Dynamically Variable Line-Size Cache Exploiting High On-Chip Memory Bandwidth of Merged DRAM/Logic LSIs

Koji Inoue† Koji Kai‡ Kazuaki Murakami†
† Dept. of Computer Science and Comm. Eng.
Kyushu University
6–1 Kasuga-koen, Kasuga,
Fukuoka 816-8580 Japan.
ppram@c.esce.kyushu-u.ac.jp
‡ Institute of Systems & Information Technologies/KYUSHU
2-1-22 Momochihama, Sawara-ku,
Fukuoka 814-0001 Japan.
kai@k-isit.or.jp

Abstract

This paper proposes a novel cache architecture suitable for merged DRAM/logic LSIs, which is called “dynamically variable line-size cache (D-VLS cache)”. The D-VLS cache can optimize its line-size according to the characteristic of programs, and attempts to improve the performance by exploiting the high on-chip memory bandwidth. In our evaluation, it is observed that the performance improvement achieved by a direct-mapped D-VLS cache is about 27%, compared to a conventional direct-mapped cache with fixed 32-byte lines.

1 Introduction

For merged DRAM/logic LSIs with a memory hierarchy including cache memory, we can exploit high on-chip memory bandwidth by means of replacing a whole cache line at a time on cache misses [5][10][11]. This approach tends to increase the cache-line size if we attempt to improve the attainable memory bandwidth. In general, large cache lines can benefit some application as the effect of prefetching. Larger cache lines, however, might worsen the system performance if programs do not have enough spatial locality and cache misses frequently take place. This kind of cache misses (i.e., conflict misses) could be reduced by increasing the cache associativity[10][11]. But, this approach usually makes the cache access time longer.

To resolve the above-mentioned dilemma, we have proposed a concept of “variable line-size cache (VLS cache)”[5]. The VLS cache can alleviate the negative effects of larger cache-line size by partitioning the large cache line into multiple small cache lines. The performance of the VLS cache depends largely on whether or not cache replacements can be performed with adequate line-sizes. However, the paper[5] did not have discussed how to determine the adequate cache-line size. Several studies have proposed cache architectures for adjusting the cache-line size to the characteristics of programs. Coherent caches proposed in [1][2] focused on shared-memory multiprocessor systems. For uni-processor systems, cache architectures exploiting varying amount of spatial locality have been proposed in [3][6][7]. These caches need tables for recording the memory access history for evicted data from the cache, or situations of past load/store operations.

This paper proposes a realistic VLS cache architecture, which is referred to as “dynamically variable line-size cache (D-VLS cache)”. The D-VLS cache changes its cache-line size at run time according to the characteristics of application programs to execute. Line-size determinator selects adequate line-sizes based on recently observed data reference behavior. Since the algorithm for determining the adequate line-sizes is very simple, the D-VLS cache has no large tables for storing the memory access history. This scheme does not require any modification of instruction set architectures. The goal of D-VLS cache is to improve the system performance of merged DRAM/logic LSIs such as PPRAM(Parallel Processing RAM)[8] or IRAM(Intelligent RAM)[9] by making good use of the high on-chip memory bandwidth.

The rest of this paper is organized as follows. Section 2 describes the concept and principle of the VLS cache. Section 3 discusses the D-VLS cache architecture. Section 4 presents some simulation results and shows the performance improvements achieved by the D-VLS cache. Section 6 concludes this paper.
2 Variable Line-Size (VLS) Cache

2.1 Terminology

In the VLS cache, an SRAM (cache) cell array and a DRAM (main memory) cell array are divided into several subarrays. Data transfer for cache replacements is performed between corresponding SRAM and DRAM subarrays. Figure 1 summarizes the definition of terms. Address-block, or subline, is a block of data associated with a single tag in the cache. Transfer-block, or line, is a block of data transferred at once between the cache and main memory. The address-blocks from every SRAM subarray, which have the same cache-index, form a cache-sector. A cache-sector and an address-block which are being accessed during a cache lookup are called a reference-sector and a reference-subline, respectively. When a memory reference from the processor is found a cache hit, referenced data resides in the reference-subline. Otherwise, referenced data is not in the reference-subline but only in the main memory. A memory-sector is a block of data in the main-memory, and corresponds to the cache-sector. Adjacent-subline is defined as follows.

- It resides in the reference-sector, but is not the reference-subline.
- Its home location in the main-memory is in the same memory-sector as that of the data which is currently being referenced by the processor.

2.2 Concept and Principle of Operations

It has been referenced at least once since it was fetched into the cache.

To make good use of the high on-chip memory bandwidth, the VLS cache adjusts its transfer-block size according to the characteristics of programs. When programs have rich spatial locality, the VLS cache would determine to use larger transfer-blocks, each of which consists of lots of address-blocks. Conversely, the VLS cache would determine to use smaller transfer-blocks, each of which consists of a single or a few address-blocks, and could try to avoid cache conflicts. Since the VLS cache can avoid cache conflicts without increasing the cache associativity, the access time of it (i.e., hit time) is shorter than that of conventional caches with higher associativity[5].

When a memory access takes place, the cache tag array is looked up in the same manner as normal caches, except that every SRAM subarray has its own tag memory and the lookup is performed on every tag memory. On cache hit, the hit address-block has the required data, and the memory access performs on this address-block in the same manner as normal caches. Otherwise, one or more address-blocks are replaced on cache replacement. For the example VLS cache shown in Figure 1, there are three transfer-block sizes as follows:

- Minimum transfer-block size (see Figure 2 (a)).
- Medium transfer-block size (see Figure 2 (b)).
Maximum transfer-block size (see Figure 2 (c)).

3 Dynamically VLS(D-VLS) Cache

3.1 Architecture

The performance of the VLS cache depends heavily on how well the cache replacement is performed with optimal transfer-block size. However, the amount of spatial locality may vary both within and among program executions. The D-VLS cache changes its cache-line size at run time according to the characteristics of programs. The line-size determinator for the D-VLS cache selects adequate line-sizes based on recently observed data reference behavior.

Figure 3 illustrates the block diagram of a direct-mapped D-VLS cache with four subarrays. The address-block size is 32 bytes, and three transfer-block sizes (32 bytes, 64 bytes, and 128 bytes) are provided. Since it does not be allowed that the medium transfer-block (64-bytes line) could not be beyond 64-byte boundary in the 128-byte cache-sector, the number of combinations of address-blocks to be involved in cache replacements is just seven rather than fifteen. The cache lookup for determining cache hit or miss is carried out as follows:

1. The address generated by the processor is divided into the byte offset within an address-block, subarray field designating the subarray, index field used for indexing the tag memory, and tag field.

2. Each cache subarray has its own tag memory and comparator, and it can perform the tag-memory lookup using the index and tag fields independently with each other.

3. One of the tag-comparison results is selected by the subarray field of the address, and then the cache hit or miss is determined.

The D-VLS cache provides the following for optimizing the transfer-block sizes at run time:

- A reference-flag bit per address-block: This flag bit is reset to 0 when the corresponding address-block is fetched into the cache, and is set to 1 when the address-block is accessed by the processor. It is used for determining whether the corresponding address-block is an adjacent-subline. On cache lookup, if the tag of an address-block which is not the reference-subline matches the tag field of the address and if the reference-flag bit is 1, then the address-block is an adjacent-subline.

- A line-size specifier (LSS) per cache-sector: This specifies the transfer-block size of the corresponding cache-sector. As described in Section 2.2, each cache-sector is in one of three states: minimum, medium, and maximum transfer-block-size states. To identify these states, every LSS provides a 2-bit state information. This means that the cache replacement is performed according to the transfer-block size which is specified by the LSS corresponding to the reference-sector.

- Line-size determinator (LSD): On every cache lookup, the LSD determines the state of the line-size specifier of the reference-sector. The algorithm is given in the next section.

3.2 Line-Size Determinator Algorithm

The algorithm for determining adequate transfer-block sizes is very simple. This algorithm is based on not memory-access history but the current state of the reference-sector. This means that no information of evicted data from the cache need be maintained. On every cache lookup, the LSD determines the state of the LSS of the reference-sector, as follows:

1. The LSD investigates how many adjacent-sublines exist in the reference-sector using all the reference-flag bits and the tag-comparison results.

2. Based on the above-mentioned investigation result and the current state of the LSS of the reference-sector, the LSD determines the next
4 Evaluations

4.1 Evaluation Models

To evaluate the effectiveness of the D-VLS cache, we simulated the following conventional fixed line-size caches and D-VLS caches.

- **FIX32, FIX64, FIX128**: Conventional 16 KB direct-mapped caches, each of which has a fixed transfer-block size of 32 bytes, 64 bytes, and 128 bytes, respectively.
- **FIX32double**: Conventional 32 KB direct-mapped cache with fixed 32-byte line size.
- **S-VLS128-32**: 16 KB direct-mapped VLS cache having three line sizes of 32 bytes, 64 bytes, and 128 bytes. S-VLS128-32 changes its line size program by program. The adequate line-size of each program is determined based on prior simulations.
- **D-VLS128-32ideal**: 16 KB direct-mapped D-VLS caches having three line sizes of 32 bytes, 64 bytes, and 128 bytes. It is an ideal D-VLS cache ignoring the hardware overhead. D-VLS128-32ideal provides a line-size specifier (LSS) for each memory-sector rather than for each cache-sector.
- **D-VLS128-32LSS1, D-VLS128-32LSS8**: 16 KB realistic direct-mapped D-VLS caches having three line sizes of 32 bytes, 64 bytes, and 128 bytes. D-VLS128-32LSS1 provides an LSS for each cache-sector, while D-VLS128-32LSS8 provides an LSS for eight consecutive cache-sectors.

4.2 Methodology

We used the following four benchmark sets, each of which consists of three programs from the SPEC92 and the SPEC95 benchmark suite, as follows:

- **mix-int1**: 072.sc, 126.gcc, and 134.perl.
- **mix-int2**: 124.m88ksim, 130.li, and 147.vortex.
- **mix-fp**: 052.alvinn, 101.tomcatv, and 103.su2cor.
- **mix-intfp**: 132.ijpeg, 099.go, and 104.Hydro2d.

The programs in each benchmark set are assumed to run in multiprogram manner on a uni processor system, and a context switch occurs per execution of one million instructions. Mix-int1 and mix-int2 contain integer programs only, and mix-fp consists of three floating-point programs. Mix-intfp is formed by two integer and one floating-point programs. We captured address traces using QPT[4] of each benchmark set for the execution of three billion instructions.

Average memory-access time ($AMT = HitTime + MissRate \times MissPenalty$) is a popular metric to evaluate the cache performance. Miss penalty in merged DRAM/logic LSIs can be a constant time regardless of the cache-line sizes, because of the high on-chip memory bandwidth. Moreover, the hit time overhead produced by the VLS caches is trivial[5]. As a result, we have decided to use cache miss rate as the performance metrics. We measured miss rate for each benchmark set using two cache simulators written in C: one for conventional caches with fixed line-sizes and the other for D-VLS caches.
4.3 Simulation Results

Figure 5 shows the simulation results for all the benchmark sets and the average of them. For each benchmark set, all results are normalized to FIX32.

First, we compare the D-VLS caches (D-VLS128-32LSS1 and D-VLS128-32LSS8) with the conventional caches. The fixed large lines in FIX128 worsen the system performance due to frequent evictions, while the D-VLS caches give remarkable performance improvements by means of the variable line-size. On average, FIX128 worsen the cache performance by about 30% while D-VLS128-32LSS1 and D-VLS128-32LSS8 improve the performance by about 27% and 22%, respectively, compared to FIX32. These performance improvements are equal or better than that of FIX32double with two times larger cache size.

Next, we compare the D-VLS caches with the statistically variable line-size cache (S-VLS128-32). On average, D-VLS128-32LSS1 and D-VLS128-32LSS8 are superior to S-VLS128-32. This is because S-VLS128-32 adjusts its line size among programs, while the D-VLS caches can adjust its line size both within and among programs.

Finally, we compare the realistic D-VLS cache (D-VLS128-32LSS1) with the ideal D-VLS cache (D-VLS128-32ideal). The difference of the performance improvements given by the realistic model and the ideal model is only 5% on average. This means that the line-size determinator can select the adequate line-sizes even if it does not accurately track the amount of spatial locality of individual memory-sectors.

5 Conclusions

In this paper, we have proposed the dynamically variable line-size cache (D-VLS cache) for merged DRAM/Logic LSIs. The line-size determinator in the D-VLS cache can detect the varying amount of spatial locality within and among programs at run time, and optimizes its cache-line size. Experimental results have shown that the realistic D-VLS caches, or D-VLS128-32LSS1 and D-VLS128-32LSS8, improve the performance by about 27% and 22%, respectively, compared to a conventional cache.

References