBC benchmarks spend a short time fetching uncompressed instructions from the ALC before a new loop is encountered and the decompression unit is invoked again resulting in low energy savings and a large performance overhead. However, EEMBC benchmarks with a high loop cache access rate achieved energy savings (for example, *PNTRCH01* with a 97% loop cache access rate [14] achieved 69% energy savings (Fig. 3 (a)) with only a small decompression overhead (Fig. 3 (b))).

Fig. 3 (a) shows that, on average, the Powerstone and MiBench benchmark suites both achieved energy savings of 30% for the system with an optimal L1 cache configuration combined with a 64-entry PLC. An additional 10% in average energy savings was gained by eliminating the decompression overhead, which would have been consumed while filling the ALC. Fig. 3 (a) shows that MiBench's *dijkstra* and *adpcm-e* benchmarks saved 56% and 38% more energy, respectively, when using the PLC instead of the ALC. Results for Powerstone's *blit* benchmark highlight the impact of the decompression overhead. For *blit*, the loop cache access rate for the 32-entry ALC is higher than the loop cache access rate for the 64-entry PLC (80% compared with 30% [14]) but by removing the decompression energy consumed during the first 2 iterations of the loop, the system with the PLC saved almost as much energy as the system with the ALC (Fig. 3 (a)).

Fig. 3 (a) also shows that, on average, for the EEMBC benchmarks, using the PLC did not result in energy savings and that the ALC outperformed the PLC. This result is expected since, for the EEMBC benchmarks, the PLC only outperformed the ALC for the 256-entry loop cache [14].

Fig. 3 (b) shows that, on average, the performance of the system increased for both the ALC and the PLC because of the large decompression overhead (the loop cache does not affect system performance since it guarantees a 100% loop cache hit rate). The average increase in performance due to decompression overhead ranged from as much as 4.7x for EEMBC benchmarks with a PLC to 1.7x for MiBench benchmarks with a PLC (Fig. 3 (b)). We also observed that using the PLC instead of the ALC reduced the decompression overhead by approximately 40% for Powerstone and MiBench benchmarks. Individual results showed that, for most benchmarks, the PLC reduced the decompression overhead but increased system performance as compared to a system with no PLC. As shown in Fig. 3, for the system with the PLC, MiBench's *adpcm-e* achieved 73% energy savings (38% more than the system with the ALC) and reduced performance overhead to only 2% more than the performance of the base system.

For our experiments, we tuned the L1 cache while keeping the loop cache size fixed to find the optimal (lowest energy) combination of L1 cache and loop cache. We compared these new L1 cache configurations to the lowest energy L1 cache configurations for a system with uncompressed instructions and no loop cache. We found that for 12 out of 31 benchmarks the new L1 cache configurations were smaller for the systems using compression compared with the L1 cache configurations for the systems not using compression. These benchmarks were able to use smaller L1 cache configurations since the L1 cache stored compressed instructions, and effectively increased the cache size. However, we did not observe a change in L1 cache configuration for systems with low loop cache access rates and no energy savings. Additionally, for some benchmarks, the optimal L1 configuration for the uncompressed system was already the smallest size (2 KB) so adding a loop cache did not result in a smaller L1 cache configuration.

We calculated the area savings gained by replacing the L1 cache storing uncompressed instructions with the smaller L1 cache storing compressed instructions combined with the loop cache for the 12 benchmarks with new optimal L1 configurations. The benchmarks that replaced an 8 KB L1 cache with a 2 KB L1 cache and loop cache achieved approximately 50% area savings. The benchmarks that replaced an 8 KB L1 cache with a 4 KB L1 cache and loop cache and replaced a 4 KB L1 cache with a 2 KB L1 cache

and loop cache achieved approximately 30% and 20% area savings, respectively. For the remaining benchmarks, the L1 cache configuration did not change, and thus adding a loop cache increased the area of the system. Some benchmarks achieved energy savings but not area savings. For example, EEMBC's *PNTRCH01* benchmark had a loop cache access rate of 97% and achieved a 69% energy savings with the ALC, but the L1 configuration was the same for both the uncompressed and compressed system, which resulted in an increase in area of approximately 14%.

V. Conclusions

We investigated the effects of combining loop caching with level one cache tuning and found that in general, cache tuning dominated overall energy savings indicating that cache tuning is sufficient for energy savings. However, we observed that adding a loop cache to an optimal (lowest energy) cache increased energy savings by as much as 26%. Finally, we investigated the possibility of using a loop cache to minimize run-time decompression overhead and quantified the effects of combining code compression with cache tuning. Our results showed that a loop cache effectively reduced the decompression overhead, resulting in energy savings of up to 73%. However, to fully exploit combining cache tuning, code compression, and loop caching, a compression/decompression algorithm with lower overhead than the Huffman encoding technique is required, and is the focus of our future work.

Acknowledgements

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.

References

[1] Albonesi, D. H. 1999. "Selective cache ways: on-demand cache resource allocation," in Proceedings of the 32nd Annual ACM/IEEE international Symposium on Microarchitecture (Haifa, Israel, November 16 - 18, 1999). International Symposium on Microarchitecture. IEEE Computer Society, Washington, DC, 248-259.

[2] Benini, L., Macii, A., and Nannarelli, A. 2001. "Cached-code compression for energy minimization in embedded processors," in Proceedings of the 2001 international Symposium on Low Power Electronics and Design (Huntington Beach, California, United States). ISLPED '01.

[3] Burger, D., Austin, T., Bennet, S. "Evaluating Future Microprocessors: The SimpleScalar ToolSet", University of Wisconsin-Madison. Computer Science Department. Tech. Report CS-TR-1308, July 1996.

[4] EEMBC. http://www.eembc.org/.

[5] Gordon-Ross, A., Vahid, F., and Dutt, N. 2005. "A first look at the interplay of code reordering and configurable caches," in Proceedings of the 15th ACM Great Lakes Symposium on VLSI (Chicago, Illinois, USA, April 17 - 19, 2005). GLSVSLI '05.

[6] Gordon-Ross, A., Viana, P., Vahid, F., Najjar, W., and Barros, E. 2007. "A one-shot configurable-cache tuner for improved energy and performance," in Proceedings of the Conference on Design, Automation and Test in Europe (DATE).

[7] Gordon-Ross, A., Vahid, F., and Dutt, N. 2004. "Automatic Tuning of Two-Level Caches to Embedded Applications," in Proceedings of the Conference on Design, Automation and Test in Europe - Volume 1 (February 16 - 20, 2004). Design, Automation, and Test in Europe. IEEE Computer Society, Washington, DC, 10208.

[8] Gordon-Ross, A., Cotterell, and Vahid, F. "Exploiting fixed programs in embedded systems: A Loop cache example," Computer Architecture Letters, Volume 1, January 2002.

[9] Guthaus, M.R., Ringenberg, J.S., Ernst,D., Austin, T.M., Mudge, T., Brown, R.B. "MiBench: A free, commercially representative embedded benchmark suite," IEEE 4th Annual Workshop on Workload Characterization, December 2001.

[10] Huffman, D. A. "A Method for the Construction of Minimum-Redundancy Codes," Proceedings of the IRE, Vol. 4D, pp. 1098-1101, Sept. 1952.

[11] Lekatsas, H.; Henkel, J.; Wolf, W. "Code compression for low power embedded system design," Design Automation Conference, 2000. Proceedings 2000. 37th , vol., no., pp.294-299, 2000.

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

[13] Nacul, A. C. and Givargis, T. 2004. "Dynamic Voltage and Cache Reconfiguration for Low Power," in Proceedings of the Conference on Design, Automation and Test in Europe - Volume 2 (February 16 - 20, 2004). Design, Automation, and Test in Europe. IEEE Computer Society, Washington, DC, 21376.

[14] Rawlins, M. and Gordon-Ross, A. 2010. "Lightweight runtime control flow analysis for adaptive loop caching," in Proceedings of the 20th Symposium on Great Lakes Symposium on VLSI (Providence, Rhode Island, USA, May 16 - 18, 2010). GLSVLSI '10.

[15] Scott, J.,Lee, L., Arends, J., Moyer, B. 1998. "Designing the Low- Power M~CORE Architecture," International Symposium on Computer Architecture Power Driven Microarchitecture Workshop, Barcelona, Spain, July 1998, pp. 145-150

[16] Wolfe, A. and Chanin, A. 1992. "Executing compressed programs on an embedded RISC architecture," in Proceedings of the 25th Annual international Symposium on Microarchitecture (Portland, Oregon, United States, December 01 - 04, 1992). International Symposium on Microarchitecture. IEEE Computer Society Press, Los Alamitos, CA, 81-91.

[17] Zhang, C., Vahid, F., and Lysecky, R. 2004. "A self-tuning cache architecture for embedded systems," ACM Trans. Embed. Comput. Syst. 3, 2 (May. 2004), 407-425.

[18] Zhang, C., Vahid, F., and Najjar, W. 2000. "A highly-configurable Cache Architecture for Embedded Systems," 30th Annual International Symposium on Computer Architecture, June 2000.