An Address Transformation Combining Block- and Word-Interleaving

Hans Vandierendonck and Koen De Bosschere
Dept. of Electronics and Information Systems
Ghent University, Sint-Pietersnieuwstraat 41, 9000 Gent, Belgium
E-mail: {hvdieren,kdb}@elis.rug.ac.be

Abstract— **As future superscalar processors employ higher issue widths, an increasing number of load/store-instructions needs to be executed each cycle to sustain high performance. Multi-bank data caches attempt to address this issue in a cost-effective way. A multi-bank cache consists of multiple cache banks that each support one load/store-instruction per clock cycle. The interleaving of cache blocks over the banks is of primary importance. Two common choices are block-interleaving and word-interleaving. Although word-interleaving leads to higher IPC, it is more expensive to implement than block-interleaving since it requires the tag array of the cache to be multi-ported.**

By swapping the bits in the effective address that are used by word-interleaving with those used by block-interleaving, it is possible to implement a word-interleaved cache with the same cost, cycle time and power consumption of a block-interleaved cache. Because this makes the L1 data cache blocks sparse, additional costs are incurred at different levels of the memory hierarchy.

Keywords— **Data cache, Multi-Banking, Block-Interleaving, Word-Interleaving.**

I. Introduction

Every new generation of superscalar processors features higher issue widths and larger instruction windows. In order to leverage this high potential instruction-level parallelism, multiple load/store-instructions have to be executed per clock cycle. The number of load/stores that can simultaneously execute is limited by the number of cache ports, i.e. the data path through which a cache memory is accessed.

A perfect multi-ported cache can execute any combination of load/stores simultaneously. Such caches are however not scalable to a high number of cache ports. Multi-bank caches were proposed to be a scalable solution [1]. In a multi-bank cache, the data is split over a number of single-ported banks that can each handle one load/store-instruction per clock cycle. The performance of a multi-bank cache is limited by bank conflicts. A bank conflict occurs when two load/stores try to access the same bank in the same cycle. As a result, the execution of one of the two instructions needs to be delayed.

The number of bank conflicts depends on the way the data is *interleaved* over the cache banks. Two schemes are commonly used. *Block-interleaving* places a whole cache block in the same bank and places consecutive blocks in consecutive banks (Figure 1, left). *Word-interleaving* spreads a cache block over multiple banks thereby placing consecutive words[1] into consecutive banks (Figure 1, middle). Word-interleaving generally allows higher instruction-level parallelism, because the presence of spatial locality makes it very likely that references to the same cache block are executed shortly after one another [2]. In a block-interleaved multi-bank cache, these references are serialised, since only one access to the same block is possible per clock cycle.

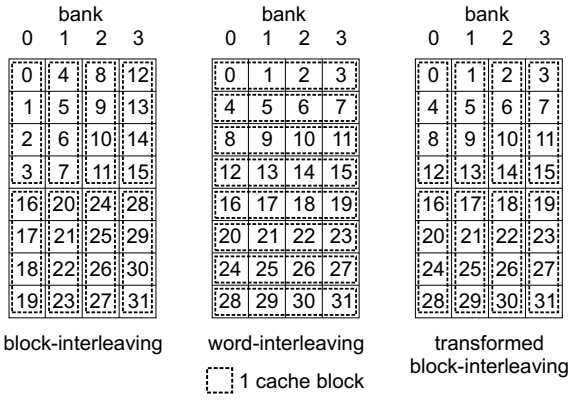

Fig. 1. Interleaving schemes in multi-bank caches.

Word-interleaved caches must support multiple tag comparisons for the same cache block. This shifts the problem of supporting multiple accesses from the data array to the tag array, making word-interleaving a less desirable solution. By multi-porting or duplicating the tag array, the required chip area, power consumption and cycle time of the cache are increased.

This paper explores a multi-bank cache design that provides the ILP benefits of word-interleaving at the same complexity as a block-interleaved cache. We achieve this by transforming the effective address before accessing a block-interleaved multi-bank cache. As a side effect, the L1 data cache blocks become sparse but, surprisingly, this can be beneficial because it sometimes *reduces* the cache miss rate and increases the parallelism in the memory system.

The remainder of this paper is organised as follows. We

Manuscript submitted: 17 Apr. 2002. Manuscript accepted: 23 May 2002. Final Manuscript received: 3 Jun. 2002.

[1] If the processor can load at most N bytes from the cache at once, then every group of N aligned bytes should be stored in the same bank. This ensures that only one data cache access is required per load instruction. We call this quantity a *word*. It is assumed to be 8 bytes in this paper.

discuss the address transformation and its interaction with the L2 cache in Section II. In Section III we show where the address transformation is applied in a microprocessor and in Section IV we present the results of a preliminary evaluation. Section V concludes the paper.

II. Address Transformation

Block-interleaving and word-interleaving differ by the address bits that are used to select the bank number (Figure 2). The bits selected by block-interleaving allow for a simpler and cheaper implementation. On the other hand, the bits selected by word-interleaving vary more between successively executed load-instructions and hence result in fewer bank conflicts. If we swap these two groups of bits prior to accessing a block-interleaved multi-bank cache (Figure 2, bottom), then we combine the benefits of block-interleaving (selecting the bank using address bits outside the block offset) with the benefits of word-interleaving (selecting the bank using rapidly varying address bits). The resulting layout of words over the cache banks is shown in Figure 1, right.

The address transformation has the side effect that the cache blocks become sparse. For most applications, this increases the cache miss rate of the L1 data cache. Hence, we may not succeed in achieving the same performance as the word-interleaved cache with duplicated or multi-ported tag arrays, but the evaluation shows that we end up close to that.

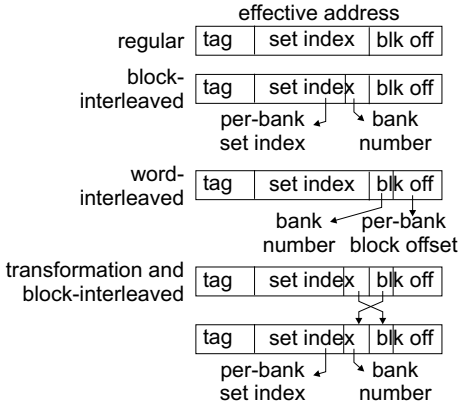

Fig. 2. Address calculation in regular and multi-bank caches.

So far, we have shown that the transformed (word-interleaved) addressing scheme can be implemented at the same cost of a block-interleaved cache. Sparse cache blocks however impact the design of the L2 cache. The L2 cache is typically shared between instructions and data. We expect a strong increase in the miss rate if we store instructions in sparse cache blocks. Hence we maintain contiguous blocks in the L2 cache. This has certain implications for the allowable block sizes. Let's assume the L1 data cache is 4-way banked and has $w = 4$ words per (sparse) cache block of B_1 bytes. The L1 instruction cache also has $w = 4$ words per cache block. Figure 3 shows which words belong to the same instruction and data cache blocks. For a word at address a, the data block consists of 4 separated words at word addresses a, $a + 4$, $a + 8$ and $a + 12$. If, however, an instruction block has to be fetched from address a, it consists of different words, i.e.: a, $a + 1$, $a + 2$ and $a + 3$. The L2 cache has to be aware whether it is fetching a data or an instruction block, so that the correct words can be returned.

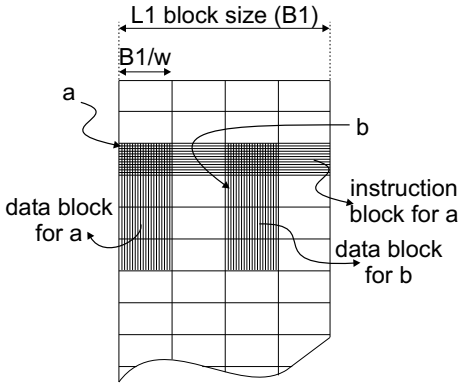

Fig. 3. Layout of L1 data and instruction blocks in memory.

It is desirable that each L1 cache block is a part of exactly one L2 cache block. Otherwise, we run into the problem that loading a cache block from L2 requires multiple L2 accesses. For a L2 cache block size of B_2 bytes, this corresponds to $B_2 \geq M \cdot B_1$, with M the number of banks in the L1 data cache. E.g.: with $M = 4$ and $B_1 = 64$ we find $B_2 \geq 256$. This choice will be far from optimal in many cases, since it can impose higher L2 cache miss rates and/or cause high L2 cache miss latencies, The desired situation is obtained when using sub-blocks in the L1 data cache. For sub-blocks of size SB_1, it suffices that $B_2 \geq M \cdot SB_1$. Sub-blocks of the same L1 block can still be placed in different L2 blocks. We use sub-blocks in the L1 data cache in our initial evaluation, because many systems already apply this.

III. Evaluation Environment

We target the interleaving technique to an aggressive dynamically scheduled superscalar processor that is organised around an instruction window and a separate load/store-queue (Figure 4). Sparse cache blocks are implemented by transforming the effective address and accessing the block-interleaved multi-bank cache with the transformed address. The transformation is applied when the load/store-instruction accesses the L1 data cache. When data is fetched from/written back to the L2 cache, the address is restored to its original form, such that the scope of the transformed addresses is limited to the light-gray box in Figure 4. The L2 cache has to be aware of the address transformation in the L1 data cache, in order to return the correct data on a miss (dark-gray box).

The processor configuration used in the evaluation is summarised in Table I. We simulated 4 SPECfp2000 and 4 SPECint2000 programs. Each benchmark was run for 500 million instructions. Traces were selected using the tech-

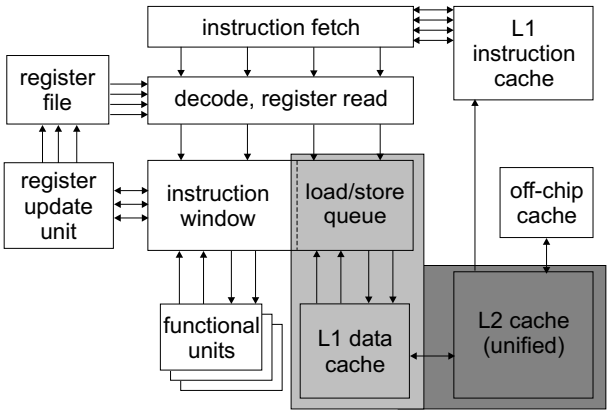

Fig. 4. Block diagram of a superscalar processor with out-of-order execution.

TABLE I
PROCESSOR CONFIGURATION.

Parameter	Value
L1 instruction cache	32 kB, 2 ways, 64 B blocks
fetch queue size	32
branch predictor	perfect
degree	8
RUU size	512
LSQ size	256
MSHRQ size	256
ALU's	8
simple INT latencies	1
INT mult./divide	7/12
FP add,cmp/mult.	1/4
FP divide/sqrt	12/18
load/store units	4
L1 data cache	32 kB, 2 ways, 64 B blocks, 32 B sub-blocks
L2 cache	128 kB, 4 ways, 128 B blocks
L2 cache latency	10 cycles
memory latency	15 + 3 cycles/8 bytes

nique described in [3], which guarantees that the program initialisation phase is skipped.

IV. Experimental Evaluation

We evaluated 4 multi-porting schemes. Perfect multi-porting assumes that any combination of loads and stores can access the cache simultaneously. Block- and word-interleaving were explained before and the block-interleaved cache with transformation (TF) is labeled "TF/Block". We did not incorporate the increased cycle time of the word-interleaved cache in our evaluation.

The IPC results (Figure 5) reveal that many programs benefit from word-interleaving over block-interleaving. With the address transformation, the performance is located between that of the block-interleaved and the word-interleaved scheme. Art and equake are exceptions because their miss rates benefit greatly from the differently shaped cache blocks. Crafty performs worse because of an increase in the miss rate.

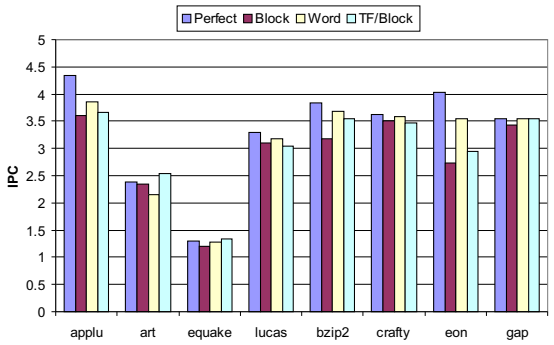

Fig. 5. IPC for different interleaving schemes.

For each attempt to issue a ready instruction, we counted the number of times a bank conflict occurred (Figure 6). These results confirm that word-interleaving performs better than block-interleaving because of fewer bank conflicts. The address transformations incur additional conflicts because multiple accesses to the same transformed block are not allowed.

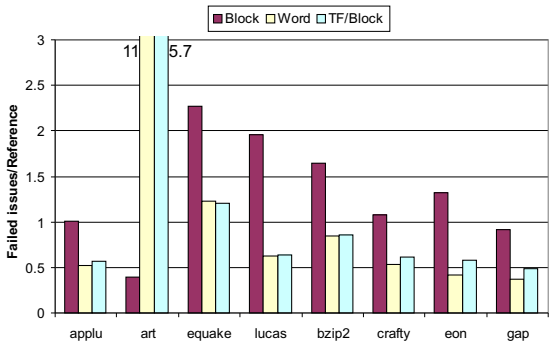

Fig. 6. Number of attempts to issue a load/store-instruction resulting in a bank conflict normalised to the number of executed load/store-instructions.

Although the sparse cache blocks may increase the number of primary cache misses (i.e. fetched blocks), the number of secondary misses (references to a block being fetched) often decreases (Figure 7). Both types of misses limit the attainable IPC. The strong decrease in secondary misses can often compensate the increase in primary misses.

V. Related Work

Sohi and Franklin investigated various aspects of data cache performance and proposed multi-bank caches to deliver high L1 data cache bandwidth [1]. Several techniques to avoid bank conflicts have been proposed. Access combining detects loads to the same cache block and

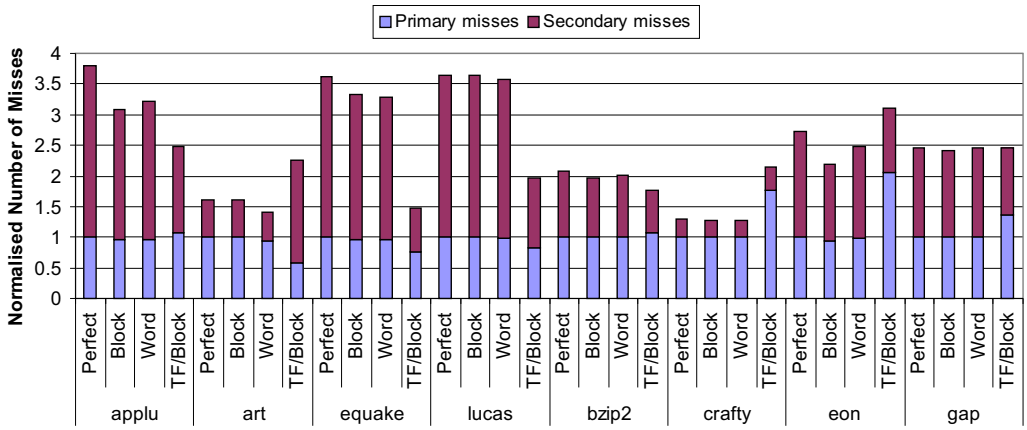

Fig. 7. Primary and secondary miss rates normalised to the primary miss rate for the perfect multi-ported cache.

performs one cache access to satisfy both [2], thereby resolving conflicting instructions that access the same cache block. In [4], various interleaving techniques are investigated as well as buffering techniques to reduce the penalty of bank conflicts. A cost/performance tradeoff is made, showing word-interleaved caches to be significantly more costly than block-interleaved caches. Bank conflicts can be avoided using bank prediction, however it performs only well in deeply pipelined machines [5].

The related techniques add extra buffers or caches in the critical path of the processor. Complexity is only added in the L2 cache in the proposed technique.

Other authors have previously studied the effects of manipulating data addresses to access multi-bank memories. There is a lot of work on XOR-based permutation schemes to interleave data over memory banks in vector processors [6], [7], [8], with the goal of mapping all elements in a strided memory access pattern onto different banks. A technique similar to the one proposed here reduces conflicts in DRAM row buffers [9].

Summary

This paper introduces an address transformation for block-interleaved multi-bank caches. By swapping the address bits used to select a bank with word-interleaving with those bits selecting a bank in block-interleaving before accessing the cache, we can combine the benefits of both techniques. As a result, the cache blocks become sparse. This introduces trade-offs between the hardware complexity, the frequency of bank conflicts and the miss rate. Surprisingly, the cache miss rates for some programs decrease when using sparse cache blocks. Furthermore, our results indicate that the amount of parallelism in the memory system is increased when using sparse cache blocks. Overall, the address transformation improves the performance of the block-interleaved multi-bank cache without increasing its complexity.

Acknowledgements

Hans Vandierendonck is supported by the Flemish Institute for the Promotion of Scientific-Technological Research in the Industry (IWT). The authors thank Smaïl Niar for his useful comments.

References

[1] G.S. Sohi and M. Franklin, "High-bandwidth data memory systems for superscalar processors," in *Proceedings of the 18th Annual International Symposium on Computer Architecture*, May 1991, pp. 53–62.

[2] J.A. Rivers, G.S. Tyson, E.S. Davidson, and T.M. Austin, "On high-bandwidth data cache design for multi-issue processors," in *Proceedings of the 30th Conference on Microprogramming and Microarchitecture*, 1997, pp. 46–56.

[3] T. Sherwood, E. Perelman, and B. Calder, "Basic block distribution analysis to find periodic behavior and simulation points in applications," in *Proceedings of theInternational Conference on Parallel Architectures and Compilation Techniques (PACT-2002)*, Sept. 2002, pp. 3–14.

[4] T. Juan, J.J. Navarro, and O. Teman, "Data caches for superscalar processors," in *ICS'97. Proceedings of the 1997 International Conference on Supercomputing*, July 1997, pp. 60–67.

[5] H. Neefs, H. Vandierendonck, and K. De Bosschere, "A technique for high bandwidth and deterministic low latency load/store accesses to multiple cache banks," in *Proceedings of the 6th International Symposium on High Performance Computer Architecture*, Jan. 2000, pp. 313–324.

[6] B. R. Rau, "Pseudo-randomly interleaved memory," in *Proceedings of the 18th Annual International Symposium on Computer Architecture*, May 1991, pp. 74–83.

[7] R. Raghavan and J. P. Hayes, "On randomly interleaved memories," in *SC90: Proceedings on Supercomputing '90*, Nov. 1990, pp. 49–58.

[8] M. Valero, T. Lang, J.M. Llaberia, M. Peiron, E. Ayguado, and J.J. Navarro, "Increasing the number of strides for conflict-free vector access," in *Proceedings of the 16th Annual International Symposium on Computer Architecture*, May 1989, pp. 372–381.

[9] Z. Zhang, Z. Zhu, and X. Zhang, "A permutation-based page interleaving scheme to reduce row-buffer conflicts and exploit data locality," in *Proceedings of the 33th Conference on Microprogramming and Microarchitecture*, Dec. 2000, pp. 32–41.