Chapter 32
A Simulation-Based Study on Memory Design Issues for Embedded Systems

Mohsen Sharifi, Mohsen Soryani, and Mohammad Hossein Rezvani

32.1 Introduction

Due to the increasing gap between the speed of CPU and memory, cache designs have become an increasingly critical performance factor in microprocessor systems. Recent improvements in microprocessor technology have provided significant gains in processor speed. This dramatic rise has increased further the gap between the speed of the processor and main memory. Thus, it is necessary to design faster memory systems. In order to decrease the processor–memory speed gap, one of the main concerns has to be in the design of an effective memory hierarchy including multilevel cache and TLB (Translation Lookaside Buffer).

On the other hand, a notable part of the computer industry nowadays is involved in embedded systems. Embedded systems play a significant role in almost all domains of human activities including military campaigns, aeronautics, mobile communications, sensor networks, and industrial local communications. Timeliness of reactions is necessary in these systems and offline guarantees have to be derived using safe methods. Hardware architectures used in such systems now feature caches, deep pipelines, and all kinds of speculations to improve average-case performance. The speed and size are two important concerns of embedded systems in the area of memory architecture design. In these systems, it is necessary to reduce the size of memory to obtain better performance.

Real-time embedded systems often have a hard deadline to complete some instructions. In these cases, the speed of memory plays an important role in system performance. Cache hits usually take one or two processor cycles, whereas cache misses take tens of cycles as a penalty of mishandling and so the speed of memory hierarchy is a key factor in the system. Almost all embedded processors have on-chip instructions and data caches. From a size point of view, it is critical for battery-operated embedded systems to reduce the amount of consumed power.

Another factor that affects cache performance is the degree of associativity. Nowadays, modern processors include multilevel caches and TLBs and their associativity is increasing. Therefore, it is critical to revisit the effectiveness of common cache replacement policies. When all the lines in a cache memory set become full and a new block of memory needs to be replaced into the cache memory, the cache controller must replace it with one of the old blocks in the cache. If the extracted cache memory line is needed in the near future, the performance of the system will be degraded. Therefore, the cache controller should extract a proper line from the cache. However, it can only guess which cache memory line should be discarded. The state-of-the-art processors employ various policies such as LRU (Least Recently Used) [1], Random [2], FIFO (First-In First-Out) [3], and PLRU (Pseudo-LRU) [4]. This shows that the selection of a proper replacement policy is still an important challenge in the field of computer architecture. All these policies, except Random, determine which cache memory line to replace by looking only at the previous references.

LRU policy increases the cost and implementation complexity. To reduce these two factors, Random policy can be used, but potentially at the expense of desirable performance. Researchers have proposed various PLRU heuristics to reduce the cost by approximating the LRU mechanism. Recent studies have considered only pure LRU policy [5–7] and have used the compiler optimization [8] or integrated approaches including hardware/software techniques [9].

One of the goals of our study is to explore common cache replacement policies and compare them with an optimal (OPT) replacement algorithm. An OPT algorithm would replace a cache line whose next reference is the farthest away in the future among all the cache lines [7]. This policy requires knowledge of the future, and hence its real implementation is impossible. Instead, heuristics have to be used to estimate it. We study OPT, LRU, a type of PLRU, Random, and FIFO policies on a wide range of cache organizations, varying cache sizes, degree of associativity, cache hierarchy, and multilevel TLB hierarchy.

Another goal in this study is to investigate the performance of the two-level TLB against the single-level TLB. This analysis is done in conjunction with cache analysis. Virtual to physical address translation is a significant operation because this is invoked on every instruction fetch and data reference. To speed up the address translation, systems provide a cache of recent translations called TLB, which is usually a small structure indexed by the virtual page number that can be quickly looked up. Several studies have investigated the importance of TLB performance [10]. The idea of multilevel TLB has been investigated in [11–13]. We compare the performance of a two-level TLB with traditional single TLB.

The performance analysis is based on *SimpleScalar*'s [14] cache simulators executing selected SPEC CPU2000 benchmarks [15]. Using a two-level TLB, we study the degree of cache associativity that is enough to offer a low miss rate with respect to SPEC CPU2000. Some prior research has investigated this subject, but only for traditional replacement policies such as LRU [5, 6]. We offer a competitive simulation-based study to reveal the relationship between cache miss rates produced by selected benchmarks and cache and TLB configurations. Additionally, we

measure the gap between miss rates of various replacement policies, especially OPT and the LRU family (such as PLRU which is less expensive than LRU). Similar measurements have been done in the literature for other modifications of LRU policy. Wong and Baer [9] demonstrated the effectiveness of a modification of the standard LRU replacement algorithm for large set associative L2 (level two) caches.

The aim of this chapter is to offer a comprehensive and simulation-based performance evaluation of the cache and TLB design issues in embedded processors such as two-level versus single TLB, split versus unified cache, cache size, cache associativity, and replacement policy.

The rest of chapter is organized as follows. Section 32.2 elaborates the problem under our study, related works on hierarchical TLB, specifications of SPEC CPU2000 benchmarks, and the reasons for selecting the benchmarks used in our study. Section 32.3 describes the setup of our experiments. Section 32.4 reports the results of our experiments, and Sect. 32.5 concludes the chapter.

32.2 Related Work

Three categories of related works have guided our study: prior research on cache replacement policies, prior research on hierarchical TLB mechanisms, and prior research on specifications of the SPEC CPU2000 benchmarks.

Several properties of the processor caches influence performance: associativity, replacement, and write policy, and whether there are separated data and instruction caches. The predictability of different cache replacement policies is investigated in [9]. The following are widely used replacement policies in commercial processors.

- LRU used in Intel Pentium and MIPS 24K/34K
- FIFO (or Round-Robin), used in Intel XScale, ARM9, ARM11
- PLRU used in Power PC 75X and Intel Pentium II, III, and IV

Cache implementations vary from direct-mapped to fully associative. With direct-mapped caches, also called one-way caches, each memory block is mapped onto a distinct cache line, whereas with fully associative cache memory, each memory block can be mapped to any of the empty cache lines. Generally, with k-way set associative cache memory, a memory block can be mapped to any of the empty lines among k cache lines within the set to which the block belongs. If all cache lines within the set are full, one of the cache lines is extracted according to a replacement policy.

As the degree of cache associativity increases, selecting an efficient replacement policy becomes more important [3]. Traditionally, most processors have chosen the LRU policy as the replacement policy. LRU replacement maintains a queue of length k, where k is the associativity of the cache. If an element is accessed that is not yet in the cache (a miss), it is queued at the front and the last element of the queue is removed. This is the least recently used element of those in the queue. At a cache

hit, the element is moved from its position in the queue to the front, effectively treating hits and misses equally. The contents of LRU caches are very easy to predict. Having only distinct accesses and a strict least recently used replacement, directly yields the tight bound k, that is, the number of distinct accesses (hits or misses) needed to know the exact contents of a k-way cache set using LRU replacement is k [16]. The weak point of this policy is its requirement of time and power.

To reduce the cost of LRU policy, Random policy [2] with lower performance is used. In this policy, the victim line is chosen randomly from all the cache lines in the set. Another candidate policy is FIFO, which can also be seen as a queue: new elements are inserted at the front evicting elements at the end of the queue. In contrast to LRU, hits do not change the queue. Real implementations use a Round-Robin replacement counter for each set pointing to the cache line that will be replaced in the future. This counter is increased if an element is inserted into a set, and a hit does not change this counter. In the case of misses only, FIFO behaves as does LRU. Thus, it has the tight bound k [16].

An approximation to the LRU mechanism is PLRU [17, 18]. This policy speeds up operations and reduces the complexity of the implementation. One of the PLRU implementations is based on using the most recently used (MRU) bits. In this policy, each line is assigned an MRU-bit, stored in a tag table. Every access to a line sets its MRU-bit to 1, indicating that the line has been recently used. PLRU is deadlock-prone. Deadlock occurs if the MRU-bits for all blocks are set to 1 and therefore none of them is ready to be replaced.

In order to prevent this situation, whenever the last 0 bit of a set is set to 1, all other bits are reset to 0. At each cache miss, the line with lowest index (in our representation the leftmost line) whose MRU-bit is 0 is replaced. As an example, we represent the state of an MRU cache set as $[A, B, C, D]_{0101}$, where 0101 are the MRU-bits and $A, \ldots, D$ are the contents of the set. On this state, an access to E yields a cache miss and new state $[E, B, C, D]_{1101}$. Accessing D leaves the state unchanged. A hit on C forces a reset of the MRU-bits: $[E, B, C, D]_{0010}$.

Figure 32.1 illustrates the MRU policy with respect to the above scenario. For the MRU replacement policy, it is impossible to give a bound on the number of

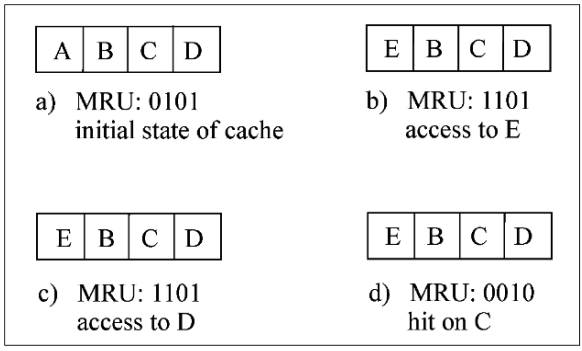

Fig. 32.1 An illustrative example of MRU replacement policy

Table 32.1 Bits used for each replacement policy

Replacement policy	Used bits
Random	Log2ways
FIFO	Nsets. Log2ways
LRU	Nsets.ways. Log2ways
MRU	Nsets.ways

accesses needed to reach a completely known cache state [16]. It seems that this method looks like LRU and FIFO; hence, we expect its performance to be better than FIFO and worse than LRU.

The LRU policy requires a number of bits to track when each cache line is accessed, whereas the MRU does not require as many bits. Indeed the MRU has less complexity than LRU. Table 32.1 shows the amount of bits used in each replacement policy [17].

Another contribution of our work is in studying two-level TLB in embedded systems and comparing its performance against single TLB. Many studies have pointed out the importance of the TLB [19–23]. Hardware TLB hierarchy and its impact on system performance are investigated in [11, 12, 20]. The advantages of multilevel TLBs over single TLBs are studied in [13, 24]. There are also real implementations of multilevel TLBs in commercial processors such as Hal's SPARC 64, IBM AS/400 Power PC, Intel Itanium, and AMD. They use either hardware or software mechanisms to update the TLB on the misses.

From the principal component analysis of raw data in [25] it is concluded that several SPEC CPU2000 benchmark programs such as *bzip, gzip, mcf, vortex, vpr, gcc, crafty, applu, mgrid, wupwise*, and *apsi* exhibit a temporal locality that is significantly worse than other benchmarks. Concerning spatial locality, most of these benchmarks exhibit a spatial locality that is relatively higher than that of the remaining benchmarks. The only exceptions are *gzip* and *bzip2* which exhibit poor spatial locality. As pointed out in [25], there is lots of redundancy in the benchmark suite. Simulating benchmarks with similar behavioral characteristics will add to the overall simulation time without providing any additional insight, so we have selected our benchmarks based on clustering results presented in [25].

We have also noticed the results presented in [26] to select our benchmarks. Some of SPEC CPU2000 benchmarks are eccentric, that is, have a behavior that differs significantly from the behavior of other benchmarks. Eccentric benchmarks are excellent candidates for case studies and it is important to include them when subsetting a benchmark suite (e.g., to limit simulation time). These benchmarks differ from the average SPEC CPU 2000 benchmark in different ways, for example, requiring high associativity or suffering from repetitive conflict misses, having low spatial locality, or benefiting from extremely large caches. For example, *crafty* has a lower cache miss rate when the block size is small. It is also somewhat more dependent on high associativity than other benchmarks. *equake* depends strongly on

the degree of associativity and suffers from repetitive conflict misses. *vpr* is highly sensitive to changes in data cache associativity, having a large number of misses.

32.3 Experimental Setup

The simulator used in our study is *SimpleScalar*. Performance of two-level TLB with different cache replacement policies were evaluated using *Sim-Cache* and *Sim-Cheetah* simulators from the Alpha version of this toolset [14].

The *Sim-Cache* engine simulates associative caches with FIFO, Random, and LRU policies. The *Sim-Cheetah* engine simulates fully associative caches efficiently, as well as simulating a sometimes-optimal replacement policy. Belady [27] calls the latter policy MIN, whereas the simulator calls it OPT. Because OPT uses future knowledge to select a replacement, it cannot be really implemented. It is, however, useful as a yardstick for evaluation of other replacement policies. We modified the original simulator to support hierarchical two-level TLB and MRU replacement policy as well as their native replacement policies, that is, FIFO, Random, LRU, and OPT and its original single TLB.

As mentioned above, we have used selected benchmarks from the SPEC CPU2000 suite as a simulation workload. Given the eccentricity of some benchmarks, we filtered out benchmarks insensitive to increases in cache associativity. We selected *vpr, gcc, crafty, eon*, and *twolf* as five integer benchmarks and *mgrid, apsi, fma3d*, and *equake* as four floating-point benchmarks. Selected benchmarks were used with reference data inputs. Instructions related to initializations were skipped and the main instructions were simulated.

For each benchmark, we performed the simulation with various L1 data cache organizations with 2-, 4-, and 8-way associativity, replacement policies as FIFO, LRU, and MRU and various instruction and data cache sizes. In our experiments all TLBs are assumed to be fully associative with the LRU replacement policy. We changed the cache sizes with 4 KB, 8 KB and 16 KB. To study the impact of 2-level d-TLB (data TLB), we compared its miss rates with that of a single TLB of the same size ($32 + 96 = 128$ entries). Also in all experiments a cache line size was 32 bytes. Relating to the cache memory, we considered two scenarios: in the first scenario, the system consists of two separate data and instruction caches in the first level (L1D and L1I) and in the second scenario, it only has a unified cache in the first level (L1U) which serves both data and instruction references. Figure 32.2 shows the above two scenarios.

The defaults used in *Sim-Cache* were as follows: 8 KB L1 instruction cache and data cache, 256 KB L2 unified cache, i-TLB (instruction TLB) with 64 entry and d-TLB with 128 entries. In all memories, the default replacement policy was LRU.

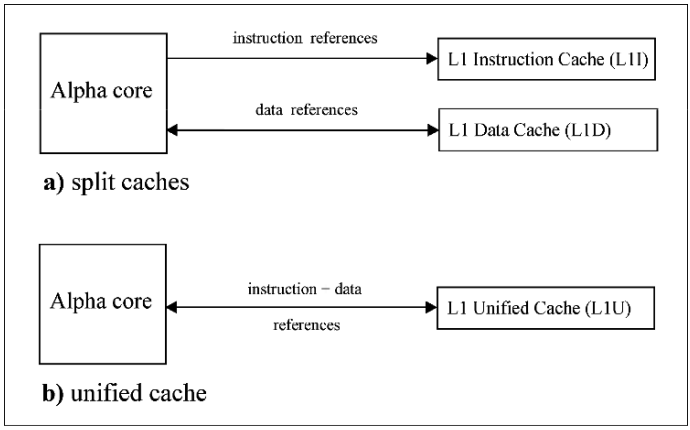

Fig. 32.2 Two scenarios for memory hierarchy

32.4 Results of Experiments

32.4.1 The Effect of Two-Level TLB on Overall Performance

Table 32.2 shows the results of using hierarchical two-level d-TLB and single level d-TLB. It shows the miss rates for each of the two levels as a percentage of the number of references to that level, as well as the overall miss rate. The overall miss rate is the percentage of references that do not find a translation in either of the two levels. The results show higher overall TLB miss rates when using two-level TLB, especially for *twolf, gcc, crafty*, and *apsi*. The average TLB miss rates for selected integer benchmarks for two-level and single TLBs are 1.91% and 1.48%, respectively, resulting in a degradation of about 0.43%. The miss rate degradation for selected floating-point benchmarks is on average 0.46%.

Despite the higher miss rates, the benefit of a two-level TLB is in reducing the access time of first level TLB and in avoiding accessing the second level. Figure 32.3 shows the normalized program execution times for the selected benchmarks. Here, the normalized execution time is the ratio of program execution time with a two-level TLB to its native execution time with single TLB. Except the two-level TLB, the other parts of simulation are the same as defaults used in the *SimpleScalar*. The two-level TLB is fully associative with LRU replacement, as is common in most commercial processors.

The average reductions in execution time, when using two-level TLB for integer and floating-point benchmarks, are about 0.32% and 0.60%, respectively. According to the results, using a 2-level TLB cannot produce a conspicuous reduction in execution time, and it degrades the overall miss rate.

Table 32.2 Miss rates of two-level TLB and single TLB

Benchmark	Miss rate of first level	Miss rate of second level	Overall miss rate of two-level TLB	Miss rate of single TLB
Twolf	3.32	44.6	1.48	1.27
Vpr	4.21	41.1	1.73	1.68
Gcc	2.19	64.84	1.42	1.13
Crafty	1.27	96.06	1.22	0.09
Eon	3.85	96.36	3.71	3.24
Mgrid	21.78	97.56	21.25	20.23
Apsi	2.60	88.77	2.30	1.98
fma3d	1.93	78.23	1.51	1.21
Equake	20.34	94.93	19.31	19.12

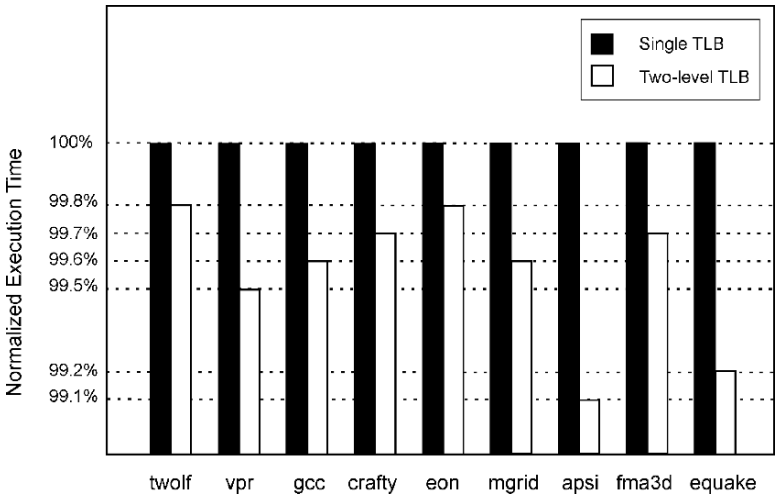

Fig. 32.3 Normalized execution time for each benchmark with two-level TLB and Single TLB

32.4.2 The Effect of Cache Associativity, Size, and Replacement Policy on Performance

Tables 32.3 and 32.4 show the average cache miss rates of integer and floating-point benchmarks for the L1 data cache. The miss rates for floating-point applications are lower than integer applications when the L1 data cache is 4 KB. For larger size L1 data caches of 8 KB and 16 KB miss rates of floating-point applications become higher. In the L1 data cache, it is hard to select a replacement policy between FIFO and Random policies, and the difference between them decreases as the cache size increases. There are applications where Random policy outperforms FIFO. The *crafty, fma3d*, and *equake* are three examples of such applications in our selected benchmarks. In other applications, FIFO has fewer misses against Random.

Table 32.3 Average L1 data cache miss rates for five SPEC CPU2000 integer applications (*vpr, gcc, crafty, eon*, and *twolf*)

		1W	2W	4W	8W	16W	32W
4 KB	FIFO	5.31	4.28	3.99	3.84	3.82	3.81
	LRU	5.31	3.99	3.57	3.40	3.29	3.29
	Random	5.31	4.36	4.09	3.98	3.92	3.92
	MRU	5.31	3.99	3.60	3.47	3.38	3.35
	OPT	5.31	3.62	3.57	2.36	2.23	2.18
8 KB	FIFO	4.12	2.94	2.65	2.53	2.49	2.45
	LRU	4.12	2.76	2.38	2.24	2.16	2.15
	Random	4.12	2.94	2.65	2.55	2.49	2.46
	MRU	4.12	2.76	2.38	2.25	2.18	2.16
	OPT	4.12	2.12	1.76	1.61	1.53	1.49
16 KB	FIFO	3.23	2.00	1.76	1.64	1.59	1.58
	LRU	3.23	1.89	1.60	1.46	1.40	1.38
	Random	3.23	2.00	1.78	1.75	1.61	1.60
	MRU	3.23	1.89	1.61	1.49	1.46	1.38
	OPT	3.23	1.89	1.28	1.17	1.12	1.10

Table 32.4 Average L1 data cache miss rates of four SPEC CPU2000 floating-point applications (*mgrid, apsi, fma3d*, and *equake*)

		1W	2W	4W	8W	16W	32W
4 KB	FIFO	5.73	4.63	4.18	3.29	3.28	3.28
	LRU	5.73	4.47	3.98	3.29	3.03	3.01
	Random	5.73	4.69	4.53	3.89	3.54	3.52
	MRU	5.73	4.53	3.99	3.29	3.03	3.03
	OPT	5.73	3.30	2.82	2.45	2.20	2.14
8 KB	FIFO	3.59	2.97	2.70	2.56	2.46	2.46
	LRU	3.59	2.80	2.45	2.30	2.26	2.25
	Random	3.59	3.13	2.92	2.80	2.55	2.46
	MRU	3.59	2.88	2.53	2.41	2.33	2.29
	OPT	3.59	2.39	2.02	1.78	1.64	1.61
16 KB	FIFO	2.20	1.93	1.84	1.83	1.80	1.74
	LRU	2.20	1.87	1.75	1.68	1.65	1.65
	Random	2.20	2.09	2.04	2.04	2.03	1.97
	MRU	2.20	1.92	1.76	1.75	1.73	1.72
	OPT	2.20	1.71	1.59	1.55	1.54	1.54

In general, we can conclude that for the rest of the benchmarks, for larger cache sizes, Random policy dominates, whereas for smaller cache sizes, FIFO dominates.

Experiments show that LRU policy is almost better than FIFO and Random, but there are some exceptions. For example, Random policy is sometimes better than LRU for *equake* and *apsi*. Compared to LRU policy, FIFO is on average about 17% worse, whereas Random is about 18% worse.

Compared to LRU policy, the performance degradation of MRU is relatively small; however, because of low complexity of MRU, we can neglect its degradation.

The gap between MRU, the best realistic replacement used in this chapter, and OPT is larger for smaller caches due to more conflict misses. The miss rate reduction is more distinct, as the size of the L1 data cache decreases. The results show that the largest reduction in miss rate is for transition from a direct-mapped to a two-way set associative L1 data cache.

For both integer and floating-point benchmarks, increased associativity has a large miss reduction with small caches. In floating-point applications, for large caches (16 KB), associativity higher than two does not efficiently reduce the miss rate, but for small caches (4 KB), the amount of miss reduction is noticeable. For L1 data cache sizes of 8 KB and 16 KB, the effect of increased associativity on miss reduction is more obvious for integer applications, than for floating-point applications.

As mentioned earlier, OPT replacement policy can be used as a yardstick to evaluate the replacement policies. From Tables 32.3 and 32.4 it can be deduced that the OPT miss rate of a certain cache size is roughly close to the MRU miss rate of a cache twice as big, with the same number of ways. For example, in Table 32.4, the miss rate of an eight-way set associative cache with size 16 KB and MRU replacement policy is 1.75%. This is approximately equal to the (1.78%) miss rate of an eight-way set associative, 8 KB, optimal replacement policy. This shows that there is still a large gap between OPT and realistic policies such as MRU. Therefore, if near optimal policy can be found in practice, the size of caches can be reduced to one-half.

32.5 The Effect of Split Cache Versus Unified Cache on Performance

Tables 32.5 and 32.6 show the results of using split data and instruction caches as the first-level caches against the common unified cache as the first-level cache. For any of the *k*-way set associative unified caches in the table, the miss rate is considerably higher compared to the aggregated miss rate of corresponding split instruction and data caches of equivalent size. The difference becomes smaller as the size of instruction and data caches increases.

Tables 32.5 and 32.6 show that for integer applications, a four-way set associative cache, on average across various cache sizes, reduces the miss rate about 12% for L1U, 12% for L1D, and 8% for L1I when compared to a two-way set associative cache. The reduction of four-way over two-way for floating-point applications for L1U, L1D, and L1I is 10%, 13%, and 9%, respectively. For integer applications the benefit of an eight-way organization, compared to two-way set associative, is 13% for L1U, 16% for L1D, and 12% for L1I, whereas for floating-point applications the benefit of eight-way over two-way is 15%, 20%, and 12% for L1U, L1D, and L1I, respectively. The results show that the gain of increasing associativity in the case of data and unified caches is more than instruction caches.

As expected, LRU and MRU replacement policies perform better than Random for data caches, although surprisingly, Random policy performs almost better than

Table 32.5 Average cache miss rates for L1I, L1D, and L1U caches for selected SPEC CPU2000 integer applications

		L1I of Size i KB			L1D of Size i KB			L1U of Size $2i$ KB		
		2W	4W	8W	2W	4W	8W	2W	4W	8W
$i = 4$	FIFO	5.52	5.55	5.61	4.28	3.99	3.84	15.30	13.98	13.82
	LRU	5.48	5.50	5.57	3.99	3.57	3.40	14.99	13.55	13.37
	Random	5.38	5.34	5.36	4.36	4.09	3.98	15.35	14.00	13.94
	MRU	5.48	5.31	5.32	3.99	3.60	3.47	14.99	13.58	13.42
	OPT	4.36	3.90	3.70	3.62	3.57	2.36	14.22	13.17	12.20
$i = 8$	FIFO	4.29	3.97	4.01	2.94	2.65	2.53	10.91	9.59	9.29
	LRU	4.25	3.90	3.91	2.76	2.38	2.24	10.74	9.31	9.23
	Random	4.01	3.61	3.47	2.94	2.65	2.55	10.91	9.62	9.50
	MRU	4.24	3.71	3.54	2.76	2.38	2.25	10.74	9.32	9.27
	OPT	3.11	2.45	2.21	2.12	1.76	1.61	9.10	7.71	7.63
$i = 16$	FIFO	2.75	2.75	2.60	2.00	1.76	1.64	7.02	5.62	5.61
	LRU	2.70	2.71	2.57	1.89	1.60	1.46	6.85	5.51	5.43
	Random	2.45	2.16	1.91	2.00	1.78	1.75	7.05	5.76	5.75
	MRU	2.71	2.24	2.21	1.89	1.61	1.49	6.87	5.60	5.48
	OPT	1.78	1.36	1.20	1.89	1.28	1.17	6.40	5.27	5.19

Table 32.6 Average cache miss rates for L1I, L1D, and L1U caches for selected SPEC CPU2000 floating-point applications

		L1I of size i KB			L1D of size i KB			L1U of size $2i$ KB		
		2W	4W	8W	2W	4W	8W	2W	4W	8W
$i = 4$	FIFO	6.82	6.86	6.92	4.63	4.18	3.29	17.07	16.11	15.65
	LRU	6.78	6.80	6.87	4.47	3.98	3.29	17.01	15.83	15.22
	Random	6.67	6.65	6.68	4.69	4.53	3.89	17.91	16.20	15.73
	MRU	6.78	6.61	6.62	4.53	3.99	3.29	17.05	15.91	15.30
	OPT	5.69	5.20	5.03	3.30	2.82	2.45	15.00	13.11	12.51
$i = 8$	FIFO	5.58	5.27	5.30	2.97	2.70	2.56	14.85	13.55	13.27
	LRU	5.53	5.24	5.26	2.80	2.45	2.30	14.11	12.80	12.50
	Random	5.31	4.89	4.76	3.13	2.92	2.80	14.97	13.61	13.30
	MRU	5.58	5.00	4.81	2.88	2.53	2.41	14.20	12.90	12.60
	OPT	4.39	3.69	3.55	2.39	2.02	1.78	12.91	9.03	8.91
$i = 16$	FIFO	3.74	3.74	3.66	1.93	1.84	1.83	10.80	8.97	8.91
	LRU	3.69	3.74	3.55	1.87	1.75	1.68	10.21	8.91	8.80
	Random	3.34	3.04	2.87	2.09	2.04	2.04	10.98	8.99	8.94
	MRU	3.69	3.26	3.25	1.92	1.76	1.75	10.29	8.92	8.85
	OPT	2.75	2.34	2.14	1.71	1.59	1.55	9.23	7.50	7.46

LRU and MRU policies for instruction caches. The temporal locality of the instruction cache is low, compared to that of the data cache. Thus, a rich replacement policy such as LRU has approximately equal performance compared to Random policy, which is a poor replacement policy.

32.6 Conclusions

The organization of cache and TLB memory is a critical issue in general-purpose embedded systems. This chapter presented a simulation-based study of the performance evaluation to find the main cache design issues such as hierarchical TLB and cache, cache size and associativity, and replacement policy in embedded processors. We selected some applications from the SPEC CPU2000 benchmark suite based on eccentricity and clustering concepts and found that the two-level TLB would not produce a significant reduction in execution time, while degrading the overall miss rate.

The experimental results showed that the gain of increasing associativity in the case of data and unified caches is more than instruction caches. In the L1 data cache and L1 unified cache, the largest miss rate reduction occurs for transition from a direct-mapped to a two-way set associative. In floating-point applications, for large caches, associativity higher than two, does not effectively reduce the miss rate, but for small caches, the amount of reduction in miss rate is noticeable.

As expected, LRU and MRU replacement policies perform better than Random for data caches, and surprisingly, for instruction caches, Random policy performs almost better than LRU and MRU. Comparing FIFO and random policies, for larger data cache sizes, Random policy dominates, whereas for smaller cache sizes, FIFO dominates. Nevertheless, in general for the L1 data cache, it is hard to select a winner replacement policy between FIFO and Random policies, and the difference between them decreases as the cache size increases.

For large caches, the MRU policy is a good approximation of LRU policy. Compared to LRU policy, MRU has less complexity and according to our results has negligible miss rate degradation.

The results of experiments also illustrated that the performance of OPT policy is nearly the same as the performance of the lower-cost MRU policy using a cache twice as big. This shows that there is still a large gap between optimal replacement policy and realistic replacement policies such as MRU. Eliminating this gap will reduce the size of caches even to one-half of their current sizes.

With respect to the attempts of memory designers to reduce the amount of power, our results offer valuable insights into the design of memory for embedded systems.

References

1. Kalavade A, Knoblock J, Micca E, Moturi M, Nicol CJ, O'Neill JH, Othmer J, Sackinger E, Singh KJ, Sweet J, Terman CJ, Williams J (2000). A single-chip, 1.6 billion, 16-b MAC/s multiprocessor DSP. *IEEE Journal of Solid-State Circuits*, 35(3), pp. 412–423.
2. Hennessy JL, Patterson D (2003). *Computer Architecture: A Quantitative Approach*. Third Edition, San Mateo. CA: Morgan Kaufmann.
3. Intel Xscale™ (2000). *Core: Developer's Manual*, December (2000). URL: http://developer.intel.com.

4. Intel Pentium 4 and Intel Xeon Processor Optimization: Reference Manual™. *Reference Manual*. URL: http://developer.intel.com.
5. Cantin JF, Hill MD (2000). Cache performance of the SPEC CPU2000 benchmarks. URL: http://www.cs.wisc.edu/multifacet/misc/spec2000cachedata/.
6. Sair S, Chamey M (2000). Memory behavior of the SPEC2000 benchmark suite. IBM Thomas J. Waston Research Center, Technical Report RC-21852.
7. Thomock NC, Flangan JK (2000). Using the BACH trace collection mechanism to characterize the SPEC 2000 integer benchmarks. *Workshop on Workload Characterization*.
8. Wang Z, McKinley K, Rosenberg A, Weems C (2002). Using the compiler to improve cache replacement decisions. *The International Conference on Parallel Architectures and Compilation Techniques*, Charlottesville, Virginia.
9. Wong W, Baer JL (2000). Modified LRU policies for improving second-level cache behavior. *The 6th International Symposium on High-Performance Computer Architecture*, Toulouse, France.
10. Jacob BL, Mudge TN (1998). A look at several memory management units: TLB-refill mechanisms, and page table organizations. *Proceedings of the Eight International Conference on Architectural Support for Programming Languages and Operating Systems*, pp 295–306.
11. Talluri M (1995). Use of superpages and subblocking in the address translation hierarchy. PhD thesis, Deptartment of CS, Univiversity of Wisconsin at Madison.
12. Nagle D, Uhlig R, Stanley T, Sechrest S, Mudge T, Brown R (1993). Design tradeoffs for software managed TLBs. *Proceedings of the 20th Annual International Symposium on Computer Architecture*, pp 27–38.
13. Chen JB, Borg A, Jouppi NP (1992). A simulation based study of TLB performance. *Proceedings of the 19th Annual International Symposium on Computer Architecture*, pp 114–123.
14. Burger D, Austin T (1997). The SimpleScalar tool set, version 2.0. Technical Report #1342, Computer Sciences Department, University of Wisconsin, Madison, WI.
15. Henning JL (2000). SPEC CPU2000: Measuring CPU performance in the new millennium. *IEEE Computer*, 33(7), pp 28–35.
16. Reineke J, Grund D, Berg C, Wilhelm R (2006). Predictability of cache replacement policies. AVACS Technical Report No. 9, SFB/TR 14 AVACS.
17. Malamy A, Patel R, Hayes N (1994). Methods and Apparatus for Implementing a Pseudo-LRU Cache Memory Replacement Scheme with a Locking Feature. United States Patent 5353425.
18. So K, Rechtshaffen RN (1988). Cache operations by MRU change. *IEEE Transaction on Computers*, 37(6), pp 700–707.
19. Anderson TE, Levy HM, Bershad BN, Lazowska ED (1991). The interaction of architecture and operating system design. *Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, Santa Clara, CA, pp 108–120.
20. Jacob B, Mudge T (1998). Virtual memory in contemporary microprocessors. *IEEE Micro*, 18(4), pp 60–75.
21. Clark DW, Emer JS (1985). Performance of the VAX- 1/780 translation buffers: Simulation and measurement. *ACM Transactions on Computer Systems*, 3(1).
22. Huck J, Hays J (1993). Architectural support for translation table management in large address space machines. *Proceedings of the 20th Annual International Symposium on Computer Architecture*, pp 39–50.
23. Rosenblum M, Bugnion E, Devine S, Herrod S (1997). Using the SimOS machine simulator to study complex computer systems. *ACM Transactions on Modeling and Computer Simulation*, 7(1), pp 78–103.
24. Austin TM, Sohi GS (1996). High bandwidth address translation for multiple issue processors. *The 23rd Annual International Symposium on Computer Architecture*.
25. Phansalkar A, Joshi A, Eeckhout L, John K (2004). Four generations of SPEC CPU benchmarks: What has changed and what has not. Technical Report TR-041026-01-1.
26. Vandierendonck H, De Bosschere K (2004). Eccentric and fragile benchmarks. *2004 IEEE International Symposium on Performance Analysis of Systems and Software*, pp 2–11.
27. Belady LA (1966). A study of replacement algorithms for a virtual-storage computer. *IBM Systems Journal*, 5(2), pp 78–101.