Reducing Traffic Generated by Conflict Misses in Caches[*]

Pepijn J. de Langen
pepijn@ce.et.tudelft.nl

Ben Juurlink
benj@ce.et.tudelft.nl

Computer Engineering Laboratory
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
P.O.Box 5031, 2600 GA Delft, The Netherlands

ABSTRACT

Off-chip memory accesses are a major source of power consumption in embedded processors. In order to reduce the amount of traffic between the processor and the off-chip memory as well as to hide the memory latency, nearly all embedded processors have a cache on the same die as the processor core. Because small caches dissipate less power and are cheaper than large caches, a small cache is preferable to a large cache. Furthermore, because set-associative caches consume more power than direct-mapped caches, a direct-mapped cache is preferable to a set-associative one. Small, direct-mapped caches generally incur many conflict misses, however. In this paper we propose and evaluate a structure called the *Conflict Detection Table* (CDT). This table can be used to determine if a memory access is expected to hit the cache. If a hit is expected and a miss occurs, then a conflict is detected and appropriate action can be taken. In addition, we propose two cache structures that employ this technique: the *Bypass in Case of Conflict* (BCC) cache and the *Sub-block in Case of Conflict* (SCC) cache. The BCC cache bypasses the cache when a conflict is detected, whereas the SCC cache fetches a sub-block of the missing cache block in such a case. Experimental results using several embedded workloads show that the BCC and SCC cache reduce the amount of traffic significantly in many cases. Furthermore, overall they incur the same number of cache misses as the direct-mapped cache. This shows that the BCC and SCC cache reduce the amount of power consumed with a negligible reduction in performance.

Categories and Subject Descriptors

B.3.2 [**Memory Structures**]: Design Styles—*Cache memories*

General Terms

Design, Measurement, Performance

Keywords

Caches, conflict misses, power reduction, embedded processors

1. INTRODUCTION

Off-chip memory transfers consume a significant amount of power, often more than the datapaths and the control units [2]. In order to limit the amount of off-chip memory traffic, it is essential that embedded processors make effective use of the on-chip cache. Embedded processors often exploit a small, direct-mapped cache, because small caches are more power efficient and are cheaper than large caches, and because set-associative caches dissipate more power than direct-mapped caches [11]. Small, direct-mapped caches, however, generally produce many conflict misses and, as a result, generate a significant amount of processor-memory traffic [3].

In this paper we present a novel technique to detect and eliminate conflict misses in the first level cache. The structure we propose is called the *Conflict Detection Table* (CDT). The CDT contains the tag part of the addresses referenced by recently executed load/store instructions and is indexed by the lower-order bits of the program counter. The idea behind the CDT is the following. If an entry corresponding to a load/store instruction is found in the CDT and the data tag stored in this entry matches the tag of the current data address, a (spatial) hit is expected because the referenced word was loaded in the cache the previous time this instruction was executed. Furthermore, if a hit is expected but the cache access yields a miss, then a conflict is detected because the word must have been replaced by another instruction.

We propose two cache structures that employ the CDT. The first, called the *Bypass in Case of Conflict* (BCC) cache, bypasses the cache when a conflict is detected. The second, called the *Sub-block in Case of Conflict* (SCC) cache, is a sector cache that fetches only the missing sector (or sub-block) when a conflict is detected. Both the BCC as well as the SCC cache are direct-mapped.

This paper is organized as follows. Section 2 briefly discusses related work. In Section 3 we explain how recurring conflict misses can be detected and appropriate action can be taken. This is experimentally verified in Section 4. Conclusions are given in Section 5.

[*] This research was supported in part by the Netherlands Organisation for Scientific Research (NWO).

CF'04, April 14–16, 2004, Ischia, Italy.
Copyright 2004 ACM 1-58113-741-9/04/0004 ...$5.00.

2. RELATED WORK

Jouppi [8] proposed employing a small (consisting of four to eight entries), fully associative *victim cache* in order to reduce conflict misses in direct-mapped caches. 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. [4] includes a mechanism that detects if a load instruction interferes with itself. This happens, for example, when a vector is accessed repeatedly and the vector is larger than the cache. In such a case, the vector displaces itself from the cache. This situation is even worse when the vector is accessed with a stride unequal to one and the stride and the cache size are not co-prime, because in this case not all blocks are used to cache the vector. This mechanism, however, does not detect cross-interference, i.e., it does not discover situations in which 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.

Tam [12] proposed the *Allocation By Conflict* (ABC) replacement policy. In this organization a 1-bit counter is associated with each cache block, which is increased (decreased) each time an acces to this block yields a miss (hit). A block is evicted from the cache only when two consecuetive accesses produce a miss. A comparison between this cache and the BCC and SCC caches will also be part of future work.

There are also static (compiler) approaches aimed at reducing conflict misses. For example, Catthoor et al. [1] 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.

3. CONFLICT DETECTION AND ELIMINATION

A conflict miss occurs when a memory word is referenced twice but is in between replaced by another word. Conflict misses occur frequently in direct-mapped caches, because each memory word is mapped to only one cache location. Consider, for example, the following simple loop

```
for (i=0; i<n; i++)
    a[i] = b[i]+c[i];
```

If the differences between the base addresses of the arrays a, b, and c are a multiple of the cache size, each a[i], b[i], and c[i] all map to the same cache line and each will replace the other in the cache so that there is no chance to exploit the spatial locality exhibited by this code. Such 'ping-pong' effects will degrade the cache performance severely.

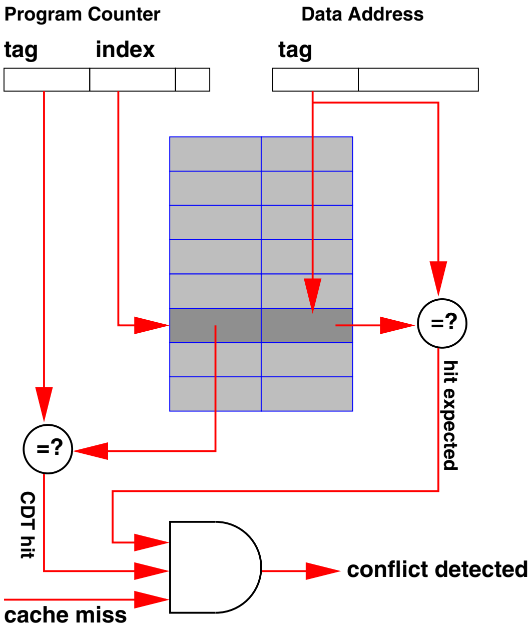

Figure 1: Conflict Detection Table (CDT)

To detect conflict misses in direct-mapped caches, we propose a small structure called the *Conflict Detection Table* (CDT). The principle idea behind the CDT is that when consecutive executions of a load/store instruction access the same cache line, a cache hit is expected for all but the first access. If a hit is expected but a cache miss occurs, a different instruction must have accessed a word that is mapped to the same cache line.

As illustrated in Figure 1, the CDT is a cache that is indexed by the lower-order bits of the program counter. Every entry contains the higher-order bits of the instruction address to determine if an access produces a hit and the tag of the address referenced the previous time the corresponding load/store instruction was executed. Each time a load/store instruction is executed, the CDT is accessed to determine if there is an entry for the current instruction. If no entry is found, one is allocated and the data tag field is set to the tag of the current address. If an entry is found, the data tag stored in this entry is compared to the tag of the current address. If they match, we expect that the requested data is already present in the cache, because it was fetched the last time this instruction was executed. From this it follows that if a cache miss is encountered, the requested data must have been replaced by a different instruction and a conflict is detected. In any case, the tag of the current address is stored in the CDT.

A conflict can only be detected when a cache miss occurs. The information about possible conflicts, therefore, does not need to be available until after the tag comparison. This implies that the CDT will not increase the time to hit the cache. Furthermore, since the amount of required logic to implement the CDT is fairly low, it will not consume much power.

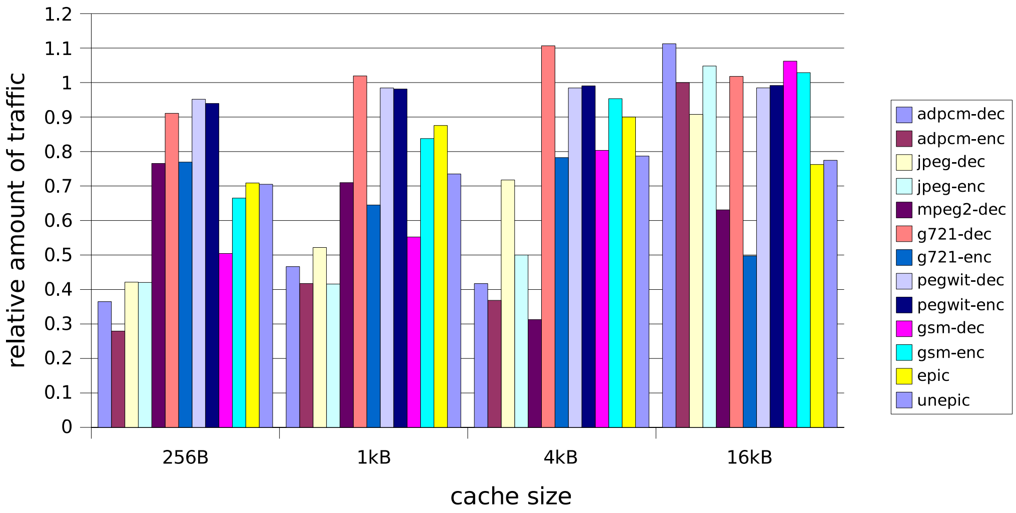

Figure 2: Amount of traffic produced by the SCC cache, relative to the amount of traffic produced by a conventional, direct-mapped cache.

When a conflict is detected, it is not known in advance which instruction will be the first to re-use this cache line. Therefore, for reducing the miss rate, it is not certain what will be most efficient: replacing the current line or bypassing the cache. For reducing the amount of traffic, however, it is almost always more efficient to fetch a smaller number of bytes instead of a whole cache line. Traffic can, therefore, be reduced by fetching only the requested word instead of the whole cache line if a conflict is detected.

We propose two cache structures that employ the DCT to detect and eliminate conflict misses. Both caches are direct-mapped but have the additional possibility to bypass the cache or to store only part of the requested cache line. The first one employs *sub-block* caching [6] and is called the *Sub-block in Case of Conflict* (SCC) cache. It fetches and stores only the missing sub-block when the CDT indicates a conflict miss. The second one uses cache bypassing and is called the *Bypass in Case of Conflict* (BCC) cache. If the CDT detects that the cache line has been replaced by another instruction, this cache will not store the following words referenced by this instruction, as long as the instruction references the same cache line.

4. EXPERIMENTAL VALIDATION

In this section we experimentally verify if the BCC and SCC cache reduce the amount of off-chip memory traffic without a significant performance degradation.

4.1 Experimental Setup

As benchmarks, we employed 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 representative of embedded multimedia applications. The *MiBench* [5] benchmarking suite, which is specifically aimed at embedded systems and also contains workloads from other application domains, was not available at the time this project was started. Moreover, *MediaBench* and *MiBench* have several benchmarks in common.

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

The cache size ranges from 256 bytes to 16 kilobytes. All caches have a line size of 32 bytes, are direct-mapped, and employ the write-back policy. The sub-block size of the SCC cache is equal to the word size. For the *Conflict Detection Table*, we have used a direct-mapped structure with 128 entries. We measured the total amount of traffic between the cache and main memory, including request (address) traffic.

For traffic, we will show the relative changes between the new (BCC/SCC) and the original direct-mapped caches. For miss rates, however, relative differences do not provide proper information. If, for example, in one case the miss rate increases from 1% to 2%, and in another case it increases from 40% to 80%, the performance penalty is far more severe with the latter one than with the first one. Therefore, one should consider absolute differences between the miss rates of two caches rather than relative differences.

4.2 Results

Figure 2 and Figure 3 depict the amount of traffic produced by the SCC cache and the BCC cache, respectively, for various benchmarks and cache sizes. In both figures, the amount of traffic is normalized to the amount of traffic generated by the conventional, direct-mapped cache.

It can be seen that in most cases both the SCC cache and the BCC cache produce significantly less traffic than a direct-mapped cache. Specifically, in 85% of all benchmark/cache size combinations, the SCC cache produces less traffic than the direct-mapped cache. The BCC cache improves upon the direct-mapped cache in 67% of all cases. Especially when the cache size is small, a traffic reduction of more than 50% can be achieved by the BCC as well as the SCC cache. On average, the amount of traffic produced by the SCC cache is 25% smaller than the amount of traffic generated by the direct-mapped cache. For the BCC

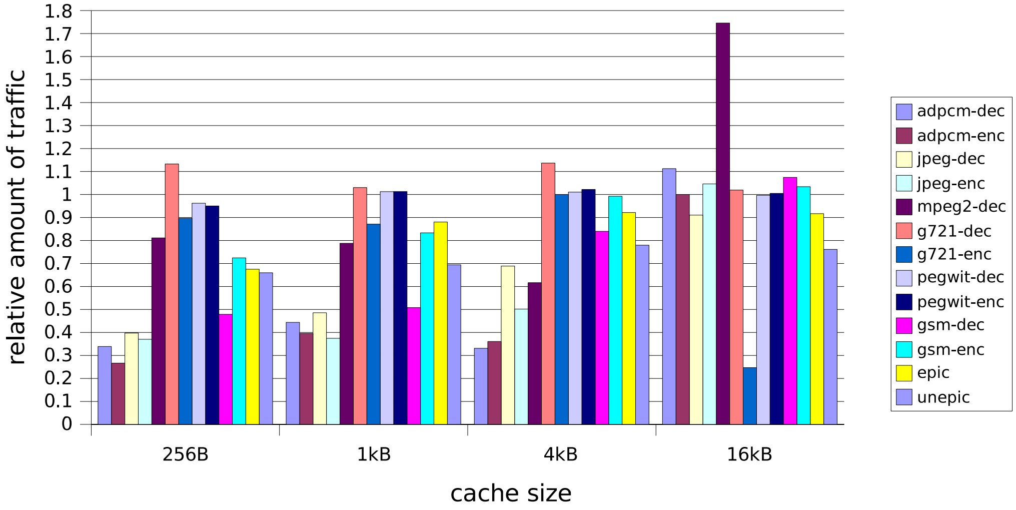

Figure 3: Amount of traffic produced by the BCC cache, relative to the amount of traffic produced by a conventional, direct-mapped cache.

cache, the average reduction is 21%. For larger cache capacities, fewer benchmarks benefit from the proposed conflict detection technique. In a small number of cases, the BCC and the SCC cache actually produce more traffic than the direct-mapped cache. In one case, namely for the `mpeg2-dec` benchmark using the BCC cache of 16 kilobytes, the BCC cache produces 75% more traffic than the direct-mapped cache. A possible explanation is that if a certain instruction decides to bypass the cache, all further executions of this instruction will also bypass the cache as long as this instruction remains in the CDT and accesses the same cache line. However, the data stored in the cache may no longer be needed. The SCC cache never generates more than 11% as much traffic as the direct-mapped cache.

The efficacy of the BCC and SCC cache is clearly very dependent on the type and amount of locality that is exhibited by an application. For some benchmarks, in particular `pegwit-dec` and `pegwit-enc`, they do not significantly produce less traffic than direct-mapped caches. In most cases, however, they reduce the amount of off-chip traffic considerably and, hence, the amount of energy consumed by an application. We further remark that although the BCC cache produces the least amount of traffic for some combinations of benchmarks and cache sizes, it is less effective than the SCC cache since it also increases the amount of traffic significantly in some cases.

The results discussed above show that the BCC and the SCC cache reduce the amount of off-chip memory traffic. However, if they would increase the miss rate significantly, no energy reduction would be achieved. To validate this, Figure 4 and Figure 5 depict the miss rates generated by the direct-mapped cache, the BCC cache, and the SCC cache, for cache capacities of 1kB and 4kB, respectively. It can be seen that in most cases the miss rates of the BCC cache and the SCC cache are comparable to the miss rate of the direct-mapped cache. In some cases, however, the miss rate of the BCC cache is significantly larger than the miss rate of the direct-mapped cache. In one case, for `mpeg2-dec` using a cache size of 1kB, it is 16% larger. As explained above, in this case bypassing the cache is clearly not effective. This also explains why the BCC produces the large amount of traffic for the `mpeg2-dec` benchmark. The SCC cache never incurs more than 8% more misses than the direct-mapped cache. Furthermore, in several cases the SCC cache performs better than the direct-mapped cache. On average, they perform equally.

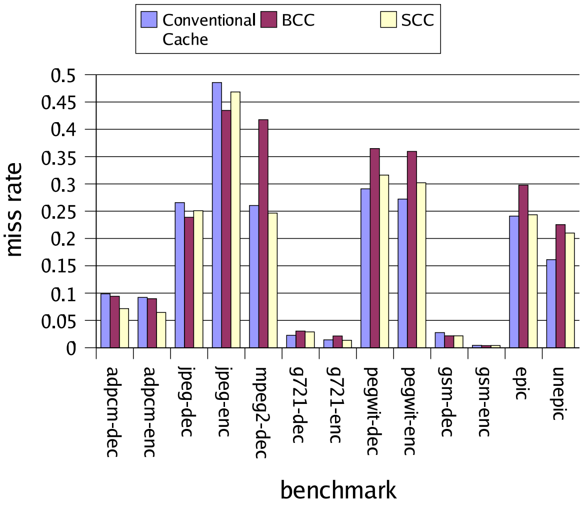

Figure 4: Miss-rates of various caches of 1kB

We conclude that the SCC cache is the most efficacious cache structure. It generates significantly less off-chip traffic than a direct-mapped cache, while performing equally as well.

5. CONCLUSIONS

We have proposed a technique which can detect and is often able to reduce the negative effects of recurring conflict misses. The proposed technique employs a small struc-

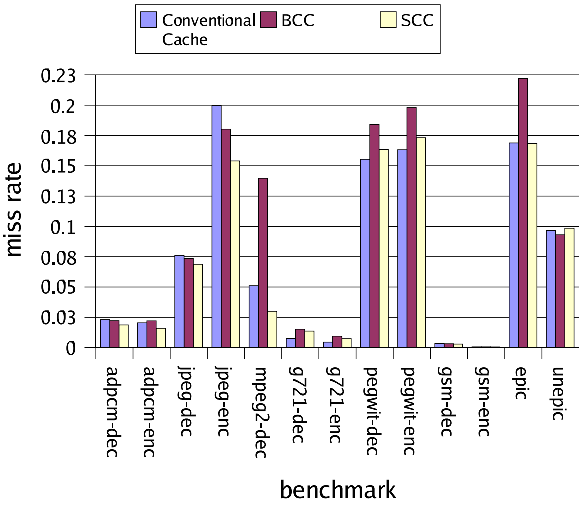

Figure 5: Miss-rates of various caches of 4kB

ture called the *Conflict Detection Table* (CDT). This conflict detection mechanism does not require much logic and we, therefore, estimate that it will not increase the cycle time. Consequently, it can easily be applied to on-chip caches that lack associativity. We have 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.

On average, the BCC cache decreases the amount of produced traffic significantly compared to the conventional direct-mapped cache. It was also shown, however, that this cache sometimes increases the amount of traffic. Furthermore, the miss rate can suffer badly from inefficiently bypassing the cache. The SCC cache also decreases the amount of produced traffic considerably in most cases. Only in a few cases, the SCC produced more traffic than a conventional direct-mapped cache. Furthermore, these increases are small. In addition, the miss rate of the SCC cache is never considerably higher than that of the conventional direct-mapped cache. On average, the SCC incurs as many cache misses as the conventional direct-mapped cache. We conclude that using the *Conflict Detection Table* to fetch sub-blocks into the cache instead of whole cache lines, significantly decreases the amount of produced traffic, and hence also the amount of power consumed.

As future work, we intend to compare these results with other dynamic caching techniques. Furthermore, a detailed power model should be used to provide more detailed information on the actual energy reduction. The conflict detection can be extended with counters to provide a better decision on what to cache. Finally, tuning the size and associativity of the *Conflict Detection Table* may improve the amount of produced traffic and the miss rate.

6. REFERENCES

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

[2] 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 *Proc. VLSI Signal Processing Workshop*, 1994.

[3] P. de Langen and B. Juurlink. Off-Chip Memory Traffic Measurements of Low-Power Embedded Systems. In *Proc. ProRISC Workshop on Circuits, Systems and Signal Processing*, pages 351–358, 2002.

[4] 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.

[5] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown. MiBench: A Free, Commercially Representative Embedded Benchmark Suite. In *Proc. Annual Workshop on Workload Characterization*, 2001.

[6] J. Hennessy and D. Patterson. *Computer Architecture (3rd ed.): A Quantitative Approach*. Morgan Kaufmann Publishers Inc., 2003.

[7] T. L. Johnson and W. 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. 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, 1990.

[9] C. Lee, M. Potkonjak, and W. 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. 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.

[11] G. Reinman and N. Jouppi. An Integrated Cache Timing and Power Model. Technical report, COMPAQ Western Research Lab, Palo Alto, California, 1999.

[12] E. Tam. *Improving Cache Performance Via Active Management*. PhD thesis, University of Michigan, Ann Arbor, 1999.