Dynamically Variable Line-Size Cache Architecture for Merged DRAM/Logic LSIs

Inoue, Koji
Department of Computer Science and Communication Engineering, Kyusyu University

Kai, Koji
Institute of Systems and Information Technologies/Kyushu

Murakami, Kazuaki
Department of Computer Science and Communication Engineering, Kyusyu University

出版情報：IEICE transactions on information and systems. E83-D (5), pp.1048-1057, 2000-05-25.
電子情報通信学会
バージョン：
権利関係：

PAPER

Dynamically Variable Line-Size Cache Architecture for Merged DRAM/Logic LSIs

Koji INOUE[†], **Koji KAI**[††], *Nonmembers*, *and* **Kazuaki MURAKAMI**[†], *Member*

SUMMARY

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 on merged DRAM/logic LSIs appropriately. In our evaluation, it is observed that an average memory-access time improvement achieved by a direct-mapped D-VLS cache is about 20% compared to a conventional direct-mapped cache with fixed 32-byte lines. This performance improvement is better than that of a doubled-size conventional direct-mapped cache[†].

key words: cache, variable line-size, merged DRAM/logic LSIs, high bandwidth

1. Introduction

Integrating processors and main memory (DRAM) on the same chip, or *merged DRAM/logic LSI*, can offer a number of advantages for breaking technological limitations of conventional system design [12]. Especially, the high on-chip memory bandwidth, which is one of the advantages of the merged DRAM/logic LSIs, will produce significant performance advantages. Because it improves data-transfer ability between the processors and the main memory dramatically.

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 [11][13][17]. 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. But, this approach usually makes the cache access time longer[4][18].

Manuscript received

[†]Department of Computer Science and Communication Engineering, Kyushu University, 6–1 Kasuga-koen, Kasuga, Fukuoka 816-8580 Japan.

[††]Institute of Systems & Information Technologies / KYUSHU, 2-1-22 Momochihama, Sawara-ku, Fukuoka 814-0001 Japan.

[†]An earlier extended-abstract version of the paper has appeared in [7].

To resolve the above-mentioned dilemma, we have proposed a concept of "variable line-size cache (*VLS cache*)"[11][6]. 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, these paper [6][11] did not have discussed how to determine the adequate cache-line size. There are at least two approaches to optimizing the cache-line sizes: one is a static determination based on compiler analysis; the other is a dynamic determination using some run-time hardware supports. It may be possible to adopt the former approach when target programs have regular access patterns within well-structured loops. However, a number of programs have non-regular access patterns. In addition, when a lot of programs run concurrently, the amount of spatial locality will vary both within and among programs.

This paper proposes one of the latter approaches, which is referred to as "dynamically variable line-size cache (*D-VLS cache*)" architecture, and evaluates the cost/performance improvements attainable by the 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 determinater selects adequate line-sizes based on recently observed data reference behavior. Since this scheme does not require any modification of instruction set architectures, the full compatibility of existing object codes can be kept. The goal of D-VLS cache is to improve the system performance of merged DRAM/logic LSIs such as PPRAM (Parallel Processing RAM)[11] or IRAM (Intelligent RAM)[12] by making good use of the high on-chip memory bandwidth.

The rest of this paper is organized as follows. Section 2 describes related work. Section 3 relates the concept and principle of the VLS cache. Section 4 discusses the D-VLS cache architecture. Section 5 presents some simulation results and shows the cost/performance improvement achieved by the D-VLS cache. Section 6 concludes this paper.

2. Related Work

Several studies have proposed coherent caches in order to produce the performance improvement of shared-

memory multiprocessor systems [1][2]. The cache proposed in [2] can adjust the amount of data stored in a cache line, and aims to produce fewer invalidations of shared data and reduce bus or network transactions. On the other hand, the VLS cache aims at improving the system performance of merged DRAM/logic LSIs by partitioning a large cache line into multiple independently small cache sublines, and adjusting the number of sublines to be enrolled on cache replacements. The fixed and adaptive sequential prefetching proposed in [1] allows us to fetch more than one consecutive cache lines. This approach needs a counter for indicating the number of lines to be fetched. Regardless of the values of memory reference addresses, the counter is always used for fetching cache lines on read misses. On the other hand, the D-VLS cache has several flags indicating the cache-line size. Which flag should be used depends on memory reference addresses. In other words, the D-VLS cache can change the cache-line size not only along the advance of program execution but also across data located in different memory addresses.

Excellent cache architectures exploiting spatial locality have been proposed in [3], [9] and [8]. The caches presented in [8] and [9] need tables for recording the memory access history of not only cached data but also evicted data from the cache. Similarly, the cache presented in [3] uses a table for storing the situations of past load/store operations. In addition, the detection of spatial locality in [3] relies on the memory access behavior derived from constant-stride vector accesses. On the other hand, the D-VLS cache determines a suitable cache-line size based on only the state of the cache line which is currently being accessed by the processor. Consequently, the D-VLS cache has no large tables for storing the memory access history. Just a single bit is added to each cache-tag for storing the memory access history.

Furthermore, the D-VLS cache attempts to make good use of the high on-chip memory bandwidth available on merged DRAM/logic LSIs. Since the high on-chip memory bandwidth allows us to transfer any number of data (up to the width of on-chip memory bus) at a time, the D-VLS cache can utilize very large cache lines, for example 128-byte cache-lines, without increasing miss penalty. The cache replacement always completes in a constant time regardless of the cache-line sizes selected.

3. Variable Line-Size (VLS) Cache

3.1 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.

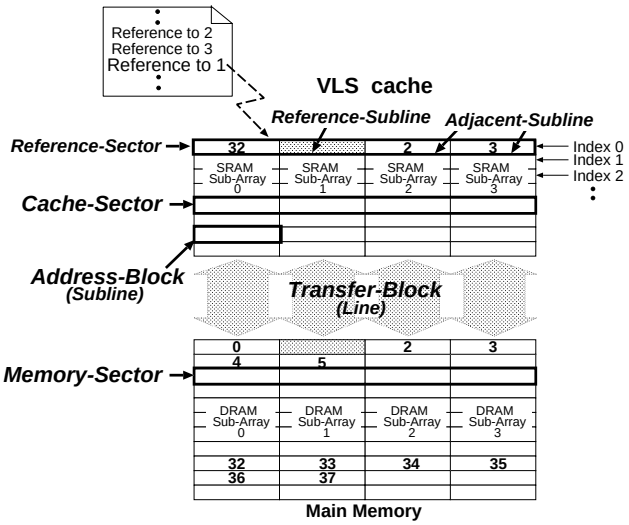

Fig. 1 Terminology for VLS Caches

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

3.2 Concept and Principle of Operations

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.

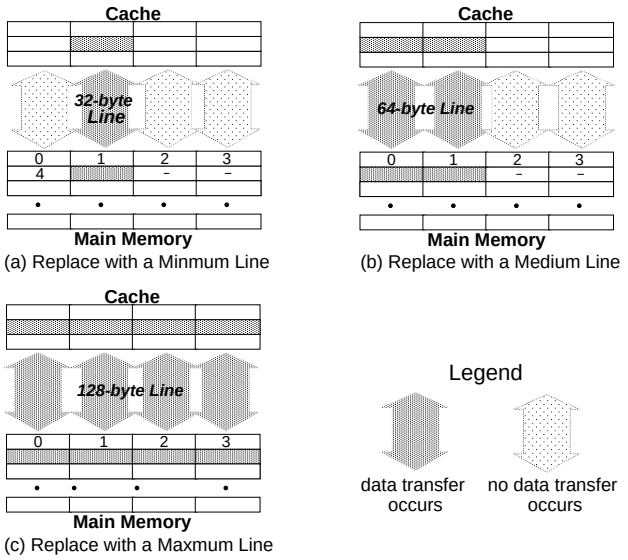

Fig. 2 Three Different Transfer-Block Sizes on Cache Replacements

The construction of the direct-mapped VLS cache illustrated in Figure 2 is similar to that of a conventional 4-way set-associative cache. However, the conventional 4-way set-associative cache has four locations where a sub-line can be placed, while the direct-mapped VLS cache has only one location for a sub-line, just like a conventional direct-mapped cache. 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[6].

The VLS cache works as follows:

1. 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.
2. 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.
3. On cache miss, a cache refill takes place as follows:
   a. According to the designated transfer-block size, one or more address-blocks are written back from the indexed cache-sector to their home locations in the DRAM main memory.
   b. According to the designated transfer-block size, one or more address-blocks (one of which contains the required data) are fetched from the memory-sector to the cache-sector.

For the example VLS cache shown in Figure 1, there are three possible transfer-block sizes as follows:

- Minimum transfer-block size, where only the designated address-block is involved in cache replacements (see Figure 2 (a)).
- Medium transfer-block size, where the designated address-block and one of its neighborhood in the corresponding cache-sector are involved (see Figure 2 (b)).
- Maximum transfer-block size, where the designated address-block and all of its neighborhood in the corresponding cache-sector are involved (see Figure 2 (c)).

4. Dynamically VLS (D-VLS) Cache

4.1 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 line-size determinater 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 we introduce the following three transfer-block sizes:

- Minimum transfer-block size (=32 bytes) involving just one ($= 2^0$) address-block,
- Medium transfer-block size (=64 bytes) involving two ($= 2^1$) address-blocks, and
- Maximum transfer-block size (=128 bytes) involving four ($= 2^2$) address-blocks.

Since it is not be allowed that the medium transfer-block misaligns with the 64-byte boundary in the 128-byte cache-sector, the number of possible combinations of address-blocks to be involved in cache replacements is just seven (four for minimum, two for medium, and one for maximum transfer-block size, respectively) rather than fifteen ($= 2^4 - 1$).

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

†Of course, the reference-flag bit corresponding to the address-block which has caused the cache miss is set to 1 when the cache replacement is performed.

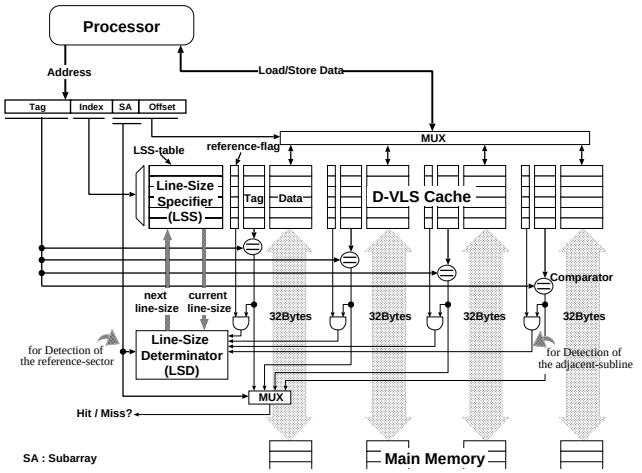

Fig. 3 Block Diagram of a Direct-Mapped D-VLS Cache

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 3.2, 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. The LSS is maintained in the LSS-table, as shown in Figure 3.

• **Line-size determinater (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.

The D-VLS cache works 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. At the same time, the LSS corresponding to the reference-sector is read using the index field from the LSS-table.
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.
4. On cache miss, a cache replacement is performed according to the state of the LSS.
5. Regardless of hits or misses, the LSD determines the state of the LSS. After that, the LSD writes back the modified LSS to the LSS-table.

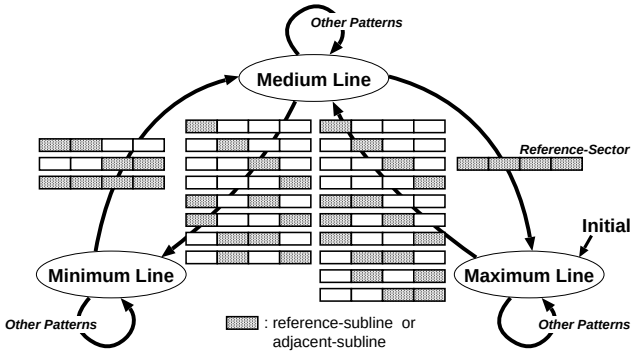

Fig. 4 State Transition Diagram

4.2 Line-Size Determinater 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 to 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 state of the LSS. The state-transition diagram is shown in Figure 4.

If there are many neighboring adjacent-sublines, the reference-sector has good spatial locality. This is because the data currently being accessed by the processor and the adjacent-sublines are fetched from the same memory-sector, and these sublines have been accessed by the processor recently. In this case, the transfer-block size should become larger. Thus the state depicted in Figure 4 transits from the minimum state (32-byte line) to the medium state (64-byte line) or from the medium (64-byte line) state to the maximum state (128-byte line) when the reference-subline and adjacent-sublines construct a larger line-size than the current line-size.

In contrast, if the reference-sector has been accessed sparsely before the current access, there should be few adjacent-sublines in the reference-sector. This means that the reference-sector has poor spatial locality at that time. In this case, the transfer-block size should become smaller. So the state depicted in Figure 4 transits from the maximum state (128-byte line)

to the medium state (64-byte line) when the reference-subline and adjacent-sublines construct equal or smaller line-size than the medium line-size (64-byte or 32-byte line). Similarly, the state transits from the medium state (64-byte line) to the minimum state (32-byte line) when the reference-subline and adjacent-sublines construct minimum line-size (32-byte line).

5. Evaluations

In this section, we discuss the effectiveness of the D-VLS cache. Before presenting the performance improvements produced by the D-VLS cache, we consider the cache-hit time, cache-cycle time, and hardware cost. In this evaluation, we represent conventional fixed line-size caches and D-VLS caches as follows:

- *FIX32, FIX64, FIX128* : Conventional 16 KB direct-mapped caches, each of which has a fixed transfer-block (line) size of 32 bytes, 64 bytes, and 128 bytes, respectively.
- *FIX32W4* : Conventional 16 KB four-way set-associative cache with fixed 32-byte line size.
- *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 cache 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 one LSS for each cache-sector, while D-VLS128-32LSS8 provides it for eight consecutive cache-sectors. That is, the total number of LSSs in D-VLS128-32LSS1 is 128 (=128 cache-sectors / 1), whereas that in VLS128-32LSS8 is 32 (=128 cache-sectors / 8).

5.1 Cache-Hit Time and Cache-Cycle Time

The structure of a direct-mapped VLS cache having 32-byte, 64-byte, and 128-byte line sizes is similar to that of a conventional 4-way set-associative cache with 32-byte fixed-lines. However, an important difference between these caches is that there is hardly any bad influence of the delay generated by the VLS cache, whereas increasing cache associativity make cache access-time longer. Thus, we can assume that the cache-hit time of a direct-mapped VLS cache having three line sizes of 32 bytes, 64 bytes, and 128 bytes is same as that of conventional direct-mapped cache with fixed 128-byte lines (FIX128) [6].

To add to the VLS cache organization, the D-VLS caches have reference-flag, LSS-table, and LSD, as shown in Section 4.1. These components do not affect the cache-hit time because they do not appear on VLS cache critical-path [6]. In this evaluation, therefore, we assume that the cache-hit time of D-VLS128-32LSS1 and D-VLS128-32LSS8 is same as that of FIX128.

From cache cycle-time point of view, however, two accesses[†] to the LSS-table might make the cache cycle-time longer. Because if the LSS-table is implemented by an SRAM array, it is very hard to complete the two SRAM accesses in a shorter processor clock cycle. There are two methods to resolve this problem: one is the pipelining of the LSS-table accesses, and the other is to implement the LSS-table using flip-flops. In this paper, the latter approach has been employed, because the former approach makes the structure and control of the LSS-table more complex.

5.2 Hardware Cost

Generally, a cache consists of an SRAM portion (data-array and tag-array) and logic portions (decoder, comparator, and multiplexors). Additionally, the D-VLS cache requires the special hardware supports, the reference-flag bits, the LSS-table and the LSD. We have calculated the size of the SRAM portion and have designed the logic portions in order to find the number of transistors for each cache. In this design, we have described the logic portions of each cache in RT-Level using VHDL (VHSIC Hardware Description language), and have translated that to a Gate-Level description using Synopsys VHDL Compiler.

For the D-VLS caches (D-VLS128-32LSS1 and D-VLS128-32LSS8), each tag field includes the 1-bit reference-flag. The LSS-table is implemented by flip-flops in order to keep the cache-cycle time, as explained earlier in Section 5.1. Since 16 KB D-VLS caches with 32-byte, 64-byte, and 128-byte lines have 128 cache-sectors (= 16KB / 128bytes), D-VLS128-32LSS1 and D-VLS128-32LSS8 require 256-bits (= 2bits×128) and 32-bits (2bits×128/8) flip-flops for the LSS-table, respectively. The LSD is independent of the number of cache-sectors which share a single LSS. We can implement the LSD with small combinational logic due to the simple algorithm for determining the adequate line-sizes, as explained earlier in Section 4.2. Table 1 shows the size of the SRAM portion and the number of transistors for the logic portions. The right-most column describes the total number of transistors including the SRAM portion where a 2-bit SRAM is assumed to be one transistor[††]. The column described as "LSS" includes both flip-flops for the LSS-table and multiplex-

[†] one for reading the LSS corresponding to the reference-sector and one for writing the modified LSS into the LSS-table.

[††] "Overall Roadmap Technology Characteristics" in [14] shows that the rate of the $LogicTransistors/cm^2$ to $CacheSRAMBits/cm^2$ from 2001 to 2007 is approximately 1:2.

Table 1 Hardware Costs

Cache Model	SRAM portion			Logic portion				**Total Hardware Cost**
	Data [bits]	Tag [bits]	**Total [bits]**	Logic [Tr]	LSD [Tr]	LSS [Tr]	**Total [Tr]**	**SRAM+Logic [Tr]**
FIX32 (direct)	131,072	9,216	**140,288**	8,354	–	–	**8,354**	**78,498**
FIX64 (direct)	131,072	4,608	**135,680**	11,310	–	–	**11,310**	**79,150**
FIX128 (direct)	131,072	2,304	**133,376**	17,988	–	–	**17,988**	**84,676**
FIX32W4 (4-way)	131,072	10,240	**141,312**	18,968	–	–	**18,968**	**89,624**
D-VLS128-32LSS1 (direct)	131,072	9,728	**140,800**	18,922	230	14,020	**33,172**	**103,572**
D-VLS128-32LSS8 (direct)	131,072	9,728	**140,800**	18,922	230	2,056	**21,208**	**91,608**

ors for selecting a LSS corresponding to the reference-sector.

It is observed that the hardware overhead of D-VLS128-32LSS1 from FIX32, FIX128 and FIX32W4 are about 32%, 22% and 16%, respectively. Since D-VLS128-32LSS8 which is a reasonable D-VLS cache does not require a lot of multiplexors for selecting a LSS, it can be implemented with slight hardware overhead. The hardware cost of D-VLS128-32LSS8 is only 17%, 8% and 2% higher than that of FIX32, FIX128 and FIX32W4, respectively. Although the D-VLS caches require more transistors than the conventional caches, this hardware overhead is trivial for the area of the entire chip of merged DRAM/logic LSIs which have not only the on-chip cache but also on-chip main memory (DRAM arrays).

5.3 Miss Rate

5.3.1 Simulation Environment

We measured miss rate using two cache simulators written in C: one for conventional caches with fixed line-sizes and the other for D-VLS (or VLS) caches with 32-byte, 64-byte, and 128-byte line sizes. In our experiments, two integer programs and a floating-point program from the SPEC92 benchmark suit [16] are executed using SPEC reference inputs. In addition, seven integer programs and four floating-point programs from the SPEC95 benchmark suite [16] are executed using SPEC training input, and SPEC test inputs, respectively. We have also simulated mpeg2encode and mpeg2decode programs from [10] using verification pictures. These programs are compiled by GNU CC with the "–O2" option, and are executed on Ultra SPARC architecture. Furthermore, in order to assume more realistic execution on general purpose processors, four benchmark sets are used, each of which consists of three programs 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.

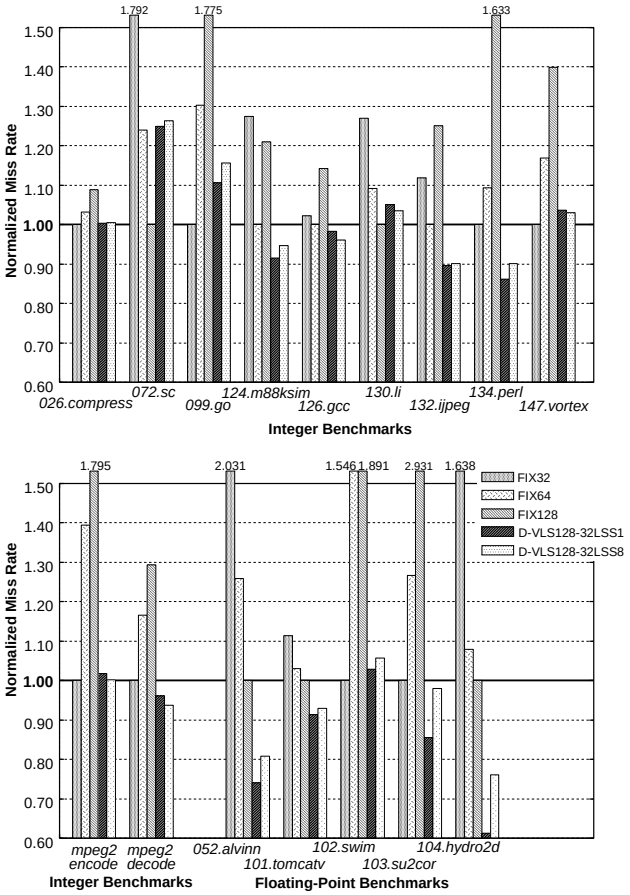

Fig. 5 Miss Rates for Benchmarks

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[5] of each benchmark set for the execution of three billion instructions.

5.3.2 Results for Benchmarks

Figure 5 presents simulation results. The left three bars for each benchmark are miss rates given by con-

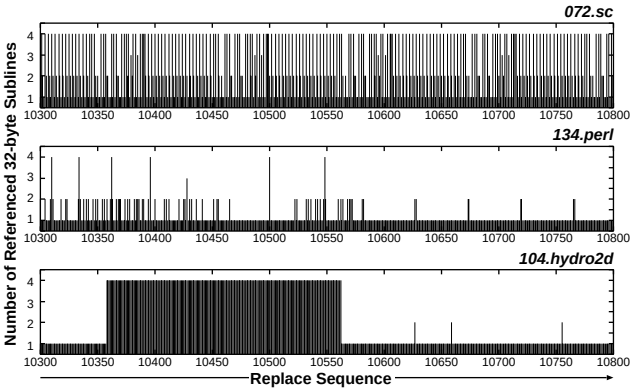

Fig. 6 Amount of Spatial Locality at a Cache-Sector

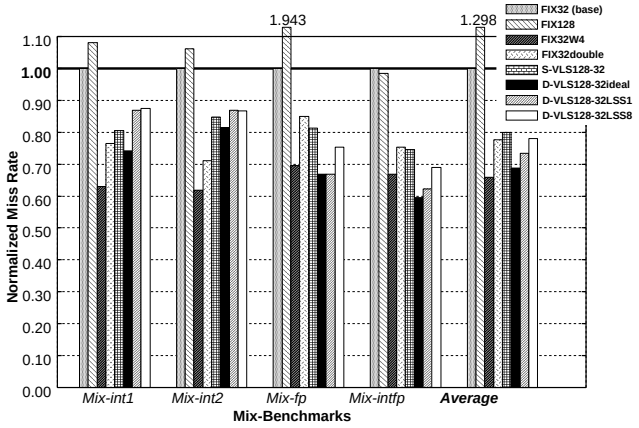

Fig. 7 Miss Rates for Benchmark sets

ventional caches, each of which has 32-byte, 64-byte, and 128-byte fixed line-size. The remaining bars to the right are results given by the D-VLS caches (D-VLS128-32LSS1 and D-VLS128-32LSS8). For each benchmark, simulation results are normalized to the miss rate produced by the conventional cache with the best line-size.

As shown in Figure 5, the best line-size is highly application-dependent. In a number of benchmarks, however, the D-VLS caches give nearly equal or lower miss rates than the conventional cache with the best line-size. Especially, for 132.ijpeg, 134.perl, 052.alvinn, and 104.hydro2d, the D-VLS caches have significant performance advantages over conventional caches. In the other benchmarks but one (072.sc), the D-VLS caches produce better results than the conventional cache with the second best line-size.

Although the D-VLS caches give good results in almost all benchmarks, they do not work better for 072.sc. In order to clarify this cause, we have analyzed the transition of the amount of spatial locality at the cache-sector which is most frequently accessed by the processor on FIX128. In this analysis, we have measured the number of 32-byte sublines referenced by the processor in a 128-byte fixed-line while the 128-byte line resides in the cache. We regard the number of the referenced 32-byte sublines as the amount of spatial locality at the cache-sector. Figure 6 presents simulation results; the horizontal axis shows cache-replacement sequence, and the vertical axis shows the number of the referenced 32-byte sublines in the 128-byte fixed-line to be evicted from the cache on each cache replacement. It is clear that the amount of spatial locality in 134.perl and 104.hydro2d are stable, whereas that in 072.sc frequently varies. On every cache lookup, the line-size determinater (LSD) tries to detect the amount of spatial locality at the reference-sector based on the number of adjacent-sublines. When the amount of spatial locality of each cache-sector varies frequently, such as 072.sc, the LSD will lack the accuracy for determining the adequate line-sizes.

5.3.3 Results for Benchmark Sets

Figure 7 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 direct-mapped caches. On average of all benchmark sets, the fixed large lines in FIX128 worsen the miss rate by about 30% from FIX32 due to frequent evictions, while the D-VLS caches give remarkable improvements by means of the variable line-size. The miss rate improvements produced by D-VLS128-32LSS1 and D-VLS128-32LSS8 are about 27% and 22%, respectively. These results are equal or better than that of FIX32double which has two times larger cache size. The conventional four-way set-associative cache with 32-byte lines (FIX32W4) gives the best result. However, the cache-hit time of set-associative caches is longer than that of direct-mapped caches. Since the cache-hit time affects all load/store operation, it has great impact on total program execution time. We will compare the FIX32W4 and D-VLS caches with average memory-access time which includes cache-hit time metric in the next section.

Next, we compare the D-VLS caches with the statically 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 determinater can select the adequate line-sizes even if it does not accurately track the amount of spatial lo-

Table 2 Cache-Hit Times

Cache Model	Hit Time [ns]
FIX32	3.852
FIX128	3.476
FIX32W4	5.958
FIX32double	4.228
D-VLS128-32LSS1	3.476
D-VLS128-32LSS8	3.476

cality of individual memory-sectors.

5.4 Average Memory-Access Time

Average memory-access time (AMT), which is an average latency required by the memory system to serve a memory reference from the processor, is a popular metric to evaluate the cache performance. AMT can be expressed as follows:

$$\begin{aligned} AMT &= HT + MR \times MP \\ &= HT + MR \\ &\quad \times 2 \times (DRAMstup + \frac{LineSize}{BandWidth}) \end{aligned}$$

,where HT, MR, and MP are cache-hit time, miss rate, and miss penalty, respectively. On cache misses, if the cache employs write back policy, two times main memory access will be occurred (one for write back and one for fill). One memory access consists of two part : the latency for an access to main memory ($DRAMstup$) and the data transfer time between the main memory and the cache ($\frac{LineSize}{BandWidth}$). $DRAMstartup$ is a constant latency which depends on the DRAM performance. In addition, the data transfer time in merged DRAM/logic LSIs can be also a constant time regardless of the cache-line sizes, because of the high on-chip memory bandwidth. In this evaluation, it is assumed that the $DRAMstup$ and $\frac{LineSize}{BandWidth}$ are 40 ns and 10 ns, respectively.

In order to find cache-hit time of conventional caches, we have used the CACTI[†][18]. The CACTI estimates the cache access-time with the detail analysis of several components, for example, sense amplifiers, output drivers, and so on. Table 2 shows the cache-hit time of conventional caches and D-VLS caches. In table 2, we have assumed that the cache-hit time of direct-mapped D-VLS caches (D-VLS128-32LSS1 and D-VLS128-32LSS8) are same as that of conventional direct-mapped cache with 128-byte lines (FIX128), as explained in Section 5.1.

Figure 8 shows average memory-access time for benchmark sets. Again, these values are normalized to those of the conventional direct-mapped cache having 32-byte fixed lines (FIX32). When we consider

[†]In this simulation, we have assumed that the process rule parameter, address width, and output-data width are 0.5 um, 32 bits, and 32 bits, respectively.

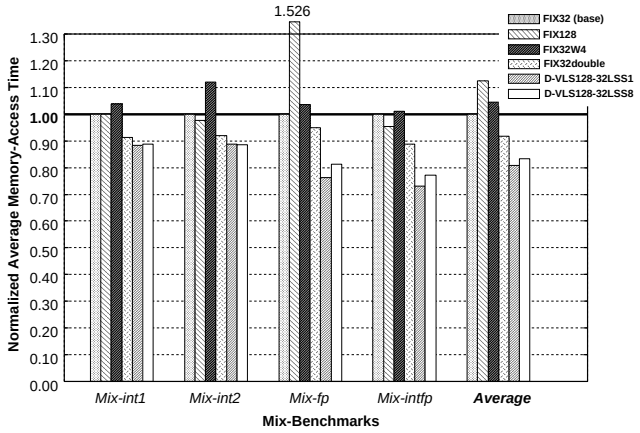

Fig. 8 Average Memory-Access Time for Benchmark sets

only miss rate, as explained in section 5.3, performance improvements produced by the conventional four-way set-associative cache (FIX32W4) is about 34%. From view point of average memory-access time, however, FIX32W4 can not give better result compared to FIX32. To keep low cache-associativity is very important to achieve high cache performance, because increasing set-associativity makes cache-access time longer [18]. On the other hand, D-VLS caches can achieve high hit rate without increasing cache-associativity and cache-size. The average memory-access time improvements of the D-VLS128-32LSS1 and D-VLS128-32LSS8 are about 20% and 17%, respectively.

5.5 Effect of Cache Size

The availability of large cache lines depends on not only the amount of spatial locality but also the cache size. When the cache size is very small, the number of large cache lines which can be held in the cache is very few. In this case, the negative effect of frequent evictions caused by large cache lines exceeds the positive effect of prefetching. Hence the small line size, which can avoid the cache conflicts, should be employed when cache size is very small. In contrast, increasing cache size increases the total number of large cache lines which can reside in the cache. As the result, conflict misses caused by the large cache lines will be reduced even if programs do not have enough spatial locality. That is, the large cache lines should be employed when the cache has enough capacity for the working-set of programs.

In order to investigate the effect of cache size on the D-VLS cache performance, we have simulated conventional caches and D-VLS caches varying the cache sizes from 4 KB to 128 KB. Figure 9 presents the average miss-rate of four benchmark sets. D-VLS128-32LSS1 and D-VLS128-32LSS8 are superior to the other conventional caches even though the cache size is varied

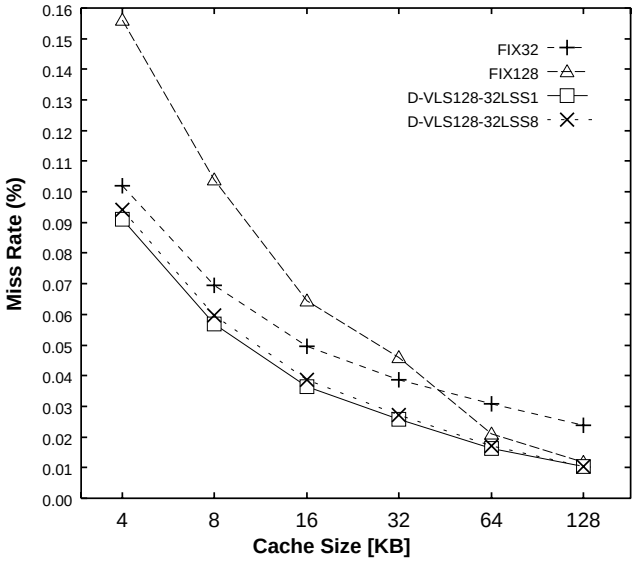

Fig. 9 Miss Rate versus Cache Size

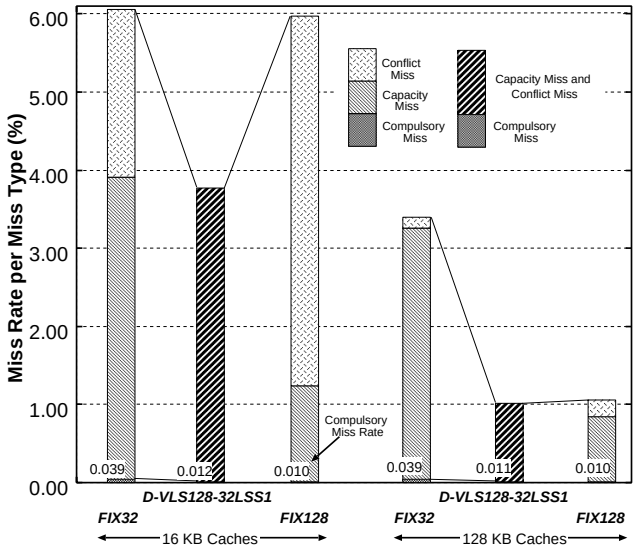

Fig. 10 Breakdown of Miss Rate

from 4 KB to 128 KB. Where the cache size exceeds 64 KB, however, the effectiveness of the D-VLS caches are very small.

The D-VLS cache attempts to reduce the conflict misses caused by large cache lines when programs have poor spatial locality. However, increasing the cache size reduces the frequent evictions caused by the large cache lines even if programs have poor spatial locality. Figure 10 shows the breakdown of miss rate for mix-intfp benchmark. It is clear that increasing the cache size reduces the conflict misses even if the fixed large-line size is employed. That is, where the cache has enough capacity for the working-set of programs, the effectiveness of the D-VLS cache is very small. Although the trend has been certainly increasing the on-chip cache size, the working-set of target application programs has been also growing. Hence, we believe that the D-VLS caches will produce the large performance improvements even if the cache size is large when the working-set of programs is big. It is our future work to evaluate the D-VLS cache using big working-set application programs.

6. Conclusions

In this paper, we have described the variable line-size cache (VLS cache) in detail, which is a novel cache architecture suitable for merged DRAM/logic LSIs. In addition, we have proposed a realistic VLS cache, called dynamically variable line-size cache (D-VLS cache). The D-VLS cache has an adjustable line-size to the characteristics of target application programs in order to make good use of the high on-chip memory bandwidth. The line-size determinater in the D-VLS cache can detect the varying amount of spatial locality within and among programs on run-time, and optimizes its cache-line size.

Experimental results showed that the realistic D-VLS cache (VLS128-32LSS1) improves the average memory-access time about 20%, and requires about 32% hardware overhead from the conventional cache. And, the reasonable D-VLS cache (VLS128-32LSS8) produces about 17% improvement while it increases the hardware cost by only 17%.

Integrating processors and main memory produces greatly performance improvements by virtue of the high on-chip memory bandwidth, so that merged DRAM/logic LSIs will become to a core device in future computer systems. Since the D-VLS cache is applicable to any merged DRAM/logic LSIs, we believe that the cache management using variable line-size is a very useful approach to improve the systems performance.

Acknowledgments

We thank Hiroto Yasuura who gave us advice on laboratory seminar. We also thank Mizuho Iwaihara Kyushu Univ. and PPRAM project team.

References

[1] Dahlgren, F., Dubois, M, and Stenstrom, P., "Fixed and Adaptive Sequential Prefetching in Shared Memory Multiprocessors," *Proc. of the 1993 International Conference on Parallel Processing*, pp.56–63, Aug. 1993.

[2] Dubnicki, C., and LeBlanc, T. J., "Adjustable Block Size Coherent Caches," *Proc. of the 19th Annual International Symposium on Computer Architecture*, pp.170–180, May 1992.

[3] Gonzalez, A. Aliagas, C. and Valero, M., "A Data Cache with Multiple Caching Strategies Tuned to Different Types of Locality," *Proceedings of International Conference on Supercomputing*, pp.338–347, July 1995.

[4] Hill, M. D., "A Case for Direct-Mapped Caches," *IEEE*

Computer, vol.21, no.12, pp.25–40, Dec. 1988.

[5] Hill, M. D., Larus, J. R., Lebeck, A. R., Talluri, M., and Wood, D. A., "WARTS: Wisconsin Architectural Research Tool Set," *http://www.cs.wisc.edu/˜larus/warts.html*, University of Wisconsin - Madison.

[6] Inoue, K., Kai, K., and Murakami, K., "High Bandwidth, Variable Line-Size Cache Architecture for Merged DRAM/Logic LSIs," *IEICE Transactions on Electronics*, Vol.E81-C, No.9, pp.1438–1447, Sept. 1998.

[7] Inoue, K., Koji, K., and Murakami, K., "Dynamically Variable Line-Size Cache Exploiting High On-Chip Memory Bandwidth of Merged DRAM/Logic LSIs," *Porc. of the 5th International Symposium on High-Performance Computer Architecture*, pp.218–222, Jan. 1999.

[8] Johnson, T. L., Merten, M. C, and Hwu, W. W., "Run-time Spatial Locality Detection and Optimization," *Proc. of the 30th Annual International Symposium on Microarchitecture*, pp.57–64, Dec. 1997.

[9] Kumar, S. and Wilkerson, C., "Exploiting Spatial Locality in Data Caches using Spatial Footprints," *Proc. of the 25th Annual International Symposium on Computer Architecture*, pp.357–368, June 1998.

[10] MPEG Software Simulation Group., "Free MPEG Softwares MPEG-2 Encoder / Decoder, Version 1.2", http://www.mpeg.org/ tristan/MPEG/MSSG/, July 1996.

[11] Murakami, K., Shirakawa, S., and Miyajima, H., "Parallel Processing RAM Chip with 256Mb DRAM and Quad Processors," *1997 ISSCC Digest of Technical Papers*, pp.228–229, Feb. 1997.

[12] Patterson, D., Anderson, T., Cardwell, N., Fromm, R., Keeton, K., Kozyrakis, C., Thomas, R., and Yelick, K., "A Case For Intelligent RAM," *IEEE Micro*, vol.17, no.2, pp.34–44, March/April 1997.

[13] Saulsbury, A., Pong, F., and Nowatzyk, A., "Missing the Memory Wall: The Case for Processor/Memory Integration," *Proc. of the 23rd Annual International Symposium on Computer Architecture*, pp.90–101, May 1996.

[14] Semiconductor Industry Association. *The National Technology Roadmap for Semiconductors,*, 1994.

[15] Shiomi, K., et al. "Development of A Standard Cell Library for VDEC (in Japanese)," *Technical Report of IEICE*, CAS97–32, VLD97–31, DSP97–46, pp.105–112, June. 1997.

[16] SPEC (Standard Performance Evaluation Corporation), *URL: http://www.specbench.org/osg/cpu92, http://www.specbench.org/osg/cpu95.*

[17] Wilson, K. M. and Olukotun, K., "Designing High Bandwidth On-Chip Caches," *Proc. of the 24th Annual International Symposium on Computer Architecture*, pp.121–132, June 1997.

[18] Wilton, S. J. E. and Jouppi, N. P., "CACTI:An Enhanced Cache Access and Cycle Time Model," *IEEE Journal of Solid-State Circuits*, vol.31, no.5, pp.677–688, May 1996.

Koji Inoue was born in Fukuoka, Japan in 1971. He received the B.E. and M.E. degrees in computer science from Kyushu Institute of Technology, Japan in 1994 and 1996, respectively. He entered Yokogawa Electric Corporation in 1996. In 1999, he joined Halo LSI Design & Technology, Inc., NY, as a circuit designer. Currently he is a Ph.D. degree in Department of Computer Science and Communication Engineering, Graduate School of Information Science and Electrical Engineering, Kyushu University. His research interests processor and cache architectures. He is a member of IPSJ.

Koji Kai received the B.E. and M.E. degrees in computer science from Kyushu University, Fukuoka, Japan, in 1989 and 1991, respectively. He has belonged to Matsushita Electric Industrial, Co., Ltd. since 1991, and is working in Institute of Systems & Information Technologies/KYUSHU as a researcher from 1996. His research interests include processor architectures, design methods of VLSI systems and hardware/software codesign. He is a member of Information Processing Society of Japan and IEEE Computer Society.

Kazuaki Murakami was born in Kumamoto, Japan in 1960. He received the B.E., M.E., and Ph.D. degrees in computer science and engineering from Kyoto University, Japan in 1982, 1984, and 1994, respectively. From 1984 to 1987, he worked for the Fujitsu Limited, where he was a Computer Architect of the mainframe computers. In 1987, he joined the Department of Information Systems of Kyushu University, Japan. He is currently an Associate Professor of the Department of Computer Science and Communication Engineering. His original research area was ILP (instruction-level parallel) processors, and in 1987, he proposed one of the first superscalar architectures. His current research focuses on the area of designing and exploiting new computer systems based on advanced VLSI and parallel-processing technologies. In 1984, he started PPRAM (Parallel Processing Random Access Memory) project and is now leading the design and implementation of PPRAM chips. He is also working on another research project, called SmartCore, which aims at developing an application-domain-specific, dynamically-reconfigurable logic core. He is a member of the ACM, the IEEE, the IEEE Computer Society, the IPSJ (Information Processing Society of Japan), and the JSIAM (Japan Society for Industrial and Applied Mathematics).