Reducing Conflict Misses in Caches

Pepijn J. de Langen Ben H.H. Juurlink

Computer Engineering Laboratory
Electrical Engineering Department
Delft University of Technology
P.O.Box 5031, 2600 GA Delft, The Netherlands
Phone: +31 15 278 1572, Fax: +31 15 278 4898
E-mail: pepijn@ce.et.tudelft.nl benj@ce.et.tudelft.nl

Abstract—

Nearly all modern computing systems employ caches to hide the memory latency. Modern processors often employ multiple levels of cache, with one or more levels on the same die as the processor core. As performance demands increase, it becomes increasingly important for an on-die cache structure to perform well. Inefficient use of the available cache space can become increasingly costly in terms of both speed [1] and power usage [4]. Speed and size constraints often make the case for small on-die caches, with little or no associativity. However, small direct-mapped caches may result in a large number of conflict misses.

In order to reduce the number of conflicts, we propose a small structure called the Conflict Detection Table (CDT), which stores the instruction and data addresses of load/store instructions. By using this table, it can be found if a memory access is expected to produce a cache hit. From this information, it can be determined if a cache miss is caused by a conflict with another instruction, and appropriate action can be taken.

We propose two caches that employ this conflict detection technique. The first cache, called the Bypass in Case of Conflict (BCC) cache, is a direct-mapped cache that bypasses the cache when a conflict is detected. The second cache, called the Sub-block in Case of Conflict (SCC) cache, is a direct-mapped cache that fetches only part of the referenced cache line, when a conflict is detected.

Experiments have shown that, for a number of applications, the BCC and the SCC can produce less traffic than direct-mapped caches. This leads to decreased power consumption without increasing the average memory access time.

Keywords— **cache; conflict misses; power reduction; embedded systems**

I. Introduction

As we have shown in earlier work [4], conflict misses in small direct-mapped caches can cause a significant amount of 'wasted' off-chip memory traffic, especially if the line size of the cache is large. Because of space and speed constraints, on-die caches are in general small and lack associativity.

Off-chip memory transfers consume a significant amount of power, compared to control units and datapaths [3]. Therefore, power reduction can be achieved by either reducing the number of conflict misses or by reducing the amount of transferred bytes per conflict miss.

This paper is organized as follows. Section II will discuss some examples of related work by other authors. In Section III we will explain how recurring conflict misses can be detected and how these can be resolved. This is experimentally verified in Section IV. Conclusions are given in Section V.

II. Related Work

Jouppi [8] proposed employing a small (four to eight entries), fully associative *victim cache* in order to reduce conflict misses in caches with low associativity. Blocks evicted from the primary cache are not immediately placed in the level-2 cache but are given a second chance in the victim cache. The victim cache is fully associative, however, and fully associative caches consume more energy than direct-mapped caches. Memik et al. [10] proposed several techniques to reduce the energy dissipated by cache organizations equipped with a victim cache.

The Dual Data Cache proposed by González et al. [5] includes a mechanism that detects if a load instruction interferes with itself. This happens, for example, when a vector is accessed and the vector is larger than the cache size. In such a case, the vector removes itself from the cache. This situation is even worse when the cache size and the vector length are not co-prime, because then not all cache blocks are used to store the vector. This mechanism, however, does not detect cross-interference, i.e., when data is replaced by data referenced by a different load instruction.

Johnson et al. [7] try not to evict a block if it is more heavily used than the arriving block that generated a miss. To do so they divide the memory into regions called *macroblocks* and employ a table called the *Memory Address*

Table (MAT) that contains information about how often each macroblock is used. If the MAT indicates that the block to be replaced is more heavily used than the arriving block, the arriving block is not stored in the cache. The MAT behaves like a cache, and it appears that it must be rather large in order to be effective. In the future we intend to compare the performance attained by the MAT with that of the BCC and SCC caches.

There are also static (compiler) approaches aimed at reducing conflict misses. For example, Catthoor et al. [2] analyze the lifetimes of array variables. Arrays that are life simultaneously are placed in memory in such a way that they cannot conflict in the cache.

III. Conflict Detecting Cache Structures

Conflict misses occur when multiple data elements are stored to the same cache line, while they have different tags. In direct-mapped caches, the stored cache line is replaced by the one that is fetched from memory.

Assume two instructions that operate concurrently on different parts of the memory. For a direct-mapped cache, it is very well possible that parts of the first region are mapped onto the same cache lines as different parts of the second region. This is especially true for when the cache is small and the data sets large. In a worst case scenario, these instructions will constantly replace cache lines that were recently filled by the other instruction. This 'round-robin' use of cache lines will increase the amount of produced traffic to a maximum and reduce the number of cache hits to a minimum.

To resolve this inefficiency in direct-mapped caches, we propose a small additional structure, called the *Conflict Detection Table* (CDT), depicted in Figure 1. The principle of the CDT is based on the fact that if an instruction makes multiple references to the same line in the cache, we expect a cache-hit on the second and further accesses to the same line by this instruction. If then a hit is expected, and the cache produces a miss, it is clear that the cache line was recently used by an different instruction.

The CDT is used to store instruction addresses and the most recently accessed data address that was requested by the corresponding load/store instruction. Since the CDT is in fact a cache itself, it may be configured direct-mapped, fully-associative, or anything in between.

On every memory access the CDT is searched for the address of the current instruction, and an entry is allocated if it is not found. If the instruction address is found in this table, the data address in the corresponding entry of this table is updated and compared to the data address that is currently requested, to detect if these are contained in the same cache line. If this is the case, we expect the currently referenced data to be already available in the cache, since it has been fetched last time this instruction was executed. From this, it follows that if a cache miss is encountered, the requested data item has been replaced by another instruction. At this point, a conflict is detected.

The detection of a conflict can only happen on a cache-miss. The information about possible conflicts, therefore, does not have to be available until after the tag comparison in the cache. This implies that the cycle time of the cache will not be increased by use of the CDT. Since the amount of required logic to implement this is fairly low, it will also not consume much power.

Whenever a conflict is detected, it is not known in advance which instruction will be the first to use the cache line again in the future. Therefore, for reduction of the miss-rate, it is not certain what will be most efficient, For the reduction of the amount of traffic, however, it is always efficient to fetch a smaller number of bytes, instead of a whole cache line. Traffic can therefore be reduced if only the requested word is fetched when a conflict is detected. When this word is fetched from the memory, it can either be placed in the cache or it can *bypass* the cache completely. In the first case, the cache employs *sub-block* caching. In the second one, the cache employs *bypassing*.

In the next section we will verify experimentally if traffic can be reduced by employing the conflict detecting technique, as explained above. Also, miss-rates for caches that employ this technique will be examined, because possible increase in delay must not be neglected.

IV. Experimental Validation

In the previous section, we proposed the *Conflict Detection Table*, which can be used to detect conflicts between different load/store instructions. Using this table, we can define caches that dynamically adjust the policy of caching if a conflict is detected.

The first cache we propose is called the *Bypass in Case of Conflict* (BCC) cache. This is a direct-mapped cache that has the ability to have memory accesses bypass the cache structure. If it is detected that the cache line has been replaced by another instruction, the cache will choose to bypass all further references by this instruction, as long as they are to the same cache line.

The second cache is called the *Sub-block in Case of Conflict* (SCC) cache. This cache is a direct-mapped one that employs sub-block caching. If a conflict is detected, this cache will only fetch and store part of the cache line. The parts of the cache line that were not fetched are then invalidated. In this case, we have chosen for a sub-block size equal to the size of a word.

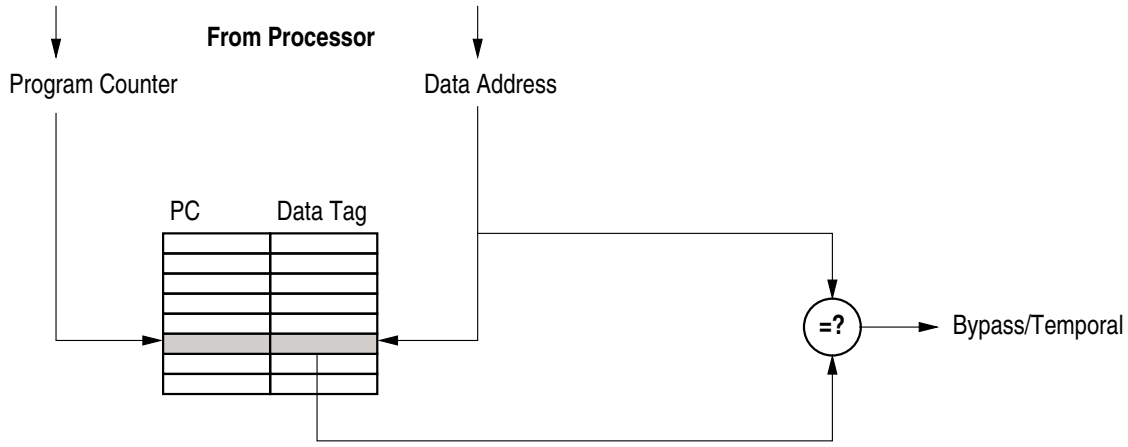

Fig. 1
CONFLICT DETECTION TABLE (CDT)

A. Benchmarks & Tools

We have benchmarked caches of sizes from 256 bytes to 128 kilobytes. For these benchmarks we have used the *MediaBench* [9] benchmarking suite, which consists of a number of audio and video codecs as well as encryption and decryption routines. These benchmarks are more representative of modern embedded applications than traditional suites like, for example, SPEC92 and SPEC95. The *MiBench* [6] benchmarking suite, which is aimed at embedded systems, was not available at the time. Also, *MediaBench* and *MiBench* share a number of common benchmarks.

A modified version of `sim-safe` from the *SimpleScalar* toolset was used to generate traces of the memory access. These traces were fed to our cache simulator, which generates statistics of occurred events. From this data, the number of transferred bytes can be computed, as well as the miss-rate.

All caches have a line size of 32 bytes, are direct-mapped, and employ a write-back policy. For the *Conflict Detection Table*, we have used a direct-mapped structure with 128 entries.

B. Results

Tables I and II list the results of experiments with the BCC cache and the SCC cache respectively. In these tables, we have compiled the relative reductions in amount of produced traffic by a cache, compared to a conventional direct-mapped cache with 32-byte cache lines.

From these tables, it is clear that, by using the *Conflict Detection Table*, the amount of traffic can be significantly reduced in many cases. Especially for small caches, a reduction in traffic of over 50% can be achieved by both the BCC and the SCC caches.

The effect of this conflict detection technique is clearly very dependent on the type and amount of locality that is exhibited by an application. For benchmarks like `pegwit-dec` and `pegwit-enc`, use of the conflict detection methods does not change the amount of produced traffic significantly. For the majority of benchmarks, however, the difference in produced traffic, compared to a direct-mapped cache, is significant.

For larger capacities, fewer benchmarks seem to benefit from this technique. In some cases, most notably for the `mpeg2-dec` benchmark using the BCC cache of 16 kilobytes, the amount of produced traffic increases significantly. For the BCC cache, this increase in traffic can reach up to 75%. For the SCC cache this maximum increase is far less, at 11%.

Overall, it can be concluded that, in the majority of cases, the amount of traffic produced by a cache can be reduced significantly by use of the conflict detection mechanism described earlier. Although the BCC cache produces the least amount of traffic for some combinations of benchmarks and sizes, this cache is less preferable than the SCC cache, since it has also shown to increase traffic significantly in some cases.

In order to show how the use of these techniques influences the cache miss-rate, we have measured how many additional cache misses are generated by these caches. From this, we then subtract the number of misses generated by the conventional direct-mapped cache, and divide the result by the total amount of memory accesses to render the increase in miss-rate. This increase in miss-rate can also be found by simply subtracting conventional cache's

Benchmark	256B	512B	1kB	2kB	4kB	8kB	16kB	32kB
adpcm-dec	66	65	56	50	67	54	-11	-11
adpcm-enc	73	68	60	40	64	41	0	0
jpeg-dec	60	57	51	50	31	7	9	-4
jpeg-enc	63	64	63	54	50	10	-5	-8
mpeg2-dec	19	27	21	16	38	34	-75	-17
g721-dec	-13	-23	-3	-10	-14	-26	-2	-3
g721-enc	10	9	13	12	0	64	75	75
pegwit-dec	4	0	-1	-1	-1	0	0	0
pegwit-enc	5	0	-1	-2	-2	-1	-1	0
gsm-dec	52	47	49	9	16	28	-7	-8
gsm-enc	28	25	17	14	1	0	-3	-4
epic	32	19	12	8	8	23	8	1
unepic	34	33	31	28	22	24	24	25

TABLE I
PERCENTAGES REDUCTION IN TRAFFIC BY USING THE BCC CACHE.

Benchmark	256B	512B	1kB	2kB	4kB	8kB	16kB	32kB
adpcm-dec	64	62	53	43	58	26	-11	-11
adpcm-enc	72	67	58	39	63	19	0	0
jpeg-dec	58	54	48	45	28	7	9	-6
jpeg-enc	58	58	58	51	50	11	-5	-8
mpeg2-dec	23	34	29	24	69	66	37	5
g721-dec	9	8	-2	-7	-11	12	-2	-3
g721-enc	23	28	36	28	22	55	50	50
pegwit-dec	5	2	2	1	2	2	2	1
pegwit-enc	6	3	2	1	1	1	1	1
gsm-dec	50	43	45	10	20	25	-6	-8
gsm-enc	33	27	16	15	5	0	-3	-4
epic	29	19	12	9	10	35	24	15
unepic	29	29	27	25	21	22	23	23

TABLE II
PERCENTAGES REDUCTION IN TRAFFIC BY USING THE SCC CACHE.

miss-rate from the new one.

In these tables, many fields contain a zero value. In these cases, the number of additional misses generated my either the BCC or the SCC cache is less than a half percent of the total number of memory references. A negative value in these tables indicates that the conflict detection mechanism was able to reduce the amount of misses is in the cache.

For some benchmarks, the BCC cache can cause up to 22% of additional misses in very small caches. For larger caches, the difference in miss-rates is only significant in few cases. Using the SCC cache, we found a maximum of 7% of additional misses in very small caches. For larger caches, we found hardly any significant difference in miss-rate. Overall, the SCC cache was found to increase the miss-rate as often as it would decreases this number.

From this, we can conclude that use of the *Conflict Detection Table* only increases the number of cache misses slightly in case of the BCC cache. With the SCC cache, the number of misses might as well be less, as it might be more than the conventional direct-mapped cache. In either case, the difference in miss-rate is be small.

Benchmark	256B	512B	1kB	2kB	4kB	8kB	16kB	32kB
adpcm-dec	-4	-1	0	0	0	0	0	0
adpcm-enc	-2	-1	0	0	0	0	0	0
jpeg-dec	-4	-3	-3	-4	0	0	0	0
jpeg-enc	-3	-4	-5	-3	-2	1	0	0
mpeg2-dec	22	20	16	12	9	7	6	0
g721-dec	22	22	1	1	1	0	0	0
g721-enc	22	22	1	1	0	0	0	0
pegwit-dec	13	11	7	5	3	2	1	0
pegwit-enc	16	12	9	6	3	2	1	1
gsm-dec	-2	-1	-1	0	0	0	0	0
gsm-enc	4	1	0	0	0	0	0	0
epic	1	5	6	6	5	4	4	3
unepic	12	8	6	2	0	-1	-1	-1

TABLE III

ADDITIONAL CACHE-MISSES IN PERCENTAGES, CAUSED BY BYPASSING IN THE BCC CACHE.

Benchmark	256B	512B	1kB	2kB	4kB	8kB	16kB	32kB
adpcm-dec	-9	-5	-3	0	0	0	0	0
adpcm-enc	-10	-5	-3	0	0	0	0	0
jpeg-dec	-6	-3	-1	0	-1	0	0	0
jpeg-enc	-2	-1	-2	-1	-5	0	0	0
mpeg2-dec	0	-3	-1	-2	-2	-2	0	0
g721-dec	4	5	1	1	1	0	0	0
g721-enc	5	5	0	0	0	0	0	0
pegwit-dec	6	4	3	1	1	0	0	0
pegwit-enc	7	5	3	2	1	1	0	0
gsm-dec	-2	-1	-1	0	0	0	0	0
gsm-enc	-1	0	0	0	0	0	0	0
epic	-1	0	0	0	0	-1	0	0
unepic	8	5	5	3	0	0	0	0

TABLE IV

ADDITIONAL CACHE-MISSES IN PERCENTAGES, CAUSED BY FETCHING SUB-BLOCKS IN THE SCC CACHE.

V. CONCLUSIONS

We have proposed a technique which can detect and is sometimes able to resolve recurring conflict misses in caches. The proposed technique employs a small structure called the *Conflict Detection Table* (CDT). This conflict detection mechanism does not require much logic and does not increase the cycle time. Therefore, it can easily be applied to on-chip caches that lack associativity.

We proposed two caches that employ the *Conflict Detection Table*: the *Bypass in Case of Conflict* (BCC) cache and the *Sub-block in Case of Conflict* (SCC) cache. Experiments with these caches showed that the amount of off-chip traffic can be reduced significantly by using the CDT.

With the BCC cache, the amount of produced traffic can be reduced significantly, compared to conventional direct-mapped caches. However, it was also shown that this cache can increase the amount of traffic as well, in some cases. Furthermore, the miss-rate can suffer badly from inefficiently bypassing of the cache.

The SCC cache also showed to significantly decrease the amount of produced traffic for in most cases. Only in few cases, the SCC produced more traffic than a conven-

tional direct-mapped cache. Furthermore, these increases are not as significant as the commonly found reductions. For the SCC cache, the miss-rate was never found to be significantly different from that of a conventional direct-mapped cache, and could as well go down as it could go up.

We conclude that use of the *Conflict Detection Table* to fetch sub-block into the cache, instead of whole cache lines, can significantly reduce the amount of produced traffic, and hence also the amount of power consumption. In general, use of the SCC reduces the amount of traffic and therefore also power consumption, without loss of performance.

REFERENCES

[1] D. Burger, J.R. Goodman, and A. Kägi. Memory Bandwidth Limitations of Future Microprocessors. In *Proc. Int. Symp. on Computer Architecture*, pages 78–89, 1996.

[2] F. Catthoor, K. Danckaert, Ch. Kulkarni, E. Brockmeyer, P.G. Kjeldsberg, T. Van Achteren, and Th. Omnes. *Data Access and Storage Management for Embedded Programmable Processors*. Kluwer Academic Publishers, 2002.

[3] F. Catthoor, F. Franssen, S. Wuytack, L. Nachtergaele, and H. De Man. Global Communication and Memory Optimizing Transformations for Low-Power Signal Processing Systems. In *VLSI Signal Processing Workshop*, 1994.

[4] P.J. de Langen and B.H.H. Juurlink. Off-chip memory traffic measurements of low-power embedded systems. In *Proc. ProRISC Workshop on Circuits, Systems and Signal Processing*, pages 351–358, November 2002.

[5] A. González, C. Aliagas, and M. Valero. A Data Cache with Multiple Caching Strategies Tuned to Different Types of Locality. In *Proc. Int. Conf. on Supercomputing*, pages 338–347, 1995.

[6] M.R. Guthaus, J.S. Ringenberg, D. Ernst, T.M. Austin, T. Mudge, and R.B. Brown. Mibench: A free, commercially representative embedded benchmark suite. In *IEEE 4th Annual Workshop on Workload Characterization*, 2001.

[7] Teresa L. Johnson and Wen mei W. Hwu. Run-time adaptive cache hierarchy management via reference analysis. In *Proc. Int. Symp. on Computer Architecture*, pages 315–326, 1997.

[8] N.P. Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *Proc. Int. Symp. on Computer Architecture*, pages 364–373, June 1990.

[9] Ch. Lee, M. Potkonjak, and W.H. Mangione-Smith. MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communicatons Systems. In *Int. Symp. on Microarchitecture*, pages 330–335, 1997.

[10] G. Memik, G. Reinman, and W.H. Mangione-Smith. Reducing energy and delay using efficient victim caches. In *Proc. Int. Symp. on Low Power Electronics and Design*, pages 262–265. ACM Press, 2003.