High Bandwidth, Variable Line-Size Cache Architecture for Merged DRAM/Logic LSIs

Koji Inoue, Koji Kai†, and Kazuaki Murakami

May 1998

Merged DRAM/logic LSIs could provide high on-chip memory bandwidth by interconnecting logic portions and DRAM with wider on-chip buses. For merged DRAM/logic LSIs with the memory hierarchy including cache memory, we can exploit such high on-chip memory bandwidth by means of replacing a whole cache line (or cache block) at a time on cache misses. This approach tends to increase the cache-line size if we attempt to improve the attainable memory bandwidth. Larger cache lines, however, might worsen the system performance if programs running on the LSIs do not have enough spatial locality of references and cache misses frequently take place. This paper describes a novel cache architecture suitable for merged DRAM/logic LSIs, called *variable line-size cache* or *VLS cache*, for resolving the above-mentioned dilemma. The VLS cache can make good use of the high on-chip memory bandwidth by means of larger cache lines and, at the same time, alleviate the negative effects of larger cache-line size by partitioning each large cache line into multiple sub-lines and allowing every sub-line to work as an independent cache line. The number of sub-lines involved when a cache replacement occurs can be determined depending on the characteristics of programs. This paper also evaluates the cost/performance improvements attainable by the VLS cache and compares it with those of conventional cache architectures. As a result, it is observed that a VLS cache reduces the average memory-access time by 16.4% while it increases the hardware cost by only 13%, compared to a conventional direct-mapped cache with fixed 32-byte lines.

1 Introduction

Recent remarkable advances of VLSI technology have been increasing processor speed and DRAM capacity dramatically. However, the advances also have introduced a large and growing performance gap between processor and DRAM, this problem is referred to as "Memory Wall" [1][13], resulting in poor total system performance in spite of higher processor performance. Integrating processors and DRAM on the same chip, or *merged DRAM/logic LSI*, is a good approach to resolve the "Memory Wall" problem. Merged DRAM/logic LSIs provide high on-chip memory bandwidth by interconnecting processors and DRAM with wider on-chip buses. In addition, the design space of memory hierarchy for merged DRAM/logic LSIs becomes so broad that the designer could choose an option from various on-chip memory-path architectures (as shown in Fig.1)[6]. Which memory-path architecture should be employed depends largely on the characteristics of target application programs. Among these candidates for on-chip memory path architectures, this paper focuses on the memory hierarchy including cache memory, because on-chip SRAM caches are still necessary for most application programs to hide long DRAM access latency even if processors and DRAM are integrated on the same chip.

For merged DRAM/logic LSIs with the memory hierarchy including cache memory, we can exploit the high on-chip memory bandwidth by means of replacing a whole cache line (or cache block) at a time

Department of Computer Science and Communication Engineering
Graduate School of Information Science and Electrical Engineering
Kyushu University
†: Institutes of Systems and Information Technologies/Kyushu
6-1 Kasuga-Koen, Kasuga, Fukuoka 816 JAPAN
E-mail: ppram@c.csce.kyushu-u.ac.jp
URL: http://kasuga.csce.kyushu-u.ac.jp/~ppram/

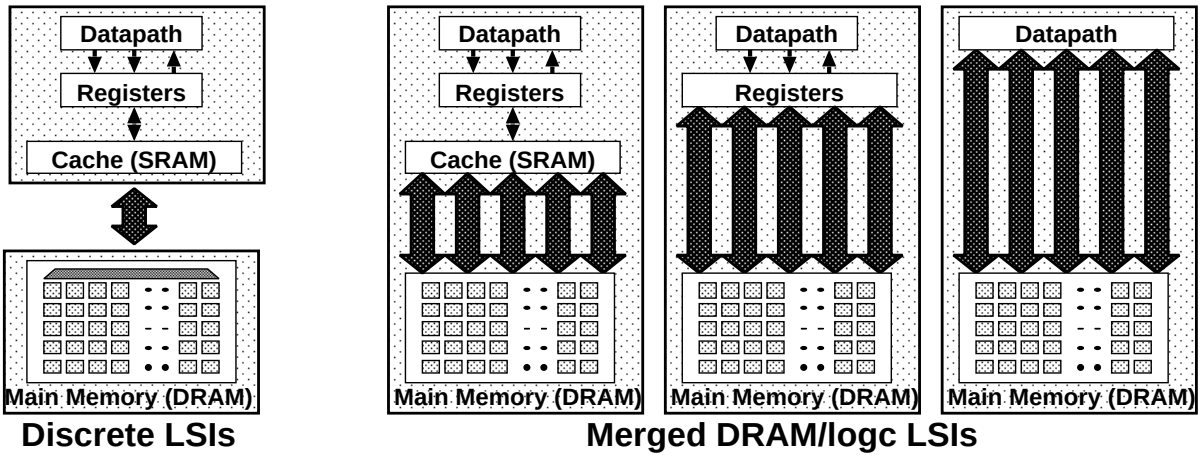

Figure 1: Memory-Path Architectures

on cache misses [7][8][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 programs with much spatial locality of references, because they provide 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, assuming the constant cache size, by increasing the cache associativity. But, this approach usually makes the cache access time longer.

There has been a proposal of the concept of variable line-size cache (*VLS cache*)[7], which was originally devised for use in the reference PPRAM ($PPRAM^R$) but is applicable to any merged DRAM/logic LSIs. The VLS cache can make good use of the high on-chip memory bandwidth by means of larger cache lines and, at the same time, alleviate the negative effects of larger cache-line size by partitioning each large cache line into multiple sub-lines and allowing every sub-line to work as an independent cache line. The number of sub-lines involved when a cache replacement occurs can be determined depending on the characteristics of application programs, as follows. If the program has rich spatial locality, larger number of sub-lines are involved on cache replacement, forming larger cache line to obtain the effect of prefetching. Otherwise, smaller number of sub-lines are involved and form themselves smaller cache lines to reduce cache conflicts. Regardless of the line sizes selected, cache replacement always complete in a constant time, because the high on-chip memory bandwidth allows to replace any number of data (up to the largest line size) at a time.

This paper describes the VLS cache in details and evaluates the cost/performance improvements attainable by the VLS cache and compares it with those of conventional cache architectures. The rest of this paper is organized as follows. Section 2 discusses the advantages and disadvantages of high on-chip memory bandwidth. Section 3 gives the concept and organization of VLS cache. Section 4 presents some simulation results and discusses the hardware cost, miss ratio, cache access time, and performance. Section 5 investigates the effects of cache size and DRAM start-up time on the performance. Section 6 describes related work. Section 7 concludes this paper.

2 Cache Memory on Merged DRAM/Logic LSIs

High on-chip memory bandwidth (which is simply noted as "high bandwidth" in the remainder of this paper) gives a dramatic improvement on data transfer ability between caches and main-memory. In this section, we discuss the advantages and disadvantages of the high bandwidth for cache memory.

2.1 Advantages of High Bandwidth

In merged DRAM/logic LSIs with the memory hierarchy including cache memory, the high bandwidth can be exploited on cache replacements. We introduce miss penalty (MP), which is the delay for a cache

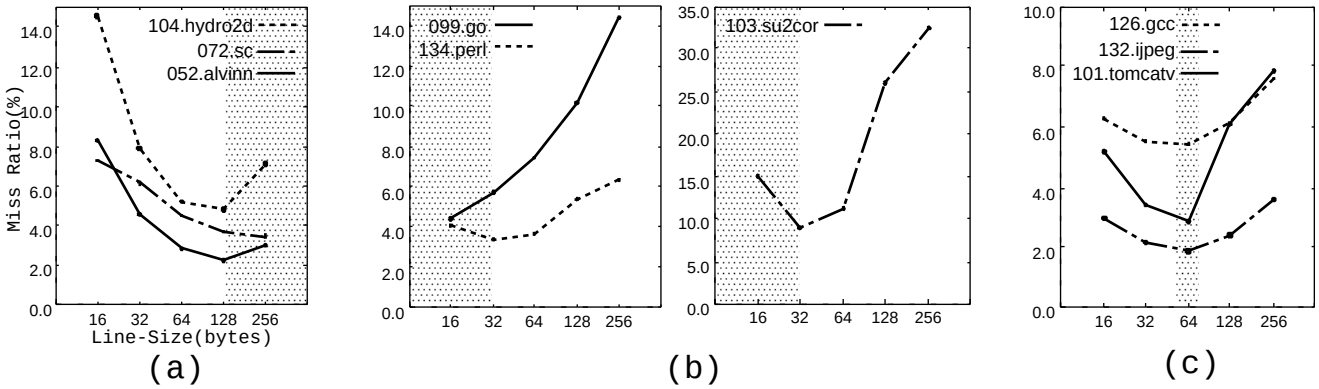

Figure 2: Miss Ratio versus Line Size

replacement, to explain an advantage of the high bandwidth. MP depends on main-memory access time (MAT) which is the latency for a reference to the main-memory. When the cache detects a miss, two main-memory accesses (for one write back and one fill[1]) are involved in the cache replacement. MP can be expressed as follows:

$$\begin{aligned} MP &= 2 \times MAT \\ &= 2 \times (DRAMstartup + \frac{LineSize}{BandWidth}). \end{aligned}$$

As shown in the above formula, MAT has two parts: the latency for an access to the main-memory ($DRAMstartup$)[2] and the transfer time for the cache line ($\frac{LineSize}{BandWidth}$). $LineSize$ and $BandWidth$ are the cache-line size and the memory bandwidth between the cache and the main-memory, respectively. Even if $LineSize$ increases within the range of $BandWidth$, assuming a constant $DRAMstartup$, MAT will not increase. However, $BandWidth$ in traditional computer systems is very small due to the I/O-pin bottleneck. Thus, MAT will increase as $LineSize$ increases, and therefore MP will also increase. In most of recent commercial processors, for example, L1 data cache employs 16-byte or 32-byte cache-line sizes due to 64-bits or 256-bits narrow memory buses.

On the other hand, $BandWidth$ of merged DRAM/logic LSIs can be enlarged dramatically due to lack of the I/O-pin limitation. The high bandwidth is easily realized by widening the on-chip buses. Therefore, the designer can increase the cache-line size within the range of the enlarged $BandWidth$ in a constant MP. Generally, large cache lines can benefit some application programs with much spatial locality of references, because they provide the effect of prefetching. Consequently, in merged DRAM/logic LSIs, the designer can positively take the advantage of spatial locality inherent in programs. For example, since instruction streams have much spatial locality in almost all programs, increasing the cache-line size is very effective for instruction caches[8].

2.2 Disadvantages of High Bandwidth

In Section 2.1, we mentioned that a great advantage of high bandwidth is the ability to increase the cache-line size in a constant miss penalty. Unfortunately, since conventional caches employ a single cache-line size, increasing the cache-line size is the only approach to make good use of the high bandwidth. However, this is not necessarily an effective approach, because it results in reducing the number of cache lines which can be held in the cache memory. So, increasing the cache-line size will worsen the miss ratio due to frequent cache conflicts if programs do not have enough spatial locality.

Actually, the spatial locality of data references depends on the characteristics of application programs, and general purpose processors have to execute a number of programs in multiprogramming environments. Fig.2 shows how the miss ratio is affected by the cache-line size in 16 KB direct-mapped data caches. *052.alvinn* and *072.sc*, from the SPEC92 benchmark program suite, are executed using the

[1] We assume that all caches in this paper adopt the write back policy.

[2] It is assumed that $DRAMstartup$ is independent of $LineSize$ because bus width inside DRAM is generally wide.

reference input. The other integer programs and floating-point programs, from the SPEC95 benchmark program suite, are executed using the training input and the test input, respectively. These programs are compiled by GNU CC with the "–O2" option, and are executed on Ultra SPARC processor. It is clear from Fig.2 that the best cache-line size for each program is very diverse. For example, the best cache-line size in Fig.2 (a) is equal to or larger than 128 bytes, that in Fig.2 (b) is equal to or smaller than 32 bytes, and that in Fig.2 (c) is just 64 bytes. If programs do not have enough spatial locality, as shown in Fig.2 (b), the system performance will get worse extremely due to cache conflicts caused by large cache lines.

3 Variable Line-Size Cache

For data caches on merged DRAM/logic LSIs, it depends on the memory access behavior whether the full utilization of high bandwidth improves the system performance or not. Increasing cache associativity is the most major method, assuming the constant cache size, to avoid cache conflicts. However, this approach usually increases the cache access time dramatically[3][12]. It is desirable, therefore, to reduce cache conflicts caused by large cache lines without increasing cache associativity. In this section, we propose a novel cache architecture for merged DRAM/logic LSIs, called *variable line-size cache* or *VLS cache*. The VLS cache has a variable line size, and can satisfy the above requirement.

3.1 Concept and Mechanism

The VLS cache has two goals in order to improve the system performance: one is positive utilization of the spatial locality inherent in programs, and the other is avoidance of cache conflicts caused by large cache lines without making the cache access time longer. To achieve these goals, the VLS cache changes its cache-line size according to characteristics of application programs. If programs have rich spatial locality, for example, the VLS cache provides the large cache lines to obtain the effect of prefetching. In this case, the high bandwidth is used in the maximum, and the advantage of spatial locality is positively utilized. Conversely, if programs have poor spatial locality, the VLS cache provides the small cache lines for avoiding cache conflicts. In this case, the high bandwidth is used as the need arises, and the frequent evictions are reduced. Since the VLS cache avoids cache conflicts by not increasing the cache associativity but changing the large cache line into the small cache lines, the access time of VLS cache is shorter than that of conventional caches with higher associativity (we will discuss the cache access time in Section 4.5).

To explain the mechanism of VLS cache, we define two terms, an *address-block* and a *transfer-block*. The address-block is a block of data associated with a single tag, and the transfer-block is a block of data moved at one time between cache and main-memory. In conventional caches, the size of address-block is equal to that of transfer-block, while a transfer-block on VLS cache can include one or more address-block(s) because the high bandwidth allows to move any number of address-blocks (within the range of the wider on-chip bus width) at a time. The number of address-blocks included in the transfer-block depends on the characteristics of application programs, and it is specified by a *line-size mode*.

In the VLS cache, an SRAM array for the cache and a DRAM array for main-memory are divided into several sub-arrays, the row size of which are equal to the address-block size. Moreover, the data transfer for an address-block is performed between corresponding sub-arrays. Fig.3 depicts the basic mechanism of a VLS cache in case that there are four SRAM sub-arrays and four DRAM sub-arrays. The VLS cache provides three line-size modes, *MinLineMode* (32-byte line), *MidLineMode* (64-byte line), and *MaxLineMode* (128-byte line). On hits, the VLS cache behaves just like a conventional 32-byte line cache regardless of the line-size modes selected. Conversely, the cache replacement on misses is performed according to the current line-size mode, as follows. When the line-size mode is MinLineMode, as shown in Fig.3 (a), a single address-block is replaced. Similarly, two and four address-blocks are replaced, as shown in Fig.3 (b) and (c), in case of MidLineMode and MaxLineMode, respectively. In the remainder of this paper, an address-block and a transfer-block are simply called a sub-line and a line, respectively (unless stated otherwise).

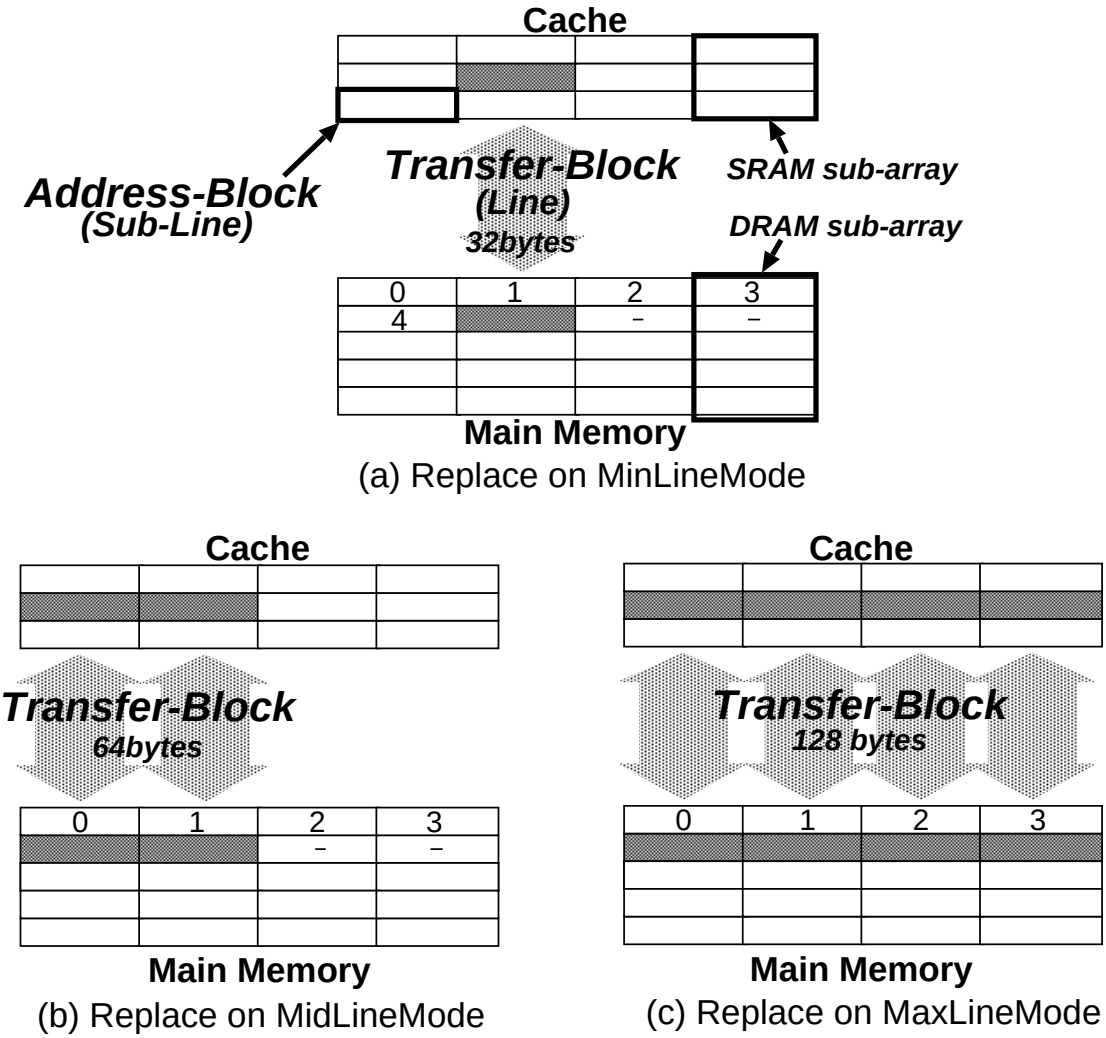

Figure 3: Basic Mechanism of VLS Caches

3.2 Architecture

Fig.4 illustrates the block diagram of a direct-mapped VLS cache which has three line sizes, 32 bytes, 64 bytes, and 128 bytes. Since the sub-line size of the VLS cache is 32 bytes, the number and the size of the tag fields are equal to those of a conventional direct-mapped cache with 32-byte lines. *StatusRegister(SR)* in the processor has a piece of information indicating the current line-size mode. This information is changed by a special instruction inserted in the top of programs. Namely, a program is executed with a single line size which is specified by the special instruction. SR keeps the information of not only the current line-size mode but also the current machine state. When a task switch occurs, the line-size mode information of the programs is saved and restored along with the current machine state by means of a conventional context saving/restoring sequence. Switching the line-size mode does not incur any extra overhead of performance.

The construction of the direct-mapped VLS cache illustrated in Fig.4 is similar to that of a conventional 4-way set-associative cache with 32-byte lines. 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. Moreover, the VLS cache can replace not only 32-byte lines but also 64-byte and 128-byte lines in a constant time, whereas the conventional 4-way set-associative cache can handle 32-byte lines only.

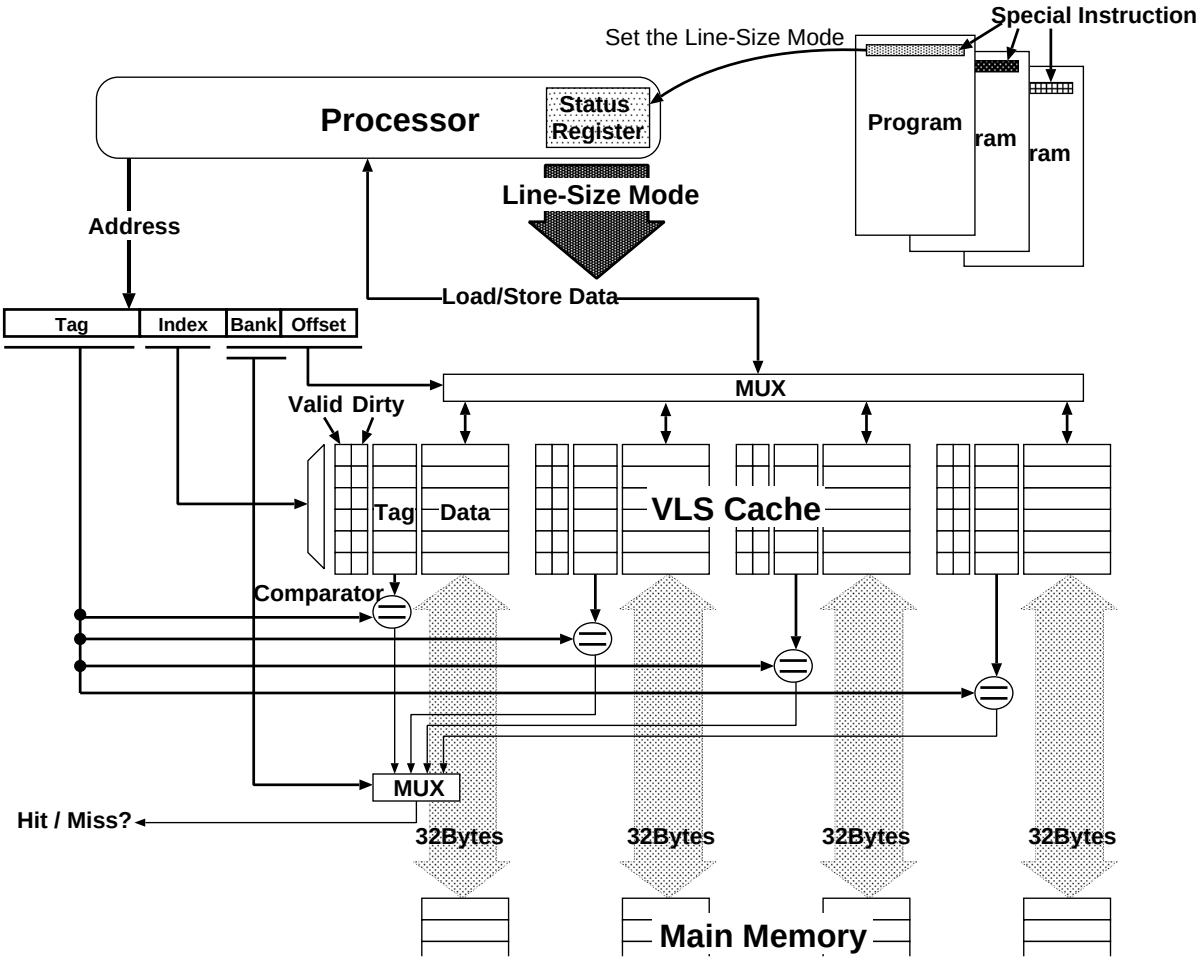

Figure 4: A Direct-Mapped VLS Cache with 32-byte, 64-byte, and 128-byte Lines

3.3 Determination of Line-Size Mode

It is important for the VLS cache to determine a suitable line-size mode to application programs. At least, we can consider the following three approaches.

1. Ideal determination based on prior simulations : Application programs are analyzed using cache simulators in advance. We determine the suitable line-size mode based on the results of the simulations.

2. Static determination based on compiler analysis : Source programs are analyzed by a special compiler. After that, this compiler determines the suitable line-size mode.

3. Dynamic determination using hardware logic : Special hardware implemented in the VLS cache determines the suitable line-size mode on run time.

On a processor employing the VLS cache, the total execution time for a program has two parts: the determination time of the suitable line-size mode and the execution time for the program. The objective of this paper is to clarify the potential effect of changing the cache-line size to the execution time. Thus, although it is not realistic to simulate programs before their executions, we have employed the first approach which can find out correctly the suitable line-size modes. We are currently developing techniques for the line-size determination based on the second and third approaches as realistic methods[5].

In case that a direct-mapped VLS cache can provide 32-byte, 64-byte, and 128-byte lines, for example, we can determine the suitable line-size mode for a program in the following manner. First, the program

is simulated three times to measure hit ratios assuming three direct-mapped caches with fixed line size of 32 bytes, 64 bytes, and 128 bytes. Then we can regard the suitable line-size mode of the program as the line size which gives the highest hit ratio of three simulations.

4 Evaluations

In this section, we discuss hardware cost, miss ratio, cache access time, and performance both for the VLS cache and for comparable conventional caches. We have made a VLS cache simulator written in C to measure miss ratios. This simulator is given address traces which are captured by QPT [4]. In this section, it is assumed that processor speed is 200 MHz, DRAM start-up time is 40 ns, and cache capacity is 16 KB. We compare following models:

- F32D, F64D, F128D : Conventional direct-mapped caches with fixed 32-byte, 64-byte, and 128-byte lines, respectively.
- F32W4 : A conventional 4-way set-associative cache with fixed 32-byte lines.
- F64W2, F128W2 : Conventional 2-way set-associative caches with fixed 64-byte, and 128-byte lines, respectively.
- V128-32D : A direct-mapped VLS cache with variable (32-byte, 64-byte, or 128-byte) line size.
- V128-32W2 : A 2-way set-associative VLS cache with variable (32-byte, 64-byte, or 128-byte) line size.

4.1 Benchmarks

In our experiments, three benchmark program sets are used, each of which consists of three programs as shown below[3]:

- *mix-int* : 072.sc, 126.gcc, and 134.perl.
- *mix-fp* : 052.alvinn, 101.tomcatv, and 103.su2cor.
- *mix-intfp* : 099.go, 132.ijpeg, and 104.Hydro2d.

Programs in each set are assumed to run concurrently on a uniprocessor system, and a context switch occurs per execution of one million instructions. Mix-int and mix-fp contain integer programs and floating-point programs, respectively. Mix-intfp is formed by two integer programs and one floating-point program. We captured address traces of each benchmark program set at the execution of three billion instructions.

4.2 Performance Model

Miss ratio is the most popular metric of cache performance. However, it is very important for cache evaluation to consider not only miss ratio but also cache access time. Since the cache access time affects all load/store operations, it has great impact on the system performance. Consequently, we use average memory-access time(AMT) as a performance metric. AMT is an average latency, as seen by the processor, required by the memory system to serve a reference from the processor. We model it as follows:

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

,where AT, MR, and MP are the cache access time, miss ratio, and miss penalty presented in Section 2.1, respectively. In this evaluation, it is assumed that buses between the caches and the main-memory operate synchronously with the processor clock. Additionally, we regard the cache access time as the

[3] Programs are compiled and executed on the same environment explained in Section 2.2.

Table 1: Hardware Costs of Various Cache Models

Cache Model	SRAM			Logic				**Total Hardware Cost SRAM+Logic [Tr]**
	Data [Tr]	Tag [Tr]	Total [Tr]	Decoder [Tr]	Comparator [Tr]	Multiplexor [Tr]	Total [Tr]	
F32D	65,536	4,608	70,144	4,158	306	3,890	8,354	**78,498**
F64D	65,536	2,304	67,840	2,694	306	8,310	11,310	**79,150**
F128D	65,536	1,152	66,688	1,236	306	16,446	17,988	**84,676**
F32W4	65,536	5,120	70,656	1,236	1,660	16,072	18,968	**89,624**
F64W2	65,536	2,432	67,968	1,236	800	16,448	18,484	**86,452**
F128W2	65,536	1,216	66,752	1,554	800	33,192	35,546	**102,298**
V128-32D	65,536	4,608	70,144	1,236	1,240	16,446	18,922	**89,066**
V128-32W2	65,536	4,864	70,4000	1,554	2,936	33,192	37,682	**108,082**

number of clock cycles for the cache hit time(HT). Consequently, the number of clock cycles for AMT can be represented as follows:

$$AMT \quad = \quad HT + MR \times 2 \times (DRAMstartup + \left\lceil \frac{LineSize}{BusWidth} \right\rceil)$$

,where $BusWidth$ is the width of the bus between the cache and the main-memory. In general, the number of load operations is much larger than that of store operations, so we assume that HT is determined by the cache read time. It is also assumed that the entire working set of programs is loaded in the on-chip DRAM main-memory before its execution.

4.3 Hardware Cost

Generally, a cache consists mainly of an SRAM portion (data-arrays and tag-arrays) and a logic portion (decoders, comparators, and multiplexors). We have calculated the capacity of the SRAM portion required in each cache model, and have designed the logic portion using VHDL in order to find the number of transistors. In this design, we have used the standard cell library for VDEC (VLSI Design and Education Center), which has been developed by Shiomi *et al.*[10].

Table.1 shows the hardware costs of the SRAM portion and the logic portion. In all cache models, the capacity of data-array is 16 KB. The right-most column describes the total number of transistors including the data-arrays. In this evaluation, a 2-bit SRAM is assumed to be one transistor[4]. In V128-32D, the capacity for tag-array is the same as that of F32D, and the number of transistors for the logic portion is almost equal to that of F32W4. Since V128-32D requires a lot of multiplexors to select a reference data in a 128-byte line, the hardware cost for its logic portion is about twice higher than that of F32D. However, we can implement V128-32D with 13% higher total hardware cost than F32D. On the other hand, the total hardware cost of V128-32W2 is larger than that of F32D by 38%. Although VLS caches require more transistors than conventional caches, the hardware overhead may be trivial for merged DRAM/logic LSIs which have not only the cache but also main-memory.

4.4 Miss Ratio

We have measured the miss ratios for three benchmark program sets. Fig.5 presents the simulation results. In V128-32D, three programs in each set are executed on the different line-size modes. For example, 099.go, 132.ijpeg, and 104.hydro2d are executed on 32-byte, 64-byte, and 128-byte line-size modes, respectively.

The miss ratios of V128-32D are lower than those of other conventional direct-mapped caches. In mix-intfp, the miss ratios of V128-32D are 1.54%, 0.72%, and 1.45% lower than those of F32D, F64D, and F128D, respectively. However, the conventional set-associative caches (F32W4, F64W2, and F128W2) are better than V128-32D. We see from these results that V128-32D could not avoid cache conflicts as well as other conventional set-associative caches. This is because that direct-mapped VLS caches have only one location for a sub-line, just like conventional direct-mapped caches.

In any benchmark program sets, V128-32W2 has a significant hit ratio advantage over V128-32D, and also much superior to the conventional set-associative caches. This is because that V128-32W2 avoids greatly the cache conflicts by virtue of its variable line size and high associativity.

[4] "Overall Roadmap Technology Characteristics" in [9] shows that the rate of the $LogicTransistors/cm^2$ to $CacheSRAMBits/cm^2$ from 2001 to 2007 is approximately 1:2.

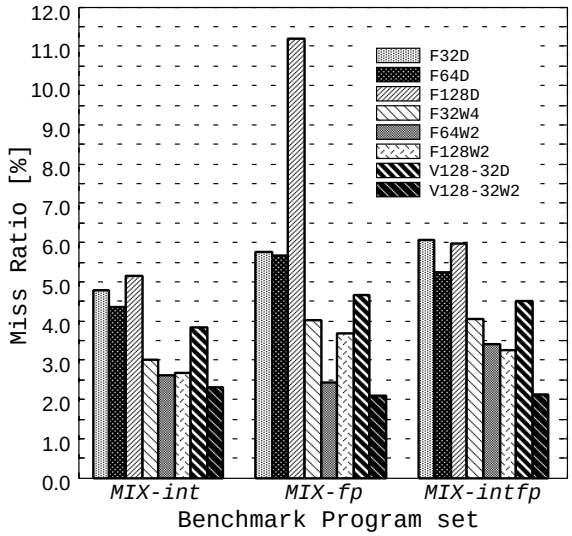

Figure 5: Miss Ratio of Various Cache Models

4.5 Cache Hit Time

Cache hit time is very sensitive to cache organizations. Fig.6 illustrates critical timing paths on conventional and proposed 16 KB caches. *MatchOut* and *DataOut* are outputs of the caches, both of which are driven by tri-state buffers. We assume that the multiplexors to select a word data are realized by the tri-state buffers. The cache hit time consists of the delay for decoder, tag read, data read, comparators, multiplexor drivers, and output drivers (match driver and data driver in Fig.6)[12]. In conventional direct-mapped caches, as shown in Fig.6 (a), the hit time is determined by either the TagSide-path or DataSide-path, while the longer path of MuxSide-path and DataSide-path determines the hit time of set-associative caches, as shown in Fig.6 (b). The structure of V128-32D is similar to that of F32W4, as shown in Fig.6 (b) and (c). In the conventional set-associative cache, the MuxSide-path often determines its hit time because control signals for selecting a word data are made after tag comparisons. However, this critical path does not appear in V128-32D because the control signals for the data selection are made from the reference address directly.

Larger cache lines, just like F128D, have two effects on the cache hit time. First, the delay for decoder is reduced by the decreased number of cache lines in the SRAM array. And second, the delay for data drivers becomes longer because the number of drivers which share an output line is increased and there is more loading at the output of each driver[12]. These features appear in not only F128D but also V128-32D. Therefore, we can regard the delay time of V128-32D's DataSide-path as that of F128D's DataSide-path. On the other hand, V128-32D's TagSide-path may be slightly longer than F128D's TagSide-path because the MatchOut of V128-32D must be selected from four comparison results. However, control signals for this selection are also made from the reference address directly. Thus, V128-32D's TagSide-path is longer than F128D's TagSide-path by the delay for a single tri-state buffer. We consider that there is hardly any bad influence of the latency caused by a tri-state buffer on the cache hit time. Consequently, it is assumed that the hit time of V128-32D is the same as that of F128D, and similarly the hit time of V128-32W2 is equal to that of F128W2.

We have used the CACTI model to find the each model's hit time[5][12]. CACTI estimates the cache access time with the detail analysis of several components, for example, sense amplifiers, output drivers, and so on. Table2 shows the hit time and the hit cycle, i.e., the required clock cycles, as seen by the 200 MHz processor, to access the cache.

[5] We assume that the process rule parameter, address width, and output-data width are 0.5 um, 32 bits, and 32 bits, respectively.

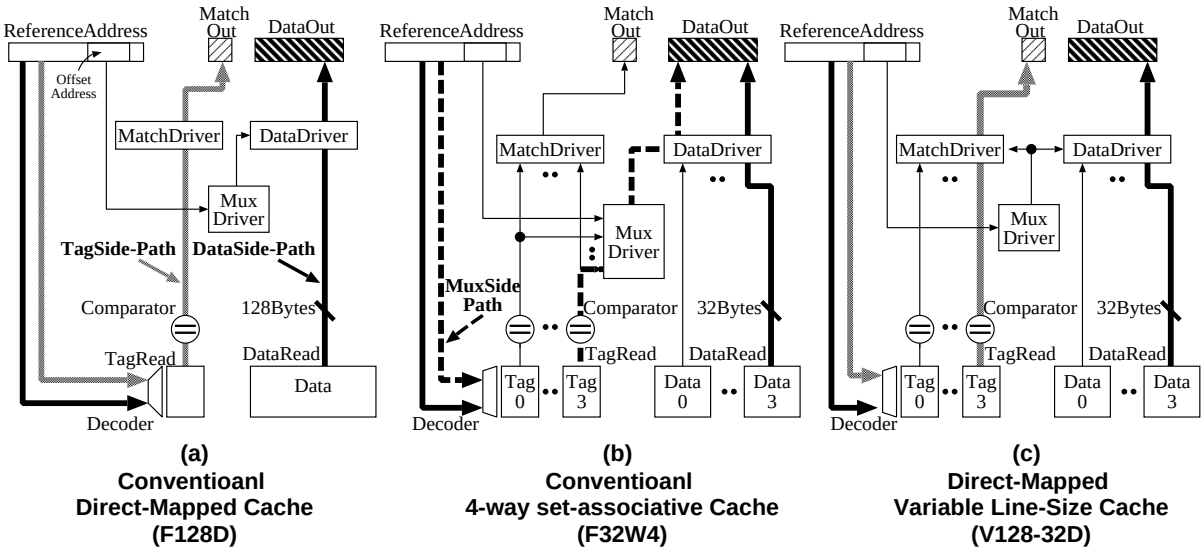

Figure 6: Critical Path of Cache Models

Table 2: Hit Times and Cycles Based on 200 MHz MPU

Model	Hit Time [ns]	Hit Cycle [cycles]
F32D	3.852	0.770
F64D	3.545	0.709
F128D	3.476	0.695
F32W4	5.958	1.192
F64W2	6.086	1.217
F128W2	7.086	1.416
V128-32D	3.545	0.695
V128-32W2	7.086	1.416

4.6 Average Access Time

We have calculated the average memory-access time based on the evaluation results in Section 4.4 and 4.5. To compare the performance of VLS cache with that of conventional caches on traditional computer systems where cache and main-memory are separated, we define a new cache model indicated as F32D-NB (Narrow Bus). F32D-NB is the same as F32D except that it does not have the high bandwidth. We assume that F32D-NB has a 64-bit off-chip memory-bus which operates synchronously with the 200 MHz processor clock. The data transfer for a line on cache models having the high bandwidth completes in one clock cycle, while that on F32D-NB needs four clock cycles (32 bytes/64 bits).

Fig.7 shows evaluation results. *"Average"* depicts the mean of the average memory-access times for three benchmark program sets. In any benchmark program sets, although V128-32D produces higher miss ratio than set-associative caches, it achieves the best performance of all caches. This is because the cache access time of V128-32D is faster than that of any set-associative caches. In average, V128-32D reduces the average memory-access time from F32D, F64D, and F128D by 16.4%, 9.1%, and 27.4%, respectively. Moreover, we see from evaluation results that exploiting both the high bandwidth and the variable line size gives dramatical performance improvements. Actually, average memory-access time of V128-32D is shorter than that of F32D-NB by 29.6% in average. In any benchmark program sets, although V128-32W2 produces the lowest miss ratios of all cache models, it does not give so high performance due to its long access time.

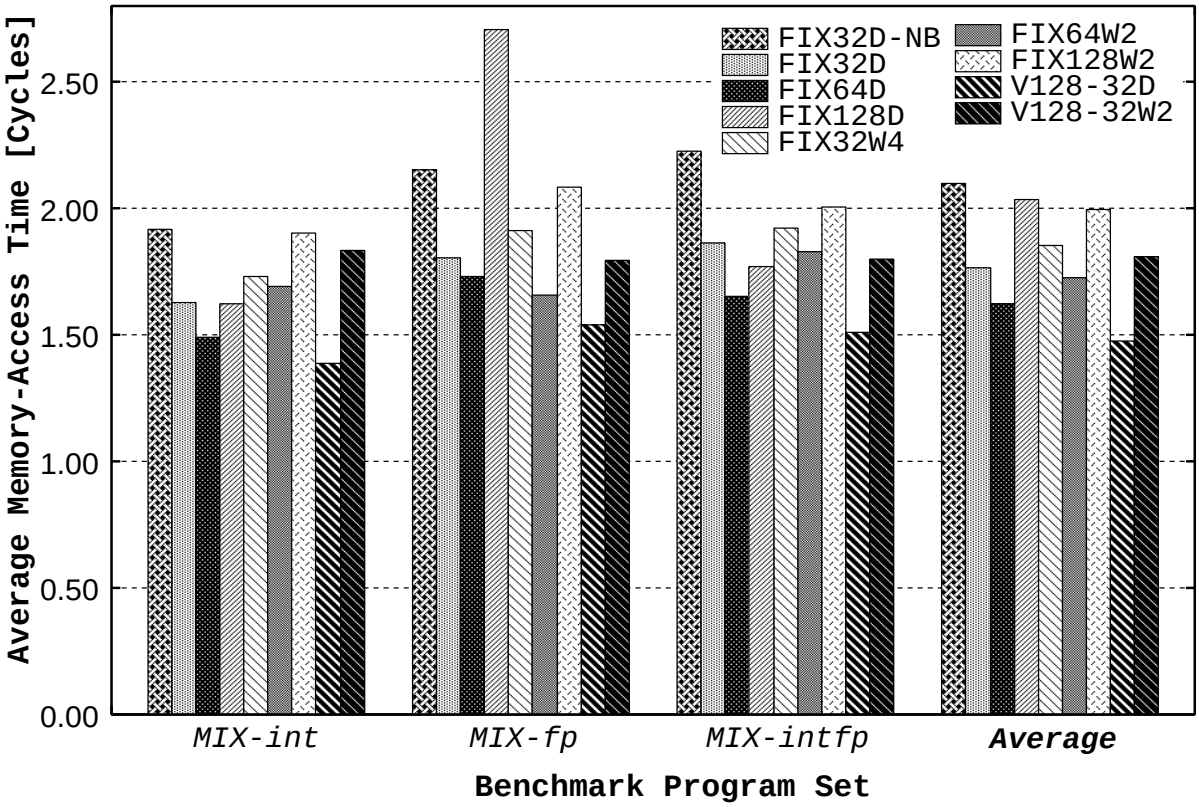

Figure 7: Average Memory-Access Time of Various Cache Models

5 Effects of Other Parameters

Thus far, we consider that the only case where cache size is 16 KB and DRAM start-up time is 40 ns. In this section, we discuss effects of the cache size and the DRAM start-up time on the performance of VLS cache. In this section, "average access time" refers to the mean of the average access times for three benchmark program sets.

5.1 Cache Size

We have simulated a direct-mapped VLS cache and conventional direct-mapped caches varying the capacities from 4 KB to 128 KB. Fig.8 shows evaluation results, which indicates that V128-32D is always superior to the conventional caches. However, the effect of the VLS cache tends to be small when the cache size is larger.

Generally, since growing cache capacity increases the number of address-blocks[6] which can be held in the cache memory, cache conflicts caused by large address-blocks will decrease. As the results, the effect of prefetching given by large address-blocks greatly exceeds the negative effect of cache conflicts. This means that the best line size enlarges along with the increase in cache capacity, regardless of the degree of the spatial locality. The VLS cache attempts to achieve high performance by utilizing the difference of suitable line size between programs. Consequently, the performance improvement given by the VLS cache becomes small when the cache has enough capacity for application programs. Actually, almost all the line-size modes of the VLS cache described in Fig.8 are MaxLineMode (128-byte lines) when the cache capacity is 128 KB.

[6] We use "address-block" to prevent confusion.

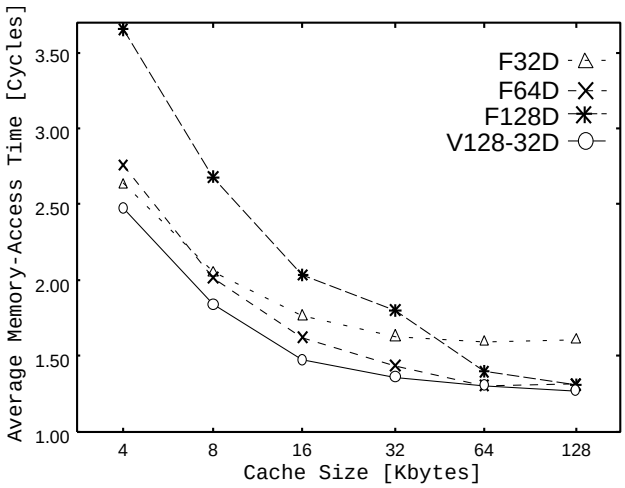

Figure 8: Average Memory-Access Time versus Cache Size

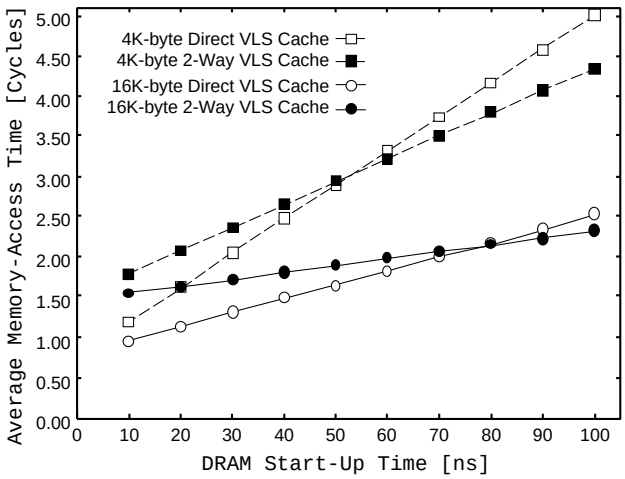

Figure 9: Average Memory-Access Time versus DRAM Start-Up Time

5.2 DRAM Start-Up Time

We have evaluated 4 KB and 16 KB VLS caches with a variety of DRAM start-up times from 10 ns to 100 ns. Fig.9 shows average memory-access times of direct-mapped and 2-way set-associative VLS caches. Regardless of the cache size, the set-associative VLS caches are superior to the direct-mapped VLS caches when the DRAM start-up is longer. This phenomena is caused by the rate of cache hit time (HT) and main-memory access time (MAT). If DRAM start-up time is longer, the average memory-access time depends largely on the main-memory access time. Thus, cache performance becomes sensitive to the miss ratio because two main-memory access are involved in a cache replacement. As the result, when the DRAM start-up time is long, set-associative VLS caches which can produce lower miss ratios give better performance than direct-mapped VLS caches.

6 Related Work

Saulsbury *et al.*[8] discussed data cache configurations on merged DRAM/logic LSIs, and proposed very large line (512 bytes) caches with a victim cache. Wilson *et al.*[11] evaluated performance of 512-byte

line caches on merged DRAM/logic LSIs and some other high bandwidth caches. These caches presented in [8] and [11] avoided cache conflicts by increasing the cache associativity. However, this approach will make the cache access time longer. On the other hand, the VLS cache avoids cache conflicts by adjusting the cache-line size to characteristics of application programs without increasing the cache associativity. Therefore, the access time of VLS cache is faster than that of the caches with higher associativity.

Dubnicki *et al.*[2] proposed coherent caches which have an adjustable cache-line size according to memory access behavior. This approach aims to produce fewer invalidations of shared data and reduce bus or network transactions on multiprocessor systems. However, the VLS cache has a quite different purpose, achieving the high hit ratio without making the cache access time longer by virtue of the high on-chip memory bandwidth realized on merged DRAM/logic LSIs.

7 Conclusions

In this paper, we have presented a novel cache architecture for merged DRAM/logic LSIs, called variable line-size cache (VLS cache). The VLS cache changes its cache-line size for each program, and therefore, it can make efficient utilization of spatial locality inherent in the programs. Especially, VLS caches are effective in multiprogramming environments.

Experimental results shown that the 16 KB direct-mapped VLS cache with 32-byte, 64-byte, and 128-byte lines (V128-32D) reduces the average memory-access time by 16.4% while it increases the hardware cost by only 13%, compared to the 16 KB conventional direct-mapped cache with fixed 32-byte lines. 16 KB 2-way set-associative VLS cache with 32-byte, 64-byte, and 128-byte lines (V128-32W2) produces much lower miss ratio than the V128-32D. However, it has not been able to achieve higher performance improvements than V128-32D due to its long access time.

In this paper, we assume that the suitable line size for each program is obtained by a number of simulations which generally require a long time. A static analysis of suitable line size is one of our ongoing works. In this approach, based on static program analysis, a compiler might specify the cache-line size at any granularity (e.g., program by program, procedure by procedure, code by code, data by data, and so on). Another ongoing work is dynamic selection of line size. The effect of wire area to total hardware cost will increase along with advances of VLSI technology. Thus, it is also our future work to evaluate hardware costs of the VLS cache considering the effect of area for widened on-chip buses.

References

[1] Burger, D., Goodman, J. R., and Kägi, A., "Memory Bandwidth Limitations of Future Microprocessors," *Proc. of the 23rd Annual International Symposium on Computer Architecture*, pp.78–89, May 1996.

[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] Hill, M. D., "A Case for Direct-Mapped Caches," *IEEE Computer*, vol.21, no.12, pp.25–40, Dec. 1988.

[4] 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.

[5] Inoue, K., Kai, K., and Murakami, K., "Dynamically Variable Line-Size Caches Exploiting High On-Chip Memory Bandwidth of Merged DRAM/Logic LSIs," *PPRAM Project's Internal Technical Reports, http://kasuga.csce.kyushu-u.ac.jp/~ppram/.*

[6] Miyajima, H., Inoue, K., Kai, K., and Murakami, K., "On-Chip Memorypath Architectures for Parallel Processing RAM (PPRAM)," *Workshop on Mixing Logic and DRAM: Chips that Compute and Remember*, http://iram.CS.Berkeley.EDU/isca97-workshop/, June 1997.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

[13] Wulf, W. A. and McKee, S. A., "Hitting the Memory Wall: Implications of the Obvious," *ACM Computer Architecture News*, vol.23, no.1, Mar. 1995.