A Capacity Co-allocation Configurable Cache for Low Power Embedded Systems

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

Abstract — **Traditional level-one instruction caches and data caches for embedded systems typically have the same capacities. Configurable caches either shut down a part of the cache to suit applications needing a small cache or employ a large cache and high associativity for applications needing to reduce miss rate and energy. However, increasing associativity is energy-costly compared with increasing capacity.**

We have extended the traditional configurable cache and made the whole on-chip cache memory capacity available to both instruction and data caches. The capacity can then be co-allocated between the data and the instruction caches. Compared with way shutdown and way concatenation, the capacity co-allocation cache provides a better solution than increasing associativity. Four out of 17 benchmarks from Mibench benefit from the capacity co-allocation cache. Energy reduction can be up to 24%, with an average of 5%, compared to a traditional configurable cache.

1 Introduction

Reducing power consumption of embedded processors is an increasingly important task for battery-powered embedded computing systems. Cache memories consume about 40% [8][16] of the total power in these systems. Power efficient cache architecture is a critical issue in the design of microprocessors for embedded computing systems.

Accessing off-chip memory is time consuming and not energy efficient due to high driving capacitance and the need for large amounts of memory storage. Reducing the miss rate of level-one caches for embedded systems can reduce the number of accesses to off-chip memory and greatly reduce power consumption. Processors for embedded computing systems typically do not have a level-two cache, which is widely available for high performance processors.

Cache size is important in determining the miss rate and power consumption. Large sized caches reduce miss rate and accesses to off-chip memory and buses but consume higher-per-cache access energy. Small sized caches have a low-per-access energy but may exhibit high miss rate and increase accesses to off-chip buses and memory. Therefore, a small sized cache may or may not result in the least overall energy consumption for a particular application.

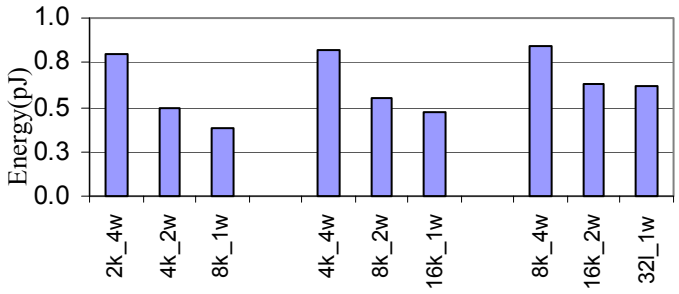

Figure 1: Energy consumption per cache access at varied cache sizes and associativities. *k* stands for kB and w stands for way, e.g. 2k_4w means a 2kB cache with associativity of 2-way.

Since an embedded system typically executes just one or several fixed sets of applications for the system's lifetime, several configurable cache architectures have been proposed to tune the cache to those applications. Cache parameters, such as cache size [1], associativity [16], line size [17], replacement policy, and buffers [11], can be set to suit a particular application to achieve overall good performance and consume less energy.

Embedded systems use the same sized separate data and instruction cache, even though the requirements for data and instructions are totally different. Some applications require a large data cache while others may require a large instruction cache. To accommodate more applications, cache capacities are typically designed as large as possible. To exploit this cache design, several low power cache schemes, such as way shut down, drowsy cache [6], and cache decay [9], have been proposed to reduce both dynamic and static energy for a particular application through shutting down or putting a part of the cache memory into a low power drowsy or decay state.

Some applications, however, may need a larger cache than is available. Way concatenation [16] can be used to increase the associativity of caches, since high associativity can further reduce miss rate but with higher power cost.

Increasing cache capacity and associativity can reduce miss rate; however, the energy cost of increasing cache capacity and associativity is different. Figure 1 shows the energy consumption of caches at varied sizes and associativities [13]. An 8kB direct-mapped cache consumes less power than a 4kB cache at 2-way and 4-way associativities and exhibits a similar miss rate to a 4kB 2-way cache [18]. Therefore, increasing cache size should be employed first before increasing associativity [18].

Based on the above observation, we propose to extend the configurable cache, so the capacity of cache memories can be co-allocated between data and instruction caches. We call it *Capacity Co-allocation Configurable Cache*. For example, one application may need only a small data cache but an instruction cache larger than the maximum available. We can allocate the memory space from the data cache to the instruction cache without shutting down part of the data cache. We do not need to increase associativity of the instruction cache and can achieve lower overall energy consumption. This is doable, since in an embedded system, data and instruction caches are laid side by side [14]. This is different than in high performance processors where data and instruction cache are

set apart, making cache capacity co-allocation difficult [10].

The rest of this paper is organized as follows. In Section 2, we review the related work. Section 3 describes the base cache architecture. We describe the design of the capacity co-allocation cache architecture in Section 4. Experimental results are presented in Section 5. We analyze the capacity co-allocation cache in Section 6 and conclude in Section 7.

2 Related Work

Both cache size and associativity can be reconfigured to suit varied applications in a highly configurable cache [16]. The reconfigurations of the instruction and the data caches are done independently and the size of both the instruction and the data caches can only be reduced from their maximal sizes. When one application needs a larger cache size than the maximum available in the configurable cache, associativity has to be increased to reduce the miss rate. Zhang, et.al, shows that increasing cache size is more efficient in terms of energy reduction than increasing associativity [18].

The way shut-down technique [1] reduces the cache capacity through shutting down the cache ways. These techniques are useful when extra cache capacity is available on chip. Our technique provides a better scheme so that the cache capacity can be co-allocated between the data and the instruction caches. There are more cache configurations in the proposed cache. For example, an existing highly configurable cache [16] with an 8kB data and instruction cache has a total of six possible configurations each, as shown in Table 1. However, the proposed cache can have 12 possible configurations for each cache. Some configurations, such as a data cache size of 12 kB, is possible only when the instruction cache size is less than or equal to 4kB.

Drowsy-cache [6] and cache-decay [9] adaptively put cache sets into a drowsy or decay state, so leakage can be reduced without significantly sacrificing performance. This means the cache capacity has not been used efficiently.

In semi-unified cache [4], the level-one data and instruction cache can be used as a backup of each other. Both the instruction and the data addresses are fed to the cache memory. When both the instruction and the data are directed to the same cache memory, a conflict occurs that cause one activity to wait for extra cycles to proceed. The proposed cache can only be used for either instruction or data, so the extra cycles can be avoided.

The CAM-based Highly Associative Cache (HAC) [5][13] is specifically designed for low power embedded systems. The CAM-based HAC reduces energy consumption through two main organizational techniques among many others. One method is to aggressively partition the cache memory into small subbanks. The small size of the subbanks reduces the energy per cache access. The other is high associativity; typically a 32-way cache is implemented on one subbank. High associativity greatly reduces the miss rate and the accesses to the off chip buses and memory. Compared with the HAC, the proposed cache does not require CAM.

Table 1: Possible Cache Configurations.

	Highly Configurable Cache					
	D$ (6 possible configurations)			I$ (6 possible configurable)		
Size (kB)	2	4	8	2	4	8
Assoc.	1	1,2	1,2,4	1	1,2	1,2,4

	Capacity Co-allocation Configurable Cache									
	D$ (12 possible configurations)					I$ (12 possible configurations)				
Size(kB)	2	4	8	12	14	2	4	8	12	14
Assoc.	1	1,2	1,2,4	1,2,4	1,2,4	1	1,2	1,2,4	1,2,4	1,2,4

3 Base Configurable Cache Organization

The proposed capacity co-allocation of level one cache is based on the highly reconfigurable cache proposed in [16], whose organization is shown in Figure 2 (only two ways are drawn to save space). The baseline uses a direct mapped cache of 8kB and a line size of 32 bytes.

The baseline exploits two techniques, namely way concatenation and way shut down. Way concatenation can be used to reconfigure the cache associativity, e.g., the baseline can be configured as direct-mapped, two-way, or four-way set associative cache. Way shutdown is used to reconfigure the cache size, which can be of 8, 4, and 2 kB. The cache size, however, cannot be larger than 8 kB, even though one of the caches, (such as data cache) may need only 2 kB. The maximum size of the instruction cannot be increased larger than 8kB. The capacity of the total cache cannot be used efficiently. The proposed cache can avoid this situation, and the cache capacity can be fully exploited to efficiently reduce the total energy consumption. We will extend the baseline to co-allocate capacity between data and instruction caches and to which we will compare our results.

The following is a brief explanation of the operation of the baseline. Details about the baseline can be found in [16]. *F0*

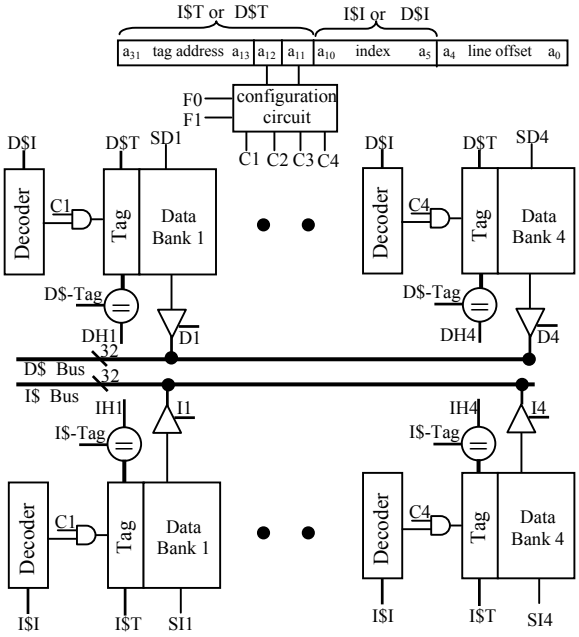

Figure 2: Organization of the baseline highly configurable cache proposed in [16].

and *F1* are two single-bit registers that can be set to configure the cache as a four, two or one way set-associative cache (the instruction and data cache have their own control circuits). Those two bits are combined with address bits *a11* and *a12* in a configuration circuit to generate four signals (*c1*, *c2*, *c3*, *c4)*, which are in turn used to control the configuration of the four ways. For example, when *F0*=0 and *F1*=0, the cache acts as a one-way cache (where that one way is four times bigger than the four-way case). Address bits a_{11} and a_{12} are decoded in the configuration circuit such that exactly one of *c1*, *c2*, *c3*, or *c4* will be logic 1 for a given address. Only one of the tag arrays and one of the data arrays are activated for a given address. Likewise, only one of the tag comparators is activated. When *F0*=1 and *F1*=1, the baseline is a four-way cache, while *F0*=1 and *F1*=0 or *F0*=0 and *F1*=1 is a two-way cache.

SD1-SD4 and *SI1-SI4* are eight single-bit registers that can be used to shut down cache ways to reduce the cache capacity. It should be noted that six, seven, and eight index bits are used for a four-way, a two-way, and a one-way cache, respectively. Also note the total cache capacity does not change when configuring the cache for four, two or one way.

4 Capacity Co-allocation for Energy Reduction

Each program has different cache capacity needs. We sought to extend the base architecture so memory could be co-allocated to take advantage of the whole capacity on the processor. The cache size can be 2, 4, 8, 12, and 14kB. Compared with the original configurable cache, there are two more possible configurations, which are 12 kB and 14 kB.

In the original configurable cache, when a capacity larger than 8kB is required, e.g. 14 kB, we have to use set associative caches, such as an 8kB 2-way cache to replace the 14Kb cache. This option, however, is power costly as shown in Figure 1. We sought to develop a cache architecture whose capacity could be co-allocated, and a particular cache size can be larger than the size of a traditional configurable cache. The primary concept is to allow cache subbanks to be co-allocated. The hardware required to support this concept is rather simple.

The proposed cache is shown in Figure 3. The co-allocation of cache capacity is based on the subbank of each cache memory. Cache memory is divided into subbanks to balance the access time, power consumption, and area [13]. For the baseline of an 8kB cache, the memory is divided into four subbanks with one subbank to be 2kB. Therefore, the co-allocation of the cache capacity is at the granularity of one subbank, which means the minimum size for a data/instruction cache is 2kB, while the maximum is 14kB.

Figure 3 shows the organization of the proposed cache. The proposed cache may have at most three subbanks of the data/instruction cache be co-allocated to instruction/data. These co-allocatable subbanks may take either the instruction address or the data address. The output of these subbanks may go to either the instruction bus or the data bus, but not both at the same time.

We will add a multiplexer in front of the cache decoders to select between two addresses and another multiplexer in front of the tag comparison. Similarly, the output of the co-allocatable subbanks can now reach both buses, depending on

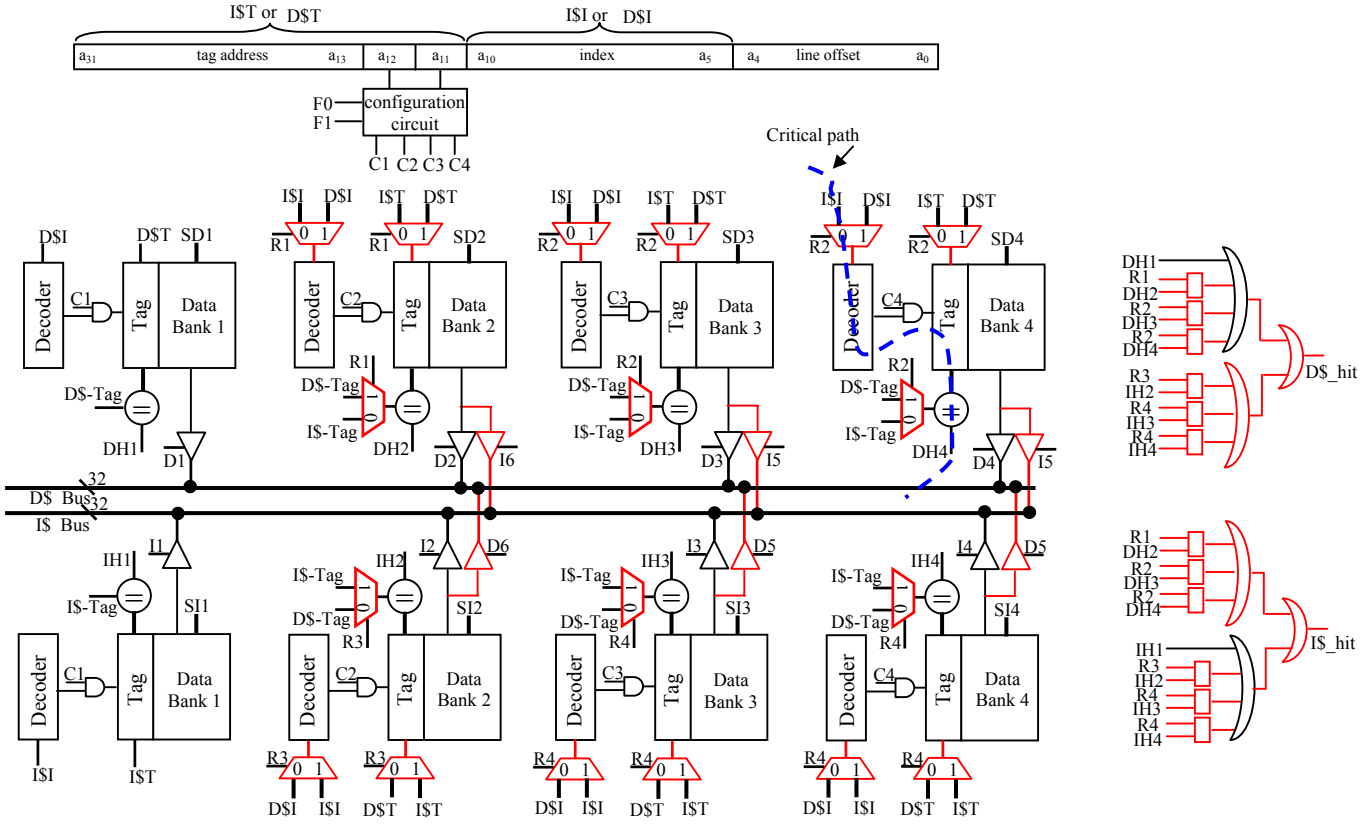

Figure 3: The organization of the capacity co-allocation configurable cache. The instruction and data caches have their own configuration control signals, C0, C1, C2, and C4.

the current subbank configuration. These changes are depicted in Figure 3 (extra logics are in red).

Single bit registers *R1*, *R2*, *R3*, and *R4* are used to control the capacity co-allocation. When *R1*=1, *R2*=1, *R3*=1, and *R4*=1, the subbanks are not co-allocated between the data and instruction cache. When *R1*=1, *R2*=0, *R3*=1, and *R4*=1, the data cache only has two subbanks. This means 4kB, while the subbanks 3 and 4 are allocated to the instruction cache, which means a 12kB. Combining the registers of *F0* and *F1*, the cache can be configured as varied sizes and associativities.

5 Experiments

5.1 Experimental Methodology

To determine the benefit of the capacity co-allocation configurable cache, we simulated 17 benchmarks selected from Mibench [7] (other benchmarks are not included due to compilation errors). We configure the SimpleScalar [2] as a single-issue in-order processor. We collect the information we needed to evaluate the energy consumption in the equation as shown in Figure 6. These include total cache accesses, execution time, and misses. The baseline does not have a level-two cache.

Through experiments, we will show that applications have varied cache capacity requirements for data and instruction. We determine the best configuration by examining all the possible configurations and pick the configuration that consumes the least energy. Energy evaluation is discussed in Section 6.2.

5.2 Experimental Results

Figure **4** shows the miss rate of both the data and instruction caches of the benchmarks simulated. The bar with "*1*" represents the miss rate of a traditional 8kB direct mapped cache of both data and instruction. While the other bars with number "2","3","4", or "5" represents the proposed capacity co-allocation cache with an instruction cache size at 2, 4, 12, and 14kB and a data cache size at 14, 12, 4, or 2 kB. All of these cache configurations are direct mapped caches.

The first observation is that some benchmarks, such as *adpcm_dec, adpcm_enc, crc,* and *susan_smooth*, a 2kB instruction cache, is enough. Increasing instruction cache capacity larger than 2kB would not bring any meaningful miss rate reduction. However, the data cache miss rate, can be significantly reduced if the memory capacity from the instruction cache can be allocated to data cache, e.g. the data cache miss rate of benchmark *adpcm_dec* and *adpcm_enc* can be reduced from 0.7% to 0.05%. However, this reduction may or may not bring meaningful energy reductions, which is discussed in Section 6.2.

The second observation is that other benchmarks, such as *bitcount,* the data cache miss rate remains unchanged when cache size is larger than 2kB. The miss rate of instruction cache achieves the lowest at cache size of 12 kB. For other benchmarks, the trade offs between the miss rate of instruction and data caches are not straightforward. For example, benchmarks *FFT* and *ghostscrip*, increasing cache size would always decrease miss rate. The best capacity allocation has to be determined through considering both performance and total energy consumption.

6 Analysis

6.1 Timing and Area Analysis

The timing of the capacity co-allocation cache is prolonged due to the extra logics on the critical path shown in Figure 3,

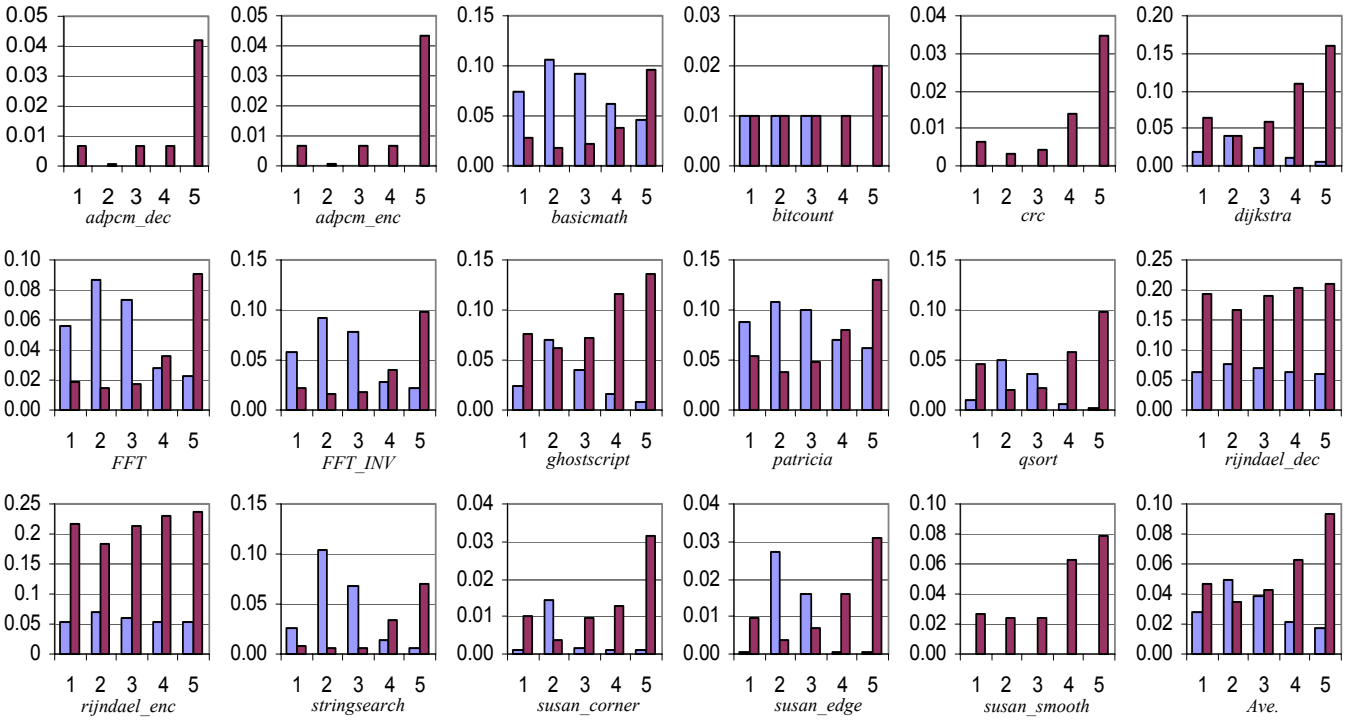

Figure **4**: Miss rates of instruction cache (left bar) and data cache (right bar) of the benchmarks simulated. In the horizontal axis, "***1"*** stands for the baseline, the size of both the instruction and data caches, which are 8kB. "***2***","***3***","***4***", or "***5***" stands for the capacity co-allocation cache with instruction cache sizes at two, four, 12, or 14kB and data cache sizes at 14, 12, four, or 2kB.

including the multiplexer to the address decoder and the extra AND and OR gates in the hit signal. We evaluate the time delay through our own layout of the configurable cache using Cadence [3] tools at 0.18 um technology. We simulated the circuits extracted from the layout using HSPICE. The dimension of our SRAM cell was 2.4µm x 4.8µm, using conventional six-transistor cells. We measured the energy of the various parts of a conventional and configurable cache at capacities of two, four, eight, 12, and 14 kB, and corresponding associativity at 1-way, 2-way, and 4-way during a cache access.

The extra delay incurred through extra logic is measured at less than 3% of the original access time. The delay of the extra logic can be tuned through carefully selecting the parameters of transistors of the multiplexer, the AND, and the OR gates. The trade off is to consider the area, time, and energy consumption.

The proposed capacity co-allocation cache aims at embedded computing systems. The cache configuration is determined through off-line simulation. It will be fixed for the lifetime of the embedded system instead of being tuned dynamically, although the cache architecture itself enables this tuning.

The extra area comes from the extra multiplexers, the AND, and the OR gates to control the hit/miss detection. The area of the control logic is very limited compared with the memory and is less than 1% of the total cache area.

6.2 Energy Savings

Energy is proportional to both power and time. Therefore, to reduce energy, we should try to reduce both of them. There are two main components that result in power dissipation in CMOS circuits: static power dissipation due to leakage current and dynamic power dissipation due to logic switching current and the charging and discharging of the load capacitance. To compare with the highly configurable cache, we consider both types of energy and use the same method as in [16] to evaluate the energy consumption. Figure 6 shows the equations we use to evaluate the energy.

We obtain the underlined terms through measurements or simulations. We measure *$_hits* and *$_misses* by running SimpleScalar [2] simulations for each cache configuration. We compute *energy_hit* of each cache configuration through the simulation of circuits extracted from our cache layout.

We use the same method to determine the *E_miss*. The *E_offchip_access* value is the energy of accessing off-chip memory, and the *E_uP_stall* is the energy consumed when the microprocessor is stalled while waiting for the memory system

$$E\_dynamic = \$\_hits * E\_hit + \$\_misses * E\_miss$$
$$E\_miss = E\_offchip\_access + E\_uP\_stall + E\_\$\_block\_fill$$
$$E\_static = cycles * E\_static\_per\_cycle$$
$$E\_mem = \$\_access * E\_\$\_access + \$\_miss * E\_misses$$
$$E\_misses = E\_next\_level\_mem + E\_\$\_block\_refill$$
$$energy\_static\_per\_cycle = k\_static * energy\_total$$
$$E\_mem = E\_dynamic + E\_static$$

Figure 6: Equations used to evaluate energy consumption. "$" stands for cache.

to provide an instruction or data. *E_cache_block_fill* is the energy for writing a block into the cache. We considered the situations of *k_miss_energy* equal to 200 in our evaluation.

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, 100 cycles on a miss. *E_static_per_cycle* is the total static energy consumed per cycle and evaluated as in [16].

The purpose of the capacity co-allocation is to select the cache configuration so the embedded system dissipates the lowest energy. We analyze the energy dissipation in this section, since the energy consumption is highly related to the execution time, e.g., the static energy is proportional to the execution time. We have to consider that the access time of the proposed capacity co-allocation cache is prolonged by 3%. The cache access is typically on the critical path of a processor, so the capacity co-allocation cache may have a 3% slower clock frequency.

We computed the total energy consumption of all the benchmarks we simulated at all possible cache configurations. These included the traditional configurable cache and the proposed capacity co-allocation cache. There are 66 possible configurations, and it is difficult to include all possible results in one figure. Therefore, we show the energy consumption of the instruction and data cache separately, computed from equations in Figure 6. Figure 5 shows the energy consumption of benchmark *basicmath,* at all possible instruction and data cache configurations. The total energy consumptions of a particular configuration will be the summation of corresponding instruction and data cache energy. This is true, since we target the capacity co-allocation cache to single issue in-order processors. In other words, the number of cache accesses, cache hits, and cache misses of the instruction/data cache is not dependent on the parameters of the data/instruction cache. The energy dissipated on the instruction cache is much higher than the data cache, since the instruction cache is accessed more frequently than the data cache.

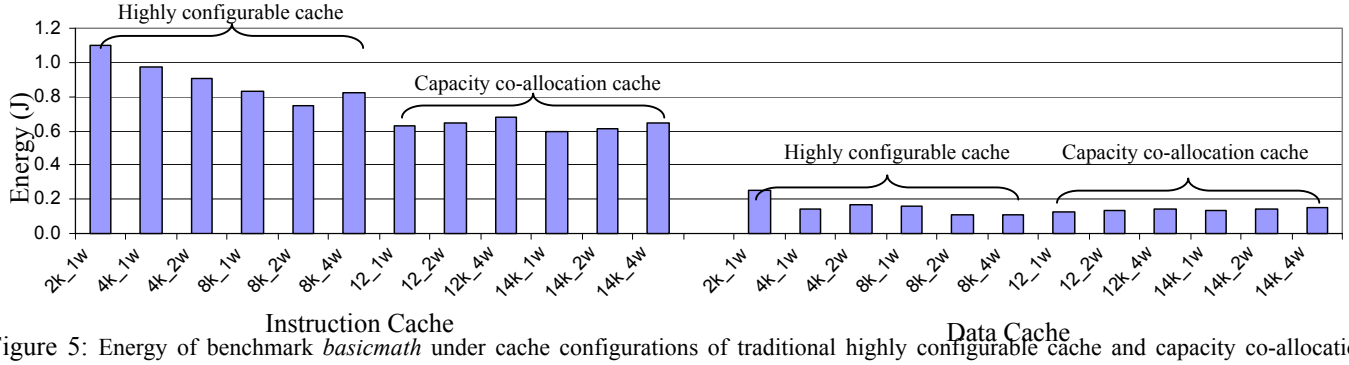

Figure 5: Energy of benchmark *basicmath* under cache configurations of traditional highly configurable cache and capacity co-allocation configurable cache considering both dynamic and static energy.

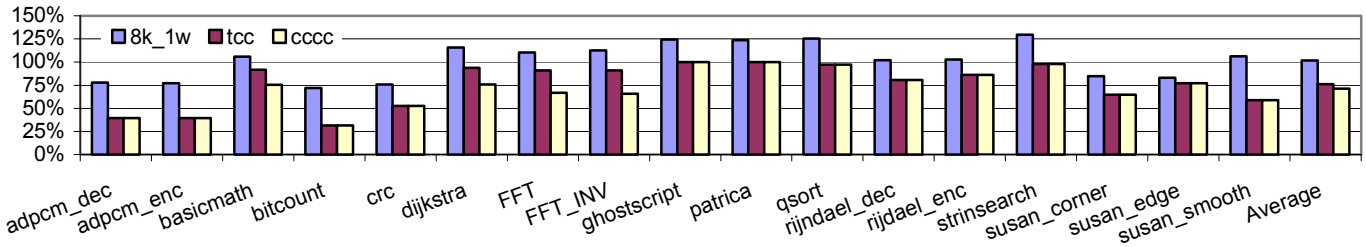

Figure 7: Energy consumption of the traditional configurable cache (tcc), the proposed capacity co-allocation configurable cache (cccc) and a conventional direct mapped cache compared to an 8 kB four-way cache with *k_miss_energy* =200.

From Figure 5, we can see that the instruction cache consumes the lowest energy at 14kB with 1-way and a data cache at 8kB with 2-way. However, this is not a viable combination, since the total capacity is larger than 16kB. For a traditional configurable cache, the lowest energy cache configuration should be an instruction cache at 8Kb 2-way and a data cache at 8kB 2-way. The total energy consumption is 0.75(E_I$) + 0.11(E_D$) = 0.86(J). With capacity co-allocation, we can choose an instruction cache at 12kB 1-way and a data cache at 4kB 1-way. The total energy consumption is: 0.61+0.13 = 0.74(J), representing 0.12/0.86 = 14% extra energy reduction over the traditional configurable cache. Without capacity co-allocation, the instruction cache has to use a 2-way set associative cache. By using capacity co-allocation, the capacity allocated to data cache is reduced; we have to use a 4kB direct mapped cache instead of the 8kB 2-way cache. However, the energy consumption of the data cache is not noticeably higher, so we can still achieve a lower energy consumption using capacity co-allocation.

Figure 7 shows the energy consumption for all the benchmarks of a conventional direct mapped cache at 8kB (*8k_1W*), the traditional configurable cache (*tcc*), and the proposed capacity co-allocation configurable cache (*cccc*) normalized to a conventional 4-way set associative cache (represented as 100%). On average, the capacity co-allocation cache achieves the lowest energy consumption, which is 30% lower than a conventional 4-way cache. Among 17 of the benchmarks simulated, four of them consumed less energy due to the capacity co-allocation and avoided using a set associative cache. Table 2 lists the best cache configurations in terms of energy of the traditional configurable cache and the capacity co-allocation configurable cache. From the table, we can see we can avoid using set associative caches for four benchmarks and reduce the energy consumption significantly.

Table 2: Cache configurations that yield lowest system energy, considering both dynamic and static energy at traditional configurable cache (*tcc_best*) and capacity co-allocation configurable cache (*cccc_best*).

Benchmark	tcc_best	cccc_best
adpcm_dec	I2K1WD4K1W	I2K1WD4K1W
adpcm_enc	I2K1WD4K1W	I2K1WD4K1W
basicmath	I8K2WD8K2W	I12K1WD41W
bitcount	I2K1WD2K1W	I2K1WD2K1W
crc	I2K1WD4K1W	I2K1WD4K1W
dijkstra	I8K2WD8K2W	I4K1WD12K1W
FFT	I8K2WD8K2W	I4K1WD12K1W
FFT_INV	I8K2WD8K2W	I12K1WD4K1W
ghostscript	I8K4WD8K4W	I8K4WD8K4W
patrica	I8K4WD8K4W	I8K4WD8K4W
qsort	I8K4WD8K2W	I8K4WD8K2W
rijndael_dec	I4K2WD2K1W	I4K2WD2K1W
rijdael_enc	I4K2WD2K1W	I4K2WD2K1W
strinsearch	I8K4WD8K1W	I8K4WD8K1W
susan_corner	I4K1WD4K2W	I4K1WD4K2W
susan_edge	I8K1WD4K2W	I8K1WD4K2W
susan_smooth	I2K1WD4K2W	I2K1WD4K2W

7 Conclusion

We have extended the traditional configurable cache so memory capacity can be co-allocated between the data and instruction caches. Compared with the traditional configurable cache, a capacity co-allocation configurable cache can achieve better performance and lower energy consumption, since we may allocate extra capacity in data/instruction cache to instruction/data, which might otherwise have to be shut down in a traditional configurable cache.

References

[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] Cadence Design Systems, http://www.cadence.com

[4] N. Drach, A. Seznec, "semi-unified caches," ICPP, pp. 25-28, 1993.

[5] A. Efthymio and J.Garside,"An Adaptive Serial-Parallel CAM Architecture for Low-Power Cache Blocks." In Proc. of ISLPED, 2002.

[6] K Flautner, NS Kim, S Martin, D Blaauw, T Mudge," Drowsy Caches: Simple Techniques for Reducing Leakage Power." ISCA, 2002.

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

[8] Intel. Intel XScale Microarchitecture, 2001.

[9] S Kaxiras, Z Hu, M Martonosi, "Cache decay: exploiting generational behavior to reduce cache leakage power." In Proceedings of ISCA, 2001.

[10] L. Jin, W. Wu, J. Yang, C. Zhang, and Y. Zhang, "Dynamic Co-allocation of Resources for Level One Caches," the 2nd International Conference on Embedded Software and Systems, Springer Verlag, December 2005.

[11] A. Malik, B. Moyer, and D. Cermak, "A low power unified cache architecture providing power and performance flexibility," ISLPED, 2000.

[12] M. Powell, S.H. Yang, B. Falsafi, K. Roy, and T.N. Vijaykumar, "Gated-Vdd: A Circuit Technique to Reduce Leakage in Deep-Submicron Cache Memories," Int. Symp. on Low Power Electronics and Design, 2000.

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

[14] S. Santhanam, et. al. "A Low-Cost, 300-MHz, RISC CPU with Attached Media Processor," IEEE Journal of Solid-State Circuit, Vol. 33, 1998.

[15] A Veidenbaum, D Nicolaescu, "Low Energy, Highly-Associative Cache Design for Embedded Processors," IEEE ICCD, 2004.

[16] C. Zhang, F. Vahid, and W. Najjar, "A Highly-Configurable Cache Architecture for Embedded Systems," ISCA, 2003.

[17] C. Zhang, F. Vahid and W. Najjar, "Energy Benefits of a Configurable Line Size Cache for Embedded Systems," IEEE International Symposium on VLSI Design, February 2003.

[18] Chuanjun Zhang, Frank Vahid and Roman Lysecky, "A Self-Tuning Cache Architecture for Embedded Systems Special Issue on Dynamically Adaptable Embedded System, ACM Transactions on Embedded Computing Systems Vol.3, No.2, May 2004, Pages1-19.