High Performance Memory Architecture for Vector Processors

著者	撫佐　昭裕
学位授与機関	Tohoku University
URL	http://hdl.handle.net/10097/39918

TOHOKU UNIVERSITY
Graduate School of Information Sciences

High Performance Memory Architecture for Vector Processors

**(ベクトルプロセッサのための高性能
メモリアーキテクチャに関する研究)**

A dissertation submitted for the degree of
Doctor of Philosophy (Information Sciences)

Department of Computer and Mathematical Sciences

by

Akihiro MUSA

January 16, 2009

High Performance Memory Architecture for Vector Processors
（ベクトルプロセッサのための高性能メモリアーキテクチャに関する研究）

Akihiro MUSA

Abstract

Computer simulations of physical phenomena and engineered systems have become widely recognized as the *third pillar* to support science and technology, in addition to theory and experiment. Complex systems of the physical phenomena and engineered systems are analyzed and better understood through computational models. Solving many of important scientific and engineering problems requires supercomputers. The supercomputers enable investigations heretofore impossible, which in turn have enabled scientific and technological advances of vast breadth and depth. Thus, supercomputing has become an indispensable tool in science and technology.

Supercomputers are categorized into vector and scalar systems. The mainstream of the supercomputers has been dominated by the commodity-based scalar systems. However, US High-End Computing Revitalization Task Force reported that there was the increasing gap between the theoretical peak performance and the sustained system performance for High End Computing systems of major US high-end computing centers. In other words, the commodity-based scalar systems have difficulty obtaining the high computation efficiency in execution of real scientific and engineering applications. Meanwhile, vector supercomputers achieve high sustained performance and high computation efficiencies in various scientific and engineering applications. Specifically, the Earth Simulator, which is the largest vector supercomputer in the world, has substantiated it in various scientific applications.

However, as advantages in VLSI technology have also been accelerating processor speeds, supercomputers have been encountering the memory wall problem. Furthermore, the number of memory ports in a processor die does not increase owing to limitations of the die area and electrical power. As a result, it is getting harder for the vector supercomputers to keep a high memory bandwidth balanced with the processor performance. The byte per flop rate (B/FLOP), the ratio of a memory bandwidth (byte/s) to computational performance (flop/s), has decreased from 8 B/FLOP to 2.5 B/FLOP in the NEC SX supercomputers during the last twenty years. There is the concern that the sustained performance of future vector supercomputers seriously goes down as the B/FLOP rate decreases.

Moreover, chip multiprocessors (CMPs) have become the mainstream in commodity-based scalar processors from 1999, and the CMP architecture is also promising for vector processor design, because the number of transistors in a vector processor has been increasing by a factor of eight for the last decade. Furthermore, many scientific and engineering applications are parallelized for multi-threads using the automatic parallelization and OpenMP. The CMP architecture will be adopted by vector supercomputers. Then, it is getting harder to keep a high memory bandwidth balanced with the improvement of their flop/s performance owing to the limited pin bandwidth. Therefore, the vector processors will be unable to outperform even commodity-based scalar processors, because the data transfer time between the main memory and the register files increases.

This dissertation has three objectives. The first objective is to clarify the relationship between the memory performance and the computational performance on the supercomputers using real scientific applications. Particularly, several important factors of memory systems are revealed for maintaining the high computation efficiency in vector supercomputers. The second objective is to establish the high performance memory architecture for maintaining the high computational performance on the future vector supercomputers. Through the experiments using real scientific applications, characteristics of the high performance memory architecture are clarified. The third objective is to clarify the relationship between the scalability of chip-multiprocessing and the memory bandwidth rate, and to establish the shared cache architecture that boosts the performance of the chip multiprocessor architecture under a low B/FLOP rate.

The aim of the high performance memory architecture is to reduce data traffic between the main memory and the vector processor, and effectively to use the memory bandwidth. Therefore, the high performance memory architecture maintains the effective memory bandwidth rate at vector register files. Then, the memory architecture actualizes the following mechanisms.

- On-chip cache mechanism
- Miss status handling registers mechanism
- Prefetch mechanism
- Selective caching mechanism

In Chapter 2, to clarify the relationship between the memory performance and the computational performance on the supercomputers, the vector supercomputers are compared against the commodity-based scalar systems using five leading scientific applications of three scientific area; electromagnetic analysis, CFD/heat analysis and seismology. These five applications are highly vectorized and parallelized. It is presented that the vector supercomputers achieve the high computation efficiency and significantly outperform the scalar systems. Concretely, the vector supercomputers achieve the computation efficiency of 40 % or more across all of the applications. Meanwhile, the scalar systems show that the computation efficiency is less than 14 %.

Then, it is clarified that the important factor affecting the computational performance on scientific applications is the memory systems. The vector supercomputers employ the interleaved memory system, while the scalar systems use the hierarchical cache memory system. The interleaved memory system organizes memory chips in banks to access multiple words at a time. The memory latency of the second memory access or later are hidden in the interleaved memory system. Moreover, memory access times are hidden by vector arithmetic operations. Therefore, the memory access times not hidden by overlapping calculations of the vector supercomputers are much shorter than those of the scalar systems. Concretely, the ratios of non-hidden memory access time to processing time of the vector supercomputers are less 30 % across all of the applications, while the ratios of the scalar systems are 50 % or more. For the three applications used in the evaluation, the ratios of the scalar systems are 80 % or more. The memory system of the vector supercomputers is more effective for scientific applications than the memory system of the scalar systems.

Moreover, it is investigated that the memory bandwidth affects the non-hidden memory access time of the vector supercomputers. The vector supercomputers as the NEC SX-7, SX-8 and the Earth Simulator have 4 B/FLOP. When the memory bandwidth is adjusted from 4 B/FLOP to 2 B/FLOP, the non-hidden memory access times of the applications become two or more times longer than the non-hidden memory access times of the 4 B/FLOP system. When the memory bandwidth is further reduced to 1 B/FLOP, the non-hidden memory access times become four or more times longer than those of the 4 B/FLOP system. As the memory bandwidth decreases, the memory read/write time increases, and the memory access time is not hidden by the pipelined vector operations. Therefore, the sustained performance is seriously affected by the B/FLOP rates, and the memory bandwidth rate of the 4 B/FLOP is essential to keep the superiority of the vector supercomputers against the scalar systems.

In Chapter 3, an on-chip cache, called vector cache, is introduced to overcome the memory wall problem in future vector supercomputers. The vector cache reuses data that have already been supplied to the vector unit, and maintains the effective memory bandwidth rate at vector register files. The vector cache employs a bypass mechanism between the vector register files and the main memory, and is controlled by software to selectively cache load/store data. The bypass mechanism and the cache work complementarily together to provide data to the register files. Furthermore, the vector cache employs miss status handling registers (MSHR) and a prefetch mechanism to improve the effect of the vector cache. The vector cache uses the MSHR to handle outstanding vector loads on cache misses, resulting in the elimination of unnecessary vector load accesses. On the other hand, the prefetch mechanism has two effects on the performance. One is that the mechanism hides the long memory latency by pipelined vector operations. The other is the same effect of the MSHR: the prefetch mechanism reduces the redundant load request between the vector cache and the main memory when multiple load instructions access the same memory data.

Then, it is presented that the vector cache has a potential for the future vector supercomputers to cover the shortage of their memory bandwidth, and the characteristics of the vector cache are clarified on the vector supercomputer. The vector cache recovers the lack of the memory bandwidth, and boosts the computation efficiencies of the 2 B/FLOP and 1 B/FLOP systems. Concretely, the vector cache increases the recovery rate of the performance in execution of the five applications by 21 % to 99 % on the 2 B/FLOP system and 9 % to 96 % on the 1 B/FLOP system. Especially, in the case where cache hit rates exceed 50 %, the 2 B/FLOP system achieves a performance comparable to the 4 B/FLOP system. The vector cache with a bypass mechanism provides the data from both the memory and the cache at once, and the sustained memory bandwidth for the register files increase.

The potential of the vector cache with the MSHR, the prefetch mechanism and selective caching is discussed for the future vector supercomputers, which have insufficient B/FLOP rates. It is shown that these mechanisms are effective for the future vector supercomputers to cover the insufficient B/FLOP rates. The effects of the MSHR are evaluated on three scientific applications. The MSHR reduces the number of load requests within the difference scheme loops which continuously load the same data, and the latency of the subsequent load requests is shortened. In the experiments, the MSHR improves the performance by 5 % to 25 % on the 2 B/FLOP system, and 4 % to 45 % on the 1 B/FLOP system. In addition, the performance of the prefetch mechanism is demonstrated. The prefetching mechanism boosts the performance by 20 % to 30 % on the 2

B/FLOP system and 20 % to 60 % on the 1 B/FLOP system. The selective caching, which is controlled by means of the bypass mechanism, is effective for efficient use of limited on-chip caches. A higher performance is obtained by selective caching, compared with all the data caching.

In Chapter 4, the performance of a chip multi vector processor (CMVP) and the effectiveness of a shared vector cache are clarified. The vector processor will employ the chip multi processor architecture in the near future. However, the CMVP does not preserve the high memory bandwidth rate owing to the memory wall problem. Thus, the CMVP employs the vector cache to improve the effective memory bandwidth rate. Moreover, because various scientific simulations have a high locality among multi-threads, the vector cache is a shared cache among vector cores.

The performance of the CMVP is evaluated using the five applications. The CMVP contains four cores and an on-chip shared cache with the MSHR. It is shown that the CMVP without the cache needs the 4 B/FLOP rate of the off-chip memory bandwidth per core for maintaining the scalability of vector processors in sustained performance. However, a future vector supercomputer will not be able to keep the 4 B/FLOP rate owing to the limited pin bandwidth. On the other hand, the evaluations of the CMVP with the shared cache have shown that the performance of the 2 B/FLOP CMVP is approximately equivalent to the performance of the 4 B/FLOP CMVP, when the date are provided to the register file from both the cache and the main memory at 2 B/FLOP rate each. Therefore, the off-chip memory bandwidth of the CMVP needs to satisfy at least 2 B/FLOP using the cache mechanism to achieve a high scalability. Moreover, the effect of the shared cache and the MSHR are evaluated using the same applications. The results show that the shared cache increases the cache hit rate and the efficiency of the applications. Meanwhile, the MSHR increases the number of opportunities to reuse data on the cache across the applications of the difference schemes.

This dissertation clarifies that the memory bandwidth rate, B/FLOP, is a primary important factor of the high computational performance on the vector supercomputers. The high performance memory architecture is introduced to overcome to the memory wall problem in future vector supercomputers. Then, the high performance memory architecture has a potential to cover the shortage of the B/FLOP rate.

Contents

Abstract **i**

1 Introduction **1**

1.1 Roles of Supercomputers 1

1.2 Vector Processors 3

1.3 Issues of Supercomputers 8

1.3.1 Issues of scalar systems 8

1.3.2 Issues of vector supercomputers 10

1.4 Objective of the Dissertation 12

1.5 Organization of the Dissertation 14

2 Memory Performance for Highly Efficient Supercomputing of Scientific Applications **15**

2.1 Introduction 15

2.2 Related Work 17

2.3 Architectural Characteristics of the Evaluated Systems 19

2.3.1 Vector supercomputers: NEC SX-7 and SX-7C 20

2.3.2 Scalar systems: NEC TX7/i9510 and SGI Altix3700 20

2.4 Benchmark Programs 22

2.4.1 GPR simulation 22

2.4.2 APFA simulation 23
2.4.3 PRF simulation 24
2.4.4 SFHT simulation 24
2.4.5 PBM simulation 25
2.5 Experimental Results and Discussion 26
2.5.1 Characterizations of benchmark programs 26
2.5.2 Efficiency of benchmark programs on the four systems . . . 27
2.5.3 Discussion on the memory performance of SX-7 and SX-7C . 30
2.5.4 Discussion on the memory performance of TX7/i9510 and Altix3700 33
2.6 Conclusions 37

3 An On-Chip Cache for the Vector Architecture 39
3.1 Introduction 39
3.2 Related Work 42
3.3 Characteristics of Scientific Applications 45
3.4 On-Chip Cache Memory for Vector Architecture 47
3.4.1 Basic mechanisms of a high performance memory architecture 47
3.4.2 Proposed vector architecture 49
3.5 Experimental Environment 53
3.5.1 Methodology 53
3.5.2 Benchmark programs 54
3.6 Performance Evaluation of Vector Cache 60
3.6.1 Relationship between efficiency and cache hit rate on Kernel loops 60
3.6.2 Relationship between efficiency and cache hit rate on the five benchmark programs 63

3.6.3 Relationship between associativity and cache hit rate 66
3.6.4 Effects of vector cache latency on performance 68
3.7 Performance Evaluation of MSHR and Prefetching Vector Cache . 72
3.7.1 Effect of MSHR on the vector cache 72
3.7.2 Effect of prefetching on the vector cache 74
3.8 Optimizations for Vector Caching: Selective Caching and Loop Unrolling 79
3.8.1 Effects of selective caching 79
3.8.2 Effects of loop-unrolling and caching 83
3.9 Conclusions 86

4 A Shared Cache for a Chip Multi Vector Processor 89
4.1 Introduction 89
4.2 Related Work 92
4.3 Characteristics of Scientific Applications 94
4.4 Chip Multi Vector Processor 95
4.4.1 Basic mechanism of Chip Multi Vector Processor 95
4.4.2 Structure of Chip Multi Vector Processor 95
4.5 Experimental Environment 98
4.5.1 Methodology 98
4.5.2 Benchmark programs 99
4.6 Performance Evaluation of CMVP 100
4.6.1 Scalability of the applications without the cache 100
4.6.2 Scalability of the applications with the vector cache 100
4.6.3 Relationship between associativity and cache hit rate 102
4.6.4 Effect of the shared vector cache 103
4.6.5 Effect of the MSHR 104

4.7 Conclusions 107

5 Conclusions 108

Acknowledgments 112

Bibliography 114

List of Figures

1.1 Block diagram of the NEC SX vector processor. 3
1.2 Vector instructions of Source Code 1.1. 5
1.3 Pipeline diagram. 5
1.4 Decrease in computation time by vectorization. 6
1.5 Trend in architectures of top 500 supercomputer sites. 8
1.6 Divergence problem for US high end computer center. 9
1.7 Trend in performance of the NEC SX series. 11

2.1 Overview of performance for the five benchmark programs on one processor, (a) computation efficiency and (b) sustained performance. 27
2.2 Ratio of non-hidden memory access time to processing time on one processor. 29
2.3 Speedup ratio in 32 processors for the five benchmark programs. . 29
2.4 Relative non-hidden memory access time as a function of the number of banks. 31
2.5 Efficiency of the GPR simulation on SX-7 and SX-7C. 32
2.6 Computation efficiency of the five benchmark programs when changing the B/FLOP rate in SX-7. 33
2.7 Correlation of cache hit rate and ratio of memory access time to processing time on TX7 and Altix. 34

2.8 Processing time of the GPR simulation: (a) TX7 and (b) Altix. . . . 35

2.9 Processing time of the APFA simulation: (a) TX7 and (b) Altix. . . . 36

3.1 Vector architecture with vector cache and memory system block diagram. 50

3.2 Instruction format of vector load/store. 50

3.3 Relationship between cache hit rate and relative memory bandwidth on Kernel loop (1). 61

3.4 Relationship between cache hit rate and computation efficiency on Kernel loop (1). 61

3.5 Relationship between cache hit rate and relative memory bandwidth on Kernel loop (2). 63

3.6 Relationship between cache hit rate and relative memory bandwidth on the five benchmark programs. 64

3.7 Computation efficiency of the five benchmark programs with/without 8 MB vector cache. 65

3.8 Recovery rate of performance on 8 MB vector cache. 66

3.9 Cache hit rate vs. associativity on the five benchmark programs. . 67

3.10 Computation efficiency of the five benchmark programs on four cache latency cases. 68

3.11 Computation efficiency of two kernel loops on four cache latency cases. 69

3.12 Computation efficiency of two kernel loops on 4 B/FLOP and 2 B/FLOP on three loop lengths. 70

3.13 (a) Computation efficiency and (b) Data transfer rate using MSHR on the PRF simulation. 73

3.14 Computation efficiency using MSHR on the GPR and SFHT simulations. 73
3.15 Computation efficiency on 2 B/FLOP and 1 B/FLOP on the GPR simulation. 74
3.16 (a) Relative non-hidden memory access time and (b) Data transfer rate on 2 B/FLOP and 1 B/FLOP on the GPR simulation. 75
3.17 Computation efficiency on 2 B/FLOP on the SFHT and PRF simulations. 76
3.18 (a) Data transfer rate on PRF and (b) Relative non-hidden memory access time rate on the SFHT simulation. 77
3.19 Computation efficiency of both MSHR and prefetch. 78
3.20 Efficiency of selective caching and cache hit rate on 2 B/FLOP system (the GPR simulation). 80
3.21 Recovery rate of performance and cache hit rate on selective caching (the GPR simulation). 81
3.22 Efficiency and cache hit rate of selective caching on performance (the PRF simulation). 81
3.23 Recovery rate of performance and cache hit rate on selective caching (the PRF simulation). 82
3.24 Efficiency and cache hit rate of outer-unrolling on performance (the PBM simulation). 84
3.25 Efficiency and cache hit rate of outer-unrolling on performance (the PRF simulation). 85

4.1 CMVP block diagram. 96
4.2 Block diagram of a core. 96

4.3 Relative performance of programs on the four-core CMVP without the vector cache. 101
4.4 Relative performance of programs on the four-core CMVP with the vector cache. 101
4.5 Cache hit rate vs. associativity on the five benchmark programs in the four core case. 103
4.6 Cache hit rate and improved efficiency on a kernel loop of the GPR simulation at the 2 B/FLOP. 104
4.7 Relative performance with/without the MSHR on the GPR, PRF and SFHT simulations at the 2 B/FLOP. 105
4.8 Cache hit rate with/without the MSHR on a kernel loop of the PBM simulation. 106
4.9 Relative performance with/without the MSHR on a kernel loop of the PBM simulation at the 2 B/FLOP. 106

List of Tables

2.1 Architectural summary of the systems. 19
2.2 Summary of benchmark programs. 22
2.3 Characterizations of the five benchmark programs on the evaluated systems. 26
2.4 Non-hidden memory access time and ratio of SX-7 for the five benchmark programs on one processor. 28
2.5 Relative non-hidden memory access time of the five benchmark programs normalized by the 4 B/FLOP case on SX-7. 32

3.1 Summary of setting parameters. 54
3.2 MSHR hit rates of three benchmark programs. 74

4.1 Summary of setting parameters. 98
4.2 Cache hit rate 105

List of Source Codes

1.1 Do Loop in Fortran code. 4

3.1 A kernel loop of FDTD (the GPR simulation). 46

3.2 Kernel Loop (1). 55

3.3 Kernel Loop (2). 55

3.4 A kernel loop of the PRF simulation. 55

3.5 A kernel loop of the PRF simulation. 56

3.6 A kernel loop of the SFHT simulation. 57

3.7 A kernel loop of the PBM simulation. 58

3.8 Outer-unrolling loop of the PBM simulation. 58

Chapter 1

Introduction

1.1 Roles of Supercomputers

Since the invention of computers, scientists, mathematicians and engineers began using revolutionary computers that rapidly performed complex calculations needed in the research and development. In the past decades, computer modeling and simulations of physical phenomena and engineered systems have become widely recognized as the *third pillar* to support science and technology, in addition to theory and experiment. Complex systems such as aircraft, proteins, human organs, global climate, space and nuclear are analyzed and better understood through computational models. With advances in computing power, scientists will be able to model such complex systems in greater detail, and eventually to couple individual models to understand the behavior of an entire system.

In the circumstances, computer simulations are performed on computing platforms ranging from simple workstations to very large and powerful systems: supercomputers. Solving many of various important scientific and engineering problems requires supercomputers. The supercomputers enable investigations heretofore impossible, which in turn have enabled scientific and technological advances of vast breadth and depth. Thus, supercomputing has become an indispensable tool in science and technology.

The Ministry of Education, Culture, Sports, Science and Technology in Japan (MEXT) has been promoting “R&D Project for Innovative Simulation Software” from 2005 [51]. In order to further contribute to the progress of science and technology and to strengthen industrial competitiveness, the project has started with some new simulation software contents, and involves the research and development of world-class multi-scale multi-physics simulation software that enables the more complex coupling models. These complex simulations need Peta flop/s performance of supercomputers, and higher supercomputers are strongly required for the future research and development.

1.2 Vector Processors

Vector supercomputers are high speed and high-end computer systems designed for the use of large-scale numerical intensive applications. In this dissertation the vector processors architecture is mainly discussed, then an overview of the vector processors architecture is presented in this section.

Vector supercomputers appeared to be CDC STAR-100 with peak performance reaching 75 Mflop/s in 1974. The vector architecture was first fully exploited in Cray-1 in 1976. Instead of leaving the data in memory like STAR-100, the Cray design had eight vector registers which held 64 64-bit words each. Cray-1 demonstrated extremely high performance in various scientific applications by the vector registers. Cray-1 normally had a performance of about 80 Mflop/s, but with up to three chains running it could peak at 240 Mflop/s. In 1980s, Japanese computer vendors, Fujitsu, Hitachi and NEC, introduced their vector processor systems, VP-200, S810 and SX-2, respectively, which were basically improved versions of Cray-1.

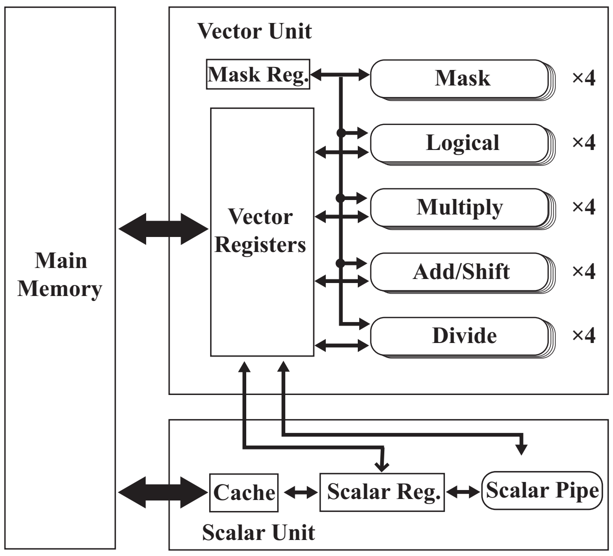


Figure 1.1: Block diagram of the NEC SX vector processor.

The vector processor simultaneously performs mathematical operations on multiple array data elements, called *vector*, by instructions named vector instructions [69]. The vector processor is categorized into SIMD (Single Instruction Multiple Data stream), and usually consists of a scalar unit and a vector unit as shown in Figure 1.1. The scalar unit is similar to an ordinary pipelined scalar processor, which executes scalar instructions for control functions, unvectorizable part of the operating system and applications. The vector unit consists of vector registers and pipelined arithmetic units. The vector registers are high speed temporary memory that holds a part of vector data on a main memory. The maximum number of elements to be held in a vector register, Maximum Vector Length, is 64 in Cray-1 and 256 in the NEC SX systems. The pipelined arithmetic units are usually made up of Add, Multiply, Divide, Logical and Shift units that are operated in a pipelined fashion, in which the vector data are input from vector registers, and results are output every clock cycle into the vector registers.

Source Code 1.1: Do Loop in Fortran code.

```fortran
DO i = 1 , N
    A(i) = X(i) + Y(i) * Z(i)
END DO
```

Vectorization is the process of converting a program to a sequence of vector instructions for executing the program in a vector processor. Source Code 1.1 shows a DO Loop in a Fortran code, and Figure 1.2 shows its converted vector instructions. Here $V0$, $V1$, $V2$, $V3$ and $V4$ indicate vector registers, which hold 256 elements in the case of the NEC SX systems. Each vector instruction processes 256 elements at once. In addition, vector supercomputers hide the memory access times by pipelined vector operations. Figure 1.3 shows a pipeline diagram of Source Code 1.1. Here, Each parallelogram shows load/store pipelines, multi pipeline, and add pipeline. The load times of arrays Z and X are hidden by the vector operations of $V0 * V1$ and $V2 + V3$.

```
loop:  VLoad V0, Yi          : V0 <--- Yi
       VLoad V1, Zi          : V1 <--- Zi
       VMulti V2, V0, V1     : V2 <--- V0 * V1
       VLoad V3, Xi          : V3 <--- Xi
       VAdd V4 ,V2, V3       : V4 <--- V2 + V3
       VStore Ai, V4         : Ai <--- V4
       Ble i, n, end         : if i = n, go to end
       be loop               : go to loop
end:
```

Figure 1.2: Vector instructions of Source Code 1.1.

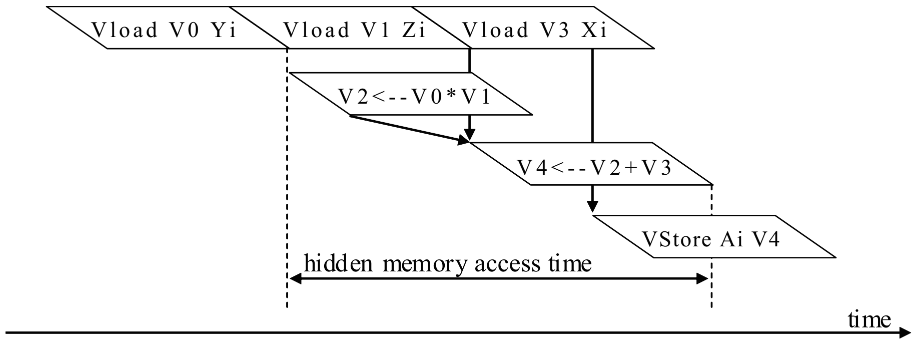


Figure 1.3: Pipeline diagram.

In order to achieve the higher sustained performance for a program, increasing a vectorization ratio is extremely important for a vector processor. The vectorization ratio α is defined by a scalar computation time ratio of vectorized calculation parts Tv to the whole calculation T as shown in Figure 1.4. The calculation time also depends on a speed ratio β , which is a ratio of vector to scalar computation performance in the vectorized calculation parts. The speed-up factor P is expressed with α and β as

$$P = \frac{1}{(1-\alpha) + \frac{\alpha}{\beta}}. \tag{1.1}$$

To improve the sustained performance, both the vectorization ratio and the speed ratio need to be increased as large as possible.

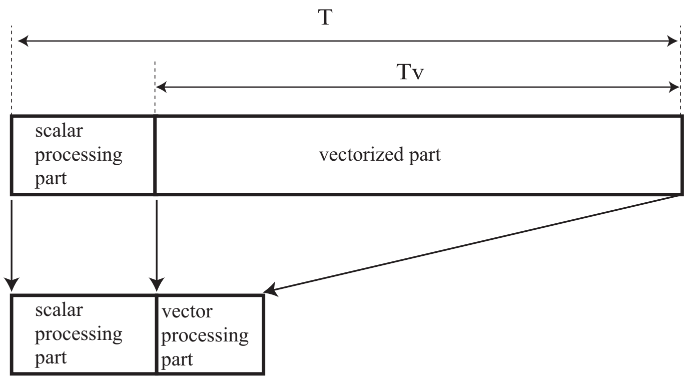


Figure 1.4: Decrease in computation time by vectorization.

Moreover, the sustained performance depends on a vector length, which is a number of iterations in DO Loop. A vector processing has an overhead time; a start-up time of vector pipeline including the preprocessing by scalar operations for vector operations. Thus, the start-up time becomes prominent in the case of the shorter vector length. The vector length generally needs 20 or more.

The memory performance is a key factor to increase the sustained performance in supercomputers. Vector supercomputers use an interleaved memory system for the main memory. The interleaved memory system organizes memory chips in banks to access multiple words at a time. The memory latency of the second memory access or later are hidden in the interleaved memory system. Here, the certain number of memory banks named *minimum number of banks* is needed to hide the memory latency (bank cycle time) by sequential memory access. The minimum number of banks N_m is

$$N_m = B_w \times B_c / D \tag{1.2}$$

where B_w is memory bandwidth (GB/s) per processor, B_c is bank cycle time of memory (ns), and D is the size of a word for floating-point data: 8 bytes. Specifically, N_m of NEC SX-7 [29] is 353 banks per processor, and N_m of NEC SX-8 [66] is 512 banks per processor. SX-7 contains 512 banks per processor, 16,384 banks per node, and SX-8 contains 512 banks per processor, 4,096 banks per node. Thus, SX-7 has a margin of 159 banks; however, SX-8 does not have a margin.

1.3 Issues of Supercomputers

1.3.1 Issues of scalar systems

Supercomputer systems are categorized into vector and scalar systems. The mainstream of supercomputers has been dominated by the commodity-based scalar systems.

On the scalar systems, the 1970s saw the emergence of the microprocessor, and RISC (reduced instruction set computer) microprocessors appeared in the mid-1980s to early-1990s. The RISC architecture has led to 20 years of sustained growth in performance at an annual rate of over 50 % [23]. Moreover, massively parallel computers appeared in 1990s. Chip multiprocessors have become the mainstream in commodity-based scalar processors from 1999. Thus, the theoretical peak performance of scalar systems has dramatically increased by the development of these computer architectures and technologies. Figure 1.5 shows the architectural classification of supercomputers in TOP500 supercomputer sites [43]. In 1993 vector supercomputers occupied 67 % of the TOP500 list, however, the number of vector supercomputers has been decreasing year by year and scalar systems have dominated in the TOP500 list.

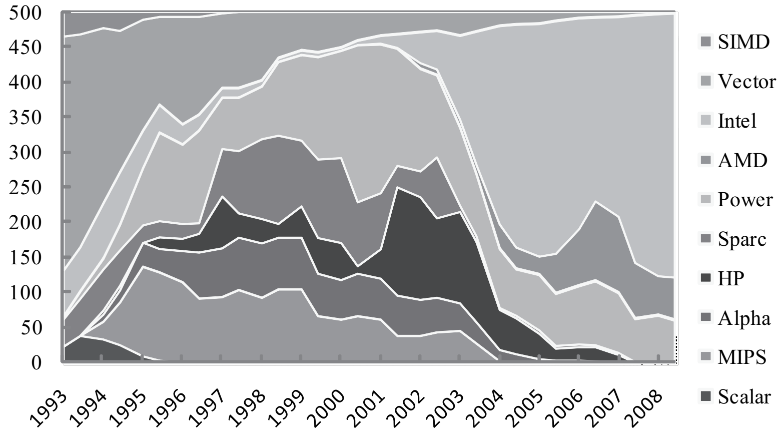


Figure 1.5: Trend in architectures of top 500 supercomputer sites.

In 2004, however, US High-End Computing Revitalization Task Force (HECRTF) reported the divergence problem that means there was the increasing gap between the theoretical peak performance and the sustained system performance for High End Computing systems of major US high-end computing centers as shown in Figure 1.6 [18]. Here, the sustained system performance is measured with a benchmark, which is specifically designed to reflect the performance of applications codes at the centers. In other words, the commodity-based scalar systems occupying the TOP500 list hardly obtain the high computation efficiency in execution of real scientific and engineering applications. HECRTF described as follows. "Continued technological improvements in microprocessor speeds driven by Moore's law result in the steeply rising upper curve of theoretical peak performance. However, the result is multiprocessor machines that are increasingly out of balance in terms of processor speed versus memory bandwidth. The imbalance produces the disappointing rise in sustained system performance displayed by the lower curve. This gap is critical because it is sustained system performance, not peak performance that is usable by applications."

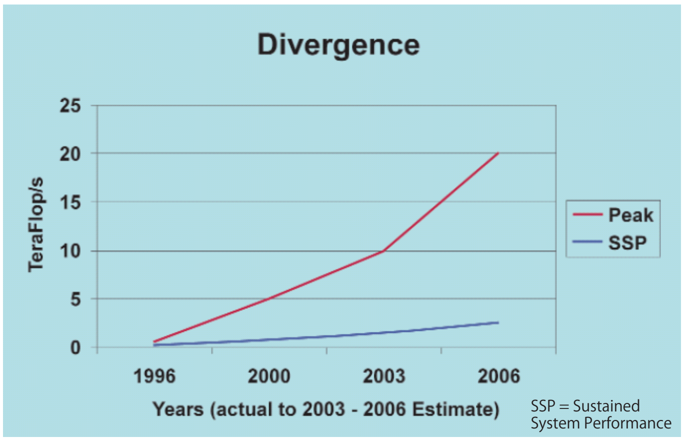


Figure 1.6: Divergence problem for US high end computer center.

1.3.2 Issues of vector supercomputers

Several researchers show that vector supercomputers achieve high sustained performance and high computation efficiencies in scientific and engineering applications [31], [38], [47], [55]. Also the Earth Simulator [10], which is the largest vector supercomputer in the world, has substantiated it in various scientific applications: sustained performances (efficiencies) of 26.58 Tflop/s (65 %) in a global atmospheric simulation [63], 16.4 Tflop/s (50 %) in a turbulence simulation [75] and 24.6 Tflop/s (75 %) in a quantum many-body problems [73]. The high sustained performance and high computation efficiencies of vector supercomputers are owing to outstanding memory performance compared to scalar systems. It will be discussed later in Chapter 2.

However, supercomputers have been encountering the memory wall problem [41], [72], because the memory performance of the supercomputers hardly follow the improvement of processor performance [23]. Moreover, the number of memory ports in a processor die does not increase owing to a limitation of the die area and electrical power. As a result, it is getting harder for vector supercomputers to keep a high memory bandwidth balanced with the processor performance. Figure 1.7 shows the trend in performance of the NEC SX systems during the last twenty years [32]. The bytes per flop rate (B/FLOP) [39], the ratio of a memory bandwidth (byte/s) to computational performance (flop/s), has decreased from 8 B/FLOP to 2.5 B/FLOP. Here, the B/FLOP is used as a technology-independent performance parameter of vector supercomputers, instead of using absolute flop/s and memory bandwidth values. Furthermore, the chip multiprocessor architecture will be adopted in vector processor design, and the gap between the memory performance and the processor performance will seriously expand. Therefore, the sustained performance begins to decrease on vector supercomputers in common with scalar systems.

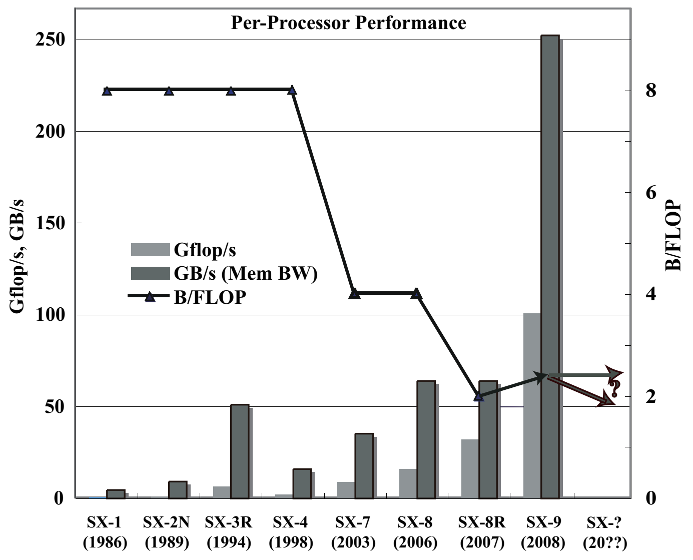


Figure 1.7: Trend in performance of the NEC SX series.

1.4 Objective of the Dissertation

Vector supercomputers encounter the memory wall problem, and the computation efficiency of the vector supercomputers decreases owing to the increasing gap between the memory bandwidth and processor performance. Then, if the memory wall problem is not be solved, future vector supercomputers hardly achieve the high computational efficiency. Hence, the main objective of this dissertation is to provide a high performance memory architecture to preserve the high computation efficiency of the future vector supercomputers. Then, this dissertation has three objectives.

The first objective is to clarify the relationship between the memory performance and the computational performance on the vector supercomputers using real scientific applications. Particularly, several important factors of memory systems are revealed for maintaining the high computation efficiency in vector supercomputers.

Then, the high performance memory architecture is proposed to overcome the memory wall problem. The aim of the high performance memory architecture is to reduce data traffic between the main memory and the vector processor, and effectively to use the memory bandwidth. Therefore, the high performance memory architecture maintains the effective memory bandwidth rate at vector register files. In the design of the memory architecture, the following mechanisms are developed.

- On-chip cache mechanism: To reuse data supplied to a vector processor
- Miss status handling registers mechanism: To reduce redundant data supplied from the main memory
- Prefetch mechanism: To hide long memory latencies

- Selective caching mechanism: To effectively use the cache capacity

The second objective certifies that the high performance memory architecture is an effective mechanism to achieve the high computation efficiency of the future vector supercomputers. Through the experiments using real application codes, characteristics of the high performance memory architecture are clarified, and effects of miss status handling registers and prefetch mechanisms are examined.

Vector processors will employ a chip multiprocessor architecture in the near future. Hence, third objective is to clarify the relationship between the scalability of chip-multiprocessing and the B/FLOP rate, and establish a shared cache architecture that boosts the performance of the chip multiprocessor architecture under low B/FLOP rate.

1.5 Organization of the Dissertation

This dissertation is organized as follows. In Chapter 1, the background of computer simulations is described, and the importance of supercomputers is mentioned to advance research and development in science and engineering. In particular, the complex simulations such as multi-scale and multi-physics simulations require Peta flop/s performance supercomputers.

In Chapter 2, the performance of supercomputers is examined from viewpoint of memory performance using real scientific applications, and the effect of memory architectures is clarified. Moreover, the effects of B/FLOP and minimum number of banks are quantitatively discussed on the sustained performance of the scientific applications in vector supercomputers.

In Chapter 3, to overcome the memory wall problem, an on-chip cache mechanism, called *vector cache*, is introduced as the high performance memory architecture of vector supercomputers. The effects of the vector cache are evaluated using two kernel loops and five leading scientific applications, and the characteristics of the vector cache is clarified. Moreover, a prefetch mechanism, miss status handling registers and selective caching are investigated to improve the effect of the vector cache.

In Chapter 4, a chip multiprocessor architecture is introduced in vector supercomputers. The scalability of scientific applications is evaluated on four cores of multi vector processor when changing the B/FLOP rate from 4 to 1. Moreover, the effect of a shared cached is discussed on the improvement of the scalability.

Finally, the conclusions of the dissertation are given in Chapter 5.

Chapter 2

Memory Performance for Highly Efficient Supercomputing of Scientific Applications

2.1 Introduction

As shown in Chapter 1, supercomputer systems are categorized into vector and scalar systems. The mainstream of supercomputers has been dominated by the commodity-based scalar systems. However, the growing gap between sustained and peak performance for real scientific applications on the scalar systems has become remarkably exposed year and year. The sustained performance of supercomputers strongly depends on their memory systems. The vector supercomputers employ the interleaved memory systems to improve the memory access performance, while the scalar systems use the hierarchical cache memory systems. In this chapter, supercomputers are evaluated from the viewpoint of memory access performance using real scientific applications, and then the requirements for high performance computing of the scientific applications are clarified. The contribution of this chapter is to quantitatively discuss the effects of B/FLOP and the number of memory banks of the vector systems on the sustained performance when executing the scientific applications in the fields of leading computational science.

The rest of the chapter is organized as follows. Section 2.2 presents related work. Sections 2.3 and 2.4 briefly describe the evaluated systems and scientific applications, respectively. In Section 2.5, performance of the memory systems on these applications is analyzed. Finally, Sections 2.6 summarizes the chapter.

2.2 Related Work

The performance characteristics of the vector supercomputers have been researched since 1980's. Fatoohi has provided simple models of the performance in the vector supercomputers, Cray-2, Cray Y-MP, EAT10-Q and NEC SX-2 using representative DAXPY-like kernel loops [16], [17]. He shows that the important factors of the sustained performance on the vector supercomputers are the average vector length, the ratio of floating point operations to memory references and the memory strides.

Shan and Strohmaier have investigated the memory performance characteristics of a modern vector supercomputer: Cray X1, and show that the average vector length and the memory bank conflicts have a great impact on the sustained performance [62]. In addition, Dunigan et al. have evaluated the performance of Cray X1, and show that the high memory bandwidth improves the performance of scientific applications [14].

Oliker et al. have compared the performance of the vector supercomputers against the scalar systems [52], [53], [55]. They evaluated the performance of the NEC SX-6 vector supercomputer and the IBM Power4 scalar system using the STREAM benchmark [40] and the NAS Parallel Benchmarks [12]. They demonstrated that SX-6 significantly outperforms the IBM Power4 system in the STREAM benchmark and the NAS parallel Benchmarks. Moreover, they have compared the application performance of the scalar systems, IBM Power, Intel Itanium2 and AMD Opteron with the performance of the vector supercomputers, Cray X1, NEC SX and Earth Simulator using leading scientific applications in four areas: atmospheric modeling, magnetic fusion, plasma physics and material science. They show that the vector supercomputers have the potential to achieve excellent performance on scientific applications owing to their higher memory bandwidth. However, they have not quantitatively discussed the effect of the memory bandwidth on their performance.

2.3 Architectural Characteristics of the Evaluated Systems

The sustained performance of supercomputers is considered to depend on their memory system. However, the effect of the memory system has not been quantitatively discussed, thus the relationship between the memory performance and the computational performance should be clarified on supercomputers using real scientific applications.

In this chapter, the performance of vector systems: NEC SX-7 [29] and SX-7C [66] are compared with the performance of scalar systems: NEC TX7 [59] and SGI Altix3700 [61] using scientific applications. SX-7 and SX-7C are representative systems in modern vector systems, and their architectures are similar to Earth Simulator. Table 2.1 summarizes the architectural characteristics of the four systems. The memory bandwidths of SX-7 and SX-7C are 5.5 and 10 times higher than the memory bandwidth of TX-7 and Altix, respectively. On the other hand, the scalar systems employ large on-chip caches to cover the lower memory bandwidth.

Table 2.1: Architectural summary of the systems.

system	CPUs per Node	Clock Freq. (GHz)	Per CPU			Processor Types
			Peak Perf. (Gflop/s)	Mem. BW (GB/s)	L3 Cache (MB)	
SX-7	32	1.1	8.83	35.3	-	Custom
SX-7C	8	2.0	16.0	64.0	-	Custom
TX7/i9510	32	1.6	6.4	6.4	9	Intel Itanium2
Altix3700	64	1.6	6.4	6.4	6	Intel Itanium2

The architecture of SX-7C is equivalent to the architecture of SX-8.

2.3.1 Vector supercomputers: NEC SX-7 and SX-7C

SX-7 and SX-7C are shared-memory vector systems. A node of SX-7 contains 32 processors with the total peak performance of 282.5 Gflop/s and a 256 GB main memory, and a node of SX-7C contains eight processors with the total peak performance of 128 Gflop/s and a 128 GB main memory. SX-7 and SX-7C run SUPER-UX (R14.1, and R15.1, respectively) a 64-bit UNIX operating system. FORTRAN compiler, FORTRAN90/SX R.316, supports ANSI/ISO Fortran95 in addition to functions of automatic vectorization and automatic parallelization.

Their processor has a vector operation unit and a 4-way superscalar operation unit. The SX-7's vector operation unit contains four vector pipes (Logical, Add/Shift, Multiply, Divide) with 144 KB vector registers, and achieves a peak performance of 8.83 Gflop/s. Similarly, the SX-7C's vector operation unit contains four vector pipes (Logical, Add/Shift, Multiply, Divide/SQRT) with 144 KB vector registers, and achieves a peak performance of 16 Gflop/s. The 4-way superscalar operation units of SX-7 and SX-7C achieve peak performances of 1.1 Gflop/s and 2 Gflop/s, respectively. The memory bandwidths of SX-7 and SX-7C are 35.3 GB/s with DDR-SDRAMs and 64 GB/s with DDR2-SDRAMs, respectively. The memory bandwidth per flop/s of SX-7 and SX-7C is 4 B/FLOP.

2.3.2 Scalar systems: NEC TX7/i9510 and SGI Altix3700

TX7/i9510 and Altix3700 are ccNUMA (cache coherent Non Uniform Memory Access) systems. A node of TX7/i9510 contains eight cells and crossbar network modules. Each cell contains four Intel Itanium2 processors and a 32 GB main memory, which are interconnected by a 6.4 GB/s bus. A computational building block of Altix3700 consists of four Intel Itanium2 processors, main memory, and two controller ASICs called the SHUB, which connect the processors and memory at 6.4 GB/s bandwidth. Altix interconnect is called the NUMAlink, a custom network in the fat-tree topology [60]. Itanium2 has 3-tier on-chip data caches consisting of 32 KB of L1, 256 KB of L2, and 6 MB (Altix) / 9 MB (TX7) of L3. Itanium2 does not use the L1 data cache to store floating-point data [25].

The performance of memory system depends on the cache system. In the case of TX-7, the memory access time is 20 times or more as long as the L3 cache access time. Therefore, when a cache hit rate is low, a memory access time becomes dominant in the total processing time; the computation efficiency on the cache based systems gets worse accordingly.

TX7 and Altix run 64-bit Linux (RedHat AS2.1 and SGI Advanced Linux Environment, respectively), and supports Fortran95 (NEC R4.3) with optimization functions and parallel processing functions specialized for Itanium2.

2.4 Benchmark Programs

Five leading scientific applications from three areas in scientific computing are used to compare the sustained performance of SX-7 and SX-7C with the sustained performance of TX7 and Altix3700. The benchmark programs have been developed by researchers of Tohoku University and are representative of each research area. Table 2.2 shows the summary of the benchmark programs whose methods are standard in individual areas.

Table 2.2: Summary of benchmark programs.

Area	Name and Description	Method Subdivision	Memory Size
Electro-magnetic Analysis	GPR simulation: Simulation of Array Antenna Ground Penetrating Radar	FDTD $50\times750\times750$	8.9 GB
	APFA simulation: Simulation of Anti-Podal Fermi Antenna	FDTD $612\times105\times505$	12 GB
CFD/Heat Analysis	PRF simulation: Simulation of Premixed Reactive Flow in Combustion	Compact Finite Difference Scheme 513×513	1.4 GB
	SFHT simulation: Simulation of Separated Flow and Heat Transfer	SMAC $711\times91\times221$	6.6 GB
Earth Science	PBM simulation: Simulation of Plate Boundary Model on Seismic Slow Slip	Friction Law 32400×32400	8 GB

2.4.1 GPR simulation

The GPR simulation is for a simulation of an array antenna SAR-GPR (Synthetic Aperture Radar - Ground Penetrating Radar), which detects anti personnel mines in shallow subsurface [33], [58]. The GPR simulation evaluates performance of the SAR-GPR in detection of buried mines. The simulation method is the three dimensional FDTD (Finite Difference Time Domain) method with Berenger's PML (Perfectly matched layer) [37]. The FDTD method is a computational electrodynamics modeling technique. The time-dependent Maxwell's equations are discretized using central-difference approximations to the space and time partial derivatives. The electric field vector components in a volume of space are solved at a given instant in time; then the magnetic field vector components in the same spatial volume are solved at the next instant in time.

The simulation space consists of two regions; air-space and subsurface space with PML of 10 layers. The performance of this code is primarily determined by the electromagnetic field calculation processes. The basic computational structure of the processes consists of triple-nested loops accessing the memory at non-stride-1 addresses; the ratio of its calculation cost to the total is 80 %. The length of the innermost loop is over 500. The computational intensity, the ratio of floating-point operations to memory references [9], is 1.1 in this code.

2.4.2 APFA simulation

Radiation patterns of an Anti-Podal Fermi Antenna (APFA) are simulated to design high gain antennas [67]. The simulation consists of two sections, a calculation of the electromagnetic field around an APFA using the FDTD method with Berenger's PML, and an analysis of the radiation patterns using the Fourier transform. The performance of the simulation is primarily determined by calculations of the radiation patterns; the ratio of its calculation cost to the total is 99 %. The computational structure of the calculations is triple-nested loops; the innermost loop is a stride-1 loop, and its length is 255. On Itanium2, the innermost loop is executed on the caches. The computational intensity in the loop is 2.25.

Therefore, this code is computational-intensive, and the performance of the code is not dominated by memory references.

2.4.3 PRF simulation

The PRF simulation provides numerical simulations of two-dimensional Premixed Reactive Flow (PRF) in combustion for the intrinsic instabilities of two-dimensional hydrogen/air premixed planar flames [68]. The simulation uses the 6th-order compact finite difference scheme and the 3rd-order Runge-Kutta method for time advancement to solve Navier-Stokes equations. The hydrogen/air detailed kinetics use Stahl and Warnatz model. Here, this simulation assumes constant density and one-step reaction.

The performance of the code is primarily determined by calculations of derivations of physical equations; the ratio of its calculation cost to the total is 50 %, and the rest of the cost has been distributed to various routines. The calculations have doubly nested loops; the loop of x-derivations induces stride-1 memory accesses, and the loop of y-derivations induces non-stride-1 memory accesses. The length of each innermost loop is 513. The computational intensity is 0.7 in this code.

2.4.4 SFHT simulation

The SFHT simulation realizes direct numerical simulations of three-dimensional laminar Separated Flow and Heat Transfer (SFHT) on surfaces of a plane [48], [74]. Fundamental equations are the continuity, momentum and energy for a three dimensional unsteady flow of incompressible viscous fluid with constant properties. The finite-difference forms are the 5th-order upwind difference scheme for space derivatives and the Crank-Nicholson method for a time derivative. The resulting finite-difference equations are solved using the SMAC method.

The performance of the code is primarily determined by calculations of the predictor-corrector methods; the ratio of its calculation cost to the total is 67 %, and the rest of the cost has been distributed to various routines. The calculations have triple-nested loops; the innermost loop needs stride-1 memory accesses, and its length is 349. The computational intensity is 1.0 in this code.

2.4.5 PBM simulation

The PBM simulation uses the three-dimensional numerical Plate Boundary Models (PBM) to explain an observed variation in propagation speed of post-seismic slip [4]. This is a quasi-static simulation in an elastic half-space including a rate- and state-dependent friction. The performance of the simulation is primarily determined by a process of thrust stress with the Green function; the ratio of its calculation cost to the total is 99 %. The computational structure of the process is a doubly nested loop which calculates a product of matrices, the innermost loop results in stride-1 memory accesses, and its length is 32400. The PBM simulation has two versions: an outer unrolling version and a non outer unrolling version. The computational intensities are 1.9 in the outer unrolling version, and 1.0 in the non unrolling version. The performance of the outer unrolling version is higher than the performance of the non unrolling version. The outer unrolling version is evaluated in this chapter.

2.5 Experimental Results and Discussion

The experiments conducted in this work measure the performance of the original source codes, which have been developed for SX-7, with optimizations of the compilers; compiler's options are high-level optimizations (SX: -C hopt, TX, Altix: -O3) and inlining subroutines. NEC compiler was used for Intel Iitanium2 on TX7 and Altix, to evaluate the performance under the same level optimizations. On SX-7 and SX-7C, the five benchmark programs are vectorized by the compiler. The benchmark programs are automatically parallelized by the compiler on the four studied systems. All the performance statistics of the four studied systems were obtained using the NEC compiler option $ftrace$ [50].

2.5.1 Characterizations of benchmark programs

To characterize computation behavior this section shows a vector operation ratio (VOR) and an average vector elements (double-precision floating-point data) per vector instruction (AVL) on the vector supercomputers, the L2 and L3 cache hit rates on Itanium2, and a parallel ratio (PR) of thread-level parallelism, which is the fraction of the code executed in parallel. As Table 2.3 shows, these five benchmark programs are highly vectorized and parallelized, and the L2 cache

Table 2.3: Characterizations of the five benchmark programs on the evaluated systems.

	SX-7/7C		TX7/i9510		Altix3700		
	VOR	AVL	L2	L3	L2	L3	PR
GPR	99.7 %	245.1	70.6 %	40.2 %	71.0 %	42.6 %	98 %
APFA	99.9 %	255.5	99.9 %	26.7 %	99.9 %	26.8 %	99 %
PRF	99.3 %	179.0	89.6 %	78.9 %	89.5 %	79.9 %	93 %
SFHT	99.4 %	192.9	92.4 %	21.8 %	92.4 %	21.7 %	98 %
PBM	99.5 %	255.5	88.7 %	54.8 %	88.9 %	63.2 %	98 %

hit rates range from 70 % to 99.9 % according to their irregularity in memory accesses.

2.5.2 Efficiency of benchmark programs on the four systems

The overall performance comparison of the four systems for the five benchmark programs is shown in Figure 2.1. The vector supercomputers achieve the high computation efficiency of 40 % or more and the higher sustained performance across all of the benchmark programs. The scalar systems show that the computation efficiency is less than 14 % across all of the benchmark programs.

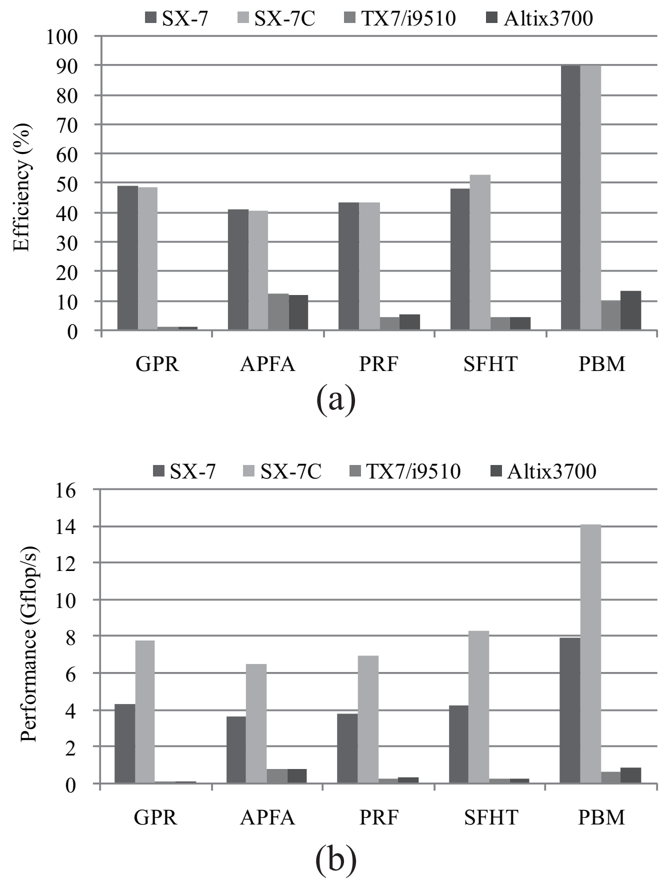


Figure 2.1: Overview of performance for the five benchmark programs on one processor, (a) computation efficiency and (b) sustained performance.

SX-7 and SX-7C hide the memory access times by the interleaved memory system and the pipelined vector operations, because VOR and AVL of the five benchmark programs are large. The memory access times not hidden by overlapping calculations of the four systems are shown in Table 2.4. The non-hidden memory access times of the vector supercomputers are much shorter than the non-hidden memory access times of the scalar systems. In particular, the non-hidden memory access time of the PBM simulation on SX-7 is 5 seconds. However, the sum of calculated floating-point data is a 13.5 trillion; 108 TB in the PBM simulation. Hiding memory access latency by pipelined vector operations works best for the PBM simulation, because the PBM simulation has the longest loop length (32400) among the five benchmark programs and further sequentially accesses memory. On the cache based systems, TX7 and Altix, the non-hidden memory access times are 600+ times longer than the non-hidden memory access times of SX-7. In the longer loops of larger simulations, the vector supercomputers are more advantageous in the performance.

Figure 2.2 shows the ratio of non-hidden memory access time to processing time for the five benchmark programs on one processor of the four systems. The processing time of the scalar systems consists mostly of the non-hidden memory

Table 2.4: Non-hidden memory access time and ratio of SX-7 for the five benchmark programs on one processor.

	GPR		APFA		PRF		SFHT		PBM	
System	Time	Ratio	Time	Ratio	Time	Ratio	Time	Ratio	Time	Ratio
SX-7	171	1	17	1	40	1	81	1	5	1
SX-7C	90	0.5	3	0.2	21	0.5	32	0.4	6	1.2
TX7	23399	137	568	34	2170	54	3674	45	5652	1082
Altix	26319	154	612	37	1745	43	3910	48	3323	636

Time : seconds

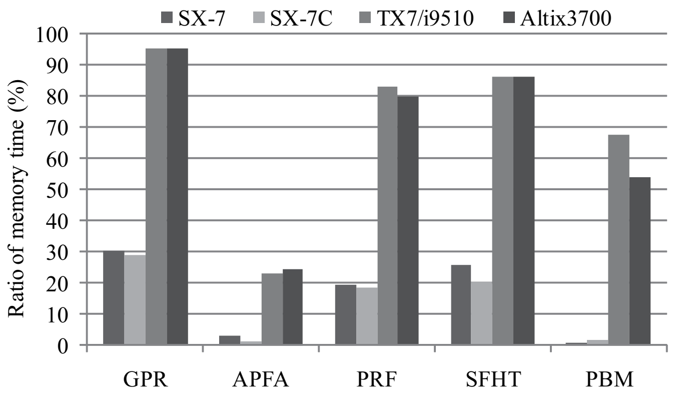


Figure 2.2: Ratio of non-hidden memory access time to processing time on one processor.

access time. The APFA simulation achieving a 99.9 % cache hit rate shows the smallest ratio of non-hidden memory access time among the five benchmark programs on the scalar systems; however the ratio is still over 20 %. The memory systems of the vector supercomputers are more effective for scientific applications than those of the scalar systems.

Figure 2.3 shows the speedup ratio in 32 processors of the studied systems for the five benchmark programs. The speedup ratio of the PRF simulation is the lowest among the five benchmark programs on each system, because the parallel ratio (PR) of the PRF is 93 % and lowest. SX-7 outperforms the other systems

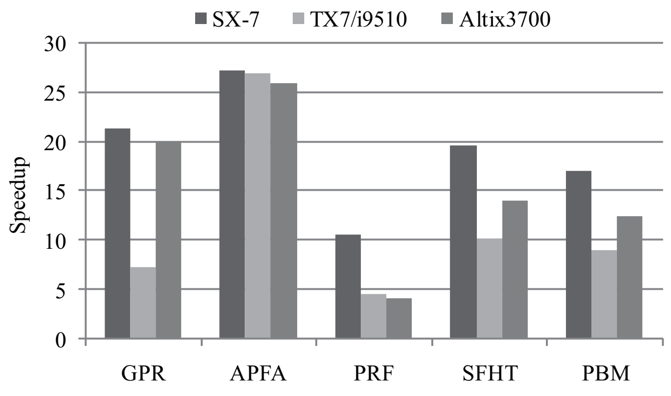


Figure 2.3: Speedup ratio in 32 processors for the five benchmark programs.

across all of the benchmark programs. TX7 and Altix utilize the same processors; however, Altix scalability is higher than TX7. This is owing to many access contentions on TX-7 bus with the lower bandwidth.

2.5.3 Discussion on the memory performance of SX-7 and SX-7C

The PRF and SFHT simulations are used to examine the non-overlapped memory access latencies when changing the number of banks per processor on SX-7. The performance of the PRF simulation is dominated by memory references, because the memory access has a 4104-byte stride, and the computational intensity is 0.7. In the SFHT simulation, the memory access needs a 16-byte stride, and the computational intensity is 1.0. Thus, the PFR simulation is more memory-intensive than the SFHT simulation. Figure 2.4 shows the non-hidden memory access time normalized by the non-hidden memory access time of SX-7 with 16K banks. The experimental results indicate that the non-hidden memory access time increases as the number of banks decreases. When one processor of SX-7 has 16K banks, the non-hidden memory access times of the PRF and SFHT simulations are 40 and 81 seconds, respectively. When one processor has 0.5K banks, the non-hidden memory access times of the PRF and SFHT simulations are 113 and 131 seconds, respectively. The non-hidden memory access time of the PFR simulation increases faster than The non-hidden memory access time of the SFHT simulation, because the strides of memory access of the PFR simulation are longer than the strides of memory access of the SFHT simulation. Then, the PFR simulation requires more banks to reduce the memory access time.

In general, various scientific applications need non-stride-1 memory accesses, and therefore the number of banks per processor needs more than the minimum

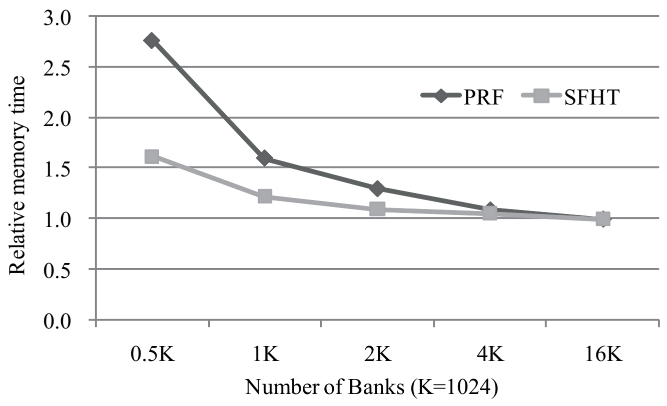


Figure 2.4: Relative non-hidden memory access time as a function of the number of banks.

number of banks to keep the higher computation efficiency. To evaluate the effect of the number of memory banks on the performance, the performance of the 8-parallel GPR simulation is compared between SX-7 and SX-7C. The memory access of the GPR simulation has a 576-byte stride. Figure 2.5 shows that the efficiency of SX-7 is 1.7 times higher than the efficiency of SX-7C in eight processors. Here, the number in each bar indicates the number of banks per processor. Although the peak performance of SX-7C is 1.8 times faster than the peak performance of SX-7, SX-7 and SX-7C are comparable in the sustained performance of the GPR simulation using eight processors. On SX-7 and SX-7C, when a benchmark program uses eight processors in a node, each processor uses 2K banks in SX-7 and 0.5K banks in SX-7C. As discussed in Section 1.2, SX-7C does not have the margin in the number of banks. In the case of non-stride-1 memory accesses, the processing time on eight processors of SX-7C increases owing to the memory access latency not hidden by the interleaved memory. On the other hand, SX-7 has the margin in the number of banks, and SX-7 is superior to SX-7C from a viewpoint of the capability to hide the memory access latency. Therefore, SX-7C requires more banks for more effective computing and scalable performance when using entire processors of one node.

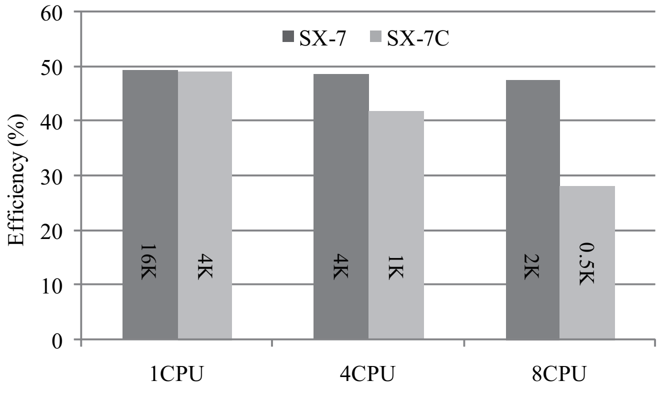


Figure 2.5: Efficiency of the GPR simulation on SX-7 and SX-7C.

Table 2.5: Relative non-hidden memory access time of the five benchmark programs normalized by the 4 B/FLOP case on SX-7.

B/FLOP	GPR	APFA	PRF	SFHT	PBM
4	1.0	1.0	1.0	1.0	1.0
2	3.5	3.0	2.2	3.5	75.6
1	9.5	11.5	5.7	10.1	316.7

It is investigated that the memory bandwidth per processor affects the non-overlapped memory access time of SX-7. Table 2.5 shows the results of relative memory access times on each application when the memory bandwidth of SX-7 is reduced by partially shutting off network switches between processors and memory units. When the memory bandwidth is adjusted to 1/2 (2 B/FLOP), the memory access time is two or more times longer than the memory access time of the 4 B/FLOP case. When the memory bandwidth is reduced to 1/4 (1 B/FLOP), the memory access time is four or more times longer than the memory access time of the 4 B/FLOP case, which is almost comparable to the cases of the scalar systems. As the memory bandwidth decreases, the memory read/write time increases, and the memory access time is not hidden by the pipelined vector operations. In the PBM simulation, the memory access time not hidden by vector operations is 316 times as long as that of the 4 B/FLOP case. Figure 2.6 shows the ratio of the sustained performance to the peak performance when changing the memory bandwidth from 4 B/FLOP to 2 B/FLOP and 1 B/FLOP in SX-7. The sustained performance of these benchmark programs seriously goes down as the memory bandwidth decreases. In particular, the performances of the GPR and PBM simulations are degraded by half and quarter when the memory bandwidth is reduced to 2 B/FLOP and 1 B/FLOP from 4 B/FLOP, because these benchmark programs are memory-intensive. Therefore, the sustained performance is seriously affected by the B/FLOP rates, and a memory bandwidth of the 4 B/FLOP is essential to keep the superiority of the vector supercomputers against the scalar systems.

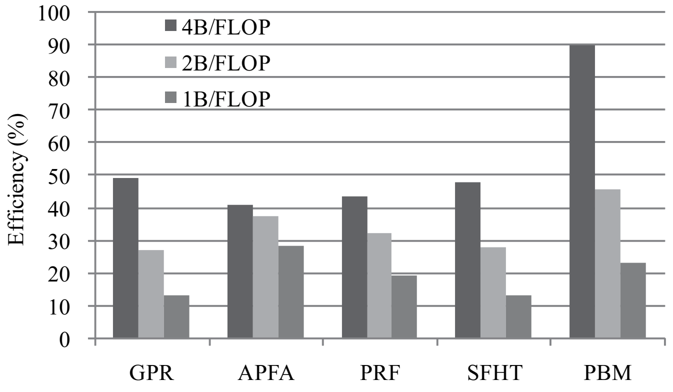


Figure 2.6: Computation efficiency of the five benchmark programs when changing the B/FLOP rate in SX-7.

2.5.4 Discussion on the memory performance of TX7/i9510 and Altix3700

The performance of TX7/i9510 and Altix3700 depends on the cache hit rate. Figure 2.7 presents the correlation between the cache hit rate and the ratio of the memory access time to processing time of the five benchmark programs on one processor; here, the cache hit rate is a sum of the L2 and L3 caches. The ratio of non-hidden memory access time becomes more than 50 % even when the cache hit rate is 95 %. Therefore, the cache hit rate needs to be almost 100 % to achieve the high computation efficiency on the cache based systems.

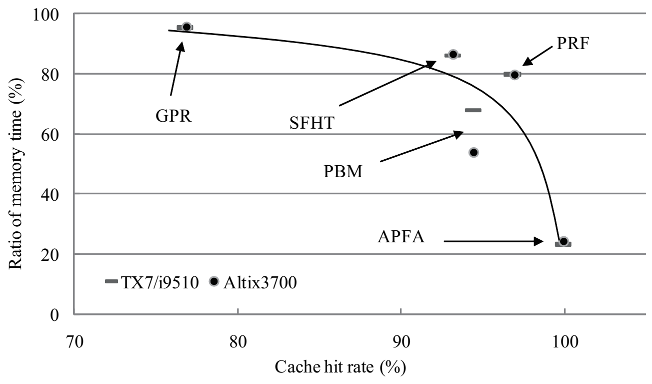


Figure 2.7: Correlation of cache hit rate and ratio of memory access time to processing time on TX7 and Altix.

The GPR simulation has a low cache hit rate and the memory-intensive. Figure 2.8 is the processing time of the GPR simulation, and shows that the memory access time of Altix decreases constantly. On the other hand, the memory access time of TX7 does not decrease in the case of eight or more processors, because the system buses of TX7 connecting processors to memory are saturated with the data transfers. On TX7 and Altix, a cell card contains processors and a main memory, which are interconnected by a 6.4 GB/s bus. The cell cards of TX7 and Altix contain four processors and two processors, respectively. In the experiment, TX7 and Altix consist of eight cell cards and 32 cell cards, respectively. In the case of eight or more processors, TX7 uses two or more processors per cell, and the bus of TX7 is more likely to saturate with data transfers between processors and a memory than that of Altix, because two or more processors of a TX7 cell share the 6.4 GB/s bus. Therefore, the experimental results suggest that it is necessary for system configurations of the cache based systems not to saturate the bus with data transfers on the bus.

Meanwhile, the APFA simulation has a high cache hit rate, and the computational intensity is 2.2. Thus, the APFA simulation is less memory-intensive than the GPR simulation. The experimental results of the APFA simulation shown in Figure 2.9 indicate that the non-hidden memory access time of TX7 and Altix decreases in the case of eight or more processors. In this case, the buses of TX7 are not saturated with data transfers between processors and a memory.

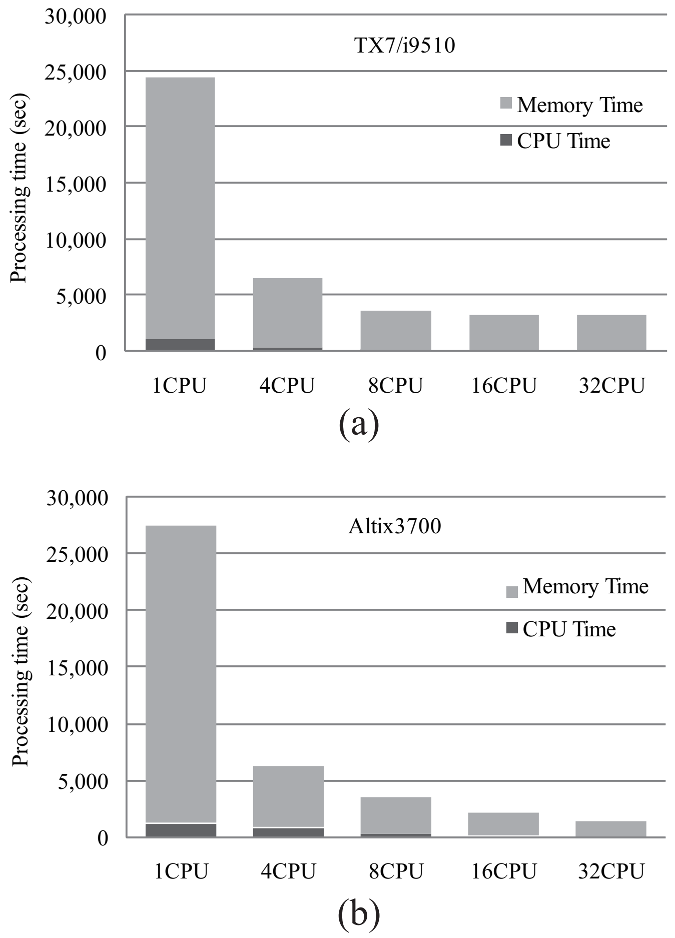
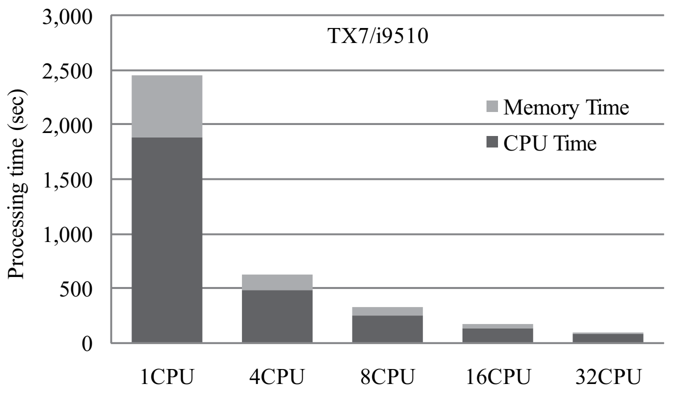
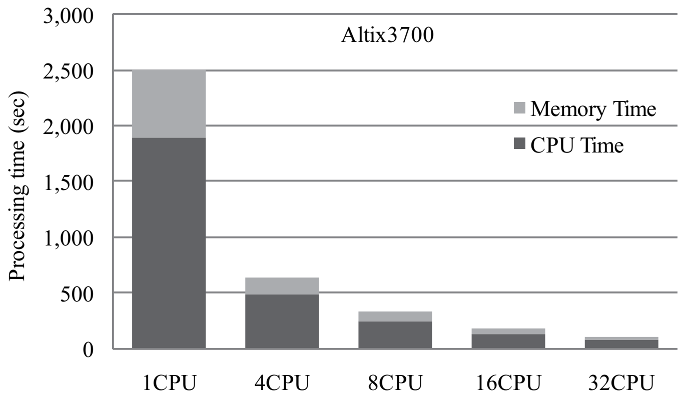


Figure 2.8: Processing time of the GPR simulation: (a) TX7 and (b) Altix.



(a)



(b)

Figure 2.9: Processing time of the APFA simulation: (a) TX7 and (b) Altix.

2.6 Conclusions

This chapter has presented the memory performance of the vector supercomputers of SX-7 and SX-7C and compared it against the cache based scalar systems of TX7/i9510 and Altix3700 using five scientific applications from three areas. The experimental results show that the vector supercomputers achieve the high efficiency and significantly outperformed the scalar systems. It has quantitatively been presented that the important factor affecting the computational performance on scientific applications is the memory performance. The vector supercomputers use the interleaved memory systems, and their memory access latencies are hidden by pipelined vector operations. It have been confirmed that the high performance of the vector supercomputers is obtained owing to a high memory bandwidth and a large number of banks. These experiments using practical application codes have shown that both a balanced performance of the high B/FLOP and the enough number of memory banks that exceeds the minimum number of banks to hide the bank cycle time are essential to achieve the higher sustained performance. Especially, the sustained performance is seriously affected by the B/FLOP rates, and a memory bandwidth of the 4 B/FLOP is essential to keep the superiority of the vector supercomputers against the scalar systems.

On the scalar systems, the computational performance depends mainly on cache hit rates. It have quantitatively been presented the correlation between the cache hit rate and the ratio of the memory access time to processing time of the five applications, and have confirmed that the cache hit rate needs almost 100 % to achieve efficient computing. Additionally, the computational performance also depends on the performance of the memory bus that connects processors and memory in a cell card. It have been demonstrated that the buses of TX7 are saturated with data transfers among processors, when two or more processors share a 6.4GB/s bus. To avoid such a situation the scalar systems would require maintaining the bus bandwidth per processor not to saturate the bus with data transfers and using cache effectively.

Chapter 3

An On-Chip Cache for the Vector Architecture

3.1 Introduction

Vector supercomputers have high computation efficiency for scientific applications [30]. In Chapter 2, it is shown that NEC SX vector supercomputers achieve high sustained performance in five leading applications owing to their high B/FLOP rates compared to scalar systems. Especially, the memory bandwidth of the 4 B/FLOP is essential to keep the superiority of the vector supercomputers against the scalar systems. However, as advantages in VLSI technology have also been accelerating processor speeds, supercomputers have been encountering the memory wall problem. As a result, it is getting harder for vector supercomputers to keep a high memory bandwidth balanced with the improvement of their flop/s performance. On the NEC SX systems, the B/FLOP rate has decreased from 8 B/FLOP to 2.5 B/FLOP during the last twenty years [32]. Then, preserving the sustained performance of future vector supercomputers requires a high performance memory architecture for maintaining the memory bandwidth of the 4

B/FLOP.

In this chapter, the following mechanisms are proposed as the high performance memory architecture.

- On-chip cache mechanism
- Miss status handling registers mechanism
- Prefetch mechanism
- Selective caching mechanism

An on-chip cache, called *vector cache*, reuses data that have already been supplied to the vector unit, and maintains the effective memory bandwidth rate at vector register files [46], [44], [45]. Here, the effective memory bandwidth rate indicates the memory bandwidth rate provided from both the main memory and the vector cache to the vector register files. The vector cache employs miss status handling registers (MSHR) [36] and a prefetch mechanism to improve the effect of the vector cache. These techniques have already been studied on many scalar architectures [15], [28], [57], [70]. Furthermore, the vector cache has a bypass mechanism and data are selectively cache to reduce capacity miss rates of the vector cache.

In this chapter, the performance of the vector cache is evaluated by using the NEC SX simulator, by limiting off-chip memory bandwidth at the 1 B/FLOP and 2 B/FLOP rates. The characteristics of the on-chip vector cache are clarified on the vector supercomputers.

The rest of the chapter is organized as follows. Section 3.2 presents related work. Section 3.3 indicates characteristics of scientific applications regarding the locality of reference for the effective use of caches. Section 3.4 describes a vector architecture with an on-chip vector cache. Section 3.5 provides an experimental methodology and benchmark programs for performance evaluation. Section 3.6 presents experimental results when executing the kernel loops and five benchmark programs on the vector architecture with the vector cache. Through the experimental results, the characteristics of the vector cache are clarified. Section 3.7 presents experimental results on the MSHR and the prefetch mechanism. In Section 3.8, the effect of selective caching is examined on the performance, in which only the data with high localities of reference are cached. In addition, the relationship between loop unrolling and the vector cache is discussed. Finally, Section 3.9 summarizes the chapter.

3.2 Related Work

Vector caches have been previously studied by many researchers using trace driven simulators of convectional vector architectures from the 1990's. This section presents related works of the vector cache.

Gee *et al.* have provided an evaluation of the cache performance in vector architectures: Cray X-MP and Ardent Titan [21], [22]. Their caches employ a full-associative cache and an n-way set associative cache. The line sizes range from 16 to 128 bytes, and the maximum cache size is 4 MB. Their benchmark programs are selected from Livermore Loops [42], NAS kernels [6], Los Alamos benchmark set [64] and real applications in chemistry, fluid dynamics and linear analysis. They shown that vector references contained somewhat less temporal locality, but large amounts of spatial locality compared to instruction and scalar references, and the cache improved the computational performance of the vector processors. Kontothanassis *et al.* have evaluated the cache performance of Cray C90 architecture using NAS parallel benchmarks [34]. Their cache is a direct-mapped cache with a 128 bytes line size, and the maximum cache size is 8 MB. They shown that the cache reduced memory traffic from off-chip memory, and a DRAM memory system with the cache was competitive to a SRAM memory system.

Fu *et al* have evaluated the effects of hardware-based prefetch mechanisms in the Alliant FX/8 vector processor system [19], [20]. They have proposed *stride prefecth* mechanism. In the vector processor, the address of loaded data is completely identified by the base address, stride and the position of the data. The *stride prefecth* mechanism takes advantage of the vector stride information specified in a vector instruction that loads or stores memory data to prefetch vector elements into the cache. Their prefetch mechanisms reduce the influence of long stride vector accesses and misses owing to block invalidations in a multiprocessor. Their benchmark programs are the Perfect Club collection of numerical programs. They indicated that the prefetch mechanisms are shown to have better performance than a non-prefetching cache.

Batten *et al* proposed a Vector Refill Unit with non-blocking cache to sustain high memory bandwidth in a cached vector machine [7]. The Vector Refill Unit pre-execute vector memory commands to detect which of the lines they will touch are not in cache and are prefetched. The Vector Refill Unit reduces costs by eliminating much of the outstanding miss state required in traditional vector architectures and by using the cache itself as a cost-effective prefetch buffer. They describe an implementation of the Vector Refill Unit within the context of the SCALE vector-thread processor [35] and provide an evaluation over a range of scientific and embedded kernels. They show an improvement in performance and a reduction in the hardware resources required to sustain high throughput in long latency memory systems.

Modern vector supercomputer Cray X1 has a 2 MB vector cache, called Ecache, organized as a 2-way set associative write-back cache with 32 bytes line size and the least recently used (LRU) replacement policy [2]. Moreover, Cray inc. announced a new vector supercomputer, BlackWidow, in 2006 [24]. It has a 512 KB L2 cache and a 8 MB L3 cache. The performance of Cray X1 has been evaluated using real scientific applications by several researchers [8], [13], [54], [62]. These studies have compared the performance of Cray X1 with that of other platforms; IBM Power, SGI Altix and NEC SX. On BlackWidow, Abts *et al* demonstrate the performance of a prototype hardware using the HPC challenge benchmark suite [1]. However, they have not quantitatively discussed the effect of the cache on its performance.

The aim of this chapter is to quantitatively investigate the effect of the vector cache using modern vector supercomputer NEC SX architecture. The basic characteristics of the vector cache are clarified and its effective usages are discussed.

3.3 Characteristics of Scientific Applications

Various scientific applications generally utilize difference schemes for simulating the physics phenomena. A part of arrays in the difference schemes has a high locality in a DO Loop. Source Code 3.1 shows an example of a difference scheme: a kernel routine of Finite Difference Time Domain Method (FDTD method). This method is used for the GPR and APFA simulations discussed in Chapter 2. In this loop, the following six arrays reuse the cached data, $H_y(i, j, k-1)$ (line 06 in Figure 3.1), $H_y(i, j, k)$ (line 08), $H_y(i-1, j, k)$ (line 08), $H_x(i, j, k)$ (line 11), $H_x(i, j, k-1)$ (line 11) and $H_z(i-1, j, k)$ (line 12). Therefore, the traffic between the main memory and the processor is decreased using the vector cache. Moreover, when the arrays H_x, H_y and H_z are selected for selective caching, then the capacity miss of these arrays is reduced.

However, $H_z(i, j-1, k)$ (line 05) and $H_x(i, j-1, k)$ (line 09) hardly reuse the data of $H_z(i, j, k)$ (line 05) and $H_x(i, j, k)$ (line 09) which have a locality on the index j, because the data of $H_z(i, j, k)$ and $H_x(i, j, k)$ are not filled into the cache yet when the subsequent load instructions of $H_z(i, j-1, k)$ and $H_x(i, j-1, k)$ are issued. Therefore, the same data are reloaded and it wastes the memory bandwidth. To solve this problem, the vector cache needs a mechanism which holds information of in-flight load requests, and the mechanism makes the subsequent load instructions access the in-flight load data from the vector cache as the in-flight load data arrive at the vector cache.

Moreover, the arrays except for H_x, H_y and H_z do not have a locality in this loop. However, the prefetch mechanism of the vector cache has the potential of hiding the memory access time of these arrays by other arithmetic operations processed, and the execution time of the loop is reduced. Therefore, the vector cache is effective for non reference locality arrays to increase the performance of a DO Loop.

Source Code 3.1: A kernel loop of FDTD (the GPR simulation).

```
DO 10 k=0,Nz
  DO 10 i=0,Nx
    DO 10 j=0,Ny
      E_x(i,j,k) = C_x_a(i,j,k)*E_x(i,j,k)
&         +C_x_b(i,j,k)*(( H_z(i,j,k)-H_z(i,j-1,k))/dy
&         -(H_y(i,j,k)-H_y(i,j,k-1))/dz-E_x_Current(i,j,k))
      E_z(i,j,k) = C_z_a(i,j,k)*E_z(i,j,k)
&         +C_z_b(i,j,k)*((H_y(i,j,k)-H_y(i-1,j,k) )/dx
&         -(H_x(i,j,k)-H_x(i,j-1,k) )/dy-E_z_Current(i,j,k))
      E_y(i,j,k) = C_y_a(i,j,k)*E_y(i,j,k)
&       +C_y_b(i,j,k)*((H_x(i,j,k)-H_x(i,j,k-1) )/dz
&         -(H_z(i,j,k)-H_z(i-1,j,k))/dx-E_y_Current(i,j,k))
10 CONTINUE
```

3.4 On-Chip Cache Memory for Vector Architecture

3.4.1 Basic mechanisms of a high performance memory architecture

Future vector supercomputers have difficulty preserving the 4 B/FLOP memory bandwidth rate owing to the memory wall problem. Thus, Memory architecture of the future vector supercomputers requires to use the memory bandwidth effectively. Then, four main mechanisms are proposed.

- On-chip cache mechanism (vector cache)
- Miss status handling registers mechanism
- Prefetch mechanism
- Selective caching mechanism

A vector cache reuses the data supplied to vector processor when the data have locality, and the traffic of the data with locality between the main memory and the processor is decreased. On vector supercomputers, the memory system employs an interleaved memory system. A vector processor has many memory ports corresponding to the interleaved memory system, and continuous data transfer is enabled. Thus, the vector cache consists of sub-caches with each memory port, and the sub-cache prevents the diminishing of the effect of the interleaved memory system. Furthermore, the vector cache employs a bypass mechanism between the main memory and vector register files. The bypass mechanism makes possible to supply data from both the main memory and the vector cache at the same time. Thus, the total amount of data provided to the vector register files in time is increased by the bypass mechanism.

A vector load/store instruction concurrently deals with 256 floating-point data in the vector architecture, then the vector cache needs to process the 256 data in continuity. The vector cache employs a non-blocking cache [15]. Moreover, the non-blocking cache employs MSHR. In a difference scheme of scientific simulations, vector load instructions often load the same data continuously. When subsequent load instructions are issued at a short time, however, the same data are not yet filled into the vector cache owing to a latency of the main memory. Thus, the subsequent load instructions cause cache misses. The MSHR makes possible for the subsequent load instructions to reuse in-flight load data. Therefore, the MSHR reduces redundant accesses to the main memory. In addition, the latency of the subsequent load requests is shortened.

A prefetch mechanism uses software-directed techniques, then a prefetch directive of a compiler and a prefetch instruction are equipped. Since only a prefetch instruction is added to the SX instruction set, the prefetch mechanism does not need to greatly modify the hardware, and the issue timing of the prefetch instruction is freely changed. On the other hand, the prefetch mechanism has two effects on the performance. One is that the mechanism hides the long memory latency by pipelined vector operations. The other is the same effect of the MSHR: the prefetch mechanism reduces redundant load requests between the vector cache and the main memory when multiple load instructions access the same data.

As the on-chip cache size is limited, selective caching mechanism is an effective approach to efficient use of the vector cache. The selective caching mechanism is software-controlled: a selective directive of a complier, and vector load-/store instructions install a flag for cache control; *cache on/off*. The data with *cache off* are supplied through the bypass between the main memory and vector register files.

3.4.2 Proposed vector architecture

Figure 3.1 shows a diagram of a vector processor architecture with a proposed on-chip vector cache. The vector processor has a vector unit, a scalar unit and an address control unit. The vector unit contains vector registers, vector arithmetic pipes and a vector cache. The scalar unit controls all the functions of the vector processor including the vector unit control in addition to the execution of scalar instructions for arithmetic operations. The address control unit queues and issues vector operation instructions to the vector unit. The vector operation instructions contain vector memory instructions and vector computational instructions. Each instruction concurrently deals with 256 double-precision floating-point data.

The main memory unit employs an interleaved memory system. The maximum B/FLOP rate between the main memory and the vector processor is 4 B/FLOP. The main memory is divided into 32 parts, each of which operates independently and supplies data to the vector processor. Thus, the main memory and the vector processor are interconnected through 32 memory ports.

Figure 3.2 shows an instruction format of vector load/store. This instruction has 32 bits. The field of *op* indicates operation codes. *Cx*, *Cy* and *Cz* specify the control function of a mask operation. *Vc* indicates the flag for cache control; *cache on/off*. *Rx* shows a vector arithmetic register, *Ry* indicates a stride of data and *Rz* is memory address of data. Here, on the prefetch instruction *Vc* is constantly *on*, and *Rx* is not used.

The vector cache is implemented in each memory port of the vector processor, thus the vector cache consists of 32 sub-caches. The sub-cache employs a

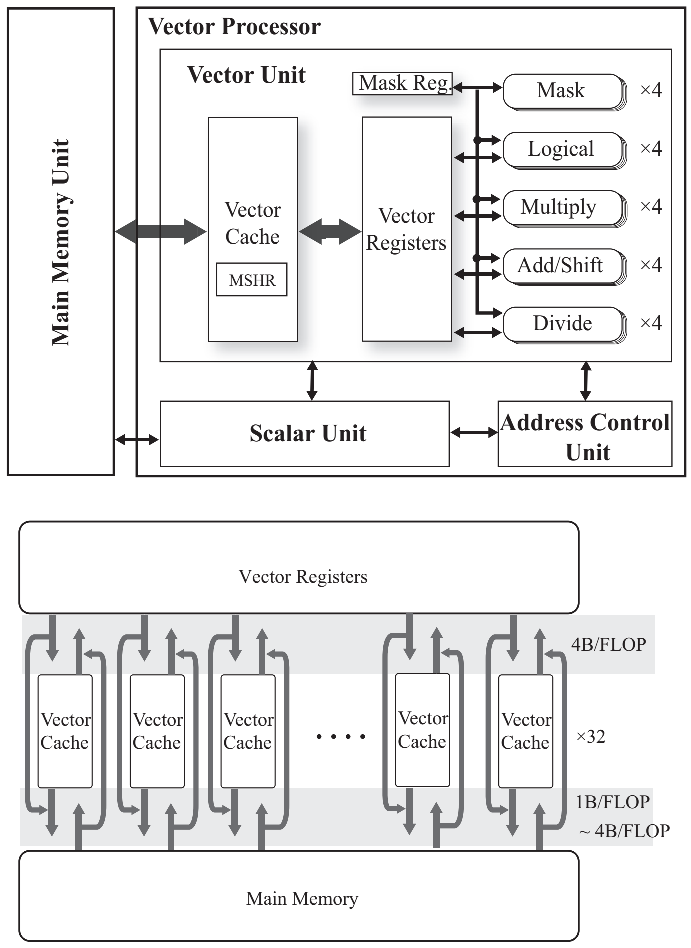


Figure 3.1: Vector architecture with vector cache and memory system block diagram.

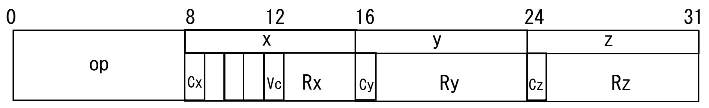


Figure 3.2: Instruction format of vector load/store.

set-associative write-through cache with the LRU replacement policy. The line size is 8 bytes; it is the unit for memory accesses. The vector cache reduces the memory access latency and the bank cycle. The memory bandwidth between the cache and vector registers is 4 B/FLOP. Moreover, the sub-cache employs a bypass mechanism between the vector register and the main memory. The bypass mechanism is controlled by the flag *Vc* of the vector load/store instructions. When the flag indicates *cache on*, the supplied data by the vector load/store instruction are cached. Meanwhile, when the flag is *cache off*, the vector load/store instruction provides data via the bypass mechanism: the data are not cached.

The vector cache employs a non-blocking cache with a MSHR. Each sub-cache has the MSHR which holds information of in-flight load requests and subsequent requests: instruction address, vector arithmetic register address and memory address of load data. When the memory address of a subsequent load request is equal to the memory address of an in-flight load data, the subsequent load request is not sent to the main memory. Then, the subsequent load requests are immediately written to the register files as the in-flight load data arrive at the MSHR.

A prefetch mechanism uses software-directed techniques. The prefetch instruction specifies memory addresses of prefetched data, and the prefetched data are transferred from the main memory to the vector cache independently of numerical pipelined operations. The prefetch instructions are issued by the address control unit.

The vector registers consist of two sets, called *vector arithmetic register* and *vector data register*. The vector arithmetic register consists of 8 sets of registers which hold a maximum of 256 double-precision floating-point data. Those registers are mainly used for the vector operations. The vector data register consists of 32 sets of registers which also hold a maximum of 256 double-precision floating-point data. The vector data register is used as buffer memory for storing data of the vector operations. The vector cache is directly connected to the vector arithmetic register and the vector data register.

The vector arithmetic pipes consist of five types of vector pipelines: Mask, Logical, Multiply, Add/Shift and Divide. Each pipeline is 4-way multiple pipelines; total 20 pipes are included in the vector unit. Vector data are input from the vector arithmetic registers in the vector arithmetic pipes, and each vector pipeline works independently or chains to operate simultaneously, then one or more results are output every clock cycle into the vector arithmetic registers or the vector data registers.

3.5 Experimental Environment

3.5.1 Methodology

A research uses a trace-driven simulator that simulates the behavior of the proposed vector architecture at the register transfer. The simulator is enhancement of the NEC SX simulator, which accurately models a single processor of the SX architecture; the vector unit, the scalar unit and the memory system. The simulator takes a system parameter file and a trace file as input, and the output of the simulator contains instruction cycle counts of a benchmark program and cache hit information. The system parameter file has configuration parameters of a vector architecture and setting parameters, i.e., the cache size, the associativity, the cache latency and the memory bandwidth.

The trace file contains an instruction sequence of a benchmark program and directives of cache control: *cache on/off*. These directives are used in selective caching and set the cache control flags in the vector load/store instructions. A benchmark program is compiled by the NEC FORTRAN compiler: FORTRAN90/SX. It supports ANSI/ISO Fortran95 in addition to functions of automatic vectorization and automatic parallelization. The executable program runs on the SX trace generator to produce the trace file. In this work, two kernel loops and the original source codes of the five benchmarks are used, and are compiled with the highest optimizations option (-C hopt [50]) and inlining subroutines.

To evaluate on-chip vector caching for future vector processors with a higher flop/s rate but a relatively lower off-chip memory bandwidth, the effects of the vector cache are evaluated on the vector processor by limiting its memory bandwidth per flop/s rate from 4 B/FLOP down to 1 B/FLOP. Here, the 4 B/FLOP case is equal to the B/FLOP rate of the SX-6, SX-7 and SX-8 systems, the 2 B/FLOP is the same as Cary X1 and BlackWidow, and the 1 B/FLOP is the same level as the

Table 3.1: Summary of setting parameters.

Base System Architecture	NEC SX-7
Main Memory	DDR-SDRAM
Memory Size	256 GB
Number of bank	16,384
Vector Cache	SRAM
Cache Size (Sub-cache Size)	256KB - 8MB (8KB - 256KB)
Cache Policy	LRU, Write-through
Associativity	2WAY, 4WAY, 8WAY
Cache Latency	15 %, 50 %, 85 %, 100 % (Rate of main memory latency)
Cache Bank Cycle	5 % of memory cycle
Line Size	8B
MSHR Entries (Sub-cache)	8192 (256)
Memory - Cache bandwidth per flop/s	1 B/FLOP, 2 B/FLOP, 4 B/FLOP
Cache - Register bandwidth per flop/s	4 B/FLOP

commodity-based scalar systems such as NEC TX7 series and SGI Altix series. The setting parameters are shown in Table 3.1.

3.5.2 Benchmark programs

To clarify the basic characteristics and validity of the vector cache, two basic kernel loops and the five benchmark programs shown in Chapter 2 are selected again.

The following kernel loops are used to evaluate the performance of the vector processor with the vector cache. Since these loops has no temporal locality, a part of $X(i)$, $Y(i)$ and $Z(i)$ data are stored on the vector cache in advance of the loop execution.

Source Code 3.2: Kernel Loop (1).

```fortran
DO  i = 1 , 25600
  A(i) = X(i) + Y(i)
END DO
```

Source Code 3.3: Kernel Loop (2).

```fortran
DO  i = 1 , 25600
  A(i) = X(i) * Y(i) + Z(i)
END DO
```

Before detailed discussions on the effects of the vector cache, the characteristics of representative routines on the benchmark programs are presented. The following routines are discussed in Sections 3.6, 3.7 and 3.8.

Source Code 3.4: A kernel loop of the PRF simulation.

```fortran
 1  do J = 1,NJ
 2    do I = 4,NI-3
 3      wDX1XC1(I,J,L) = 1.D0/3.D0
 4      wDX1XC2(I,J,L) = 1.D0
 5      wDX1XC3(I,J,L) = 1.D0/3.D0
 6      wPHIX12(I,J) = (DX1AA(I)*wPHIX11(I-2,J)
 7  &     +DX1BB(I)*wPHIX11(I-1,J)+DX1DD(I)*wPHIX11(I+1,J)
 8  &     +DX1EE(I)*wPHIX11(I+2,J)) *DELX_INV
 9    end do
10  end do
```

Source Code 3.4 shows a difference scheme loop of the PRF simulation. As the arrays of $DX1AA(I)$, $DX1BB(I)$, $DX1DD(I)$ and $DX1EE(I)$ (line 06, 07 and

08 in Source Code 3.4) are defined in the preceding loop and each size is 4 KB only. Thus, these arrays are always on the vector register files, and these arrays do not need to access the cache and the main memory in the loop. The array $wPHIX11$ (line 06, 07 and 08) has a high locality regarding the index i. On the SX architecture, a vector load instruction transfers 256 floating point data from the main memory to the register files at once. Thus, a part of $wPHIX11(i-2,j)$ data is reused by the subsequent load of $wPHIX11(i-1,j)$, $wPHIX11(i+1,j)$ and $wPHIX11(i+2,j)$. However, the data is hardly reused and the cache hit rate is 5 % in this loop, because the data is not filled into the cache yet when the subsequent load instructions are issued. Hence, this problem is solved by the MSHR and the prefetch mechanism in the vector cache.

Source Code 3.5: A kernel loop of the PRF simulation.

```
DO KK = 2,NJ
  DO I = 1,NI
    wDY1YC3(I,KK-1,L) = wDY1YC3(I,KK-1,L) * wDY1YC2(I,KK-1,L)
    wDY1YC2(I,KK,L)   = wDY1YC2(I,KK,L) - wDY1YC1(I,KK,L)
&                   * wDY1YC3(I,KK-1,L)
    wDY1YC2(I,KK,L)   = 1.D0 / wDY1YC2(I,KK,L)
    wDY1YC(I,KK,L)    = (wPHIY12(I,KK) - wDY1YC1(I,KK,L)
&                   * wDY1YC1(I,KK-1,L)) * wDY1YC2(I,KK,L)
  END DO
END DO
```

Moreover, Source Code 3.5 is one of kernel loops of the PRF simulation. The size of each array in the PRF simulation is 18 MB, and many loops are doubly nested loop. Thus, the arrays are treated as two dimensions array, and the size of cached data per array is 2 MB only. $wDY1YC1$, $wDY1YC2$ and $wDY1YC3$

in Source Code 3.5 are defined in the preceding loop. In addition, $wDY1YC1$ and $wDY1YC2$ have the spatial locality, and these arrays reuse the cached data. Here, $wDY1YC3(I, KK-1, L)$ (line 05 in Source Code 3.5), $wDY1YC2(I, KK, L)$ (line 06), $wDY1YC1(I, KK, L)$ (line 07) and $wDY1YC2(I, KK, L)$ (line 08) reuse data on the register files. Thus, these arrays do not access the cache. This loop is discussed for selective caching in Section 3.8.

Source Code 3.6 shows one of kernel loops of the SFHT simulation. The array of Phi has a locality, and $Phi(i, j-1, k)$ (line 04 in Source Code 3.6), $Phi(i-1, j, k)$ (line 05) and $Phi(i, j, k-1)$ (line 06) reuse the cached data. The other arrays, AN, AS, AE, AW, AT, AB, AP, RGN and DIV do not have a locality. In Section 3.7, however, these arrays are prefetched. Then, the prefetch mechanism is an effective way for hiding the memory access time of these arrays by other arithmetic operations processed.

Source Code 3.6: A kernel loop of the SFHT simulation.

```
 1    do 110 k=KST(l),NK,2
 2      do 110 j=JST(l),NJ,2
 3        do 110 i=IST(l),NI,2
 4          res=(AN(i,j,k)*Phi(i,j+1,k)+AS(i,j,k)*Phi(i,j-1,k)
 5    &       +AE(i,j,k)*Phi(i+1,j,k)+AW(i,j,k)*Phi(i-1,j,k)
 6    &       +AT(i,j,k)*Phi(i,j,k+1)+AB(i,j,k)*Phi(i,j,k-1)
 7    &       +AP(i,j,k)*Phi(i,j,k)-DIV(i,j,k))/AP(i,j,k)
 8          res=res1*RGNc(i,j,k)
 9          Phi(i,j,k)=Phi(i,j,k)-res1
10  110 continue
```

The PBM simulation has two versions: a non outer unrolling version and an outer unrolling version. Source Code 3.7 shows the main routine of the non outer unrolling version. The inner loop, index j, is vectorized and arrays gd_dip and $wary$ are stored in the vector cache by vector load instructions. However, gd_dip is spilled from the cache, because gd_dip needs 7.6 GB. On the other hand, $wary$ is held in the cache, because its needs only 250 KB and the array is defined in the preceding loop.

Source Code 3.7: A kernel loop of the PBM simulation.

```
do i=1,ncells
  do j=1,ncells
    wf_dip(i)=wf_dip(i)+gd_dip(j,i)*wary(j)
  end do
end do
```

Source Code 3.8: Outer-unrolling loop of the PBM simulation.

```
1 do i=1,ncells,4
2   do j=1,ncells
3     wf_dip(i)=wf_dip(i)+gd_dip(j,i)*wary(j)
4     wf_dip(i+1)=wf_dip(i+1)+gd_dip(j,i+1)*wary(j)
5     wf_dip(i+2)=wf_dip(i+2)+gd_dip(j,i+2)*wary(j)
6     wf_dip(i+3)=wf_dip(i+3)+gd_dip(j,i+3)*wary(j)
7   end do
8 end do
```

Moreover, Source Code 3.8 shows an outer-unrolled 4 times in the PBM simulation. In the outer-unrolling case, array $wary$ of the outer loop index i (line 03 in Source Code 3.8) reuses the data of the index $i-1$ on the cache, and arrays $wary$ (line 04, 05 and 06) reuse the data on register files of $wary$ (line 03), hence, memory references are reduced. However, the cache capacity for gd_dip increases owing to an increase in the number of arrays referenced in the innermost loop. This problem is discussed in Section 3.8.

3.6 Performance Evaluation of Vector Cache

The performance of the proposed vector processor architecture is evaluated by using the benchmark programs, and the basic characteristics of the vector cache are clarified in this section. Here, the vector cache does not employ the MSHR and the prefetch mechanism.

3.6.1 Relationship between efficiency and cache hit rate on Kernel loops

In this subsection, the execution of the kernel loop (1) is simulated on the vector processor with the vector cache. Since this loop has no temporal locality, a part of $X(i)$ and $Y(i)$ data is stored on the vector cache in advance of the loop execution. In the following two ways, the data for caching are selected to change the range of cache hit rates, and their effects on performance are examined. One is to store both $X(i)$ and $Y(i)$ with the same index i on the cache, and load instructions of both $X(i)$ and $Y(i)$ are set to *cache on*, named Case 1. The other is to cache only $X(i)$, and load instructions of $X(i)$ and $Y(i)$ are set to *cache on* and *cache off*, respectively. This is Case 2. Here, the cache associativity is 2 way, and the cache latency is 15 % of the main memory latency in the setting parameters.

Figure 3.3 shows the relationship between the cache hit rate and the relative memory bandwidth. The relative memory bandwidth is obtained by normalizing the effective memory bandwidth of the systems with the vector cache by the effective memory bandwidth of the 4 B/FLOP system without the cache. Figure 3.4 shows the relationship between the cache hit rate and the computation efficiency. Figure 3.3 indicates that the relative memory bandwidth of each case increases as the cache hit rate increases. Therefore, the vector cache is one of the promising solutions to cover a lack of the memory bandwidth. Similar to Figure 3.3, the

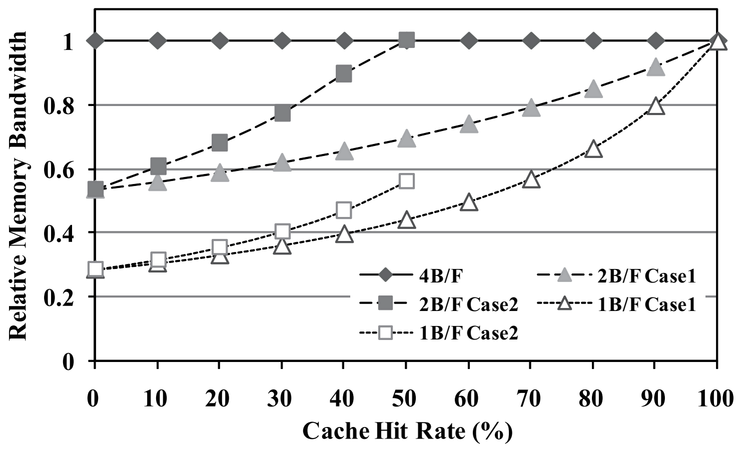


Figure 3.3: Relationship between cache hit rate and relative memory bandwidth on Kernel loop (1).

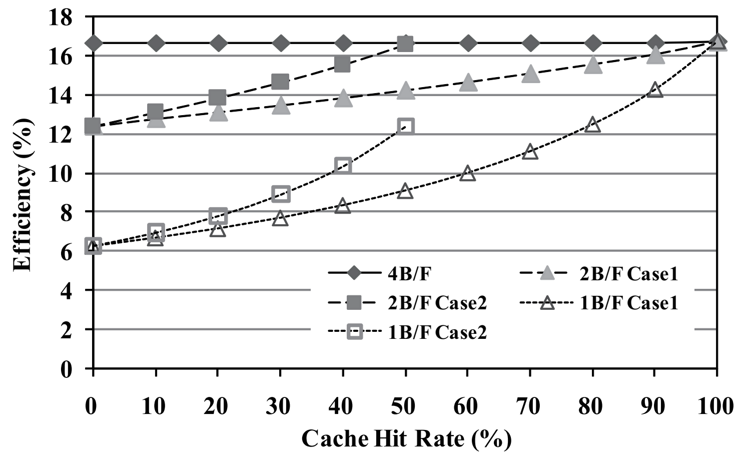


Figure 3.4: Relationship between cache hit rate and computation efficiency on Kernel loop (1).

computation efficiency of each case in Figure 3.4 improves as the cache hit rate increases.

Figures 3.3 and 3.4 show the correlation between the relative memory bandwidth and the computation efficiency. On both the 2 B/FLOP and 1 B/FLOP systems, the performance of Case 2 is greater than the performance of Case 1. In particular, Case 2 with the vector cache of the 50 % hit rate on the 2 B/FLOP system achieves the same performance of the 4 B/FLOP system. On the 4 B/FLOP system, each of $X(i)$ and $Y(i)$ is provided at a 2 B/FLOP rate on average. When the cache hit rate is 50 % in Case 2, all data of $X(i)$ are supplied directly from the vector cache at the 2 B/FLOP rate and all data of $Y(i)$ are supplied from the memory at the 2 B/FLOP rate through the bypass mechanism. Consequently, each of $X(i)$ and $Y(i)$ is provided at the 2 B/FLOP rate on the vector registers, and the total amount of data provided to vector registers in time is equal to the total amount of data in the 4 B/FLOP system. However, the 1 B/FLOP system with vector caching with the 50 % hit rate does not achieve the performance of the 4 B/FLOP system. Because $Y(i)$ is provided at a 1 B/FLOP rate, the total B/FLOP rate at the vector registers does not reach the 4 B/FLOP rate. On the other hand, in Case 1, the data of $X(i)$ and $Y(i)$ are supplied from either of the vector cache or the memory at the same index i. While supplying the data from the memory, $X(i)$ and $Y(i)$ have to share the memory bandwidth rate. Therefore, the 1 B/FLOP and 2 B/FLOP systems need a cache hit rate of 100 % to achieve a performance of the 4 B/FLOP system.

Similarly, the performance of Kernel (2) is examined on the 2 B/FLOP system in the following three cases. In Case 1, all of $X(i)$, $Y(i)$, and $Z(i)$ are cached with the same index i in advance. In Case 2, both $X(i)$ and $Y(i)$ are provided from the vector cache, and therefore the maximum cache hit rate is 66 %. In Case 3, only $X(i)$ is in the cache, and hence the maximum cache hit rate is 33%. Figure 3.5 shows the relationship between the cache hit rate and the change in the relative memory bandwidth regarding Kernel (2). On the 4 B/FLOP system, each of $X(i)$, $Y(i)$ and $Z(i)$ is provided at a 4/3 B/FLOP rate on average. The performance of Case 2 is comparable to the performance of the 4 B/FLOP system when the cache hit rate is 66 %, because the B/FLOP rate of each data is the 4/3 B/FLOP rate on average. However, in Case 3, both $Y(i)$ and $Z(i)$ are provided from the memory, and $Y(i)$ and $Z(i)$ have to share the 2 B/FLOP rate at the vector register. As a result, the relative memory bandwidth never reaches the memory bandwidth of

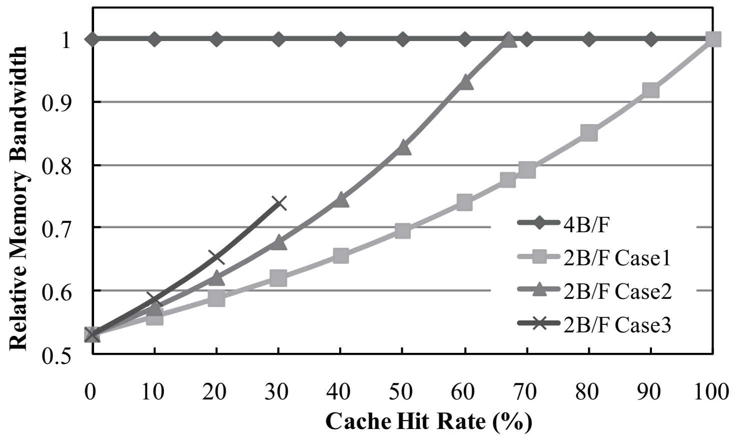


Figure 3.5: Relationship between cache hit rate and relative memory bandwidth on Kernel loop (2).

the 4 B/FLOP system.

These results indicate that the vector cache has a potential for the future vector supercomputers to cover the shortage of memory bandwidth. In additions, all data do not need to be cached from the discussions on Figures 3.3 and 3.5. The key data determining the performance needs to be cached to make good use of limited on-chip capacity.

3.6.2 Relationship between efficiency and cache hit rate on the five benchmark programs

The execution of the five benchmark programs is simulated with the vector cache varying memory bandwidth per flop/s rates; 1 B/FLOP and 2 B/FLOP. Here, all vector load/store instructions are set to *cache on*, the cache associativity is 2 way, and the cache latency is 15 % of the main memory latency in the setting parameters.

Figure 3.6 shows the relationship between the cache hit rate of the 8 MB vector cache and the relative memory bandwidth. Figure 3.6 indicates that the vector cache improves the effective memory bandwidth, depending on the cache hit rate of the benchmarks. Cache hit rates vary from 13 % in the SFHT simulation to 96 % in the APFA simulation, because these depend on the locality of reference in individual applications.

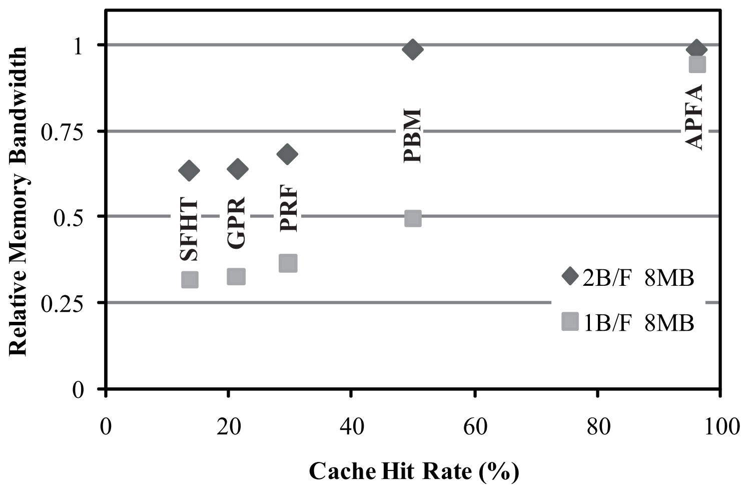


Figure 3.6: Relationship between cache hit rate and relative memory bandwidth on the five benchmark programs.

In the APFA simulation, the relative memory bandwidth is 0.96 at the 2 B/FLOP system with 96 % hit of the 8 MB cache, resulting in an effective memory bandwidth almost equal to the 4 B/FLOP system. In this case, most data are provided from the cache. The PBM simulation of the 2 B/FLOP system reaches the effective memory bandwidth of the 4 B/FLOP system at the cache hit rate 50 %. In this case, all data of *wary* in Source Code 3.7 are supplied directly from the vector cache at the 2 B/FLOP rate and all data of *gd_dip* are supplied from the main memory at the 2 B/FLOP rate. Therefore, the total amount of data provided to vector registers in time is equal to the total amount of data in the 4 B/FLOP system. The vector cache with a bypass mechanism provides the data from both the memory and the cache at once, and the sustained memory bandwidth for the registers increase.

The relative memory bandwidths in others benchmark programs, the SFHT, GPR and PRF simulations, are over 0.6 on the 2 B/FLOP system and 0.3 on the 1 B/FLOP system. Here, the relative memory bandwidth without the cache is 0.5 on the 2 B/FLOP system and 0.25 on the 1 B/FLOP system. Thus, the improvement of the relative memory bandwidth is over 20 % using the 8 MB cache.

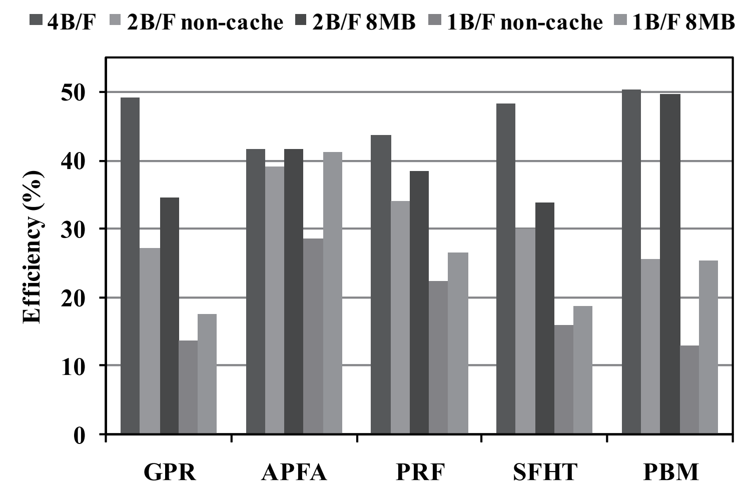


Figure 3.7: Computation efficiency of the five benchmark programs with/without 8 MB vector cache.

Figure 3.7 shows the computation efficiency of the five benchmark programs on the 4 B/FLOP system without the cache, the 2 B/FLOP and 1 B/FLOP systems with/without the 8 MB cache. Figure 3.7 indicates that the efficiency of the 2 B/FLOP and 1 B/FLOP systems is increased by the cache. In particular, the 2 B/FLOP system with the cache achieves the same efficiency as the 4 B/FLOP system in the APFA and PBM simulations. Figure 3.8 shows the relationship between the cache hit rate of the 8 MB cache and the recovery rate of the performance by the cache from the 2 B/FLOP and 1 B/FLOP systems to the 4 B/FLOP system. The recovery rate goes up as the cache hit rate increases. The vector cache increases the recovery rate by 21 % to 99 % on the 2 B/FLOP system and 9 % to 96 % on the 1 B/FLOP system, depending on the data access locality of each benchmark program. The results indicate that the vector cache has a potential for the future vector supercomputers to cover the shortage of memory bandwidth on real scientific applications.

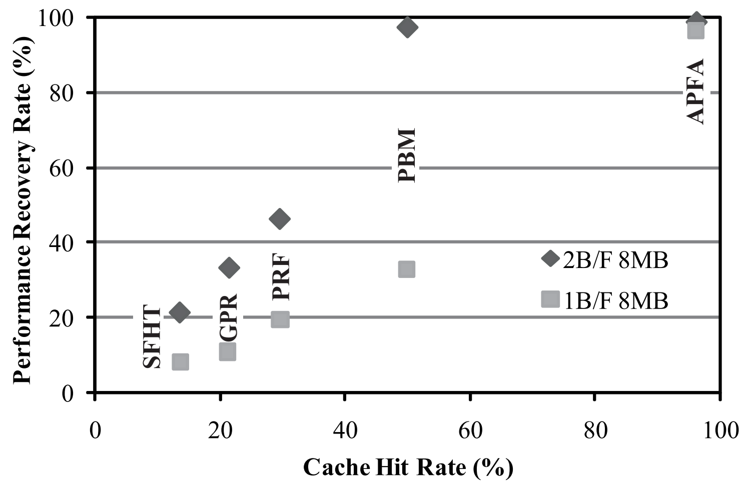


Figure 3.8: Recovery rate of performance on 8 MB vector cache.

3.6.3 Relationship between associativity and cache hit rate

In this subsection, the effect of the associativity is examined ranging from 2 to 8 ways on the cache hit rate. Here, the cache latency is 15 % of the main memory latency in the setting parameters.

Figure 3.9 shows that the cache hit rates on each set associative cache, covering cache sizes from 256 KB to 8 MB. Four benchmark programs, the APFA, PRF, SFHT and PBM simulations, approximately have constant cache hit rates across three associativity cases. The cache hit rates vary with the associativity in the GPR simulation.

The cache hit rate is dependent on among placements of the arrays on memory, memory access patterns of the programs and the associativity. In the GPR simulation, its basic computational structure consists of triple nested loops. The loops access many arrays at intervals of 584 bytes stride addresses, and the cached

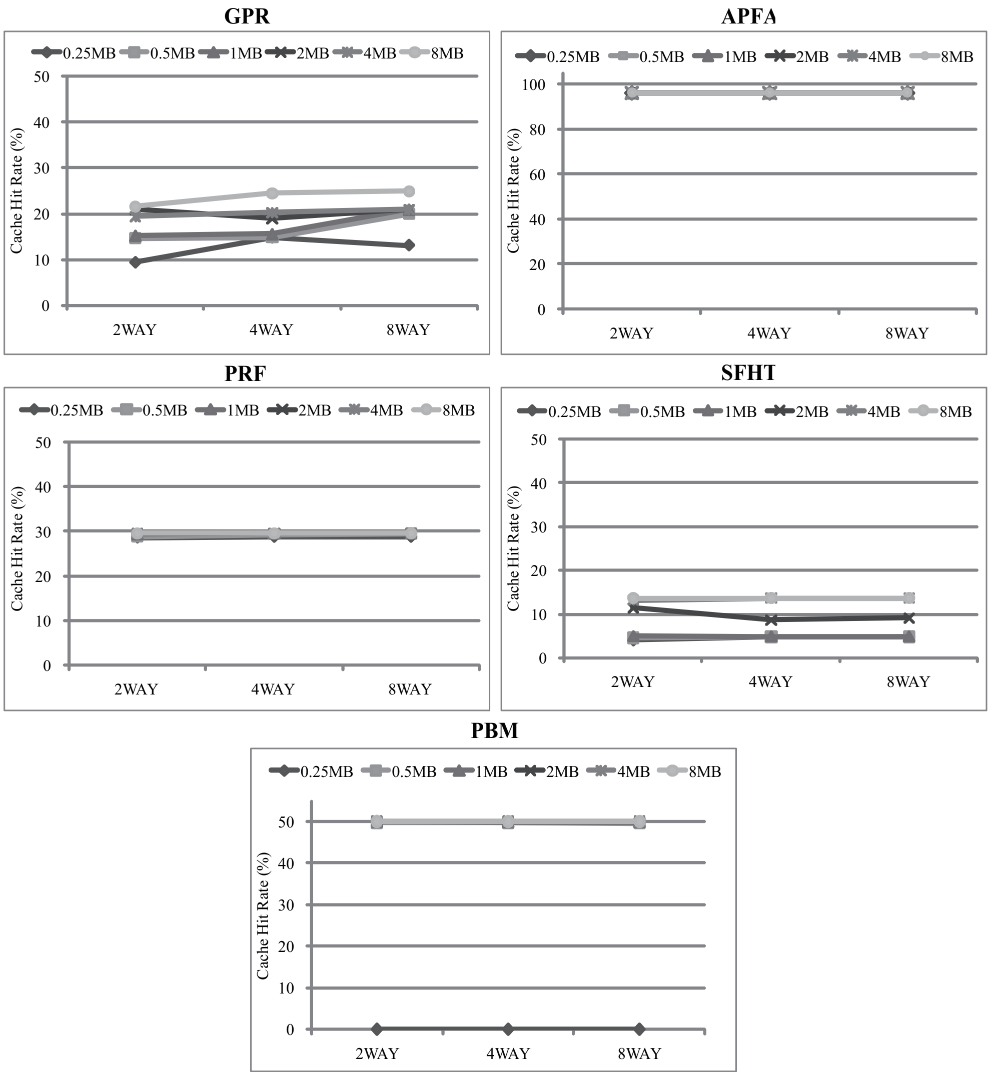


Figure 3.9: Cache hit rate vs. associativity on the five benchmark programs.

data are frequently replaced. Therefore, the cache hit rate varies with the cache capacities and the associativity. In the SFHT simulation, the memory access patterns have 8 bytes stride accesses. On the 2 MB case, the cached data which will be reused on the 2-way set associative cache are evicted by some other data. Thus this is owing to a placement of memory addresses of the arrays. Other programs have constant cache hit rates across three associativity cases. These memory access patterns are 8 bytes or 16 bytes stride accesses, and the placement of the arrays on memory does not cause any conflict of reused data.

These results provide that the associativity has effects on the cache hit rates when a loop accesses many arrays at intervals of large stride addresses as shown in the GPR simulation. Therefore, the vector cache requires 8-way associativity cache.

3.6.4 Effects of vector cache latency on performance

In general, the latency of the on-chip cache is considerably shorter than the latency of the off-chip memory. The effects of the cache access latency are investigated on the performance of the five benchmark programs.

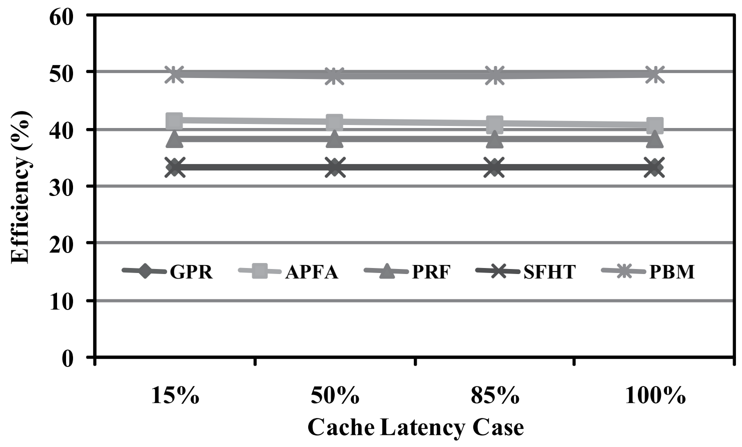
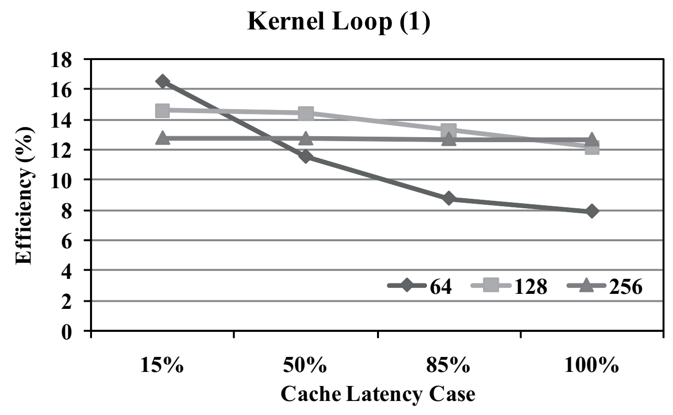


Figure 3.10: Computation efficiency of the five benchmark programs on four cache latency cases.

Figure 3.10 shows the relationship between the computation efficiency of the benchmark programs and four cache latencies; 15 %, 50 %, 85 % and 100 % of the memory access latency on the 2 B/FLOP system with the 2 MB cache. The five benchmark programs have the same efficiency when changing the cache latency, because the vector architecture hides the memory access times by pipelined vector operations when a vector operation ratio and a vector loop length of benchmark programs are enough large. In addition, the vector cache consists of 32 sub-caches between vector register files and the main memory via each memory



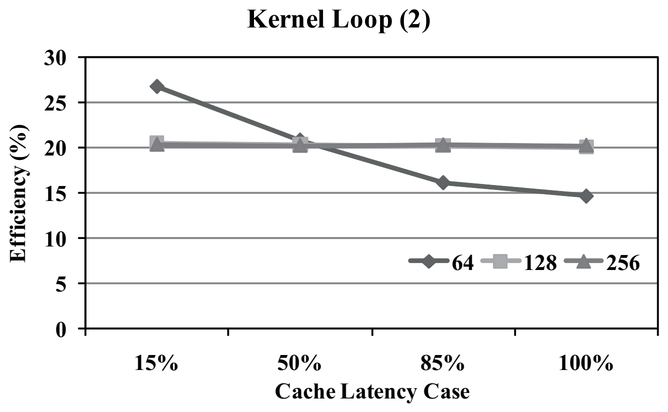


Figure 3.11: Computation efficiency of two kernel loops on four cache latency cases.

port. The sub-caches connected to 32 memory ports provide multiple words at a time. The cache latency of the second memory access or later is hidden in the sub-cache system.

For comparison, the performance of shorter loop length cases is examined using Kernel loops (1) and (2) on the 2 B/FLOP system when changing the cache access latency. Figure 3.11 shows that the relationship between the computation efficiency and four cache latency cases in Kernel loops. Here, three loop length cases are examined in the 100 % cache hit rate; $1 \leq i \leq 64$, $1 \leq i \leq 128$, $1 \leq i \leq 256$. On the shorter loop length cases, Figure 3.11 indicates that the vector cache latency is an importance factor in the performance, when pipelined vector operations do not hide the memory access time. Especially, on the loop

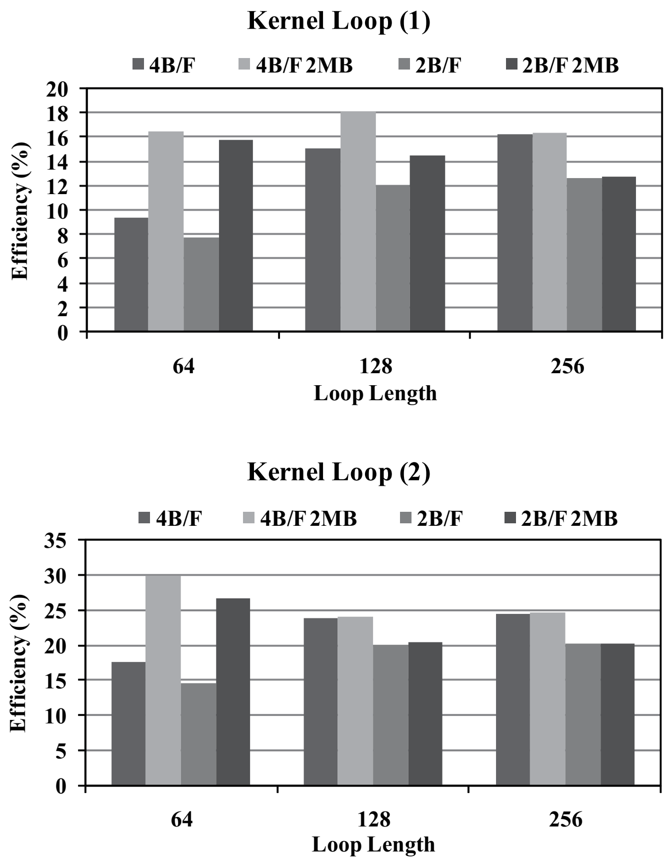


Figure 3.12: Computation efficiency of two kernel loops on 4 B/FLOP and 2 B/FLOP on three loop lengths.

length 64 case, the 15 % latency case has two times higher efficiency than the 100 % case in Kernel (1), and the 15 % latency case has 12 % higher efficiency in Kernel (2). However, the longer the loop length, the lower the effect of the cache latency.

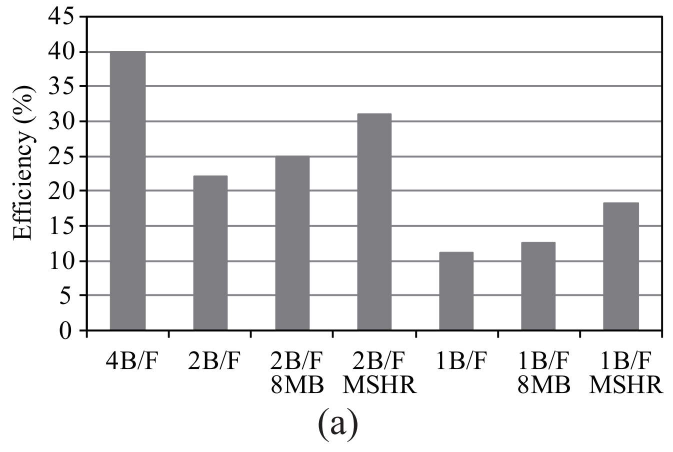
Figure 3.12 shows that the computation efficiency of Kernel loops in the 4 B/FLOP and 2 B/FLOP systems with/without the vector cache. On both the loop lengths 64 and 128 of Kernel (1), the efficiency of the 4 B/FLOP system with the cache is higher than the efficiency of the 4 B/FLOP system without the cache, and the loop length 128 case of the 4 B/FLOP system with the cache achieves the highest efficiency. Meanwhile, in Kernel (2), the loop length 64 case of the 4 B/FLOP system with the cache has the highest efficiency. In these kernel loops the memory access time of the shorter loop length case is not entirely hidden by pipelined vector operations only, then the ratio of memory access time to the processing time becomes the smallest owing to the decrease in memory latency by the cache. Therefore, the performance of shorter loop length cases is sensitive to the vector cache latency, and the vector cache boosts the performance of the benchmark programs with short vector lengths on the 4 B/FLOP system.

3.7 Performance Evaluation of MSHR and Prefetching Vector Cache

This section shows the effects of the MSHR and prefetching on the performance of three benchmark programs: the PRF, GPR, and SFHT simulations.

3.7.1 Effect of MSHR on the vector cache

The MSHR has the potential for increasing the performance in difference schemes of scientific applications. Figure 3.13 shows (a) the computation efficiency and (b) the data transfer rate using the MSHR on the PRF simulation for the difference scheme loop. The sustained performance with the MSHR is 1.25$\times$ and 1.45$\times$ higher than the sustained performance without the MSHR on the 2 B/FLOP and 1 B/FLOP systems, respectively. The computation efficiency is 31 % on the 2 B/FLOP system, and 18 % on the 1 B/FLOP system. The MSHR does not supply redundant data from the main memory, and hence the data transfer rates are improved as shown in Figure 3.13 (b). The data transfer rates on the 2 B/FLOP and 1 B/FLOP systems reach 78 % and 45 % of the data transfer rate of the 4 B/FLOP system, respectively. Figure 3.14 shows the computation efficiency increased by the MSHR for difference scheme loops of the GPR and SFHT simulations. On the GPR simulation, the sustained performance with the MSHR is 1.11 times higher than the sustained performance without the MSHR. In the SFHT simulation, the sustained performance is improved by the MSHR by 4 to 5 %. Here, Table 3.2 shows the MSHR hit rate, a fraction of memory accesses solved by the MSHR, on the three benchmark programs. It is showed that the MSHR is an effective way to handle the loops of the difference schemes.



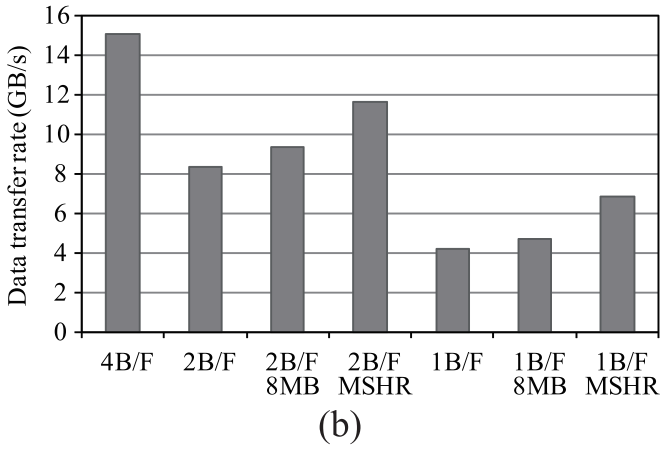


Figure 3.13: (a) Computation efficiency and (b) Data transfer rate using MSHR on the PRF simulation.

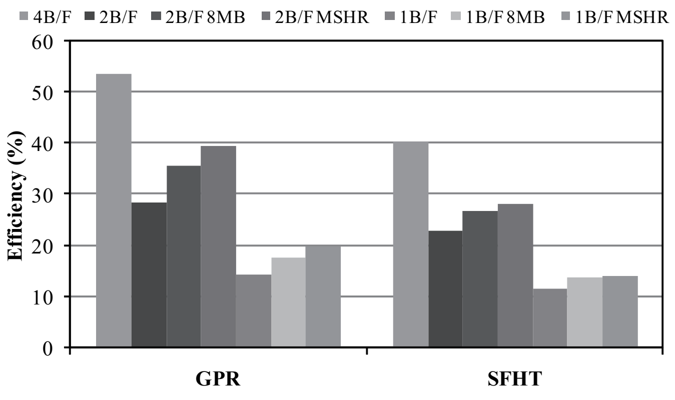


Figure 3.14: Computation efficiency using MSHR on the GPR and SFHT simulations.

Table 3.2: MSHR hit rates of three benchmark programs.

Names	GPR	PRF	SFHT
Hit Rate (%)	11	47	4

3.7.2 Effect of prefetching on the vector cache

The prefetch mechanism is provided to increase the effect of the vector cache on the 2 B/FLOP and 1 B/FLOP systems. Several memory instructions are executed by the prefetch mechanism to obtain the effects of the following two types. TYPE I is that the prefetch mechanism reduces the memory access time by issuing the load instructions enough before they are needed, because the memory access time is hidden by other arithmetic operations processed simultaneously. TYPE II is the same effect of the MSHR: the prefetch mechanism reduces the number of load requests to the main memory by removing redundant load instructions accessing the same memory address.

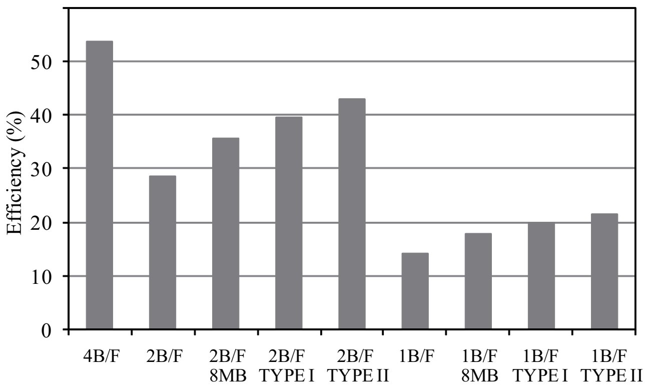
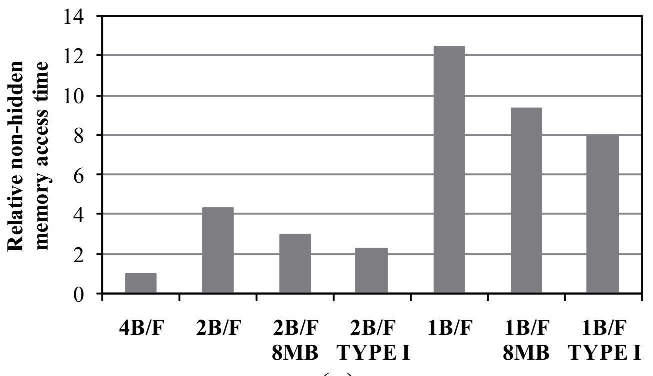
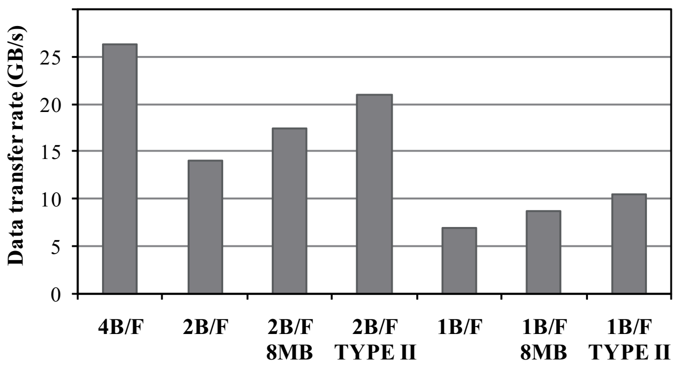


Figure 3.15: Computation efficiency on 2 B/FLOP and 1 B/FLOP on the GPR simulation.

In the GPR simulation (Source Code 3.1), the prefetching arrays of TYPE I: $E_x(i, j, k)$ (line 04 in Source Code 3.1), $E_z(i, j, k)$ (line 07) and $E_y(i, j, k)$ (line 10) and of TYPE II: $H_z(i, j, k)$ (line 05) and $H_x(i, j, k)$ (line 09) are selected. These



(a)



(b)

Figure 3.16: (a) Relative non-hidden memory access time and (b) Data transfer rate on 2 B/FLOP and 1 B/FLOP on the GPR simulation.

array data are prefetched at one iteration ahead of outer index i. Figure 3.15 shows the computation efficiency in this loop using the prefetch mechanism on the 2 B/FLOP and 1 B/FLOP systems. In TYPE II, the computation efficiencies are 43 % on the 2 B/FLOP system, and 22 % on the 1 B/FLOP system. The prefetching mechanism improves the performance by 20 %. Regarding TYPE I, a 10 % improvement in performance is obtained by prefetching. Figure 3.16 shows (a) the relative non-hidden memory access time on TYPE I and (b) data transfer rate on TYPE II in this loop. Here, the relative non-hidden memory access time is obtained by normalizing the non-hidden memory access time of each system by the non-hidden memory access time of the 4 B/FLOP system. The prefetch of TYPE I reduces the non-hidden memory access time by 20 % on both the 2

B/FLOP and 1 B/FLOP systems. On the other hand, as shown in Figure 3.16 (b), the effect of TYPE II is equivalent to the effect of the MSHR, and TYPE II increases the data transfer rate of the 2 B/FLOP and 1 B/FLOP systems to 7.3 GB/s and 3.6 GB/s, respectively. Therefore, the prefetch mechanism hides the memory access time by other arithmetic operations, and increase the data transfer rate on the vector supercomputer.

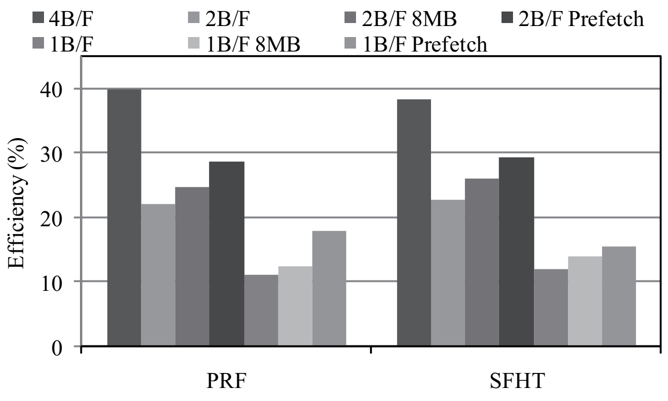
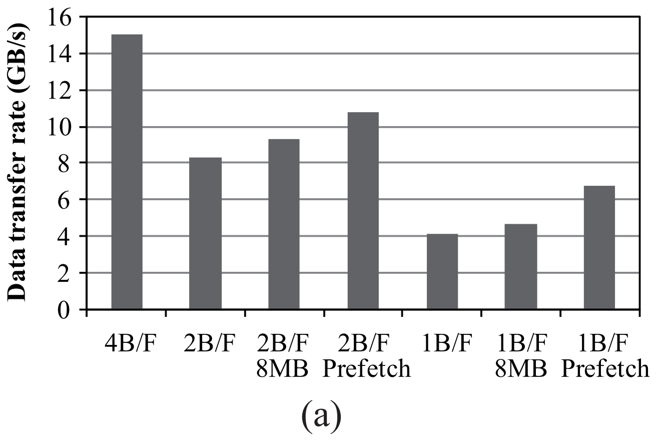


Figure 3.17: Computation efficiency on 2 B/FLOP on the SFHT and PRF simulations.

Similarly, the effect of the prefetch mechanism is simulated for the loop of the PRF simulation in Source Code 3.4 and the loop of the SFHT simulation in Source Code 3.6. Array $wPHIX11(i+2,j)$ is prefetched on the PRF simulation. This prefetch type is TYPE II. AN, AE, AT, AP, RGN and DIV are prefetched on the SFHT simulation. This prefetch is classified into TYPE I. These array data are prefetched at one iteration ahead of the outer index i. Figure 3.17 shows the computation efficiency in two loops using the prefetch mechanism on the 2 B/FLOP and 1 B/FLOP systems. In the PRF simulation, prefetching achieves 1.3$\times$ and 1.6$\times$ performance improvements on the 2 B/FLOP and 1 B/FLOP systems, respectively. On the SFHT simulation, the performance is 1.3 times higher than the performance of the non-prefetch case. Figure 3.18 shows (a) the data



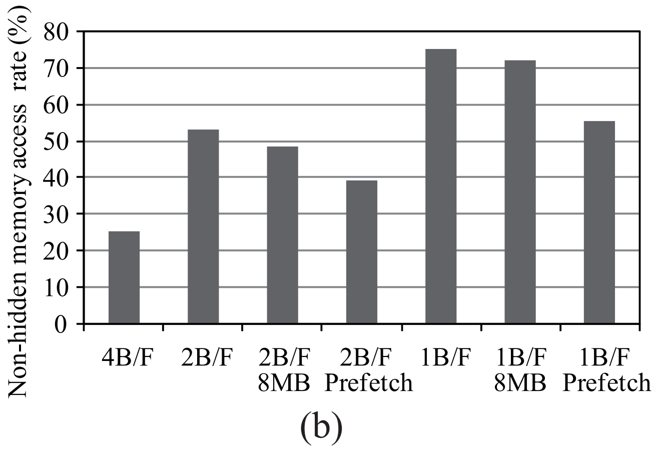


Figure 3.18: (a) Data transfer rate on PRF and (b) Relative non-hidden memory access time rate on the SFHT simulation.

transfer rate on the PRF simulation and (b) the relative non-hidden memory access time on the SFHT simulation. The data transfer rate improvements are 1.5 GB/s and 2.1 GB/s from the non-prefetch case on the 2 B/FLOP and 1 B/FLOP systems, respectively. The non-hidden memory access times are reduced to 70 % of the non-prefetch case on the 2 B/FLOP and 1 B/FLOP systems. Just as the GPR simulation, the prefetch mechanism of TYPE I hides the memory access time by other arithmetic operations. Moreover, the prefetch mechanism of TYPE II increases the data transfer rate.

Finally, the effects of both the MSHR and the prefetch mechanism are evaluated for the three benchmark programs. Figure 3.19 shows the computation efficiencies of the benchmark programs with the MSHR and the prefetch mechanism (P + M in Figure 3.19 indicates the case of using both the MSHR and the prefetch mechanism). The GPR simulation has the synergistic effects of the MSHR and the prefetch mechanism. The performance improvement by using both the MSHR and the prefetch mechanism are 10 % on the GPR simulation. On the PRF simulation, the computation efficiency of the MSHR case is 1.9 % higher than the computation efficiency of both the MSHR and the prefetch case. This is because the busy time of the address control unit is increased by prefetching; the prefetch on the PRF simulation is the same effect of the MSHR, and the prefetch instructions are issued to the address control unit. Besides, as the MSHR effect is small in the SFHT simulation, the synergistic effect of the MSHR and the prefetch mechanism is unavailable. The results suggest that the effects of both the MSHR and prefetch mechanism are more dependent on the characteristics of benchmark programs, however, both the MSHR and prefetch mechanism have the potential for the vector cache to increase the sustained performance.

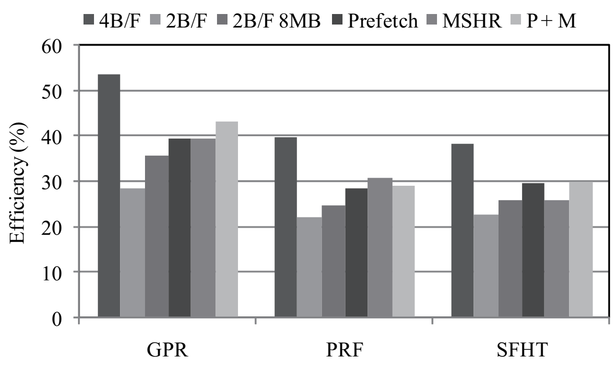


Figure 3.19: Computation efficiency of both MSHR and prefetch.

3.8 Optimizations for Vector Caching: Selective Caching and Loop Unrolling

This section discusses some optimization techniques for the vector cache to reduce cache miss rates on benchmark programs.

3.8.1 Effects of selective caching

As the on-chip size is limited, selective caching is an effective approach to efficient use of the on-chip cache, which is expected to reduce capacity miss rates. The selective caching is evaluated using two benchmark programs, the GPR and PRF simulations. The simulation methods of the two benchmark programs are Finite Difference Time Domain (FDTD) method and Compact Finite Difference Scheme of flow and heart, respectively. Then a part of arrays has a high locality in these cases.

In the GPR simulation, the size of each array used is 330 MB, then the cache is not able to store all the arrays owing to cache size limitation. However, the difference scheme as shown in Source Code 3.1 generally has temporal locality in accessing arrays; H_x, H_y and H_z. These arrays are selected for selective caching, then these arrays require the cache capacity of 2.7 MB to reuse the data with locality. Figure 3.20 shows the effect of selective caching on the 2 B/FLOP system with the vector cache in the GPR simulation. Figure 3.21 shows the recovery rate of the performance from the 2 B/FLOP system without the cache to the 4 B/FLOP system by selective caching. The cache sizes are varied from 256 KB up to 8 MB. Here, “ALL” in the figures indicates that all the arrays in the loop are cached. “Selective” shows that some of the arrays are selectively cached. “Cache Usage Rate” indicates the ratio of number of cache hit references to the

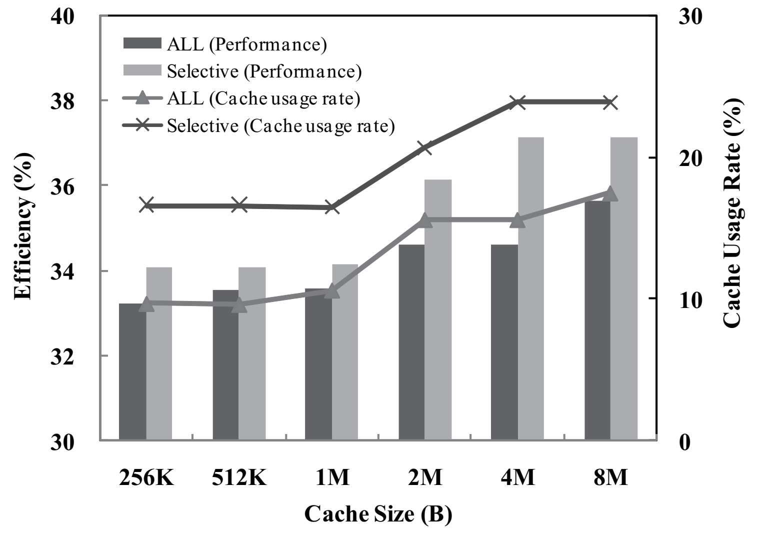


Figure 3.20: Efficiency of selective caching and cache hit rate on 2 B/FLOP system (the GPR simulation).

total memory references.

In the 256 KB cache case, the efficiencies of "ALL" and "Selective" are 33.3 % and 34.1 %, and the cache usage rate are 9.6 % and 16.6 %, respectively. The arrays $H\_y(i,j,k)$, $H\_y(i-1,j,k)$ (line 08 in Source Code 3.1), $H\_x(i,j,k)$ (line 11) and $H\_z(i-1,j,k)$ (line 12) load data from the cache on "Selective." However, the arrays $H\_y(i,j,k-1)$ (line 06) and $H\_x(i,j,k-1)$ (line 11) cause cache misses, because the cache capacity is small, and the reused data of these arrays are replaced by other array data. In "ALL," $H\_y(i-1,j,k)$ (line 08) and $H\_z(i-1,j,k)$ (line 12) cause cache misses, because the reused data of these arrays are replaced by other array data. Figures 3.20 and 3.21 indicate that the performance and the cache usage rate increase as the cache size increase. On the 4 MB cache, the cache hit rate of "Selective" is 24 % and its efficiency is 3 % higher than the efficiency of "ALL." The recovery rate of performance is mere 34 %. In this case, the arrays H_x, H_y and H_z load all the reusable data from the cache by the selective caching. On this difference scheme, array data with temporal locality do not require a large cache capacity. However, the GPR simulation does not achieve

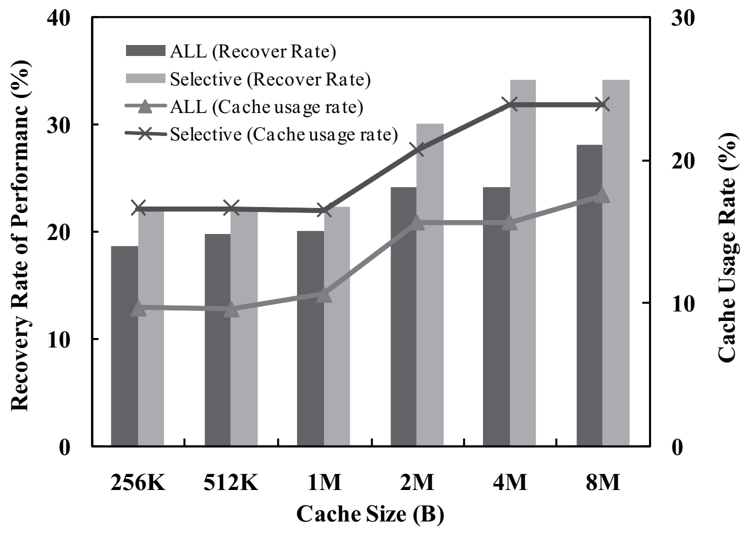


Figure 3.21: Recovery rate of performance and cache hit rate on selective caching (the GPR simulation).

high cache usage rates and recovery rates of performance by selective caching, because this loop has many non-locality arrays; C_x_a, E_x, $E\_x\_Current$ etc.

Similarly, the effect of the selective caching is simulated for the loop of the PRF simulation in Source Code 3.5. The arrays $wDY1YC1$ and $wDY1YC2$ have the spatial locality. These arrays are selected for selective caching.

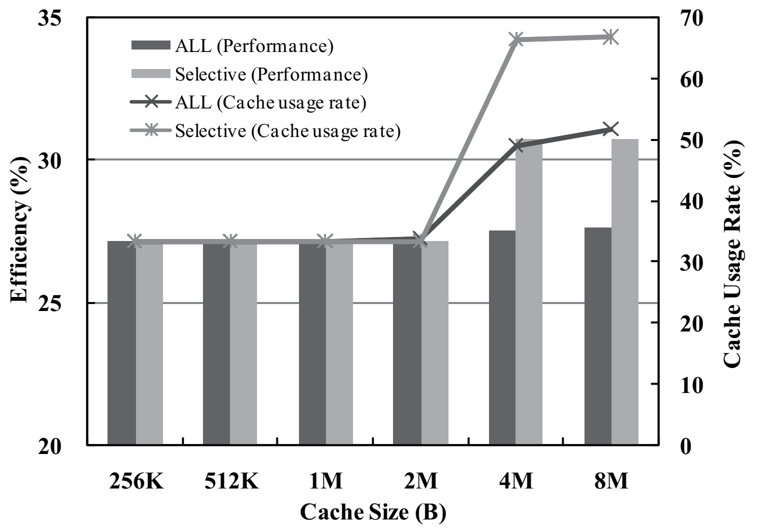


Figure 3.22: Efficiency and cache hit rate of selective caching on performance (the PRF simulation).

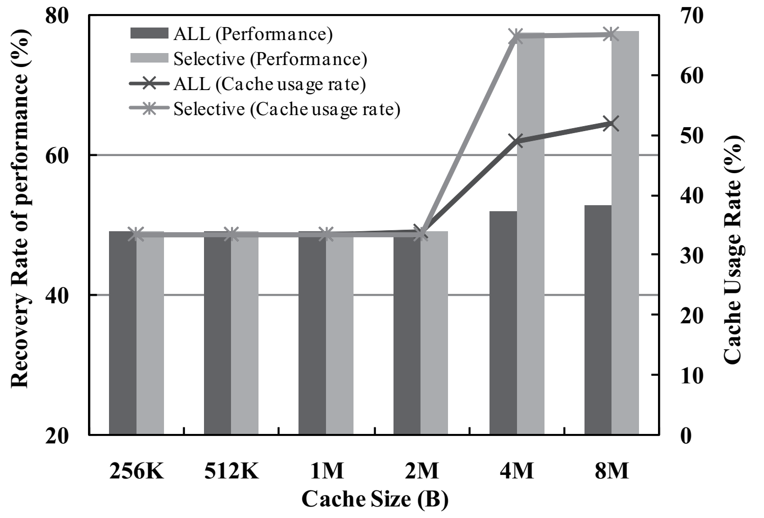


Figure 3.23: Recovery rate of performance and cache hit rate on selective caching (the PRF simulation).

Figures 3.22 and 3.23 show the efficiency and the recovery rate of the performance from the 2 B/FLOP system to the 4 B/FLOP system by selective caching in the PRF simulation. Figure 3.22 indicates that the cache hit rates and efficiency of “Selective” are the same as the cache hit rates and efficiency of “ALL” until 2 MB cache size. The cache hit rate is 33 %. The arrays $wDY1YC2(I, KK-1, L)$ (line 03) and $wDY1YC1(I, KK-1, L)$ (line 08) are cache hit arrays in each case, because these arrays require the cache capacity of 72 KB to reuse the data with locality. Over 4 MB cache size, however, “Selective” achieves a 66 % cache hit rate, resulting in a 30+ % improvement in efficiency. Furthermore, Figure 3.23 indicates that the recovery rate is 78 %. In this case, $wDY1YC2(I, KK, L)$ and $wDY1YC1(I, KK, L)$ (line 04) reuse the data defined in the preceding loop. Therefore, on selective caching, array data are reused between loops, when the array is small.

The results of the GPR and PRF simulations show that the selective caching improves the performance of the benchmark programs. Compared with the GPR simulation, the PRF simulation has higher values regarding both the cache usage rate and improved efficiency, because the ratio of arrays with locality per loop on the PRF simulation is higher than the ratio of arrays with locality per loop on the GPR simulation. In the GPR simulation, the reused data are only defined in the loop, and the cache miss always occurs in this loop. For example, arrays $H\_y(i,j,k)$ (line 08 in Source Code 3.1) hits the data of $H\_y(i,j,k)$ (line 06), but $H\_y(i,j,k)$ (line 06) does not hit the data owing to their first accesses (cold miss). On the other hand, the cached data of the PRF simulation are provided in the immediately preceding loop, and therefore, cache misses do not occur like the first access in the PRF simulation.

3.8.2 Effects of loop-unrolling and caching

Loop unrolling is effective for higher utilization of vector pipelines, which is a basic loop optimization. However, loop unrolling also needs more cache capacity to capture all the arrays of unrolled loops. Therefore, the relationship between loop unrolling and vector caching is clarified.

Figure 3.24 shows the efficiency and the cache hit rate of the outer-unrolling case on the PBM simulation with the 2 B/FLOP system. "2B/F" indicates the without-cache case, and the efficiency constantly increases as the degree of unrolling increases. The cache hit rate of the 0.5 MB cache case is 49 % on the non-unrolling case and becomes poor as the loop unrolling proceeds, because the cached data of $wary$ in Source Code 3.7 are evicted from the cache by gd_dip. Thus, the efficiency of the 0.5 MB cache case achieves 49 % on the non-unrolling case, and it is similar to the efficiency of the 2 B/FLOP system with unrolling. In this case, a conflicting effect between caching and unrolling appears. Moreover, the cache hit rates of the 8 MB cache case decrease as the degree of unrolling increases. The cached data remain on the cache in this case, but the cache hit rate decreases owing to a decrease in the number of *wary* references. However, the efficiency is approximately constant; 48 % or 49 % because unrolling covers the losing effect of caching. This case shows that the effects of caching are comparable to that of unrolling.

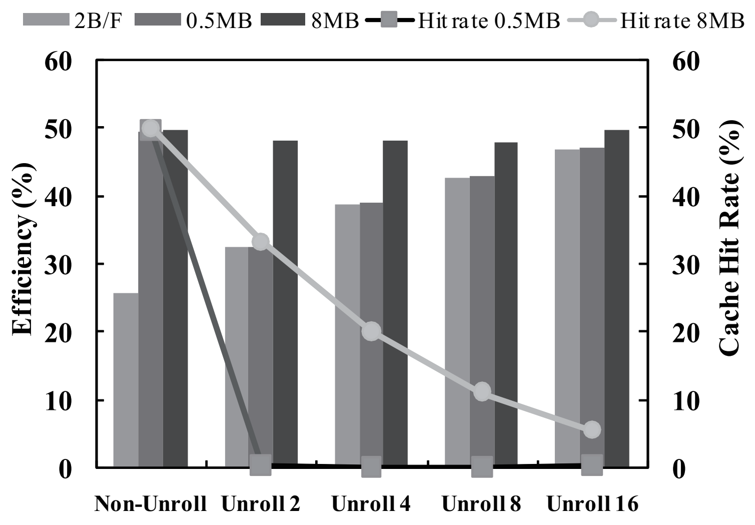


Figure 3.24: Efficiency and cache hit rate of outer-unrolling on performance (the PBM simulation).

Similarly, the effect of the unrolling is simulated for the loop of the PRF simulation in Source Code 3.5. Figure 3.25 shows the efficiency and the cache hit rate of outer-unrolling with the 2 B/FLOP system. The cache hit rates of both the 0.5 MB and 8 MB caches gradually reduce as the degree of unrolling increase, because of an increase in the cache miss rate. In this case, the highest efficiency is 33 % on the 8 MB cache with the unrolling degree of 4 since the gain by loop unrolling covers the loss owing to the increase in the miss rate. It is higher than the selective caching case; it is 30.7 % (Ref. Figure 3.22). Thus, this case indicates the synergistic effect of both caching and unrolling.

These results indicate that loop unrolling has both conflicting and synergistic effects with caching. Loop unrolling is a basic optimization and has beneficial effects in various applications. Therefore, caching has to be used carefully as a complementary tuning option for loop unrolling.

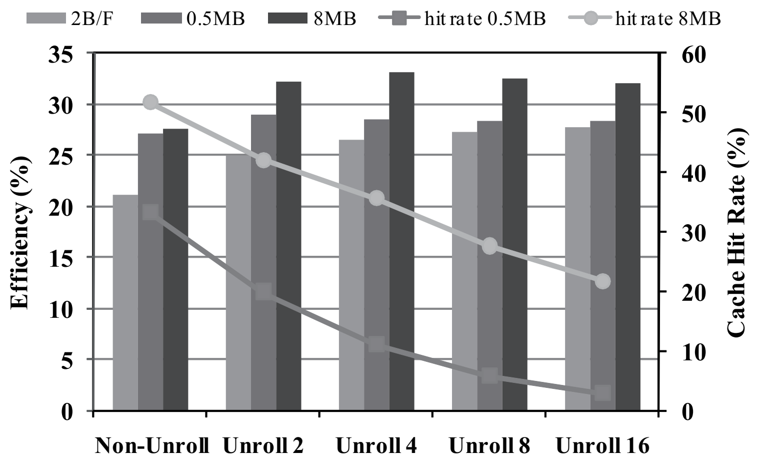


Figure 3.25: Efficiency and cache hit rate of outer-unrolling on performance (the PRF simulation).

3.9 Conclusions

This chapter has presented that the vector cache has a potential for the future vector supercomputers to cover the shortage of their memory bandwidth, and clarifies the characteristics of an on-chip vector cache with the bypass mechanism on the vector supercomputer NEC SX architecture. The relationship between the cache hit rate and the performance is demonstrated using two DAXPY-like loops. Moreover, this chapter has clarified the same relationship found on the five benchmark programs. The vector cache recovers the lack of the memory bandwidth, and boosts the computation efficiencies of the 2 B/FLOP and 1 B/FLOP systems. The degree of the contribution of the vector caches highly depends on the characteristics of the applications, and the vector cache increases the recovery rate of the performance in execution of the five applications by 21 % to 99 % on the 2 B/FLOP system and 9 % to 96 % on the 1 B/FLOP system. Especially, when cache hit rates are 50 % or more, the 2 B/FLOP system achieves a performance comparable to the 4 B/FLOP system. The vector cache with a bypass mechanism provides the data from both the memory and the cache at once, and the sustained memory bandwidth for the registers increase.

This chapter has also discussed the relationship between performance and cache design parameters such as cache associativity and cache latency. The effect of the cache associativity from 2-way to 8-way is examined. The associativity has effects on the cache hit rates when a loop accesses many arrays at intervals of large stride addresses. In addition, the effect of the cache latency is examined on the performance when changing it to 15 %, 50 % , 85 % and 100 % of the memory latency. It is demonstrated that the computational efficiencies of five benchmark programs are constant across these latency changes, when the vector loop lengths of the benchmark programs are 256 or more. In these cases, the latency is hidden by pipelined vector operations. However, in the case of shorter vector loop lengths, the cache latency affects the performance, and the 15 % latency case of Kernel (1) has two times higher efficiency than the 100 % case in the 2 B/FLOP system. In addition, the 4 B/FLOP system is also boosted owing to the effect of the short latency of the cache.

Moreover, this chapter has discussed the potential of on-chip vector cache with the MSHR and the prefetch mechanism for the future vector supercomputers, which have insufficient memory bandwidth per flop/s rates. The effects of the MSHR are evaluated on three scientific applications. The MSHR reduces the number of load requests on the difference scheme loops which continuously load the same data, and the latency of the subsequent load requests is shortened. Thus, the MSHR improves the performance by 5 % to 25 % on the 2 B/FLOP system, and 4 % to 45 % on the 1 B/FLOP system. In addition, this chapter have demonstrated the performance of the prefetch mechanism under the two types; TYPE I: the prefetch mechanism hides the memory access time by other arithmetic operations, and TYPE II: the prefetch mechanism reduces the number of load requests to the main memory by removing redundant load instructions accessing the same memory address. The prefetching mechanism boosts the performance by 20 % to 30 % on the 2 B/FLOP system and 20 to 60 % on the 1 B/FLOP system.

Finally this chapter has discussed selective caching and the relationship between loop-unrolling and caching. This chapter has shown that selective caching, which is controlled by means of the bypass mechanism, is effective for efficient use of the limited capacity of the on-chip caches. Two cases are examined; the ratios of arrays with locality per loop are higher and lower cases. In each case, the higher performance is obtained by selective caching, compared with all the data caching. In addition, the loop unrolling is useful in the improvement of performance, and caching is complementary to the effect of loop unrolling.

Chapter 4

A Shared Cache for a Chip Multi Vector Processor

4.1 Introduction

Thanks to advances in circuit integration technologies, chip multiprocessors (CMPs) have become the mainstream in commodity-based scalar processors. Eight-core CMPs are already found in the commercial market. CMP-based vector processors have not been found in the commercial market yet. However, the CMP architecture is also promising for vector processor design, because the number of transistors in a vector processor has been increasing by a factor of eight for the last decade. In the cases of the NEC SX vector supercomputers, the vector processor of the SX-7 system released in 2001 consists of 60 million transistors, and the vector processor of the SX-9 system released in 2008 has 350 million transistors manufactured using the 65 nm technology. Thus, vector pipelines will be added in a vector processor. However, many scientific and engineering applications are parallelized for multi-threads using the automatic parallelization and OpenMP.

The computational granularity of the multi-threads is greater than the granularity of loop vectorization, and it is more effective that the CMP architecture is adopted by vector supercomputers. Therefore, the CMP architecture will be mainstream of future vector supercomputers.

A characteristic of modern vector supercomputers is their high off-chip memory bandwidth, which brings significant advantages of vector supercomputers over the scalar-based systems [52], [53]. In Chapter 2, the ratio of memory bandwidth to the floating-point operation rate needs to reach 4 B/FLOP to keep a high sustained performance in execution of real scientific applications. When a vector processor employs a multi core processor chip, however, it is getting harder to keep a high memory bandwidth balanced with the improvement of their flop/s performance owing to the limited pin bandwidth. If the memory bandwidth is not enough for multi vector core processors, the vector processors would be unable to outperform even commodity-based scalar processors.

In Chapter 3, it is indicated that an on-chip vector cache improves the effective memory bandwidth rate of a single-core vector processor, when its off-chip memory bandwidth is decreased. Then, it is clarified that the vector cache increases the sustained performance to a certain degree on the 1 B/FLOP and 2 B/FLOP systems. In this chapter, the vector cache is adopted by a chip multi vector processor (CMVP) to cover its limited off-chip memory bandwidth. In particular, the effects of the on-chip shared vector cache are discussed on the performance of the CMVP when executing real scientific applications.

The rest of the chapter is organized as follows. Section 4.2 presents related work. Section 4.3 indicates the locality of a difference scheme in a multi-threading case. Section 4.4 describes the design of a CMVP architecture discussed in this chapter. Section 4.5 provides an experimental methodology and benchmark programs for the evaluation of the CMVP. Section 4.6 presents experimental results when executing five scientific applications on the vector architecture. The effects of a shared cache on the performance of the CMVP are discussed. Finally, Section 4.7 summarizes the chapter.

4.2 Related Work

The investigations on a chip multi vector processor are unpublished up to the present time. In this chapter, the papers which are discussed about a shared cache in scalar systems are described.

On studies of shared caches, Nayfeh *et al.* have investigated the performance of three memory architectures on a multiprocessor: shared-primary cache, shared-secondary cache, and shared memory [49]. Their CPU and the parallel memory references are modeled using Mipsy and the SimOS simulator, respectively. They consider a 16 KB 2-way set-associative L1 cache and a 512 KB 2-way set-associative L2 cache. Their benchmark programs are selected from SPEC92 [11] and SPLASH [65]. They indicated that the shared-primary cache and shared-secondary cache architectures have a potential for increasing the sustained performance of the benchmark programs.

Although CMPs have been studied by many researchers, most investigations of CMPs are concerned with scalar processors. Peng *et al.* have evaluated the memory performance and scalability in commodity-based scalar processors [56]: Intel Core 2 Duo [26], Intel Pentium D [27] and AMD Athlon 64X2 [3]. The aim of their papers is to demonstrate that the computational performance and scalability on CMPs are impacted by the memory hierarchy architecture among three processors. Their benchmark programs are selected from SPEC CPU2000, SPEC CPU2006, SPEC jbb2005 [11], SPLASH2 [71] and BioPerf [5]. They shown that Core 2 Duo has the best performance for most of the benchmarks, because Core 2 Duo employs the shared L2 cache that stores the data shared by multiple threads.

So far, there is no chip vector multiprocessor for high end computing systems so called supercomputers, however, the CMP architecture is definitely the key technology for future vector processor design. As Chapter 3 has indicated that an on-chip cache has the potential for keeping the performance when their off-chip memory bandwidth is limited, the concepts of a shared cache mechanism for scalar-based CMPs are applied to a CMVP for increasing the performance. Thus, a CMVP with a shared cache mechanism is designed, and its performance using scientific and engineering applications is evaluated.

4.3 Characteristics of Scientific Applications

Various scientific applications generally utilize difference schemes for simulating the physics phenomena. When the applications of difference schemes are parallelized for multi-threading, a part of arrays has a high locality among threads. The example of a difference scheme is Source Code 3.1 in Chapter 3: it is a kernel routine of Finite Difference Time Domain Method (FDTD method). The outermost loop, index k, is parallelized by multi-threading. In the case of single thread execution, array elements $H\_y(i,j,k)$ (line 06 in Source Code 3.1) and $H\_x(i,j,k)$ (line 11) are cached, but they are not reused as $H\_y(i,j,k-1)$ (line 06) and $H\_x(i,j,k-1)$ (line 11), because they are replaced by other data before reused. In the case of multi-threading execution, however, they are reused as $H\_y(i,j,k-1)$ and $H\_x(i,j,k-1)$ by another thread. In the difference schemes, one thread often reuses data on a shared cache that are fetched by another thread.

As shown in Chapter 3, it is indicated that the MSHR is a necessary function for difference schemes to effectively utilize their locality of reference. Since a latency of main memory is much longer than CPU cycles, the MSHR effectively handles subsequent vector loads of the same data, whose fetch request is outstanding to the memory. This is because each thread of a difference scheme loads the same data at the same time on multi-threading, even though the reused data are not yet filled into the cache when the subsequent load instructions are issued. When loop k in Source Code 3.1 is parallelized, two array elements, $H\_y(i,j,k)$ (line 06 in Source Code 3.1) and $H\_x(i,j,k)$ (line 11) are cached and reused. However, without the MSHR, they are not reused on the cache as $H\_y(i,j,k-1)$ (line 06) and $H\_x(i,j,k-1)$ (line 11).

4.4 Chip Multi Vector Processor

4.4.1 Basic mechanism of Chip Multi Vector Processor

On a chip multi vector processor, maintaining the memory bandwidth rate to 4 B/FLOP becomes more difficult. As shown in Chapter 3, the vector cache improves the effective memory bandwidth rate of a single-core vector processor. Therefore, the vector cache with the bypass mechanism and the MSHR mechanism is a key technology for the CMVP. The constitution of the vector cache is not a private cache but shared cache, because difference schemes of scientific simulations have a high locality among multi-threads.

A core of the CMVP makes the same composition as the processor as shown in Chapter 3. Each core has 32 memory ports. The core and the vector cache are interconnected through 32 crossbar switches at 4 B/FLOP. Each crossbar switch has a priority control mechanism of data transfer from the cores to the vector cache. When two or more cores send data at once, the data are forwarded to the vector cache according to a priority policy. In this chapter, the priority level of the data transfer is as follows. $Core0 > Core1 > Core2 > Core3$

4.4.2 Structure of Chip Multi Vector Processor

Figure 4.1 shows the CMVP block diagram, which has four vector cores and a shared vector cache. The vector core and the vector cache are similar to the architecture shown in Chapter 3. Figure 4.2 shows the block diagram of a core of the CMVP. Each core works independently. The core has a vector unit, scalar unit and an address control unit. The vector unit contains four parallel vector pipe sets, each of which has five types of vector arithmetic pipes (Mask, Logical, Add/Shift, Multiply, Divide), and vector registers.

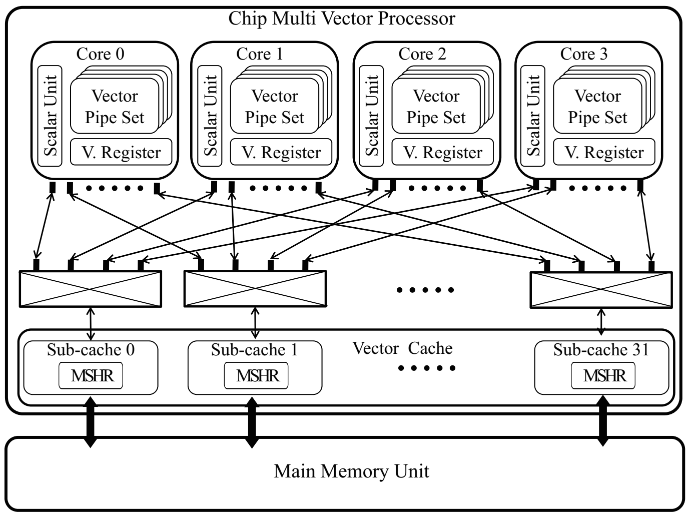


Figure 4.1: CMVP block diagram.

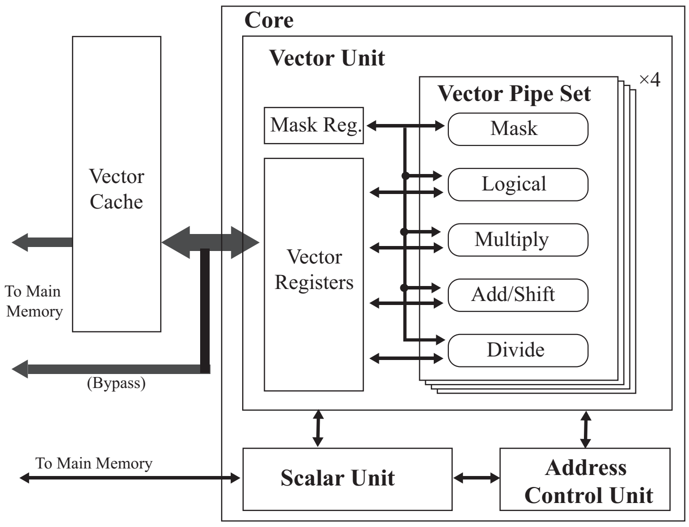


Figure 4.2: Block diagram of a core.

The vector cache consists of 32 sub-caches. The sub-cache is a non-blocking cache and includes a tag array, a data array and a MSHR. The sub-cache employs a set-associative write-through cache with the LRU replacement policy. The line size is 8 bytes. The vector cache reduces the memory access latency by 85 % and the bank cycle by 95 %. The memory bandwidth between the vector cache and the vector core is 4 B/FLOP.

The main memory unit is an interleaved memory system and is divided into

32 parts, each of which operates independently and supplies data to the cores. The maximum memory bandwidth rate, B/FLOP, between the main memory and the vector processor is 4 B/FLOP.

4.5 Experimental Environment

4.5.1 Methodology

A trace-drive simulator is developed for simulating the behavior of the proposed CMVP architecture at the register transfer level. The simulator is enhancement of the NEC SX simulator as shown in Chapter 3. Particularly, this simulator deals with a parallelized program by multi-threads of DO Loop level using the automatic parallelization and OpenMP.

To evaluate the CMVP as a future vector processor with a higher flop/s rate but a relatively lower off-chip memory bandwidth, the effects of the vector cache are evaluated on the CMVP by limiting its memory bandwidth per flop/s rate from 4 B/FLOP down to 1 B/FLOP.

Table 4.1: Summary of setting parameters.

Base System Architecture	SX-7
Number of Core	1, 2, 4
Main Memory	DDR-SDRAM
Memory Size	256 GB
Number of bank	16,384
Vector Cache	SRAM
Total Size (Sub-cache)	256 KB - 8 MB (8 KB - 256 KB)
Associativity	2WAY, 4WAY, 8WAY
Cache Policy	write-through, LRU replacement
Line Size	8 bytes
Cache Latency	15 % of main memory latency
Cache Bank Cycle	5 % of main memory cycles
MSHR Entries (Sub-cache)	65,536 (2048)
Memory – Cache bandwidth per flop/s	1 B/FLOP, 2 B/FLOP, 4 B/FLOP
Cache – Register bandwidth per flop/s	4 B/FLOP

The setting parameters are shown in Table 4.1. Here, the NEC SX-7 architecture [29] is adopted as a vector core, and the vector cache size is 8 MB, which is the same as Cray BlackWidow's cache capacity. The bandwidth per flop/s between the main memory and the vector cache indicates a value per the four-core CMVP. In addition, on the single core case, Core 0 is simulated in Figure 4.1 that exclusively uses the full bandwidth between the main memory and the vector cache. On the two cores simulation, Core 0 and Core 1 share the bandwidth.

4.5.2 Benchmark programs

The performance of the CMVP with the shared vector cache is evaluated by using the five benchmark programs as shown in Chapter 2. Here, the four benchmark programs, the GPR, APFA, PRF and SFHT simulations, utilize difference schemes, and the PBM simulation does not use the difference schemes.

As shown in Section 4.3, difference schemes have the high locality among threads. The PBM simulation has also a high locality of reference in multithreading. In Source Code 3.7 of the PBM simulation, the outer loop, index i, is parallelized by multi-threading. The array $wary(j)$ has a capacity of 250 KB, and the array gd_dip accessed by four threads spills the array $wary(j)$ from the vector cache, when the cache size is below 1.25 MB. However, each thread accesses the same array data $wary(j)$ at once. Hence, using the MSHR, three threads of the subsequent load instruction reuse the cached data of the array $wary(j)$.

4.6 Performance Evaluation of CMVP

In this section, the performance of the CMVP is evaluated using the benchmark programs, and the potential of the shared cache with the MSHR for the CMVP is clarified.

4.6.1 Scalability of the applications without the cache

First, the performance of five benchmark programs is evaluated when changing the B/FLOP rate from 4 down to 1 in the four-core CMVP without the vector cache. Figure 4.3 shows the relative performance of the five benchmark programs. Here, the relative performance is obtained by normalizing the sustained performance of each case by the sustained performance of a single core at the 1 B/FLOP rate. When the B/FLOP rate is 4, the speedups of the five benchmark programs are greater than 3.6 on the four-core CMVP. The 4 B/FLOP CMVP scales well, because each program has a high parallel ratio. However, the amount of the supplied date from the memory is insufficient on the 1 B/FLOP and 2 B/FLOP CMVPs. Thus, the scalability of the CMVP decreases as the B/FLOP rate decreases. Especially, the relative performance on the four-core CMVP in the GPR and PBM simulations, which are memory intensive, only achieve speedups of 1.1 at the 1 B/FLOP rate and 2.2 at the 2 B/FLOP rate. Therefore, these results indicate that B/FLOP rates seriously affect the scalability of the CMVP.

4.6.2 Scalability of the applications with the vector cache

Figure 4.4 shows the relative performance on the vector cache with the MSHR when the B/FLOP rates are reduced from 4 to 1. Here, the relative performance is obtained by normalizing the sustained performance of each case by the sustained

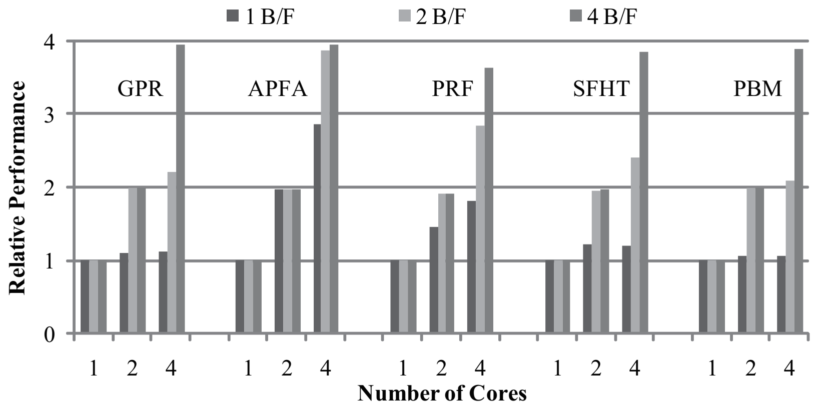


Figure 4.3: Relative performance of programs on the four-core CMVP without the vector cache.

performance of a single core without the cache at the 1 B/FLOP rate. The scalability of the five benchmark programs is improved by the cache mechanism. Especially, the performances of the 1 B/FLOP and 2 B/FLOP CMVPs for the APFA simulation are comparable to the performance of the 4 B/FLOP CMVP. For the PRF and PBM simulations, the performance of the 2 B/FLOP CMVP approximately achieves the performance of the 4 B/FLOP CMVP.

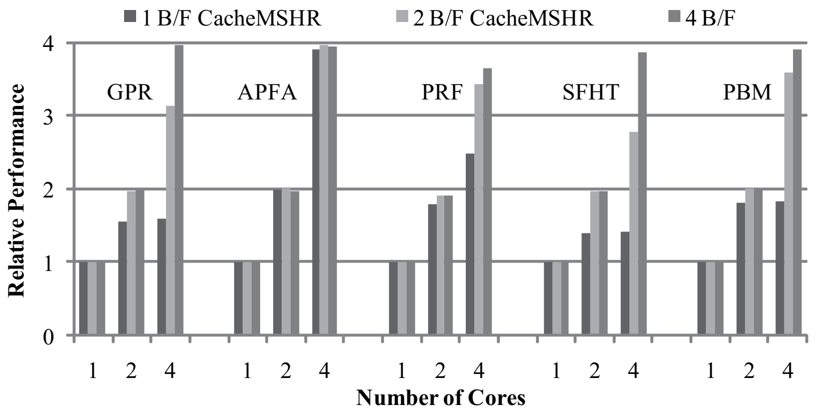


Figure 4.4: Relative performance of programs on the four-core CMVP with the vector cache.

The relative performances of the 1 B/FLOP CMVP are smaller than the relative performances of the 2 B/FLOP CMVP. This is because the 1 B/FLOP CMVP does not provide data to vector registers at the rate of 4 B/FLOP unless all the data are on the cache, even though the 2 B/FLOP CMVP can. Especially, the performance of the 2 B/FLOP CMVP is approximately equivalent to the performance of the 4 B/FLOP CMVP, when the data are provided to the register file from the cache and the memory at 2 B/FLOP rate each. Therefore, the off-chip memory bandwidth per CMVP needs to satisfy at least 2 B/FLOP to achieve a high scalability.

4.6.3 Relationship between associativity and cache hit rate

In Chapter 3, it is provided that the associativity has effects on the cache hit rates in the single core case. In this subsection, the effect of the associativity is examined ranging from 2-way to 8-way on the cache hit rate in the multi core case. set

Figure 4.5 shows that the cache hit rate on each set associative cache by changing cache sizes from 256 KB to 8 MB. Three benchmark programs, the GPR, PRF and SFHT simulations approximately have constant cache hit rates across three associativity cases. In the APFA and PBM simulations the cache hit rates vary with the associativity. Particularly, in the APFA simulation the cache hit rate increase by 10 % from a 2-way set associative cache to an 8-way set associative cache on the 256 KB cache capacity. Then, in the PBM simulation the cache hit rate increase by 4 % from a 2-way set associative cache to an 8-way set associative cache on the 2MB cache capacity.

On the multi-core case, each core accesses the cache at once, and the cache access patterns become random. Then, the placement of the arrays on the cache causes the conflict of reused data. Therefore, CMPV requires 4-way or more associative cache.

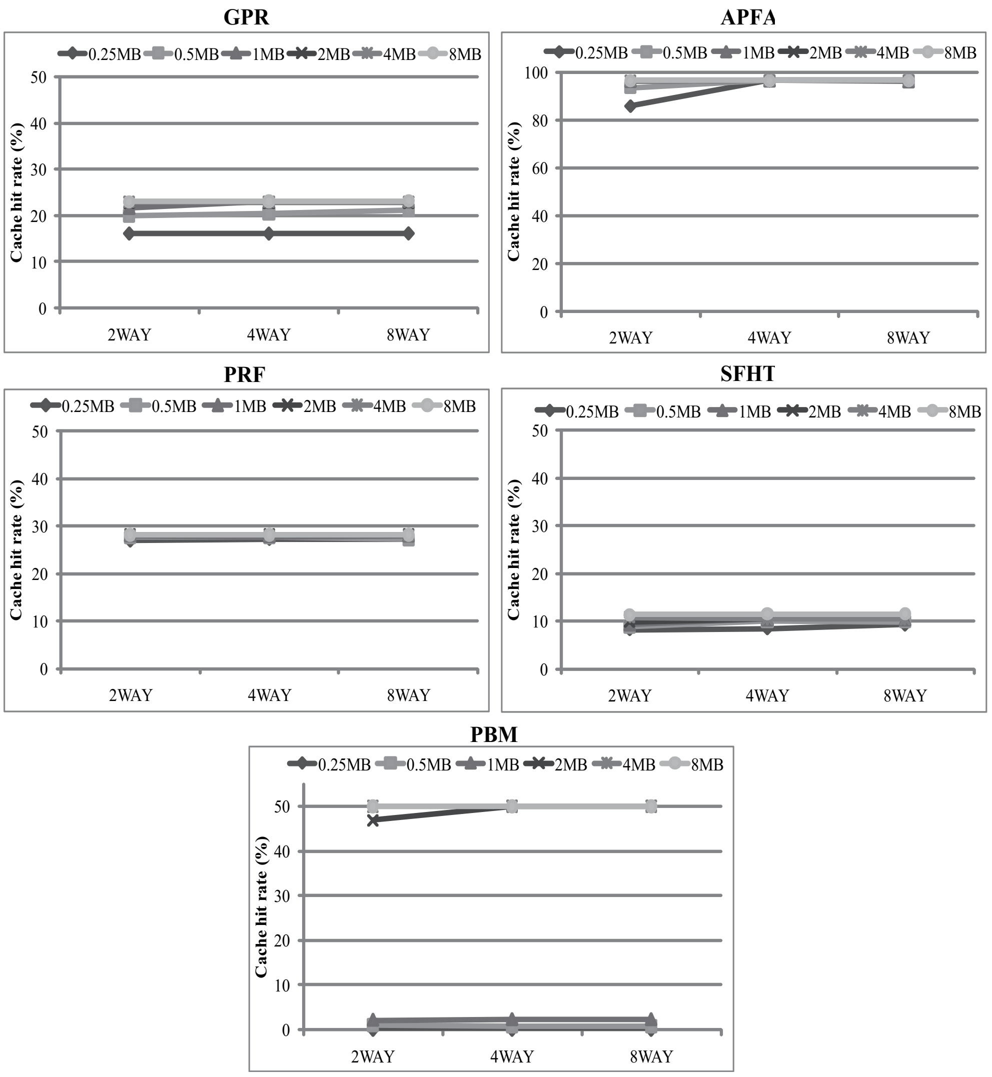


Figure 4.5: Cache hit rate vs. associativity on the five benchmark programs in the four core case.

4.6.4 Effect of the shared vector cache

For multi-threaded programs of various difference schemes, a thread reuses the data on the cache previously loaded by another thread. Here, the effect of the shared cache with the MSHR is shown using the GPR simulation, which uses the FDTD method, as shown in Subsection 4.3. Figure 4.6 indicates the cache hit rates and the improved efficiencies per core by the shared cache at the 2 B/FLOP rate. The cache hit rate increases by 6.5 % from the one-core case to the four-core case, and the improved efficiency per core is increased by 3.3 %, because the array elements $H_y(i, j, k-1)$ (line 06 in Source Code 3.1) and $H_x(i, j, k-1)$ (line 11) reuse the cached data on the shared cache. For the accesses of Core 1 to array element $H_y(i, j, k-1)$ at index $k = n$, Core 1 reuses the cached data of the array element $H_y(i, j, k)$ that has been loaded at index $k = n-1$ by Core 0. Consequently, sharing the cache among vector cores increases the cache hit rate of difference schemes, resulting in performance improvement.

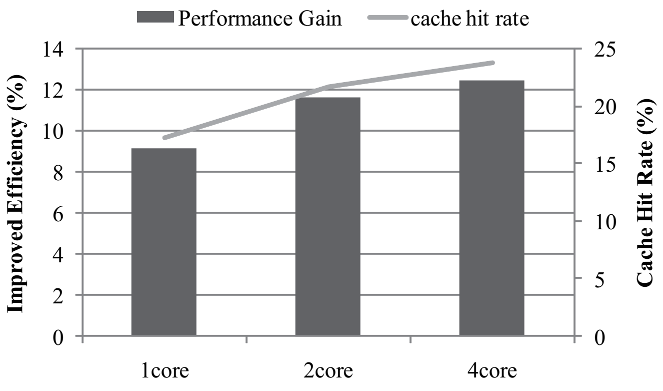


Figure 4.6: Cache hit rate and improved efficiency on a kernel loop of the GPR simulation at the 2 B/FLOP.

4.6.5 Effect of the MSHR

For both multi-threaded and vectorized programs of difference schemes that continuously load the same data, the MSHR has the potential for increasing the performance. Figure 4.7 shows the relative performance with/without the MSHR on the four-core CMVP at the 2 B/FLOP rate. Here, the relative performance is obtained by normalizing the sustained performance of each case by the sustained performance of the four-core CMVP without the cache at the 2 B/FLOP rate. The improvements of the relative performance obtained by the MSHR are from 6 % in the SFHT simulation to 17 % in the GPR simulation. Table 4.2 shows the cache hit rates of three applications with/without the MSHR. The MSHR improves the cache hit rates by 4.9 % to 6.4 %. Therefore, the MSHR increases the number of opportunities to reuse data on the cache across the applications of the difference schemes.

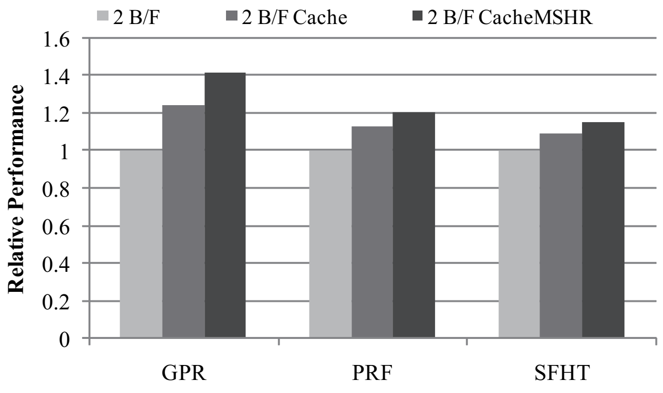


Figure 4.7: Relative performance with/without the MSHR on the GPR, PRF and SFHT simulations at the 2 B/FLOP.

Table 4.2: Cache hit rate

	GPR	PRF	SFHT
Cache	23.0 %	21.1 %	6.2 %
Cache + MSHR	27.9 %	27.5 %	11.5 %

In the PBM simulation, which does not use the difference schemes, the MSHR is also an effective way to increase the performance of the CMVP. Figure 4.8 indicates the cache hit rates with/without the MSHR on the four-core CMVP when changing the cache size from 256 KB to 8 MB. The MSHR improves the cache hit rates by 33 % when the cache size is below 1 MB. Moreover, Figure 4.9 shows the relative performance with/without the MSHR on the four-core CMVP at the 2 B/FLOP rate. Here, the relative performance is obtained by normalizing the sustained performance using the cache without the MSHR at the 2 B/FLOP rate. The improvements of the relative performance using the MSHR is 50 % below 1 MB cache. Therefore, the MSHR has the potential to increases the sustained performance when each thread share in the same data of the vector cache.

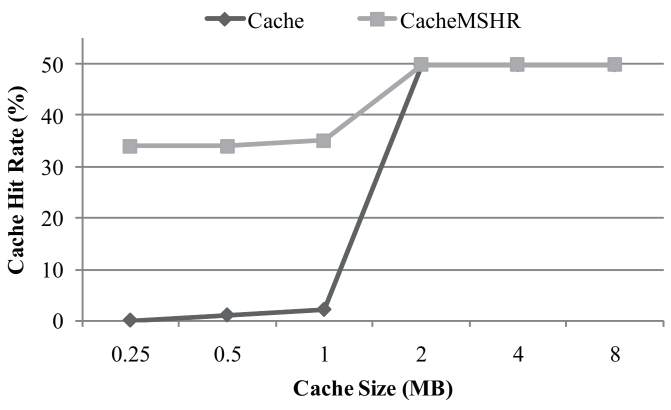


Figure 4.8: Cache hit rate with/without the MSHR on a kernel loop of the PBM simulation.

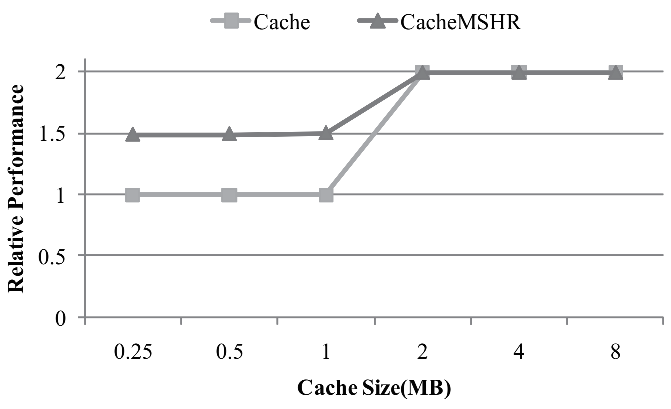


Figure 4.9: Relative performance with/without the MSHR on a kernel loop of the PBM simulation at the 2 B/FLOP.

4.7 Conclusions

In this chapter, the performance of the CMVP is evaluated using five scientific applications. The CMVP contains four vector cores and an on-chip shared cache with the MSHR. It is shown that the CMVP without the cache needs the 4 B/FLOP rate of the off-chip memory bandwidth per core for maintaining the scalability of vector processors in sustained performance. However, a future vector supercomputer will not be able to keep the 4 B/FLOP rate owing to the limited pin bandwidth. Therefore, the on-chip shared cache is proposed for the CMVP. The evaluations of the CMVP have shown that the off-chip memory bandwidth on the CMVP should satisfy at least 2 B/FLOP using the cache mechanism to achieve a high scalability. . This chapter has discussed the relationship between performance and cache design parameters: cache associativity. The effect of the cache associativity is examined setting from 2-way to 8-way. The cache hit rate is sensitive to the associativity in the two benchmark programs. The results indicate that the performance cache of CMVP depend on the cache associativity.

Moreover, the effect of the shared cache and the MSHR are evaluated using the scientific applications. The results show that the shared cache increases the cache hit rate and the efficiency of the GPR simulation by 6.5 % and 3.3 %, respectively. Meanwhile, the MSHR is effective for both the multi-threaded and the vectorized programs. The cache hit rates increase by 4.9 % on the GPR simulation and by 33 % on the PBM simulation, and the relative performance improves by 14 % on the GPR simulation and by 50 % on the PBM simulation

Chapter 5

Conclusions

Vector supercomputers achieve the high computation efficiency by the high B/FLOP rate in scientific and engineering applications. However, the vector supercomputers encounter the memory wall problem, and the computation efficiency decreases owing to the widening gap between the memory bandwidth and processor performance. In the circumstances, this dissertation proposes a high performance memory architecture to overcome the memory wall problem on vector supercomputers. The high performance memory architecture is a vector cache mechanism, which has four mechanisms: bypass mechanism, miss status handling register, prefetch mechanism, selective caching. The objective of this dissertation is to clarify the following points.

- Performance of vector supercomputers is maintained by memory performance, particularly the B/FLOP rate.
- A proposed high performance memory architecture has the potential of increasing the relative B/FLOP rate for future vector supercomputers.
- Scalability of a chip multi vector processor on future vector supercomputers is maintained by memory performance, particularly the B/FLOP rate.

- A shared vector cache has the potential of increasing the relative B/FLOP rate on the chip multi vector processors.

Furthermore, this dissertation clarifies the characteristics of the high performance memory architecture.

In Chapter 2, the memory performance of vector supercomputers and scalar systems is examined across five scientific applications. The vector supercomputers achieve the high computation efficiency and significantly outperform the scalar systems in the scientific applications, because the vector supercomputers use the interleaved memory systems and their memory access latencies are hidden by pipelined vector operations. The performance of the interleaved memory systems is preserved by the memory bandwidth and enough number of memory banks. In particular, the B/FLOP rate needs to reach 4 to preserve the computational performance in the vector supercomputers.

In Chapter 3, the potential of the proposed vector cache is discussed for future vector supercomputers, whose memory bandwidth rates will decrease at 2 B/FLOP or lower. The aim of the vector cache is to reduce data traffic between the main memory and the vector processor, and effectively to use the memory bandwidth. Thus the vector cache consists of sub-caches, which prevent the diminishing of the effect of the interleaved memory system. Furthermore, the vector cache employs a bypass mechanism between the main memory and vector register files, then the total amount of data provided to the vector register files is increased by the bypass mechanism. These mechanisms are examined using the five scientific applications. The vector cache increases the recovery rate of the performance in execution of the five applications by 21 % to 99 % on the 2 B/FLOP system and 9 % to 96 % on the 1 B/FLOP system. Especially, as cache hit rates are 50 % or more, the 2 B/FLOP system achieves a performance comparable to the 4 B/FLOP system.

Moreover, the MSHR, the prefetch mechanism and the selective caching are introduced in the vector cache to improve the effect of the vector cache. These mechanisms reduce redundant data supplied from the main memory and hide the latency of the main memory. Furthermore, these mechanisms become effective use for the cache capacity. The experimental results indicate that the MSHR improves the computational performance by 5 % to 25 % on the 2 B/FLOP system, and 4 % to 45 % on the 1 B/FLOP system. The MSHR reduces the number of load requests within the difference scheme loops which continuously load the same data, and the latencies of the subsequent load requests are shortened. In the prefetch mechanism, the performance of the prefetch mechanism is demonstrated under the two types; TYPE I: the prefetch mechanism hides the memory access time by other arithmetic operations, and TYPE II: the prefetch mechanism reduces the number of load requests to the main memory by removing redundant load instructions accessing the same memory address. The prefetching mechanism boosts the performance by 20 % to 30 % on the 2 B/FLOP system and 20 % to 60 % on the 1 B/FLOP system. In the selective caching, two cases are examined; the ratios of arrays with locality per loop are higher and lower cases. In each case, the higher performance is obtained by selective caching, compared with all the data caching.

In Chapter 4, the performance of the chip multi vector processor (CMVP) and the effectiveness of the shared vector cache are discussed. A vector processor will employ the chip multi processor architecture in the near future. However, the CMVP does not preserve the high memory bandwidth rate owing to the memory wall problem. Thus, the vector cache is a key to improve the effective memory bandwidth rate for the CMVP. Moreover, because difference schemes of scientific simulations have a high locality among multi-threads, the vector cache employs a shared cache. Then, the performance of the CMVP is evaluated using the five applications.

The CMVP without the cache needs the 4 B/FLOP rate of the off-chip memory bandwidth per core for maintaining the scalability of vector processors in sustained performance. When the B/FLOP rate is 4, the speedups of the four programs are greater than 3.6 on the four-core CMVP, however, the scalability of the CMVP decreases as the B/FLOP rate decreases. Especially, the relative performance on the four-core CMVP in the memory intensive programs only achieve speedups of 1.1 at the 1 B/FLOP and 2.2 at 2 B/FLOP rates. Meantime, by the shared cache mechanism with MSHR, the scalability of the four programs is improved. Especially, the performances of the 1 B/FLOP and 2 B/FLOP CMVPs for the APFA simulation are comparable to the performance of the 4 B/FLOP CMVP. For the PRF and PBM simulations, the performance of the 2 B/FLOP CMVP approximately achieves the performance of the 4 B/FLOP CMVP. Therefore, the off-chip memory bandwidth on the CMVP needs to satisfy at least 2 B/FLOP using the cache mechanism to achieve a high scalability.

As described above, the memory bandwidth rate, B/FLOP, is a primary important factor of the high computational performance on the vector supercomputers. The high performance memory architecture is introduced to overcome to the memory wall problem in future vector supercomputers. Then, the high performance memory architecture has a potential to cover the shortage of the B/FLOP rate. In particular, the bypass and MSHR mechanisms are fundamental to the hardware mechanism of the future vector supercomputers.

Acknowledgments

I have obtained a lot of supports and kindly help from many people at Tohoku University and NEC corporation. I can hardly complete this study without them. I would like to express my sincere gratitude to all of them.

First of all, I would like to express my sincere thanks to my research supervisor, Professor Hiroaki Kobayashi, who conducted me to research fields, and has supported and encouraged me throughout years. I would like to thank Professor Kazuhiro Nakahashi and Professor Satoru Yamamoto for their thoughtful review of this dissertation and their helpful comments. I wish to express my appreciation to Associate Professor Hideaki Goto, Associate Professor Hiroyuki Takizawa and Assistant Professor Ryusuke Egawa for their valuable and helpful suggestions and comments.

I would like to express my appreciation to Research Associate Koki Okabe for his worthy suggestions and discussion. I would like to thank Professor Motoyuki Sato, Professor Akira Hasegawa, Professor Goro Masuya, Professor Terukazu Ota and Professor Kunio Sawaya for their advice about applications. I am pleased to acknowledge the assistance of Mr. Yoshiei Sato and Mr. Takashi Soga. They are member of Supercomputer team of Kobayashi Laboratory.

Special thanks go to Mr. Takaaki Shimizu, Mr. Satoru Hirokami and Mr. Kiyoshi Sugiyama of NEC Corporation for their support and encouragements. Thanks go to Ms. Yoko Isobe and Mr. Satoshi Jinguji of NEC Computertechno for their support and suggestions. I am also pleased to acknowledge the assistance of Mr. Matsuo Aoyama and Mr. Hirofumi Akiba of NEC Software Tohoku.

Finally, I express the deep appreciation to my family, who bring someone peace and comfort. I want to express my gratitude to my wife Ms. Katsumi Musa as a life partner. I also want to thank my daughter Ms. Yumiko Musa and my son Mr. Yuki Musa.

Bibliography

[1] D. Abts et al. "The Cray BlackWidow: A Highly Scalable Vector Multiprocessor". In *Proceedings of the ACM/IEEE SC2007 conference*, Reno, USA, November 2007.

[2] D. Abts, S. Scott, and D. J. Lilja. "So Many States, So Little Time: Verifying Memory Coherence in the Cray X1". In *International Parallel and Distributed Processing Symposium*, April 2003.

[3] AMD. "AMD Athlon 64X2 Dual-Core Product Data Sheet". Technical report, AMD, 2007. http://www.amd.com/us-en/Processors/ProductInformation/.

[4] K. Ariyoshi, T. Matsuzawa, and A. Hasegawa. "The key frictional parameters controlling spatial variation in the speed of postseismic slip propagation on a subduction plate boundary". *Earth and Planetary Science Letters*, 256:136–146, 2007.

[5] D. A. Bader, Y. Li, T. Li, and V. Sachdeva. "A Benchmark Suite to Evaluate High-Performance Computer Architecture on Bioinformatics Applications". http://www.bioperf.org/.

[6] D. H. Bailey. "NAS kernels". http://www.netlib.org/benchmark/nas.

[7] C. Batten, R. Krashinsky, S. Gerding, and K. Asanovic. ”Cache Refill/Access Decoupling for Vector Machines”. In *Proceedings of the 37th International Symposium on Microarchitecture*, pages 331–342, Portland, 2004.

[8] R. Biswas et al. ”NAS Experience with the Cray X1”. Technical report, NASA Ames Research Center, September 2005.

[9] D. Callahan, J. Cocke, and K. Kennedy. ”Estimating Interlock and Improving Balance for Pipelined Architectures”. *Journal of Parallel and Distributed Computing*, 5:334–358, 1994.

[10] The Earth Simulator Center. ”Outline of the Earth Simulator Project”. In *Annual Report of The Earth Simulator Center (April 2002 - March 2003)*, pages 2–7, 2003.

[11] Standard Performance Evaluation Corporation. ”SPEC’s Benchmarks and Published Results”. http://www.spec.org/benchmarks.html.

[12] NASA Advanced Supercomputing Division. ”NAS Parallel Benchmarks”. http://www.nas.nasa.gov/Software/NPB.

[13] T. H. Dunigan Jr., M. R. Fahey, J. B. White III, and P. H. Worley. ”Early Evaluation of The Cray X1”. In *Proceedings of the ACM/IEEE SC2003 conference*, Phoenix, USA, November 2003.

[14] T. H. Dunigan Jr., J. S. Vetter, J. B. White III, and P. H. Worley. ”Performance Evaluation of The Cray X1 Distributed Shared-Memory Architecture”. In *IEEE MICRO*, volume 25, pages 30–40, 2005.

[15] K. I. Farkas and N. P. Jouppi. ”Complexity/Performance Tradeoffs with Non-Blocking Loads”. In *The 21th International Symposium on Computer Architecture (ISCA)*, Chicago, USA, April 1994.

[16] R. A. Fatoohi. "Vector Performance Analysis of Three Supercomputers: Cray-2, Cray Y-MP and ETA10-Q". In *Proceedings of Supercomputing ‘ 89*, 1989.

[17] R. A. Fatoohi. "Vector Performance Analysis of The NEC SX-2". In *Proceedings of Supercomputing ‘ 90*, 1990.

[18] High-End Computing Revitalization Task Force. "Federal Plan for High-End Computing". Technical report, Executive Office of the President, Office of Science and Technology Policy, May 2004. Report of the High-End Computing Revitalization Task Force.

[19] J. W. C. Fu and J. H. Patel. "Data Prefetching in Multiprocessor Vector Cache Memories". In *18th Annual International Symposium on Computer Architecture*, pages 54–63, 1991.

[20] J. W. C. Fu and J. H. Patel. "Data Prefetching Strategies for Vector Cache Memories". In *5th International Parallel Processing Symposium*, May 1991.

[21] J. D. Gee and A. J. Smith. "Vector Processor Caches". Technical Report UCB/CSD-92-707, University of California, October 1992.

[22] J. D. Gee and A. J. Smith. "The Effectiveness of Caches for Vector Processors". In *The 8th international Conference on Supercomputing 1994*, pages 333–343, July 1994.

[23] J. K. Hennessy and D. A. Patterson. *"Computer Architecture: A Quantitative Approach, fourth edition"*. Morgan Kaufman, 2007.

[24] Cray Inc. ”Next-Generation Cray Supercomputer Will Deliver Leading-Edge Vector Processing and Integrated Scalar Capability.”. http://investors.cray.com/phoenix.zhtml?c=98390&p=irol-newsArticle&ID=876494.

[25] Intel. *”Intel Itanium Architecture Software Developer’s Manual”*, 2002.

[26] Intel. ”Intel Core2 Extreme Processor X6800 and Intel Core2 Duo Desktop Processor E6000 and E4000 Series Datasheet”. Technical report, Intel, 2008.

[27] Intel. ”Intel Pentium D Processor 800 Sequence Datasheet”. Technical report, Intel, 2008.

[28] R. E. Kessler. ”The Alpha 21264 microprocessor”. *IEEE Micro*, 19(2):24–36, 1999.

[29] K. Kitagawa, S. Tagaya, Y. Hagiwara, and Y. Kanoh. ”A Hardware Overview of SX-6 and SX-7 Supercomputer”. *NEC Research & Development*, 44:2–7, 2003.

[30] H. Kobayashi. ”Implication of Memory Performance in Vector-Parallel and Scalar-Parallel HEC”. In M. Resch et al., editors, *High Performance Computing on Vector Systems 2006*, pages 21–50. Springer-Verlag, 2006.

[31] H. Kobayashi, R. Egawa, H. Takizawa, K. Okabe, A. Musa, T. Soga, and Y Shimomura. ”First Experiences with NEC SX-9”. In M. Resch et al., editors, *High Performance Computing on Vector Systems 2008*, pages 3–11. Springer-Verlag, 2008.

[32] H. Kobayashi, A. Musa, Y. Sato, H. Takizawa, and K. Okabe. ”The potential of On-chip Memory Systems for Future Vector Architectures”. In M. Resch

et al., editors, *High Performance Computing on Vector Systems 2007*, pages 247–264. Springer-Verlag, 2007.

[33] T. Kobayashi, X. Feng, and M. Sato. ”FDTD simulation on array antenna SAR-GPR for land mine detection”. In *Proceedings of SSR2003: 1st International Symposium on Systems and Human Science*, pages 279–283, Osaka, Japan, November 2003.

[34] L. I. Kontothanassis, R. A. Sugumar, G. J. Faanes, J. E. Smith, and M. L. Scott. ”Cache Performance in Vector Supercomputers”. In *Proceedings of the 1994 conference on Supercomputing*, pages 255–264, Washington D.C., 1994.

[35] R. Krashinsky et al. ”The Vector-Thread Architecture”. In *The 31th International Symposium on Computer Architecture (ISCA)*, Munich, Germany, June 2004.

[36] D. Kroft. ”Lockup-Free Instruction Fetch/Prefetch Cache Organization”. In *The 8th International Symposium on Computer Architecture (ISCA)*, pages 81–87, PA, USA, 1981.

[37] K. S. Kunz and R. J. Luebbers. *”The Finite Difference Time Domain Method for Electromagnetics”*. CRC Press, Boca Raton, FL, 1993.

[38] P. Lammers and U Kuster. ”Recent Performance Results of the Lattice Boltzmann Method”. In M. Resch et al., editors, *High Performance Computing on Vector Systems 2006*, pages 51–59. Springer-Verlag, 2006.

[39] J. McCalpin. ”Memory bandwidth and machine balance in current high performance computers”. In *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, pages 19–25, December 1995.

[40] J. D. McCalpin. ”STREAM: Sustainable memory bandwidth in high performance computers”. http://www.cs.virginia.edu/stream.

[41] S. A. McKee. ”Reflections on the Memory Wall”. In *Proceedings of the 1st conference on Computing frontiers*, Ischia, Italy, April 2004.

[42] F.H. McMahon. ”Livermore loops”. http://www.netlib.org/benchmark/livermore.

[43] H. Meuer, J. Dongarra, E. Strohmaier, and H. Simon. ”TOP 500 Supercomputer Sites”. http://top500.org.

[44] A. Musa, Y. Sato, R. Egawa, H. Takizawa, K. Okabe, and H. Kobayashi. ”An On-Chip cache Design for Vector Processors”. In *Proceedings of the 8th MEDEA workshop on Memory performance: Dealing with Applications, systems and architecture*, pages 17–23, Romania, 2007.

[45] A. Musa, Y. Sato, R. Egawa, H. Takizawa, K. Okabe, and H. Kobayashi. ”Characteristics of an On-chip Cache on NEC SX Vector Architecture”. *Interdisciplinary Information Sciences*, In printing, 2009.

[46] A. Musa, Y. Sato, T. Soga, R. Egawa, H. Takizawa, K. Okabe, and H. Kobayashi. ”Effects of MSHR and Prefetch Mechanism on an On-Chip Cache of the Vector Architecture”. In *Proceedings of the 6th International Symposium on Parallel and Distributed Processing and Application 2008*, pages 845–858, Italy, 2006.

[47] A. Musa, H. Takizawa, K. Okabe, T. Soga, and H. Kobayashi. ”Implications of Memory Performance for Highly Efficient Supercomputing of Scientific Applications”. In *Proceedings of the 4th International Symposium on Parallel and Distributed Processing and Application 2006*, pages 845–858, Italy, 2006.

[48] M. Nakajima, H. Yanaoka, H. Yoshikawa, and T. Ota. ”Numerical Simulation of Three-Dimensional Separated Flow and Heat Transfer around Staggerd Surface-Mounted Rectangular Blocks in a Channel”. *Numerical Heat Transfer (PartA)*, 47:691–708, 2005.

[49] B. A. Nayfeh, L. Hammond, and K. Olukotun. ”Evaluation of Design Alternatives for a Multiprocessor Microprocessor”. In *The 23th International Symposium on Computer Architecture (ISCA)*, pages 67–77, PA, USA, 1996.

[50] NEC. *”FORTRAN90/SX Programmer’s Guide”*, 2006.

[51] Ministry of Education, Culture, Sports, Science, and Technology (Japanese Govemment). ”White Paper on Science and Technology 2006”, 2006.

[52] L. Oliker, A. Canning, J. Carter, and J. Shalf. ”Scientific Computations on Modern Parallel Vector System”. In *Proceedings of the ACM/IEEE SC2004 conference*, Pittsburgh, USA, November 2004.

[53] L. Oliker, A. Canning, J. Carter, J. Shalf, and D. Skinner. ”Evaluation of Cache-based Superscalar and Cacheless Vector Architectures for Scientific Computations”. In *Proceedings of the ACM/IEEE SC2003 conference*, Phoenix, USA, November 2003.

[54] L. Oliker et al. ”A Performance Evaluation of the Cray X1 for Scientific Applications”. In *VECPAR’04: 6th International Meeting on High Performance Computing for Computational Science*, Valencia, Spain, June 2004.

[55] L. Oliker et al. ”Leading Computational Methods on Scalar and Vector HEC Platforms”. In *Proceedings of the ACM/IEEE SC2005 conference*, Seattle, USA, November 2005.

[56] L. Peng, J-K. Peir, T. K. Prakash, Y-K. Chen, and D. Koppelman. ”Memory Performance and Scalability of Intel’s and AMD’s Dual-Core Processors: A Case Study”. In *26th IEEE International Performance Computing and Communications Conference (IPCCC)*, pages 55–64, New Orleans, LA, 2007.

[57] V. Santhanam, E. H. Gornish, and W. Hsu. ”Data Prefetching on the HP PA-8000”. In *The 24th International Symposium on Computer Architecture (ISCA)*, Denver, USA, June 1997.

[58] M. Sato, T. Kobayashi, Z. Zeng, G. Fang, and X. Feng. ”High Resolution GPR System for landmine detection”. In *International Conference Requirements and Technologies for the Detection, Removal and Neutralization of Landmines and UXO*, pages 548–553, Brussele, Belgium, September 2003.

[59] T. Senta, A. Takahashi, T. Kadoi, T. Itoh, S. Kawaguchi, and Y. Kishida. ”Itanium2 32-way Server System Architecture”. *NEC Research & Development*, 44:8–12, 2003.

[60] SGI. ”White paper: SGI NUMAlink Industry Leading Interconnect Technology”. Technical report, SGI Inc., 2005.

[61] SGI. ”SGI Altix 4700 Servers and Supercomputers”. Technical report, SGI Inc., 2007. http://www.sgi.com/pdfs/3867.pdf.

[62] H. Shan and E. Strohmaier. ”Performance Characteristics of the Cray X1 and Their Implications for Application Performance Tuning”. In *18th Annual International Conference on Supercomputing*, pages 175–183, Malo, France, 2004.

[63] S. Shingu et al. ”A 26.58 Tflops Global Atmospheric Simulation with the Spectral Transform Method on the Earth Simulator”. In *Proceedings of the ACM/IEEE SC2002 conference*, Baltimore, USA, November 2002.

[64] M. L. Simmons and H. J. Wasserman. ”Los Alamos National Laboratory computer benchmarking 1986”. http://lib-www.lanl.gov/cgi-bin/getfile?00319242.pdf.

[65] J. P. Singh, W. D. Weber, and A. Gupta. ”SPLASH: Stanford Parallel Applications for Shared-Memory”. *Computer Architecture News*, 20(1):5–44, 1992.

[66] S. Tagaya, M. Nishida, T. Hagiwara, T. Yanagawa, Y. Yokoya, H. Takahara, J. Stadler, and M. Galle. ”The NEC SX-8 Vector Supercomputer System”. In M. Resch et al., editors, *High Performance Computing on Vector Systems*, pages 3–24. Springer-Verlag, 2005.

[67] Y. Takagi, H. Sato, Y. Wagatsuma, K. Mizuno, and K. Sawaya. ”Study of High Gain and Broadband Antipodal Fermi Antenna with Corrugation”. In *2004 International Symposium on Antennas and Propagation*, volume 1, pages 69–72, 2004.

[68] K. Tsuboi and G. Masuya. ”Direct Numerical Simulations for Instabilities of Remixed Planar Flames”. In *The Fourth Asia-Pacific Conference on Combustion*, pages 136–139, Nanjing, China, November 2003.

[69] T. Watanabe. *”Instruction Set Architecture for a Series of Vector Processors and Their Performance Evaluations”*. PhD thesis, Tohoku University, 2005.

[70] S. P. V. Wiel and D. J. Lilja. ”When caches aren’t enough: Data prefetching techniques”. *IEEE Computer*, 30(7):23–30, 1997.

[71] S. C. Woo, M. Ohara, M. Torrie, Singh J. P., and A. Gupta. ”The SPLASH-2 Programs: Characterization and Methodological Considerations”. In *The 22th International Symposium on Computer Architecture (ISCA)*, Santa Margherita Ligure, Italy, June 1995.

[72] W. Wulf and S. McKee. ”Hitting the wall: Implications of the obvious.”. *ACM SIGArch Computer Architecture*, 23(1):20–24, March 1995.

[73] S. Yamada, T. Imamura, T. Kano, and M. Machida. ”High-Performance Computing for Exact Numerical approaches to Quantum Many-Body Problems on the Earth Simulator”. In *Proceedings of the ACM/IEEE SC2006 conference*, Tampa, USA, November 2006.

[74] H. Yanaoka, H. Yoshikawa, and T. Ota. ”Direct Numerical Simulation of Turbulent Separated Flow and Heat Transfer over a Blunt Flat Plate”. *Journal of Heat Transfer*, 125:779–787, 2003.

[75] M. Yokokawa et al. ”16.4-Tflops Direct Numerical Simulation of Turbulence by a Fourier Spectral Method on the Earth”. In *Proceedings of the ACM/IEEE SC2002 conference*, Baltimore, USA, November 2002.

Publications

Journal Papers

[1] Akihiro Musa, Yoshiei Sato, Ryusuke Egawa, Hiroyuki Takizawa, Koki Okabe, Hiroaki Kobayashi.

"The Potential of On-Chip Memory Systems for Future Vector Architectures."

Accepted for the publication in *Interdisciplinary Information Sciences (IIS)*. (Chapter 3)

Conference Papers

[1] Akihiro Musa, Hiroyuki Takizawa, Koki Okabe, Takashi Soga, Hiroaki Kobayashi.

"Implications of Memory Performance for Highly Efficient Supercomputing of Scientific Applications."

Proceedings of *The 4th International Symposium on Parallel and Distributed Processing and Application (ISPA06)*, pp. 845-858, December 2006. (Chapter 2)

[2] Hiroaki Kobayashi, Akihiro Musa, Yoshiei Sato, Hiroyuki Takizawa, Koki Okabe, Hiroaki Kobayashi.

"The Potential of On-Chip Memory Systems for Future Vector Architecture."

Proceedings of *The 7th Teraflop Workshop (High Performance Computing on Vector Systems 2007)*, pp. 249-264, April 2007. (Chapter 3)

[3] Akihiro Musa, Yoshiei Sato, Ryusuke Egawa, Hiroyuki Takizawa, Koki Okabe, Hiroaki Kobayashi.

"An On-Chip Cache Design for Vector Processors."

Proceedings of *The 8th MEDEA workshop on Memory performance: Dealing with Applications, systems and architecture*, pp. 17-23, September 2007. (Chapter 3)

[4] Akihiro Musa, Yoshiei Sato, Takashi Soga, Koki Okabe, Ryusuke Egawa, Hiroyuki Takizawa, Hiroaki Kobayashi.

"A Shared Cache for a Chip Multi Vector Processor."

Proceedings of *The 9th MEDEA workshop on Memory performance: Dealing with Applications, systems and architecture*, pp. 24-29, October 2008. (Chapter 4)

[5] Akihiro Musa, Yoshiei Sato, Takashi Soga, Koki Okabe, Ryusuke Egawa, Hiroyuki Takizawa, Hiroaki Kobayashi.

"Effects of MSHR and Prefetch Mechanisms on an On-Chip Cache of the Vector Architecture."

Proceedings of *The 6th International Symposium on Parallel and Distributed Processing and Application (ISPA08)*, pp. 335-342, December 2008. (Chapter 3)

[6] Akihiro Musa, Yoshiei Sato, Ryusuke Egawa, Hiroyuki Takizawa, Koki Okabe, Hiroaki Kobayashi.

"Early Evaluation of On-Chip Vector Caching for the NEC SX Vector Architecture."

The poster presentation at *The Premier International Conference on High Performance Computing, Networking, Storage and Analysis (SC07)*, November 2007. (Chapter 3)

[7] Akihiro Musa, Yoshiei Sato, Takashi Soga, Ryusuke Egawa, Hiroyuki Takizawa, Koki Okabe, Hiroaki Kobayashi.

"Caching an on Chip Multi Vector Processor."

The poster presentation at *The Premier International Conference on High Performance Computing, Networking, Storage and Analysis (SC08)*, November 2008. (Chapter 4)

[8] Hiroyuki Takizawa, Koki Okabe, Akihiro Musa, Takashi Soga, Yoshiaki Matsumura, Manabu Ito, Hiroaki Kobayashi.

"Performance Evaluation of SX-7 Using Real Simulation Codes."

ハイパフォーマンスコンピューティングと計算科学シンポジューム *HPCS2006*（ポスタープレゼンテーション）, 1 月 2006. (Chapter 2)

[9] 佐藤 義永, 撫佐 昭裕, 江川 隆輔, 滝沢 寛之, 岡部 公起, 小林 広明.

“ベクトルプロセッサ用キャッシュメモリの性能評価”

ハイパフォーマンスコンピューティングと計算科学シンポジューム *HPCS2008*（ポスタープレゼンテーション）, 1 月 2008. (Chapter 3)

[10] 佐藤 義永, 撫佐 昭裕, 江川 隆輔, 滝沢 寛之, 岡部 公起, 小林 広明.

“ベクトルプロセッサ用キャッシュメモリにおける MSHR の性能評価”

次世代スーパーコンピューティング・シンポジューム *2008*（ポスタープレゼンテーション）, 9 月 2008. (Chapter 3)

Technical Report

[1] 小林　広明, 岡部　公起, 撫佐 昭裕, 曽我　隆, 松村　佳昭, 伊藤　学.

“実シミュレーションコードによる大規模科学計算システムの性能評価”

SENAC,　Vol.38, No.4, pp. 39-59, 2005. (Chapter 2)