An Efficient Direct Mapped Instruction Cache for Application-Specific Embedded Systems

Chuanjun Zhang
Computer Science and Electrical Engineering Department
University of Missouri-Kansas City
Kansas City, MO 64110
zhangchu@umkc.edu

Abstract
Caches may consume half of a microprocessor's total power and cache misses incur accessing off-chip memory, which is both time consuming and energy costly. Therefore, minimizing cache power consumption and reducing cache misses are important to reduce total energy consumption of embedded systems. Direct mapped caches consume much less power than that of same sized set associative caches but with a poor hit rate on average. Through experiments, we observe that memory space of direct mapped instruction caches is not used efficiently in most embedded applications. We design an **efficient cache** – a configurable instruction cache that can be tuned to utilize the cache sets efficiently for a particular application such that cache memory is exploited more efficiently by index remapping. Experiments on 11 benchmarks drawn from Mediabench show that the efficient cache achieves almost the same miss rate as a conventional two-way set associative cache on average and with total memory-access energy savings of 30% compared with a conventional two-way set associative cache.

Categories and Subject Descriptors
B 3.2 [Memory Structures]: Design Style, Cache memories

General Terms
Design, Algorithm

Keywords
Instruction cache, low power cache, efficient cache design

1. Introduction
Cache may consume up to 50% of a microprocessor's total power [8][10], including both dynamic and static power. Cache misses incur accessing to off-chip memory, which is both time consuming and energy costly, because of the high capacitance of off-chip buses and large storage of off-chip memory. Embedded processors need low power and low energy techniques to prevent overheating and to prolong battery life.

Direct-mapped (DM) caches are popular in embedded microprocessor architectures due to their simplicity. A DM cache consumes less power per access than a same sized set-associative cache that has to access multiple cache ways simultaneously. Furthermore, a DM cache does not have a multiplexor that is used to select the desired way data among cache ways as in a set associative cache, thus a DM cache has faster access time, which is often on the microprocessor's critical path and thus influences clock frequency.

DM caches have poor hit rates on average than the same sized set associative caches. A low hit rate means more access to off-chip memory and off-chip buses, which are both energy and time costly. One way to solve this problem is to use set associative caches. Set associative caches have a low miss rate hence less accesses to off-chip memory, but set associative caches have a much higher per access energy consumption than that of a same size direct mapped cache. A two way set associative cache consumes around 40% more energy than a same sized direct mapped cache [9].

A particular embedded system may run one or just a few applications in the system's lifetime. Embedded system designers are increasingly using application-specific architectures tuned to that one or few applications (e.g., Tensilica's application-specific processor [11]). We examine the usage of direct-mapped instruction caches in embedded systems for particular applications and observe that cache memory is not used efficiently during the execution of most applications. Many cache sets are not used efficiently for a particular application. We name these sets as *underused* sets. Furthermore, we notice that some cache sets are *overused* and accessed more frequently than average. We name these cache sets as *overused sets*. Because an embedded system will typically execute one or just a few fixed applications during its lifetime, we design a configurable instruction cache that the accesses to *overused cache sets* are reduced and hence the conflict miss rate. Furthermore, the reduced cache accesses are remapped to the *underused cache sets* to improve the hit rate. Balancing the accesses to all cache sets would not increase the per access dynamic and static energy consumption of the instruction cache. However, the hit rate of the efficient cache is increased therefore the accesses to off-chip memory are reduced, which means the execution time of an application is reduced hence the total static energy, since static energy consumption is proportional to the execution time. Dynamic energy is also reduced due to fewer visits to off-chip bus and memory, which are power costly.

The rest of the paper is organized as follows. We discuss related work in Section 2. We present data on our observation of the *overused* and *underused* cache sets in Section 3,. We discuss the design of our efficient cache in Section 4 and describe how to use an efficient cache in a configurable pre-fabricated platform in Section 5. We discuss experimental results in Section 6. The limitations of the efficient cache are discussed in Section 7. We conclude the paper in Section 8.

2. Related Work
Peir[5] found that 40% of the cache frames in a direct-mapped data cache contain less recently used sets during the execution of TPC-C benchmarks. He developed an adaptive group-associative cache

CODES+ISSS'05, Sept. 19–21, 2005, Jersey City, New Jersey, USA.

that dynamically identified these less accessed cache sets and utilized these cache sets to approximate the global least-recently used replacement policy to improve the performance of a direct-mapped cache. His technique improved the hit rate of a direct-mapped cache to that of a same sized four-way set-associative cache. Another approach, cache set decay [4], dynamically turns off cache sets that have not been visited for a designated period, reducing the L1-cache leakage energy dissipation by 4x in SPEC2000 applications.

Compared with our efficient cache, Peir's design maps cache sets to less frequently used cache sets to reduce conflict miss in a direct-mapped cache and thus requires a complex design, such as to distinguish a hit as a hit on a direct-mapped location or a hit in an out of position cache set that needs an extra cycle to access. The extra cycle would prolong the execution time and consume more energy even if the hit rate is improved. The cache decay method requires special circuit techniques to make the cache sets consume less static energy. These techniques may not easily available to embedded system designers.

Several researchers proposed to tune cache size to fit a particular application. Albonesi [1] dynamically tuned the size of a set-associative cache by shutting down cache ways to save dynamic power. Zhang [12] proposed a configurable cache whose size and number of ways can be tuned to a particular application to save dynamic and static power. Both of them tune the cache size at the granularity of cache ways, which may incur extra misses and need special shut down circuits.

An optimal direct mapped cache index mapping[3] is proposed to reduce direct mapped cache's miss rate of a **synthesizable core** for embedded systems. A heuristic is also proposed to determine the optimal index mapping for a particular application.

Our efficient cache is intended to be used in a **pre-fabricated platform**. A pre-fabricated platform is a chip that has already been designed but is intended for use in a variety of possible applications with advantages of mass production and shorter time-to-market. To perform efficiently for the largest variety of applications, recent platforms come with parameterized architecture that a designer can configure for his/her particular set of applications. Recent architectures include cache parameters [8][12][1] that can be configured by setting a few configuration register bits. We therefore developed a configurable efficient instruction cache whose index mapping can be configured by setting bits in a configuration register.

3. Overused/underused cache sets observation

Through experiments, we observe that some cache sets are overused and visited much more than average for an application. Furthermore, some other cache sets are underused and accessed much less than average.

Figure 1 shows one example of the benchmark *adpcm_enc*, drawn from the MediaBench [7] benchmark suite. The *underused* sets are scattered among the whole cache space, which means that simple shut down schemes, such as shutting down a cache subarray, may incur extra misses because useful cache sets may be taken out unexpectedly.

The other observation we make is that even at the subarray level, there are still *overused* and *underused* subarrays. These *overused* cache subarrays generate conflict misses that otherwise would be reduced if the accesses to theses cache subarrays are reduced.

Based on the above two observations, we design an efficient instruction cache architecture for embedded systems in which the accesses to *overused* cache subarrays are reduced and the accesses to *underused* subarrays are increased without increasing the cache access time. The benefit of our efficient cache is that we reduce the miss rate of efficient cache without increasing the cache's size, associativity, or cache access time.

4. Efficient Instruction Cache Design

4.1 Basic design

Figure 2 shows the design of our efficient instruction cache. Cache memories are divided into subarrays to achieve the best trade-off of area, performance (access time), and power consumption [9]. There are four subarrays in a conventional four and eight Kbyteyte direct-mapped cache [9]. The original subarray decoders, four two-input AND gates, have been replaced by configurable decoders. Each subarray has two rows of configurable decoders, while each row has five bits. The contents in the new configurable decoders, which are shown in Figure 2, implement the same decoding function as the original four two-input AND gates.

In Figure 3, we show two cases that the contents of the configurable decoder is configured to effectively use the cache space. In Figure 3(a), the accesses to subarray *0* is reduced by extending the decoder from "*00*" to "*000*". The accesses to subarray *1* is increased by re-mapping "100" from the original subarray *0* to the subarray *1*.In Figure 3(b), the accesses to the subarray *0* is reduced by extending the decoder from "*00*" to "0000", which means that only one fourth of the original address space is mapped to the subarray *0*. The other addresses, "*0100*", "*1000*", and "*1100*" are remapped to subarray *1, 2,* and *3*, respectively. The "X" in Figure 3 means "don't care." The two XCAM bits of the configurable decoders in addition to the second

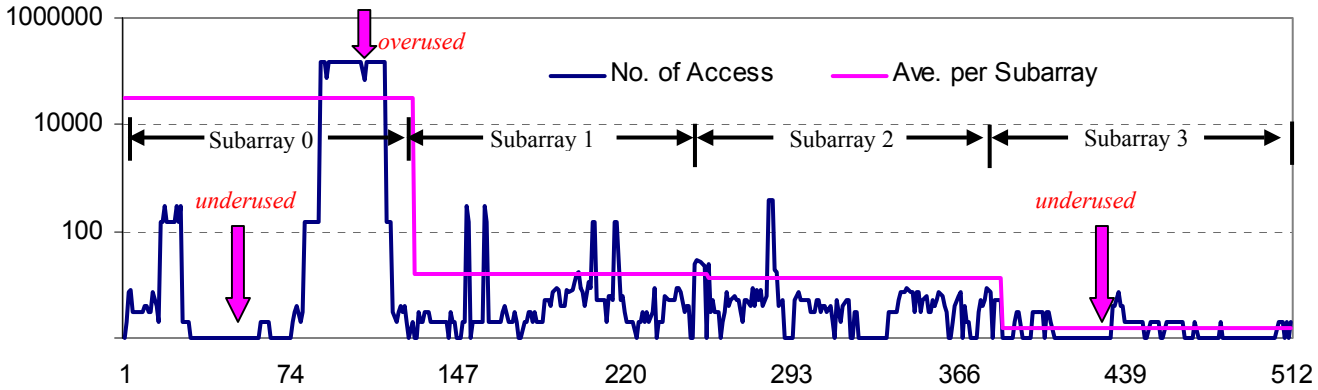

Figure 1: Instruction cache set access counts for *adpcm_enc* from Mediabench for an 8 Kbyte cache.

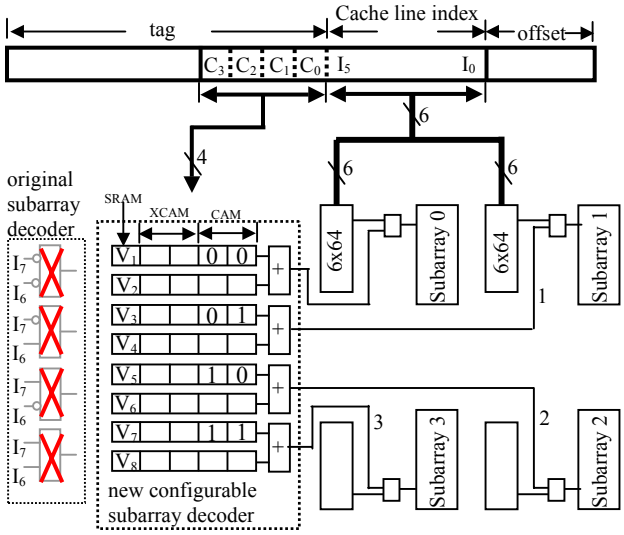

Figure 2:The organization of an *efficient cache*.

row of the decoder may need don't care functions. The V_i , *i = 1, 2,…8,* are valid signals of the configurable decoder. When V_1, V_3, V_5, and V_7 are zeros, the two XCAM bits of the corresponding decoders are not used; otherwise, the original decoder is extended to include the two XCAM bits. Therefore, accesses to corresponding subarrays are reduced. When V_2, V_4, V_6, and V_7 are zeros, the corresponding decoders are not used; otherwise increasing the width of the decoder from one row to two rows increases accesses to corresponding subarrays.

4.2 Cache access time

To avoid increasing the cache access time, the new configurable decoders must run as fast as or faster than the 6-to-64 decoders. The programmable decoder consists of two rows of decoders. The first row has two conventional CAM cells, two XCAM cells that are conventional CAM cells with "don't care" function implemented, one SRAM for the valid bit of the decoder. The second row has one bit SRAM for valid bit and four XCAM cells. We use standard ten-transistor CAM cells, shown Figure 6. The XCAM cell needs an extra NMOS transistor, shown Figure 6, to implement the "don't care" function. HSPICE simulation shows that we can easily choose appropriate parameters of the transistors in the SRAM, CAM, and XCAM cells to make the comparison of the programmable decoder runs faster than that of the 6-to-64 decoder.

To avoid faulty cache hits, tag length must be extended to include the subarray index. In a four-subarray direct-mapped cache, the proposed cache tag is two bits longer than that of the original tag. This will increase the time spent on activating the word line, bit lines of tags, and the time needed to compare the tags. We collect access times for both the tag and the data side for our *efficient cache* with four subarrays at cache size of 4Kbyte, 8Kbyte, and 16Kbyte, using CACTI model at technology 0.18µm. Table 1 shows the cache access times. From the table, we can see that the data side access time is still longer than that of the tag side after we extend the length of the tag by two bits.

4.3 Power per cache access

The extra power consumption comes from the fact that we use eight rows of five-bit long programmable decoders to replace the

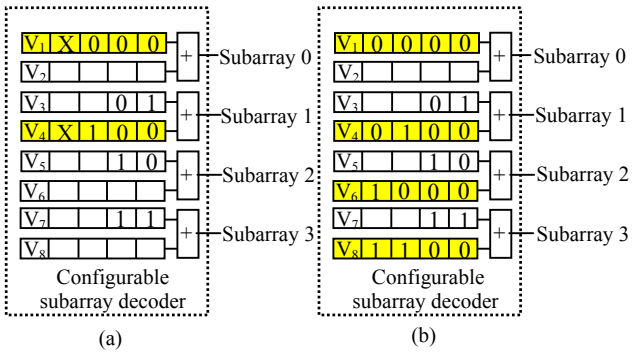

Figure 3:Two examples of the configurable subarray decoder.

original four AND gates decoders. We measure the power consumption of the programmable decoders using Cadence layout tools at technology 0.18um. The power consumption per access of the decoders is 0.70pJ. The total power consumption of an 8 Kbyte direct mapped cache is 0.38nJ per cache access from our own cache layout. Therefore, the power overhead due to replacing the original decoder with programmable decoders is 0.70pJ/0.38nJ = 0.18%.

5. Using an Efficient Instruction Cache

The proposed efficient cache is intended to be used in a pre-fabricated platform that has already been designed but is intended for use in a variety of possible applications. To perform efficiently for the largest variety of applications, recent platforms come with parameterized architectures that a designer can configure for his/her particular set of applications.

Figure 4 shows the procedure of designing an efficient instruction cache. First, we need a simulation tool to profile the application and an algorithm to search for the optimal configurable subarray decoders. The usage of the cache sets is recorded from the simulation tool. Then the number and location of the *overused* and *underused* cache sets are fed to the algorithm to set up the configurable decoder that produces the desired index mapping cache structure.

First, we need to determine both the *overused* and *under used* subarrays. Let SA_0, SA_1, SA_2, and SA_3 denote the accesses to subarray *0,1,2,* and *3*, respectively. *Ave* represents the average access to the subarrays. We define a subarray to be *overused* when $SA_i / Ave > threshold_1$, and a subarray to be *underused* when $SA_i / Ave < threshold_2$, where $i = 0,1,2,3$. We determine the two thresholds through experiments and find out that $threshold_1 = 1.3$ *and* $threshold_2 = 0.8$ are appropriate for the benchmarks we simulated. We point out that these two thresholds just tell us whether there exist chances to balance the subarray accesses; therefore, we tend to select a small value of $threshold_1$ and large value of $threshold_2$.

Last, we need an algorithm to search for the optimal configurable subarray decoder. There are two rows of decoders for a subarray with each decoder has four bits but only the two XCAM bits are configurable for the first row. The possible configurations of the two XCAM bits in Figure 2, defined as decoder configuration (DC) are: DC = {*xx, x0, 0x, x1, 1x, 00,01,10, 11*}. *xx* means the decoder remain unchanged, *x0, 0x, x1*, and *1x* means the address space mapped to the overused subarray decoders is reduced by half, while *00,01,10*,and *11* means that only one fourth of the original address space is mapped to the original overused

cache size (KB)	original tag side access time(ns)	tag side access time when two bits longer(ns)	original data side access time (ns)	access time change
4	0.83	0.85	0.87	***NO***
8	0.86	0.87	0.96	***NO***
16	0.93	0.95	1.13	***NO***

Table 1: Access Time of our proposed cache.

subarray and three fourths of the address space to the overused subarray is remapped to underused subarray.

Figure 5 shows the algorithm we use to determine the optimal subarray decoder configuration. The inputs include both the *overused, OS,* and *underused* subarrays, *US*. The set of *OS* may be empty, no overused subarrays, or contain at most three subarrays, so does the *US*. The outputs are the XCAM bits of the first-row decoder configuration (FDC_k) of the overused subarray and second row decoder configuration (SDC_k) of the underused subarray. The algorithm simply tries all possible decoder configurations and selects the configuration that has the lowest miss rate.

In terms of running time complexity, although the algorithm has four loops, the number of subarrays is just four (for cache size is less than or equal to 64Kbyte, which is larger than typical cache size of embedded systems [12]). There are nine possible decoder configurations of the first row decoder, FDC_k, and the second row decoder, SDC_k. However, the FDC_k and SDC_k are dependent of each other. For example, assume subarray 0 is overused and subarray 3 is underused, then we can only reduce the access to subarray *0* by half. The configuration of the first row decoder can be any of the four possibilities: *0x, x0, 1x, x1*, correspondingly, the second row decoder of subarray 3 must be *1x, x1, 0x, x0*, respectively. In other words, if the first row decoder is *0x*, then the second row decoder of subarray 3 must be *1x*. For one of the worst case, where there are one overused subarray but three under used subarrays, the total simulations is 3x4 = 12 when reducing the access by half, and 4! = 24 when reducing the access by three fourths. Therefore, the total simulation we need to run is 36. It takes around six hours on a Pentium IV machine using SimpleScalar[2] simulation tool to search for the optimal configurable decoder.

6. Experiments

To determine the benefits of our efficient cache, we simulated a variety of benchmarks for a variety of cache sizes using SimpleScalar [2]. The benchmarks included programs from MediaBench [7] (*adpcm_enc, adpcm_dec, epic_enc, epic_dec, jpeg_enc, jpeg_dec, mpeg2, pegwit_enc, pegwit_dec*). We used the data sets that came with each benchmark as program stimuli.

6.1 Results

We show, in Figure 7 and Figure 9, the reduction of instruction miss rate with respect to a 4 Kbyteyte and 8 K byte direct-mapped cache for all the benchmarks, respectively. 4K-E and 8K-E represent the instruction cache miss rate our efficient cache at cache size of 4 Kbyteyte and 8 Kbyteyte, respectively. 4K-2W and 8K-2W represent the miss rate of two way set associative cache at size of 4 Kbyteyte and 8 Kbyteyte, respectively.

6.2 Observations

From Figure 7 and Figure 9, we can observe that the miss rate reduction of our efficient cache is almost as good as that of a

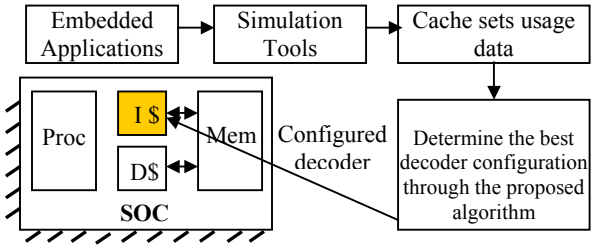

Figure 4: Design procedure for an efficient cache.

Algorithm

Input: overused subarrays: $os_i \in OS$, $0 \le i \le 2$

Input: underused subarrays: $us_j \in US$, $0 \le j \le 2$

Output: decoder configuration FDC_k and SDC_k, *k=0,1,2,3*

initialize both FDC_k and SDC_k to *xx*

for each $os_i \in OS$

for each $FDC_k \in DC$

for each $us_i \in US$

for each $SDC_k \in DC$

calculate miss rate through simulation

select the FDC_k and SDC_k with the smallest miss rate

Figure 5: Algorithm to determine the optimal subarray decoder configuration

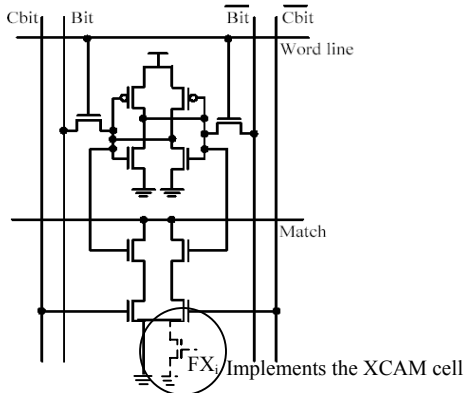

Figure 6: CAM cell circuitry. One extra NMOS is used to implement "don't care" CAM cell. Don't care means that the "Match" is always high.

conventional two-way set associative cache on average at cache size of 4 Kbyteyte and 8 Kbyteyte.

Table 2 lists the contents of the configurable decoder for all the benchmarks we simulated at cache size of 4Kbyteyte. The result for cache size at 8Kbyteyte is not shown due to space limit.

6.3 Energy Evaluation

We consider both dynamic and static energy consumption of in our energy evaluations. We used similar energy evaluations as shown in [12]. Our equation for computing the total energy due to memory accesses is as follows:

$$energy\_mem = energy\_dynamic + energy\_static$$

$$where: \quad energy\_dynamic = \underline{cache\_hits} * \underline{energy\_hit} + \underline{cache\_misses} * energy\_miss$$

$$energy\_miss = energy\_offchip\_access + energy\_uP\_stall + energy\_cache\_block\_refill$$

$$energy\_static = cycles * energy\_static\_per\_cycle$$

We obtain the underlined terms through measurements or simulations. We compute *cache_hits*, *cache_misses*, *and cycles* by running SimpleScalar [2] simulations. We compute *energy_hit* and *energy_cache_block_refill* using the CACTI [9] model. The *energy_offchip_access* value is the energy due to accessing off-chip memory and the *energy_uP_stall* is the energy consumed when the microprocessor is stalled to wait for the memory system to provide an instruction. *energy_cache_block_fill* is the energy for refilling a cache block due to a cache miss. These terms highly depend on particular memory and microprocessor. Instead of evaluating the energy to a particular microprocessor system and off chip memory configuration, we try to figure out a method that may be used for different microprocessors and off chip memory configurations. We examined the three terms of *energy_offchip_access*, *energy_uP_stall*, and *energy_cache_block_fill* for typical commercial memories and microprocessors, and found that *energy_miss* ranged from 50 to 200 times bigger than *energy_hit*. Thus, we redefined *energy_miss* as:

$$energy\_miss = k\_miss\_energy * energy\_hit$$

we considered the situations of *k_miss_energy* equal to 50 and 200.

Finally, *cycles* is the total number of cycles for the benchmark to execute, as computed by SimpleScalar, using a cache with single cycle access on a hit and using 100 cycles on a miss. *energy_static_per_cycle* is the total static energy consumed per cycle. This value is also highly system dependent, so we again consider a variety of possibilities, by defining this value as a percentage of total energy including both dynamic and static energy:

$$energy\_static\_per\_cycle = k\_static * energy\_total$$

k_static is a percentage that we can set. To consider the CMOS technology trend, we evaluate the situations where *k_static* is 30% and 50% of the total energy.

6.4 Energy savings

We show the energy savings of all benchmarks for cache sizes ranging from 4 Kbyteyte to 8 Kbyteyte when *k_static=50%, k_miss_energy=200* in Figure 8 and Figure 10**.** Energy data at other combinations of *k_miss_energy* and *k_static* are not shown due to space limit. The energy savings are on average 12% and 8% compared with a same sized conventional direct mapped cache. The negative value stands for that the efficient cache consumes less energy than that of a conventional direct mapped cache. Compared with the conventional two-way set associative caches, which have almost the same miss rate as our efficient cache, the energy consumption of the two-way set associative cache is on average 20% and 15% more than a direct mapped cache.

Therefore, our efficient cache can achieve almost the same miss rate but consume 30% and 23% less energy than a conventional two-way set associative cache at cache size of 4 Kbyte and 8 Kbyte, respectively.

Compared with conventional direct mapped caches, the energy savings of the efficient cache come form the fact that accesses to off chip buses and memory are reduced due to the reduction of the miss rate. Compared with two way set associative cache, the efficient cache has almost the same miss rate hence consumes comparable off chip energy. However, the efficient

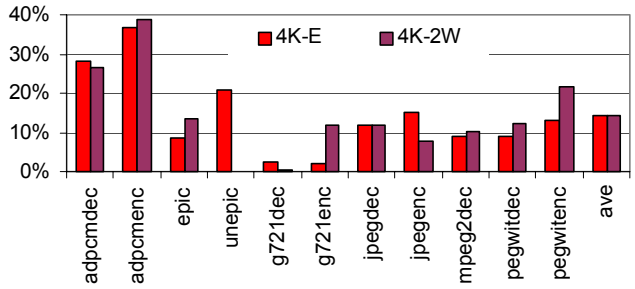

Figure 7: Miss rate reductions of Mediabench benchmarks at cache size of 4 Kbyteyte compared with same size direct mapped cache.

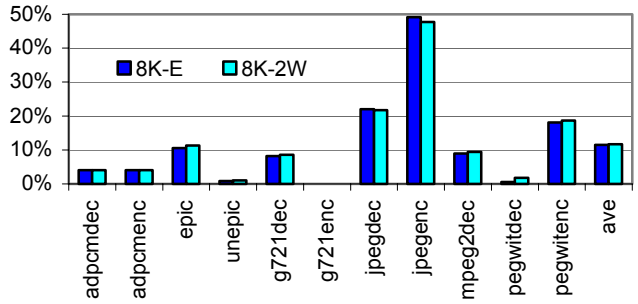

Figure 8: Miss rate reductions of Mediabench benchmarks at cache size 8 Kbyteyte compared with same size direct mapped cache.

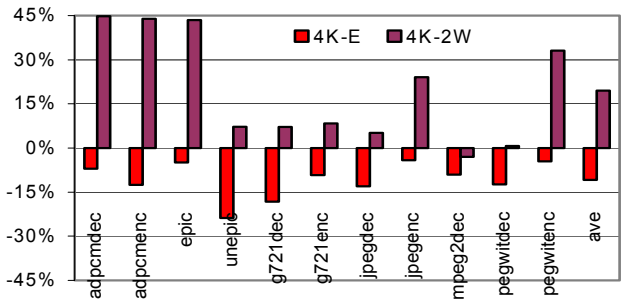

Figure 9:Energy savings of the efficient cache at cache size 4Kbyteyte compared with direct mapped cache respectively.

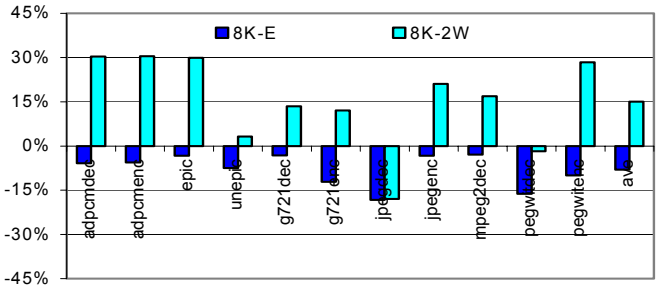

Figure 10: Energy savings of the efficient cache at cache size 8Kbyteyte compared with direct mapped cache.

cache consumes much less energy per cache access than that of a two way set associative cache. Therefore, the total energy is reduced.

7. Limitations of the efficient cache

The design of the efficient cache requires simulating the applications beforehand to determine what is the best index decoding schemes that can be implemented by the configurable decoder. However, different input data sets may change the distribution of both *overused* and *underused* cache sets. The efficient cache may experience a worse miss rate if the best index decoding changes, causing both performance and power overhead. If the input data sets changes are not frequent, we may choose a set of best decoding schemes for each input data sets and store the best decoding schemes in a special memory buffer, therefore the best decoding is used for different input data sets. On the other hand, we may use the original decoding scheme for applications whose best decoding highly depends on input data sets.

To determine the prevalence of this situation, we simulated all the benchmarks using a secondary input data set that comes with Mediabench. We haven't found that the best index decoding has been changed due to the change of the input data sets for the benchmarks we simulated.

8. Conclusion

We have designed an efficient instruction cache for application-specific embedded systems. We observe that there are cache sets that are overused by some applications during execution and generate conflict misses. In addition, some other cache sets are underused that the miss rate can be reduced if the underused cache space is used efficiently. We proposed to re-map the cache memory reference from the *overused subarrays* to *underused subarrays* for a particular application. We showed that an efficient cache achieves almost the same hit rate of a conventional same sized two-way set associative cache while consumes 25% less energy.

	subarray0	subarray1	subarray2	subarray3
adpcmdec	1x000	1x001	0xx10	0xx11
	0xxxx	0xxxx	1x100	1x101
adpcmenc	1x000	0xx01	0xx10	0xx11
	0xxxx	0xxxx	1x100	0xxx
epic	10000	0xx01	0xx10	0xx11
	0xxxx	10100	11000	11100
unepic	1x000	1x001	0xx10	0xx11
	0xxxx	0xxxx	1x101	1x100
g721dec	0xx00	0xx01	1x010	1x011
	0xxxx	1x1x1	0xxxx	0xxxx
g721enc	0xx00	0xx01	1x010	0xx11
	0xxxx	1x101	0xxxx	0xxxx
jpegdec	1x000	0xx01	0xx10	0xx11
	0xxxx	1x100	0xxxx	0xxxx
jpegenc	0xx00	0xx01	0xx10	1x011
	0xxxx	0xxxx	1x111	0xxxx
mpeg2dec	1x000	1x001	0xx10	0xx11
	0xxxx	0xxxx	1x101	1x100
pegwitdec	1x000	0xx01	0xx10	0xx11
	0xxxx	0xxxx	0xxxx	1x100
pegwitenc	0xx00	0xx01	1x010	1x011
	1x1x1	0xxxx	0xxxx	0xxxx

Table 2: The configurable decoder contents of the efficient cache at 4 Kbyteyte.

Reference

[1] D.H. Albonesi, "Selective Cache Ways: On-Demand Cache Resource Allocation," Journal of Instruction Level Parallelism, May 2000.

[2] D. Burger and T.M. Austin, "The SimpleScalar Tool Set, Version 2.0," Univ. of Wisconsin-Madison Computer Sciences Dept. Technical Report #1342, June 1997.

[3] T. Givargis, "Improved Indexing for Cache Miss Reduction in Embedded Systems," Design Automation Conference, Anaheim CA, 2003.

[4] S. Kaxiras, Z. Hu, and M. Martonosi, "Cache Decay: Exploiting General Behavior to Reduce Cache Leakage Power," International Symposium on Computer Architecture, 2001.

[5] J. Peir, Y. Lee, W. Hsu, "Capturing Dynamic Memory Reference Behavior with Adaptive Cache Topology," in the Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems, 1998.

[6] L. Lee, B. Moyer, J. Arends, "Instruction Fetch Energy Reduction Using Loop Caches For Embedded Applications with Small Tight Loops," International Symposium on Low Power Electronics and Design, 1999.

[7] C. Lee, M. Potkonjak and W. Mangione-Smith, "MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems," International Symposium on Microarchitecture, 1997.

[8] A. Malik, B. Moyer and D. Cermak, "A Low Power Unified Cache Architecture Providing Power and Performance Flexibility," International Symposium on Low Power Electronics and Design, June 2000.

[9] G. Reinmann and N.P. Jouppi. CACTI2.0: An Integrated Cache Timing and Power Model, 1999. COMPAQ Western Research Lab.

[10] S. Segars, "Low power design techniques for microprocessors," International Solid-State Circuits Conference Tutorial, 2001.

[11] Tensilica Inc. http://www.tensilica.com/.

[12] C. Zhang, F. Vahid, and W. Najjar, "A Highly-Configurable Cache Architecture for Embedded Systems," International Symposium on Computer Architecture, 2003.