TOHOKU UNIVERSITY

Graduate School of Information Sciences

Design Space Exploration of Vector Architectures for Multimedia Applications

(マルチメディアアプリケーションのためのベクトルアーキテクチャ設計)

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

Department of Computer and Mathematical Sciences

by

Ye GAO

14 January 2014

Design Space Exploration of Vector Architectures for Multimedia Applications

Ye Gao

Abstract

People have never ceased to aspire to a higher quality of media services. In order to realize the high quality media service, next generation multimedia applications (MMAs) will process an unprecedented amount of media data in real time. For example, a next generation ultra-high definition video (8k video) needs to process 64x more data than standard definition video, and a 3D computer visual algorithm for super resolution images needs to process 25 - 250x more data than the current 3D computer visual algorithms. In order to efficiently process these media data, both a high computing performance and a high data transfer performance are required for the processors that execute the high quality MMAs.

The hardware approaches to high performance media processing are mainly classified into three categories: application-specific integrated circuits (ASICs), accelerators, and ISA extensions to general-purpose processors (GPPs). This dissertation focuses on an ISA extension to enhance the potential of GPPs on media processing because of the following two reasons. First, there are characteristics on the kernels of MMAs. A portion of MMAs kernels has a complicated control flow while some kernels require high throughput for data processing. A GPP enhanced with an ISA extension could efficiently execute two kinds of kernels. The GPP is in charge of executing kernels with complicated control flow, and the ISA extension is in charge of kernels with high throughput data processing. Second, with an increment of the varsities of MMAs, different kinds of MMAs are expected to be executed on one platform. A GPP enhanced with an ISA extension can satisfy this requirement, because its high programmability leads to a high productivity to design various applications. Therefore, a GPP with an ISA extension is the main target discussed in this dissertation.

Recently, the peak computing capability of GPP increases significantly by integrating a large number of cores and/or increasing the width of SIMD execution units. However, it is not effective in the future. With the failure of the Dennard scaling, core count scaling may be in jeopardy, and it is not easy anymore to increase the computing performance by straightforwardly increasing the number of cores or the width of SIMD execution units. It becomes more and more crucial for processor designers to pay attention to improve the efficiency of the processor in terms of computational efficiency and power efficiency.

Under these situations, this dissertation focuses on the vector architecture, which is known for its high computational efficiency and high power efficiency, and designs a vector extension to support the next generation media processing. The vector architecture is considered a potential candidate for media processing because it can make a good use of a plenty of data level parallelism (DLP) involved in MMAs. The objective of this dissertation is to enhance GPPs with a media-oriented vector ISA extension (MVPX), which can accelerate a wide range of MMAs at high power efficiency.

In order to achieve the objective, MVPX should overcome at least three issues as follows. First, the conventional vector architecture cannot execute MMAs with short vectors efficiently. Second, conventional multi-banked cache memories for vector architectures cannot achieve a high data transfer performance and low power consumption at the same time for MMAs. Third, conventional methods to find the most power-efficient configurations cost too much time to find the most power efficient configuration of the vector architecture for MMAs.

In order to overcome these problems, the design space of the vector architecture is explored to find the power efficient solution for a wide range of MMAs. First, the instruction issue policies of vector architecture have been explored, and an out-of-order vector processing mechanism (OVPM) has been proposed to improve the computational efficient of MMAs with short vectors by reducing the pipeline stalls. MMAs contain packs of independent data that can be processed with the same operation in parallel. The pack is called a vector in this chapter. The vector architecture can efficiently process the vectors in MMAs. This is because it costs cycles on data dependency checking, pipeline startup and loop control instructions only once per vector processing. By efficiently processing the vectors, the vector architecture can potentially achieve a high computing potential for MMAs. However, the conventional vector architecture cannot efficiently execute MMAs with short vectors. Most of the conventional vector architectures employ the in-order vector processing mechanism (IVPM). Although IVPM causes pipeline stalls due to keeping the program sequence, for applications with long vectors, making a good use large vector registers can hide those stalls. However, for applications with short vectors, the large vector registers cannot be efficiently utilized and thus expose the pipeline stalls and memory access latencies. The exposure leads to degradation of computational efficiency.

In order to reduce the pipeline stalls, this dissertation proposes OVPM. By using OVPM, even though a certain vector instruction is stalled, the subsequent vector instruction enables to be executed, overtaking the stalled vector instruction. In this way, the pipeline stalls can be reduced, which leads to the improvement of computational efficiency. In order to reduce the power consumption of OVPM, it shares the renaming unit and commit unit with the GPP. This is because the heavy usages of the two units by vector instructions and scalar instructions are in the different execution phases. As the evaluation results, OVPM could achieve 3.25x higher computational efficiency than IVPM only with a 7% power increase. Therefore, MVPX should adopt an out-of-order vector issue policy in order to improve the power efficiency.

Second, the cache line sizes of multi-banked cache memory have been explored, and a multi-banked cache memory for MVPX, called MVP-cache, has been proposed. MMAs require a high data transfer performance. Conventional vector architectures usually adopt a multi-banked cache memory to improve the data transfer performance. However, conventional multi-banked cache memories of vector processors cannot transfer data efficiently for MMAs. In order to adopt multi-banked cache memories to MMAs, there are at least two requirements that should be considered. There are vectors of various lengths involved in MMAs. Hence, the first requirement is that the multi-banked cache memory should efficiently transfer vectors of various lengths. Moreover, as the main execution environment of MMAs, desktop computers invest so much energy consumption in a cache memory. Hence, the other requirement is that the multi-banked cache memory should achieve a low energy consumption. However, conventional multi-banked cache memories either have a low data transfer efficiency for short vectors or cost high energy consumption on their tag arrays.

In order to achieve the high data transfer performance at a low power consumption, this dissertation proposes a multi-banked cache memory called MVP-cache. Unlike conventional multi-banked cache memories that consist of one data array and one tag array, MVP-cache associates one tag array with multiple independent data arrays of small-sized cache lines. As a result, MVP-cache consumes less static power on its tag arrays. At the same time, MVP-cache can also achieve a high efficiency on short vector data transfers because the flexibility of data transfers can be improved by controlling data transfers of each data array. As the evaluation results, MVP-cache can achieve a comparable performance with the other competitive cache organizations while the energy consumption of MVP-cache is smaller than those of the other. MVP-cache can also improve the power efficiency of MVPX.

Third, the numbers of parallel pipelines in each VFU and cache ports have been explored, and a performance-power optimization method (PPoM) has been proposed in order to find the power-efficient configuration with a short estimation time. MMAs have a various hardware requirement to MVPX to improve their performance. For some memory-intensive MMAs, they require a large number of cache ports to improve the data transfer performance. Meanwhile, the computationally-intensive MMAs require a large number of parallel vector arithmetic units to improve the computing performance. If MVPX employs a large amount of hardware to fit the maximum hardware requirement, MVPX wastes its power consumption for the MMAs which do not need so much hardware. Meanwhile, if MVPX employs a small amount of hardware, it loses performance for the MMAs which have a high hardware requirement.

In order to match the various demands of MMAs, this dissertation proposes a PPoM for MVPX in order to find the most power efficient configuration for each MMA. PPoM contains a performance estimation model of the vector architecture to estimate the execution cycles and performance bottleneck of a certain MMA. The performance estimation model is established based on the estimation and comparison of the enqueue and dequeue speed of vector instruction issue queues. According to the estimated executed performance bottleneck, the hardware resources will be increased to remove the bottleneck. In this way, the proposed PPoM can find the optimal or sub-optimal configuration at runtime. As the evaluation result, PPoM could obtain the most power-efficient configuration for seven of nine MMAs, which takes less estimation and simulation time than conventional approaches.

Enhanced with these proposed solutions, it is possible for MVPX to accelerate a wide range of MMAs with a high power efficiency.

Acknowledgements

This dissertation would not have been carried out without a lot of support of many people. The author would like to acknowledge all of them gratefully

First of all, I would like to express grateful gratitude to Professor Hiroaki Kobayashi, my supervisor, who was abundantly helpful and offered invaluable assistance. I benefited immensely from not only his support, counsel, and encouragement, but also introduction to research fields during the past eight years. I would like to thank Professor Michitaka Kameyama and Professor Takafumi Aoki for their thoughtful review of this dissertation and their helpful comments. I wish to express my gratitude to Associate Professor Hiroyuki Takizawa and Associate Professor Ryusuke Egawa for their technical and personal support and their hot encouragement.

I would like to thank Associate Professor Hideaki Goto, Associate Professor Kentaro Sano, Assistant Professor Kazuhiko Komatsu and Dr. Masayuki Sato for their valuable and helpful comments. Thanks go out to Ms. Maki Takahashi, Ms. Rikako Hasegawa, and all the members of Cyberscience Center of Tohoku University for their support to my research activities and my comfort laboratory life.

I would also like to express my appreciation to all the members of Kobayashi, Goto, Takizawa and Egawa Laboratory, Graduate School of Information Sciences, Tohoku University. Gratefully thanks go to Mr. Yusuke Funaya, Dr. Yoshiei Sato and Dr. Musa for their kind advice to my research and life Sincerely thanks go to Noki Shoji to establish the power model of the vector processor.

I express the deep appreciation to my family. I want to thank my parents for their affectionate encouragement and much support. I also want to thank my wife for emotional support.

Ye GAO

Janurary 2014

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

People have never ceased to aspire to a higher quality of media services. In order to realize the high quality media service, next generation multimedia applications (MMAs) will process an unprecedented amount of media data in real time. For example, a next generation ultra-high definition video (8k video) needs to process 64x more data than standard definition video [1] [2], and a 3D computer visual algorithm for super resolution images [3] needs to process 25 - 250x more data than the current 3D computer visual algorithms. In order to efficiently process these media data, both a high computing performance and a high data transfer performance are required for the media processors that execute the high quality MMAs.

In addition to the performance requirements from MMAs, the media processors should also achieve a high power efficiency. Desktop computers are considered as the main computer systems to execute the high quality MMAs. Since the size of desktop computers cannot be significantly varied, the performance of cooling units of the desktop computers is limited. Hence, in order to avoid the overheat of the processors, their power budgets, in terms of thermal design power (TDP), are also restricted. Consequently, it is required for the processors to achieve a power-efficient execution for MMAs.

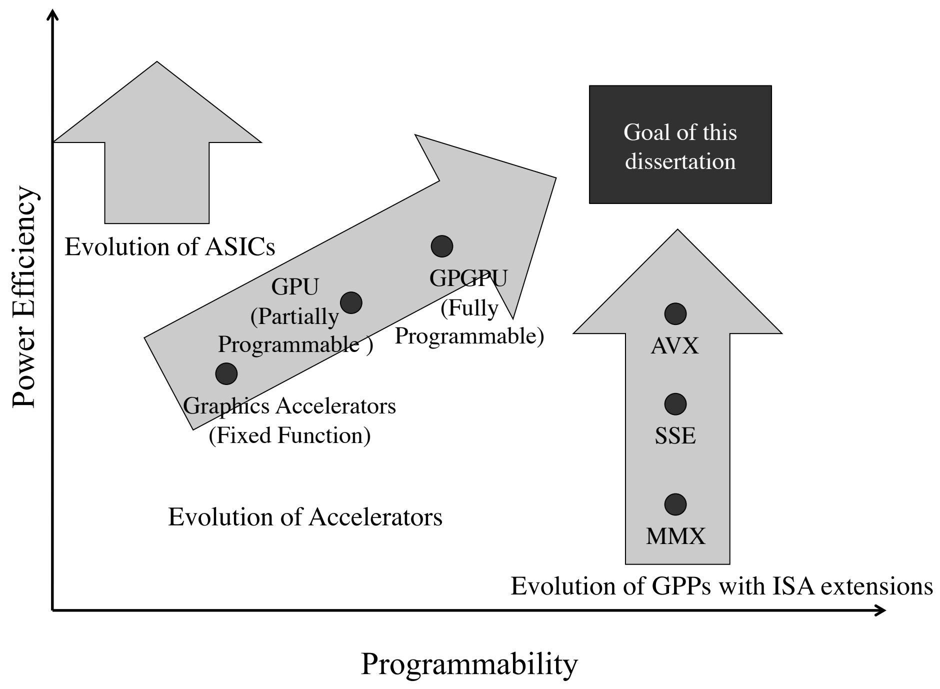

Goal of this dissertation
Evolution of ASICs
GPU (Partially Programmable)
GPGPU (Fully Programmable)
Graphics Accelerators (Fixed Function)
Evolution of Accelerators
AVX
SSE
MMX
Evolution of GPPs with ISA extensions
Power Efficiency
Programmability

Figure 1.1: *Goal of this Dissertation.*

The processors to execute MMAs are also required to have a high programmability. In the future, more and more MMAs will be executed on one platform. For example, people play 3D games, edit videos and pictures and listen to music on their desktop computers. This requires that a processor to execute MMAs should have a high programmability. Many researchers have described the benefits of high programmability [4] [5] [6]. The benefits include time-to-market by parallel developing hardware and software, easiness to fix software and prototyping bugs and variety of executable applications. Consequently, the processors to execute MMAs should have a high programmability as well as high power efficiency.

The hardware approaches for high performance media processing are mainly classified into three categories: application-specific integrated circuits (ASICs) [7], accelerators [8] [9] [10], and SIMD ISA extensions to general-purpose processors (GPPs) [11] [12]. Figure 1.1 shows the evolutions of these three approaches. ASICs aspire to extremely high power efficiency for one MMA. However, the lack of hardware adaptivity to various MMAs makes the ASIC approach mismatch the high programmability requirement.

The accelerators such as graphics processing units (GPUs) could achieve high power efficiency. Since limited programmability of accelerators hinders their uses on various applications, they evolve to improving their power efficiency and programmability in order to accelerate to a variety of applications. However, the accelerator approach lacks the support to legacy MMAs. The abandon of existing MMAs is not a wisdom decision while it cost too much human resource and time to redesign the legacy programs for the new accelerators [13].

The SIMD ISA extensions of GPPs could achieve a high programmability [14]. Compared to the other two approaches, SIMD ISA extensions have two advantages. One is that a GPP with an SIMD ISA extension enables to execute legacy MMAs without the redesign of the applications [15]. The other one is that a GPP with an ISA extension could efficiently execute vectorizable and sequential kernels, both of which are existed in MMAs [16]. Therefore, this dissertation focuses on the ISA extension approach to the performance improvement of MMAs.

Recently, the peak computing performance of the SIMD extensions increase significantly by increasing the width of SIMD execution units. For example, Intel's SIMD extensions MMX [17], SSE [18], AVX [12], AVX2 [19] adopt SIMD width of 64-bit, 128-bit, 256-bit and 512-bit, respectively. However, with the failure of Dennard scaling [20], straightforwardly increasing the width of SIMD execution units cannot be improve the power efficiency any more [21] [22]. It is necessary to consider an efficient way to use the SIMD execution units. One approach is to deeply pipeline the SIMD execution units and use a large amount data to fill up the pipelines.

Indeed, there is such an architecture. That is the vector architecture [23]. The vector architecture is usually used in high performance computing systems, or so called supercomputers [24] [25] [26]. The applications in the high performance computing domains employ packs of independent data that can be processed with the same operation in parallel. The packs of data are called vector, and the number of data in a pack is called vector length in this dissertation. Since the applications in high performance computing domains usually have a long vector in their algorithms [27] [28], the vector architecture usually employs large vector registers to store the long vectors and fill up the deeply pipelined vector functional units (VFUs) and vector load and store unit (VLSU). In this way, the stalls due to data dependency and latencies of VFUs and VLSU enable to be hide, thus leading to a high efficient use of VFUs and VLSU. Since MMAs also contains a mass of data parallel processing, the vector architecture is also a promoting approach to improving the power efficiency of ISA extensions for MMAs.

1.2 Objective of the Dissertation

The objective of this dissertation is to establish a design methodology for microprocessors that accelerate a wide range of MMAs at high power efficiency. To this end, this dissertation proposes a media-oriented vector ISA extension (MVPX) to enhance GPPs. In order to achieve a power-efficient execution by using MVPX, there are at least three issues as follows.

First, MMAs with short vectors cannot be efficiently executed on the conventional vector architectures, which adopt an in-order issue policy. In the case of short vector processing, the pipeline stalls due to the in-order issue policy and memory access latencies are easily to be exposed. The exposure causes the degradation of computational efficiency, leading to low power-efficient execution for MMAs with short vectors.

Second, conventional multi-banked cache memories for vector architectures cannot achieve a high data transfer performance and low power consumption at the same time for MMAs. In order to improve the data transfer performance, conventional vector architectures employ a multi-banked cache memory. For a multi-banked cache memory, cache line size is a key factor to affect the data transfer performance and power consumption. A multi-banked cache memory with small sized cache lines (MBC-S) could achieve a high data transfer performance for various MMAs, while a multi-banked cache memory with large sized cache lines (MBC-L) could achieve a low power consumption on its tag arrays. However, MBC-S consumes a large power consumption on tag arrays, and MBC-L has an inefficiency on short vector processing. Therefore, conventional multi-banked cache memories cannot achieve a high data transfer performance at a low power consumption.

Third, conventional vector architectures cannot achieve a power-efficient execution for all MMAs. Although different MMAs have various hardware requirements, conventional vector architectures fix the amounts of hardware resources. Therefore, in order to improve the power efficiency of MVPX, it is necessary to reconfigure the amounts of hardware resources at runtime to obtain the most power-efficient configuration of each

MMA. However, when applying conventional parameter searching methods to the vector architecture designed for MMAs, they costs too long time to find the most power-efficient configuration.

In order to resolve these problems, this dissertation explores the design space of vector architectures. Design space exploration means the process of analyzing several ”functionally equivalent” implementation alternatives to identify an optimal solution [29]. The design spaces discussed in this dissertation include four important factors. They are issue policies of the vector architectures, cache line size of multi-banked cache memory, the number of parallel pipelines in each VFU, and the number of cache ports. The issue policies of vector architecture should be explored because they could potentially affect the power efficiency of short vector processing. In addition, the cache line sizes are explored in order to improve power-efficient data transfer for all MMAs. At last, the numbers parallel pipelines and cache ports should be explored in order to find the proper amounts of hardware resources to realize power-efficient execution for any MMAs. These four factors form two 3D design spaces as shown in Figure 1.2. By exploring the design space of vector architectures, it is possible to clarify what kind of approaches is effect to improve the power efficiency of MVPX.

The design space exploration of vector architectures is divided into three stages, each of which proposes an approach to the power efficiency improvement of MVPX. In the first stage, the instruction issue policies of vector architecture will be explored, and an out-of-order vector processing mechanism (OVPM) is proposed to improve the computational efficient by reducing the pipeline stalls. By using OVPM, even though a certain vector instruction is stalled, the subsequent vector instruction enables to be executed, overtaking the stalled vector instruction. In this way, the pipeline stalls can be reduced, which leads to the improvement of power efficiency of MVPX.

In the second stage, the cache line sizes of multi-banked cache memory will be explored, and a multi-banked cache memory for MVPX, called MVP-cache, will be proposed. MVP-cache associates single tag array with multiple data array. It is possible to achieve a high data transfer performance for short vectors and low power consumption on tag array. This is because the relationship between the size of cache lines and the number of tags is decoupled. As a result, MVP-cache could improve data transfer performance of short vectors by adopting a small cache line size. MVP-cache could also reduce the power consumption by increase the number of data arrays that associated with the same tag array. In this way, MVP-cache could achieve a power-efficient data transfer for MMAs.

In the third stage, the numbers of parallel pipelines in each VFU and cache ports will be explored, and a performance-power optimization method (PPoM) will be proposed in order to find the most-power efficient configuration for each MMA at the runtime. PPoM contains an analytical model established for MVPX to estimate the execution cycles and performance bottleneck of a certain MMA. The numbers of parallel pipelines and cache ports will be searched base on the greedy algorithm. By using PPoM, the most power-efficient configuration can be find at runtime.

1.3 Organization of the Dissertation

The rest of the dissertation is organized as follows. Chapter 2 proposes OVPM in order to improve the potential of vector architecture on short vector processing. Chapter 3 presents a multi-banked cache memory that associates one tag array with multiple data arrays. Such a multi-banked cache memory reduces the power consumption on tag arrays and improves the data transfer performance on short vectors. Chapter 4 describes PPoM for MVPX. By this method, it is possible to find the power-optimal or sub-optimal configuration at runtime. Chapter 5 concludes the dissertation and describes the future work of this dissertation.

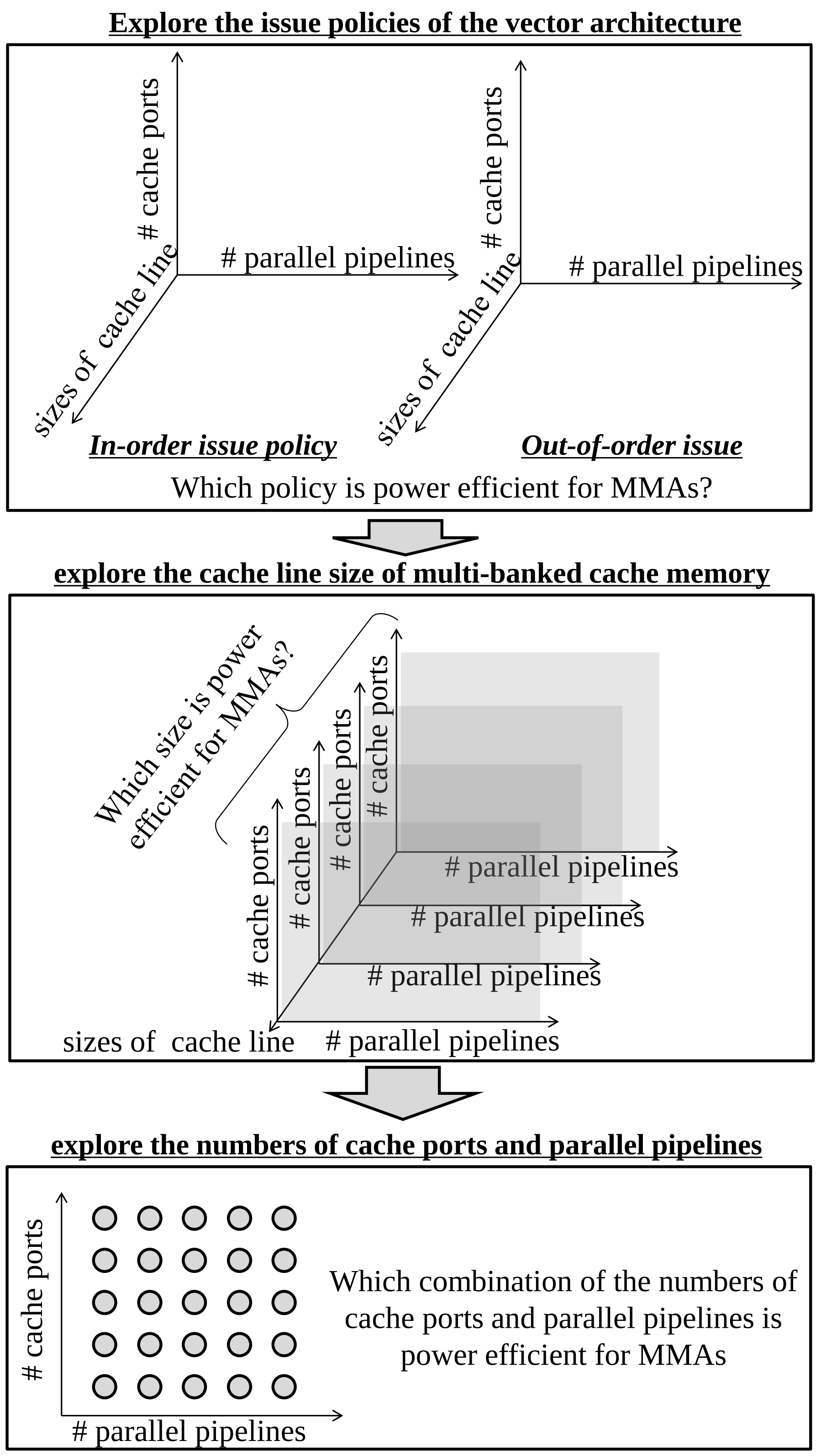

Explore the issue policies of the vector architecture
# cache ports
# parallel pipelines
sizes of cache line
In-order issue policy
# cache ports
# parallel pipelines
sizes of cache line
Out-of-order issue
Which policy is power efficient for MMAs?
explore the cache line size of multi-banked cache memory
Which size is power efficient for MMAs?
# cache ports
# cache ports
# cache ports
# cache ports
# parallel pipelines
# parallel pipelines
# parallel pipelines
# parallel pipelines
sizes of cache line
explore the numbers of cache ports and parallel pipelines
# cache ports
# parallel pipelines
Which combination of the numbers of cache ports and parallel pipelines is power efficient for MMAs

Figure 1.2: *Design Space Exploration of the Vector Architecture.*

Chapter 2

An Out-of-Order Vector Processing Mechanism

2.1 Introduction

MMAs contain packs of independent data that can be processed with the same operation in parallel [30]. The pack is called a *vector* in this chapter. The vector architecture can efficiently process the vectors in MMAs. This is because it costs cycles on data dependency checking, pipeline startup and loop control instructions only once per vector processing [15]. By efficiently processing the vectors, the vector architecture can potentially achieve a high computing potential for MMAs.

Although conventional vector architectures are good at processing long vectors, MMAs with short vectors cannot be efficiently executed on them. If the vector length is short, the pipeline latencies and memory access latencies are easy to be exposed. Such exposures make VFUs and memory buses underutilized, and thus, cause the degradation of computational efficiency. The objective of this chapter is to improve the computational efficiency of MVPX on MMAs with short vectors. In order to achieve the objective, the instruction issue policy of MVPX is explored.

Most of the conventional vector architectures employ the in-order vector processing mechanism (IVPM). Although IVPM causes pipeline stalls due to keeping the program sequence, for applications with long vectors, making a good use large vector registers can hide those stalls. However, for applications with short vectors, the large vector registers cannot be efficiently utilized and thus expose the pipeline stalls and memory access latencies. The exposure leads to degradation of computational efficiency. Therefore, OVPM is proposed in this chapter in order to improve the computational efficiency of MMAs with short vectors.

The rest of this chapter is organized as follows. Section 2.2 confirms the data level parallelism involved in MMAs and analyzes their vector lengths. Section 2.3 clarifies the inefficiency of IVPM. Section 2.4 proposes the architecture of OVPM. Section 2.5 evaluates the performance and power consumption of OVPM, and Section 2.6 concludes this chapter.

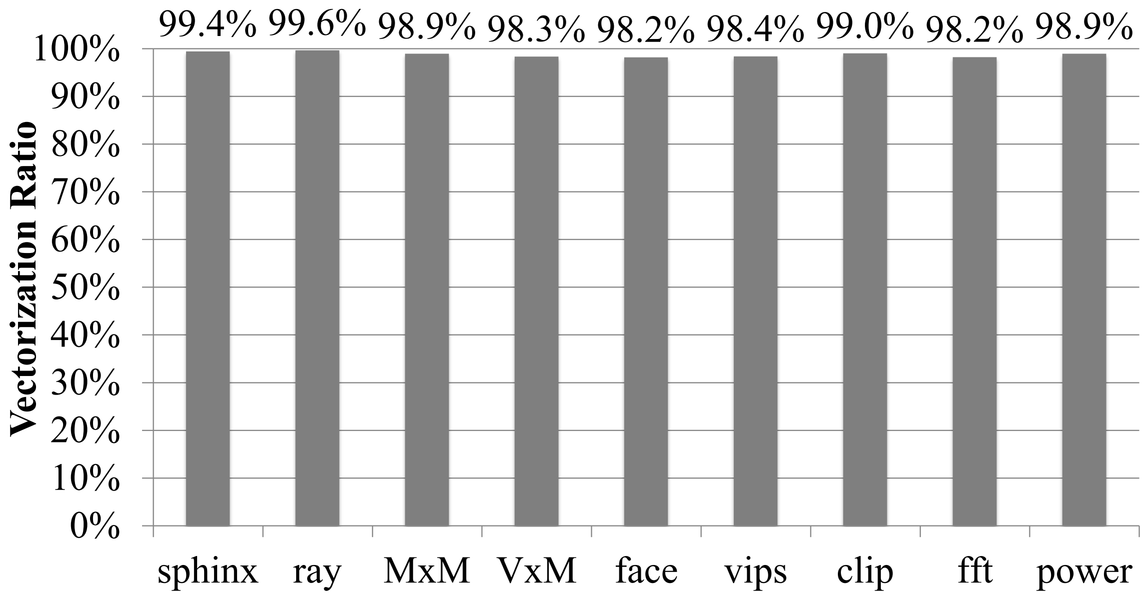


Figure 2.1: *Vectorization Ratio.*

2.2 DLP Potential of MMAs

In this section, the potential of DLP in MMAs is discussed based on their vectorization ratios and vector lengths. Then, the problems of conventional vector architectures on executing MMAs are clarified.

2.2.1 Vectorization Ratio

In this subsection, the vectorization ratio is used to verify that vector architectures can potentially achieve high performances on MMAs. As the preliminary evaluation, this dissertation uses nine MMAs to evaluate the potential of DLP in MMAs. The detail of the benchmark programs describes in Table 2.3.

The vectorization ratio is defined as the ratio of the vectorized calculation time to the overall calculation time [24]. According to Amdahl's law [31], the performance speedup can be expressed as Equation (2.1).

$$Speedup = \frac{1}{(1-P)+\frac{P}{N}} \ , \tag{2.1}$$

where P indicates the proportion of a program that can be parallelized, and N is the speedup ratio of the parallel part.

In the case of vector processing, P is the vectorization ratio. Therefore, a high vectorization ratio means that the application has a potential to achieve a high performance by using vector architectures. According to Amdahl's law, there is no significant speedup by using the vector architectures, unless the vectorization ratio is very high, e.g. 90% or more. Furthermore, a high vectorization ratio also indicates that a compiler has been able to exploit abundant DLP. A high vectorization ratio is also crucial to take advantage of the computational performance of vector architectures. Therefore, there are software tuning techniques, such as loop sequence changing and inline expansion, to increase the vectorization ratio.

Figure 2.1 presents the vectorization ratios of some typical multimedia application kernels [32] [33] [3]. After some simple software tunings, the vectorization ratios of all the benchmarks exceed 98%. Besides, the vectorization ratio becomes higher than 99% in three of the nine benchmarks. These results show that MMAs contain massive DLP, which can be easily exploited by a compiler with simple software tunings, and thus have a potential to achieve a high performance by vector processing.

2.2.2 Vector Length

Vector Length of MMAs

A vector length in this chapter means the number of elements simultaneously operated by vector architectures. The performance of MMAs on the vector architecture also depends on their vector lengths, because the overheads for the execution of scalar instructions and the vector pipeline latency are required for starting up a vector pipeline. Thus, a long vector length is needed to hide these overheads. Generally speaking, the vector length

Table 2.1: *Vector Length.*

Benchmarks		sphinx	face	ray	vips	M x M	V x M	clip	fft	power
	MVL32	32	28.83	31.76	26.33	31.25	31.25	32	32	32
Average	MVL64	64	57.67	63.53	39.5	62.5	62.5	64	32	32
Vector	MVL128	128	86.5	120	79	125	125	64	32	32
Length	MVL256	256	173	216	79	250	250	64	32	32
	MVL512	512	173	360	79	500	500	64	32	32
Vector Length		4096	173	1080	79	1000	1000	64	32	32

should be longer than 20 [28] in order to hide the start-up overhead. Table 2.1 shows the vector length of benchmark programs. Most of the applications have a long vector, meaning that the MMAs have a potential to hide the vector pipeline start-up overheads.

Average Vector Length

The average vector length (AVL) is the average number of data parallel operations per vector instruction and is defined by

$$AVL = \frac{VL}{V_{num}} \ , \tag{2.2}$$

where V_{num} is the number of vector instructions and VL denotes the vector length. V_{num} is calculated as follows:

$$V_{num} = \left\lceil \frac{VL}{MVL} \right\rceil \ , \tag{2.3}$$

where MVL is the maximum number of elements that can be processed by one vector instruction. If AVL is close to MVL, which implies the vector hardware resources are efficiently utilized.

Table 2.1 also describes AVL in different values of MVL. For MxM, *ray* and VxM, AVL is always close to MVL. In *face* and *vips*, AVL becomes obviously shorter than MVL

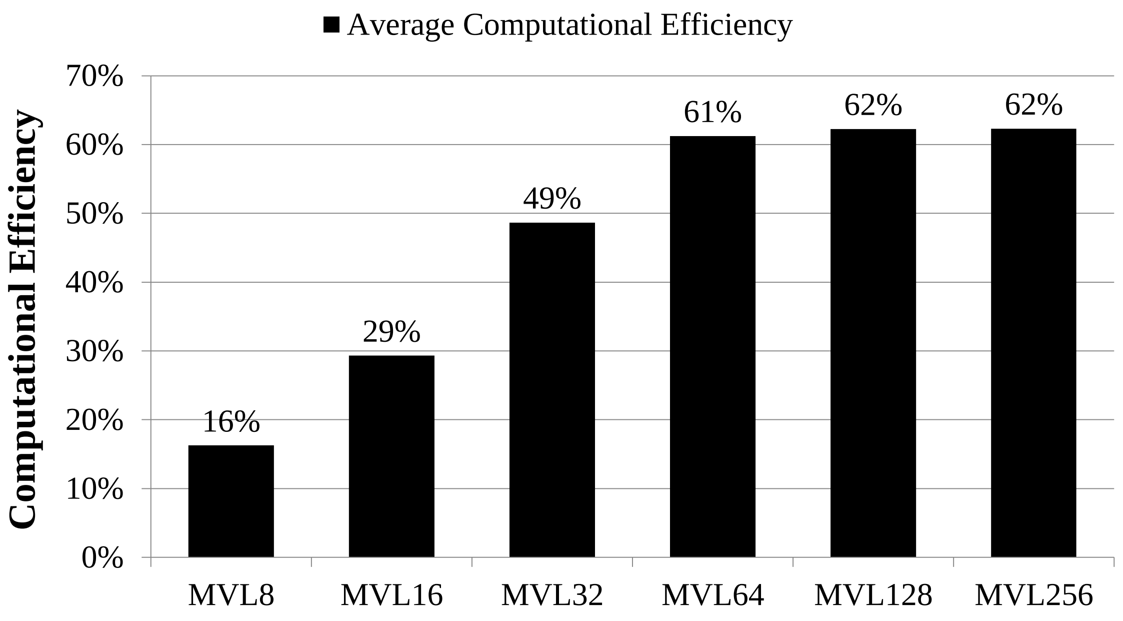


Figure 2.2: *Effect of MVL to Computational Efficiency and AVL.*

if MVL is longer than the vector lengths of those programs. Although vector length of raytracer is slightly longer than that of *M*x*M*, its V_{num} of *ray* is larger than *M*x*M*'s. As a result, AVL of *ray* is smaller than that of *M*x*M*.

In order to investigate the effect of changing the vector length, the performance of SIMD architectures is evaluated in the difference cases of MVL by using nine media applications listed in Table 2.3. In the evaluation, the number of parallel degree is set at eight, which refers to the recent SIMD extensions and assumes that there are no pipeline stalls and sufficient data transmission capability with a view to purely observing the influence of AVL. The average computational efficiency and AVL of all benchmark programs with the change of MVL are shown in Figure 2.2. $MVL8$ means that at most eight elements can be processed by one instruction, which is the configuration of modern SIMD extensions, such as Intel's AVX [12], when processing floating point elements. The configurations of MVL that is longer than 64 represent current vector machines. For example, Cray X1 [26] and NEC SX-9 [25] can process at most 64 and 256 elements respectively by one instruction.

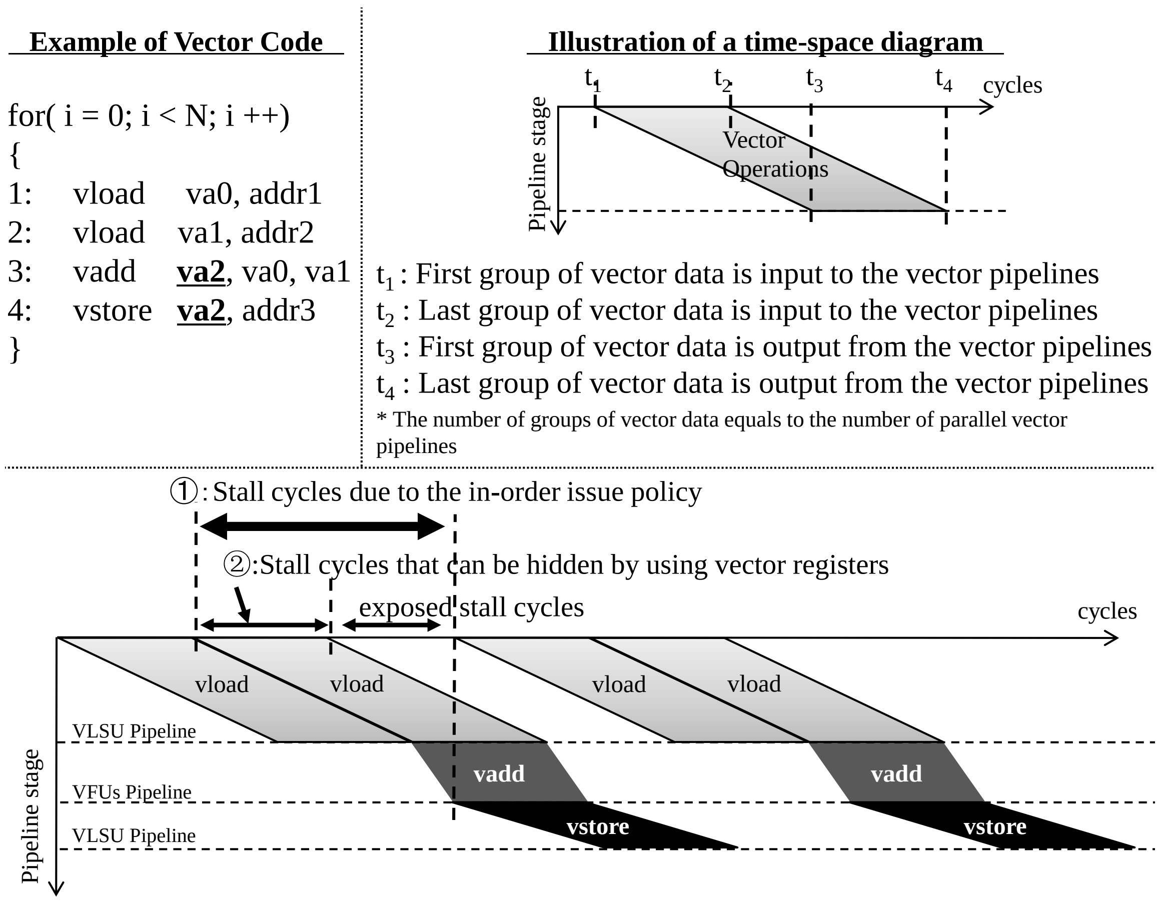
The evaluation result shows, as a popular approach to accelerate MMAs, existing

SIMD extensions have a limitation to process a large amount of data. Even the latest SIMD extension, AVX, can only process eight double precision floating data by one instruction. Such a short MVL is not enough for the next generation MMAs. Different from SIMD extensions, the MVL of vector architectures is much longer than that of SIMD extensions. AVL enables to become longer with the increment of MVL, according to Equations (2.2) and (2.3). Longer AVL enables to keep VFUs pipelines and VLSU busy. Thus, vector architectures can tolerate long memory access latencies and more efficiently use VFUs.

Moreover, compared with SIMD extensions, vector architectures also support more kinds of memory access patterns including stride memory access and indexed memory access and a masked vector processing mechanism. In addition, vector architectures have a vector length register that specifies the vector length for the current operation [15]. The flexible vector length makes vector architectures more easily accommodate programs that naturally have shorter vectors than the maximum vector length supported by hardware. Therefore, vector architectures have a high potential to improve the power efficiency of MMAs.

2.3 Problems of IVPM

Vector architectures can tolerate the long access latency naturally by employing large vector register files and a deeply pipelined VLSU [15]. Since one vector instruction processes a large number of data to fill in the VLSU pipeline stages, the memory access latency of these data can be hidden in a pipeline fashion. This method is effective for MMAs with long vectors because there are sufficient data to fill up the VLSU pipeline stages. However, when a MMA with short vectors is executed, the pipeline stalls would frequently occur, and thus expose the long access latency. This problem severely degrades the performance especially if the vector architecture adopts only in-order vector processing mechanism (IVPM).


Example of Vector Code

```
for( i = 0; i < N; i ++)
{
1:    vload    va0, addr1
2:    vload    va1, addr2
3:    vadd     va2, va0, va1
4:    vstore   va2, addr3
}
```

Illustration of a time-space diagram

t_1 : First group of vector data is input to the vector pipelines
t_2 : Last group of vector data is input to the vector pipelines
t_3 : First group of vector data is output from the vector pipelines
t_4 : Last group of vector data is output from the vector pipelines
* The number of groups of vector data equals to the number of parallel vector pipelines

Figure 2.3: *Inefficiency of IVPM.*

To discuss the inefficiency of IVPM, the behavior of a simple vector program is shown in Figure 2.3 by using a time-space diagram. In the time-space diagram, each parallelogram shows a vector operation. Variables t_1 to t_4 denote four special moments of vector operations. The detailed description of them is shown in Figure 2.3.

In Figure 2.3, a vector store instruction in the first iteration is stalled in the issue stage. It blocks the subsequent instructions, such as the vector load instruction in the second iteration, from being issued. The same stall will occur in each iteration, resulting in inefficient use of memory bandwidth and a large performance loss. The behavior of IVPM is summarized as follows.

1. *The first and second instructions*: Vector load instructions, *vload*, are decoded and executed by generating the memory addresses at the address generation unit (AGU).

2. *The third instruction*: A vector addition instruction, *vadd*, is decoded and then stalled in the issue stage until $va0$ and $va1$ are ready.

3. *The fourth instruction*: A vector store instruction *vstore* is stalled in the decode stage unless the previous instruction *vadd* is issued, because of the in-order execution policy for the conventional vector architectures. Then, it is stalled in the issue stage, waiting for the results of the *vadd* due to the input data dependency.

4. *The fifth instruction*: The first instruction of the second iteration (*vload*) is stalled in the decode stage, unless the last instruction (*vstore*) of the first iteration is issued, no matter whether its operands are ready or not. This makes the first *vload* of the second iteration expose the memory latency. The exposure of memory latency occurs in each iteration. The subsequent instructions show the same behavior as these five instructions.

The exposed stall cycles due to IVPM can be expressed as the following equation.

$$stall_{exposed} = stall_{total} - stall_{hide}, \tag{2.4}$$

where $stall_{total}$ denotes the total stall cycles due to IVPM, and $stall_{hide}$ represents the stall cycles that can be hidden by using vector register files. In Figure 2.3, $stall_{total}$ and $stall_{hide}$ are denoted by ① and ②, respectively. From Figure 2.3, it is obvious that $stall_{total}$ is the pipeline latency of VFUs and VLSU before the stalled instruction. Hence,

$$stall_{total} = \sum Latency_{pipeline}. \tag{2.5}$$

On the other hand, $stall_{hide}$ is the chime of a vector instruction [23], which can be expressed as follows.

$$stall_{hide} = Chime_{vector} = \frac{min(LL, MVL)}{N_{parallel}}, \tag{2.6}$$

where $N_{parallel}$ denotes the parallel degree of pipelined VFUs. Using Equations (2.5) and

(2.6), Equation (2.4) can be rewritten as the following expression.

$$stall_{exposed} = \sum Latency_{pipeline} - \frac{min(LL, MVL)}{N_{parallel}}, \tag{2.7}$$

where LL stands for a loop length of the program and MVL means the maximum number of vector elements that can be processed by one vector instruction. Equation (2.6) shows that the effect of hiding stalls by using vector register files is positively related to the vector length, and Equation (2.7) means that the shorter vector length is, the more stalls will be exposed. That is why IVPM is inefficient on executing MMAs with short vectors.

2.3.1 Related Work

So far, many approaches have been proposed for vector architectures to hide the long access latency. Kozyrakis et al. have proposed delayed vector pipelines [34]. The delayed vector pipelines enable a vector arithmetic instruction to be issued right after its previous instruction and delay its actual execution for the worst case memory access latency. In this way, the issued arithmetic instruction would not block the successive vector memory instruction, thus hiding the access latencies. However, recent vector architectures have cache memories, and the delayed vector pipelines cannot be applied to vector architectures with a cache memory because the postponed execution for the worst case memory access latency eliminates the effects of cache hits.

Decoupled executions are also proposed for vector architectures [35] [36]. The decoupled architectures separate a vector instruction sequence into multiple streams. By simultaneously executing the multiple streams, a memory access stream could keep issuing no matter whether the other streams are blocked or not. The successively issued vector memory instructions enable to transfer data by only paying for access latency once. However, instructions in each stream should still be issued strictly following the program sequence, where it is possible to cause pipeline stalls and expose the access latency. Therefore, a new approach should be considered to efficiently avoid pipelines stalls and tolerate long access latency.

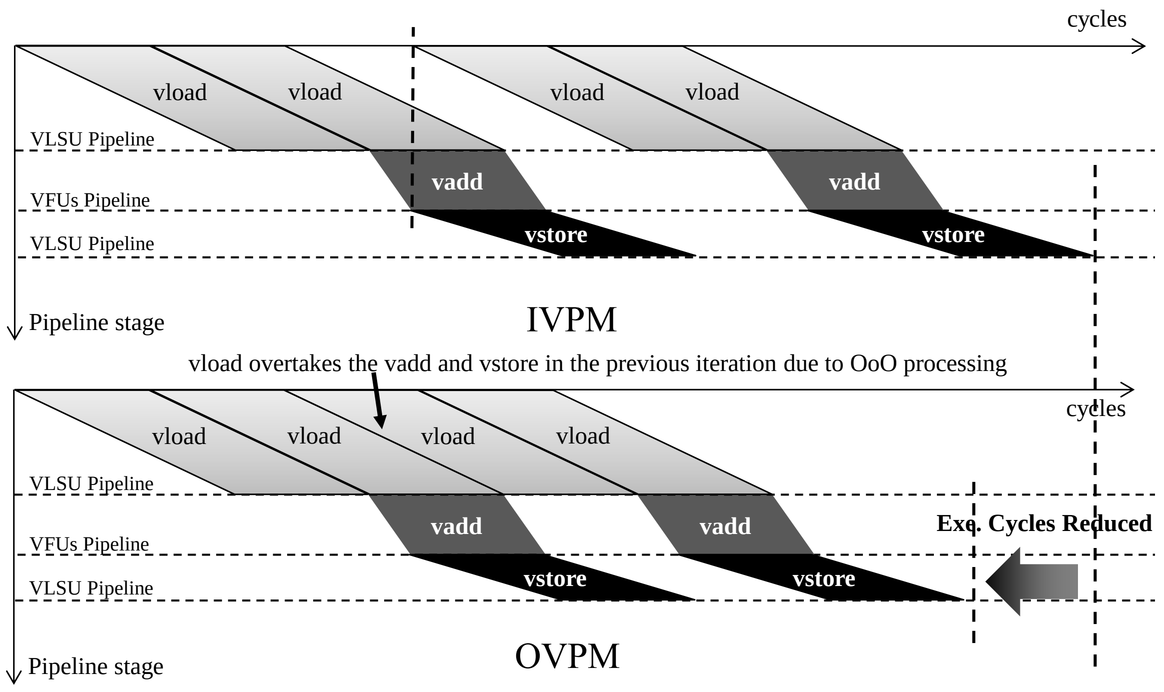


Figure 2.4: *Potential of OVPM.*

2.4 OVPM

MMAs with short vectors cannot be executed efficiently due to the exposure of pipeline stalls. An out-of-order execution is one of the most effective ways to eliminate pipeline stalls and hide memory access latencies. Figure 2.4 compares IVPM and OVPM using time-space diagrams. In OVPM, an instruction is issued as soon as its operands are ready, even though its precedent instructions have not been issued yet. Therefore, the issue and execution of instruction *vload* in the second iteration can overtake those of the *vadd* and *vstore* instructions in the first iteration. In this way, even MMAs with short vectors can also efficiently use deeply pipelined VLSU to hide the access latencies. One of the biggest problems to apply an out-of-order execution to a vector architecture is its high power consumption. However, with the development of process technologies, the hardware becomes cheaper and cheaper than before. The control units for reducing pipeline stalls do not consume so much power as before. Moreover, OVPM can potentially improve the utilization of memory bandwidth and shorten the execution time of MMAs. Therefore,

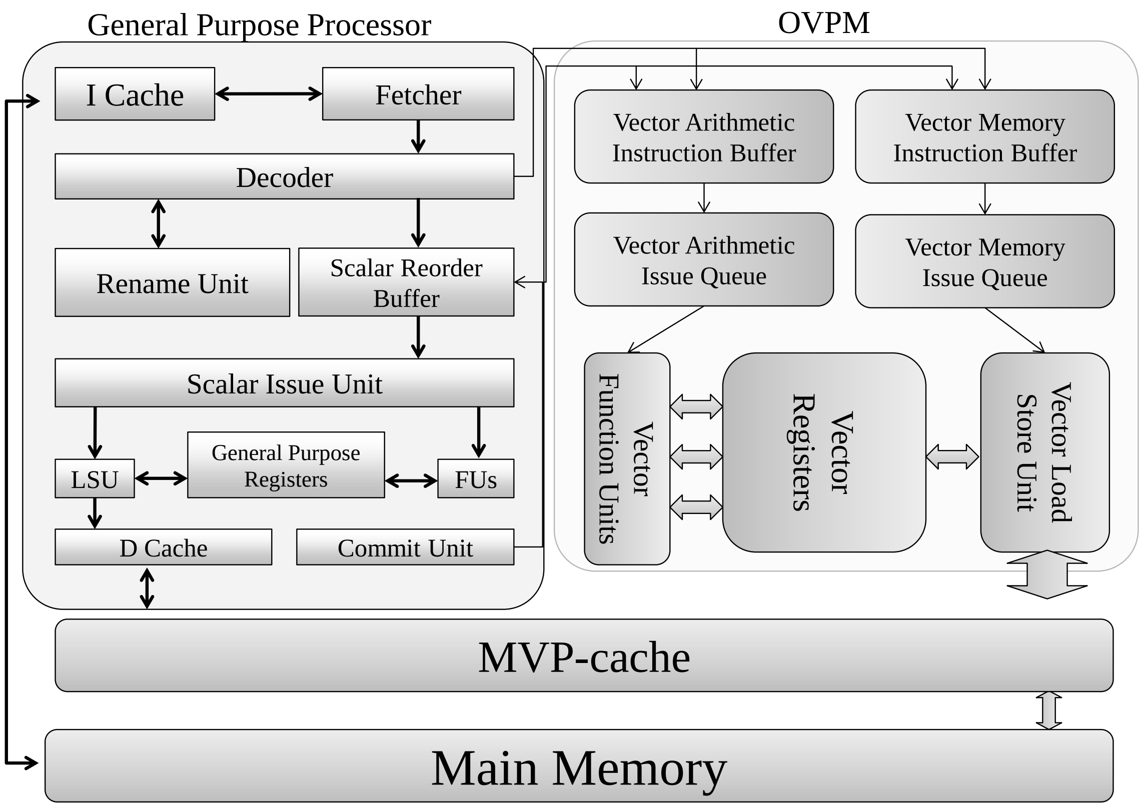


Figure 2.5: *Block Diagram of OVPM.*

from the viewpoint of energy consumption, it is worth to adopt OVPM.

2.4.1 Overview of MVPX

Figure 2.5 illustrates the block diagram of MVPX. White blocks represent the components of GPP, while gray blocks presents the components of MVPX. Additionally, two instruction buffers are installed to schedule vector instructions and realize the out-of-order issues. The vector execution unit consists of parallelized vector lanes. In each lane, a deeply pipelined functional unit and a vector instruction chaining mechanism are used to improve the computational efficiency. VLSU manages the data transfer between the memory sub-system and vector register files. It mainly contains the parallelized address generation units for vector memory instructions and the vector load/store queues, which keep the information for memory accesses.

2.4.2 Out-of-Order Vector Processing Mechanism

Three units are necessary to realize OVPM. They are a renaming unit, a reorder unit and a commit unit. The renaming unit is used to remove the name dependencies and check the true data dependence among vector instructions. The commit unit is used to manage the retirement of vector register files in the register alias table. Only after all preceding instructions have been committed, the vector register file used in the vector instruction is returned to the free list. Since the heavy usages of the rename unit and commit unit by scalar instructions and vector instruction are in the different execution phase of MMAs, OVPM and the GPP could share the rename unit and commit unit without performance degradation. In this way, the power consumption of OVPM could be reduced.

In order to issue vector instructions in an out-of-order fashion, OVPM employs two new instruction buffers: a vector arithmetic instruction buffer (VAIB) and a vector memory instruction buffer(VMIB), in the vector datapath as the reorder buffers. Vector instructions in the two buffers would be traversed to detect the states of the operands. When operands of a vector instruction have been prepared, it could be issued no matter whether there are its precedent vector instructions in the program sequence or not.

There are two approaches to out-of-order execution. One approach is implemented by using reservation stations, such as Intel's P6 [37]. The other approaches is implemented by using the physical register files, such as Alpha21264 [38]. This chapter adopts the latter one because it is more power efficient than the former one for vector architectures. The re-order buffers of the reservation station based approach need to hold all the input/output vector data. Hence, those buffers need to hold multiple copies of the same vector data in different entries if the input operands of the two vector instructions are the same. Holding the duplicated data in the re-order buffers consumes a lot of power, especially for long vector data. Meanwhile, the re-order buffers of the physical register file based approach only hold the pointers to input/output physical vector register files to store the vector data. In this way, the physical register file based approach does not need any copy of vector data. As a result, the physical register file based approach consumes less power in re-order buffers than the reservation station based approach.

The behavior of vector instructions in the datapath is shown as follows, referring to Figure 2.5. Firstly, the instruction fetcher of the scalar unit takes instructions from the instruction cache memory and sends them to the decoder. Then, the decoder detects the class of the instructions. If a decoded instruction is a vector arithmetic instruction, the decoded information of the instruction is stored in VAIB. If a decoded instruction is a vector memory access instruction, its decoded information is stored in VMIB. In the instruction issue stage, an instruction in VAIB is delivered to the vector arithmetic issue queue (VAIQ) as long as the input/output operands of the first instruction stored in VAIB are ready for execution. The vector instructions in VAIQ wait for the availability of VFUs. If they have available access ports, instructions are issued from VAIQ until the number of issued instructions reaches the issue width. A memory access instruction is divided into two sub-operations, an address generation operation and a memory access operation. In the address generation operation, the instruction is also forwarded to VAIQ to wait for the availability of AGU. In the memory access operation, the instructions in VMIB are delivered to the vector memory instruction issue queue (VMIQ) as long as the operands, such as addresses and output/input registers, of the memory access instructions are ready. The memory accesses are performed by VLSU. Finally, the results from the execution stage and memory access stage update the corresponding registers.

2.5 Performance Evaluations

2.5.1 Experimental Methodology

Table 2.2: *Configuration of MVPX and Memory Sub-System.*

Parameters	**Value**
Process Technology	32 nm
Processor Frequency	1GHz
Vector ALU Pipeline Latency	10 cycles
Vector Multiplier Pipeline Latency	15 cycles
Vector Division Pipeline Latency	20 cycles
Number of Vector Lanes	8
Number of Architectural Vector Registers	16
Number of Physical Vector Registers	96
Entries per Vector Register	128 entries
The size of VMIB and VAIB	128 entries
Number of Cache Ports	16
Cache Capacity	2 MB
Cache Bandwidth	128 bytes/cycle
Cache Line Size of MVP-cache	8 bytes
MVP-cache Access Latency	30 cycles

A simulator of vector extension with OVPM is developped based on the SimpleScalar toolset [39] to investigate its performance on media workloads. McPAT0.8 [40] is used to simulate OVPM power consumption. The specifications of OVPM are listed in Table 2.2. The GPP is modeled as a 4-way superscalar processor, and OVPM employs 8-way parallel pipelined VFUs.

Nine multimedia benchmark programs in Table 2.3 are used to evaluate OVPM. The benchmark programs of *clip*, *fft* and *power* are three hot kernels of the 3-D computer visual algorithm for super resolution images [3]. *M*x*M* and *V*x*M* are programs of matrix multiplications, which are commonly used in MMAs. The other four benchmark programs are the kernels of MMAs selected from the PARSEC benchmark suite [32] and the ALP-bench benchmark suite [33]. Both of them include emerging MMAs that contain massive

Table 2.3: *Benchmark Programs.*

Benchmarks	Categories	Vector Length
sphinx	speech recognition	4096
face	face recognition	173
ray	animation	1080
vips	image processing	79
clip	computer vision	64
fft	computer vision	32
power	computer vision	33
MxM	Matrix Multiplication	1000
VxM	Vector-Matrix Multiplication	1000

DLP. The benchmark programs are compiled by a PISA cross-compiler, so as to generate assembly codes defined in the SimpleScalar toolset. For the vector codes, there are many mature vector compilers such as NEC's C compiler of SX-9 (sxcc) [41]. However, since SimpleScalar does not support the cross-compilation with sxcc, the benchmark programs are firstly vectorized automatically by using sxcc. Then, referring to the assembly code generated by sxcc, vector instructions are manually inserted into the assembly code files generated by the PISA cross-compiler. Finally, the assembly code files are translated to binary codes as the inputs of the simulator.

In order to take a closer look at the evaluation results, the benchmark programs are classified into the short vector category and the long vector category. The short vector category contains the benchmark programs *face*, *vips*, *clip*, *fft*, and *power*. The other benchmark programs including *ray*, *V*x*M*, *M*x*M* and *sphinx* are classified to the long vector category.

2.5.2 Evaluation Results

Computational Efficiency of IVPM and OVPM

The computation efficiencies of IVPM and OVPM are compared in order to demonstrate the effective of OVPM. Here, the computational efficiency is defined by the ratio of sus-

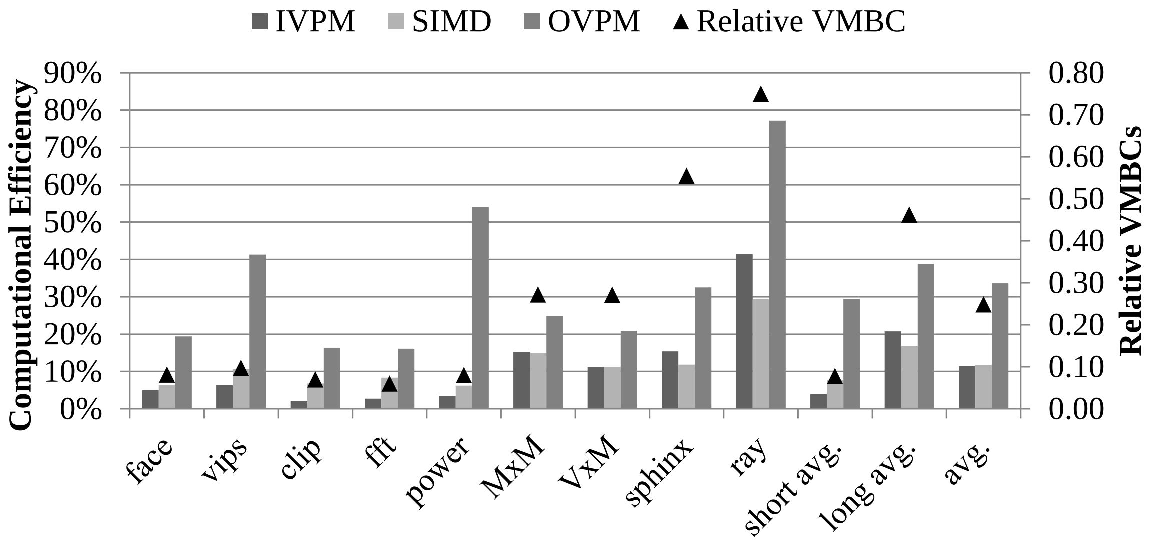


Figure 2.6: *Computational Efficiency of IVPM and OVPM.*

tained performance to peak performance. The higher the computational efficiency means the better use of vector execution units. Specially, the computational efficiency of 100 % means that the MMA always makes the full use of the vector execution units during the execution. The evaluation results show the maximum performance of IVPM with OVPM when changing the size of vector registers. The sizes of vector registers for IVPM and OVPM are 512 entries and 128 entries, respectively. This evaluation also compares the proposal with a SIMD extension, which one of the most popular approaches to media processing. According to the configuration of the latest SIMD extensions, Intel¡¯s AVX2 [19], it is assumed that the SIMD extension in this evaluation can process up to eight double-precision floating- point values or 512-bit integer values by one instruction, and it can also perform an out-of-order execution.

Figure 2.6 shows that the computational efficiency of OVPM is always higher than that of IVPM. The improvement of computational efficiency is derived from the good use of VFUs due to avoiding some of the pipeline stalls. With the help of the out-of-order issue policy, even if OVPM employs vector register of a smaller size, it still enables to effectively hide the memory access latency and pipeline latency of VFUs. Specially, the benchmark programs classified to the short vector category show significant performance improvement. In those benchmark programs, short vector operations cannot be always overlapped with memory operations in the case of IVPM. As a result, they easily expose the pipeline stalls, leading to the low computational efficiency. OVPM could efficiently reduce these stalls, and thereby allow vector extension to achieve a much higher efficiency on processing short vectors. In spite of the significant performance improvement, their computational efficiencies are still less than 50 %. This is because their vectorization ratios are lower than those of other benchmark programs.

OVPM also shows a higher performance than the SIMD extension for both of the MMAs in the short and long vector categories. This is because the limited number of elements that can be processed by one instruction restricts the performance improvement by the SIMD extension. OVPM can handle up to 128 vector data by one instruction, while SIMD can only process eight elements at most. Even for the MMAs in the short vector category, the vector lengths of these applications are still longer than the vector length that can be processed by one SIMD instruction. Therefore, the SIMD extension cannot make a good use of DLP in the MMAs. In contract to the SIMD extension, OVPM efficiently exploits the DLP in MMAs. As a result, the deep vector pipeline can be filled in to tolerate the long access latencies, and thus improve the performance of MMAs. This evaluation confirms that OVPM has a higher performance than a currently popular approach to MMAs.

To confirm that OVPM improve the computational efficiency of MMAs with short vectors by reducing the pipeline stalls, the stalled cycles of VLSU pipeline are evaluated, in terms of vector memory blank cycles (VMBCs). VMBCs are the accumulation of the exposed VLSU pipeline stalls. For a given program, a VMBC becomes smaller if the period of underutilizing memory bandwidth is shorter. Therefore, a small VMBC means an efficient use of memory bandwidth. Figure 2.6 shows the relative VMBCs of OVPM against IVPM. VMBCs of all the benchmark programs are reduced when employing OVPM, especially for the programs classified to short vector categories. Compared

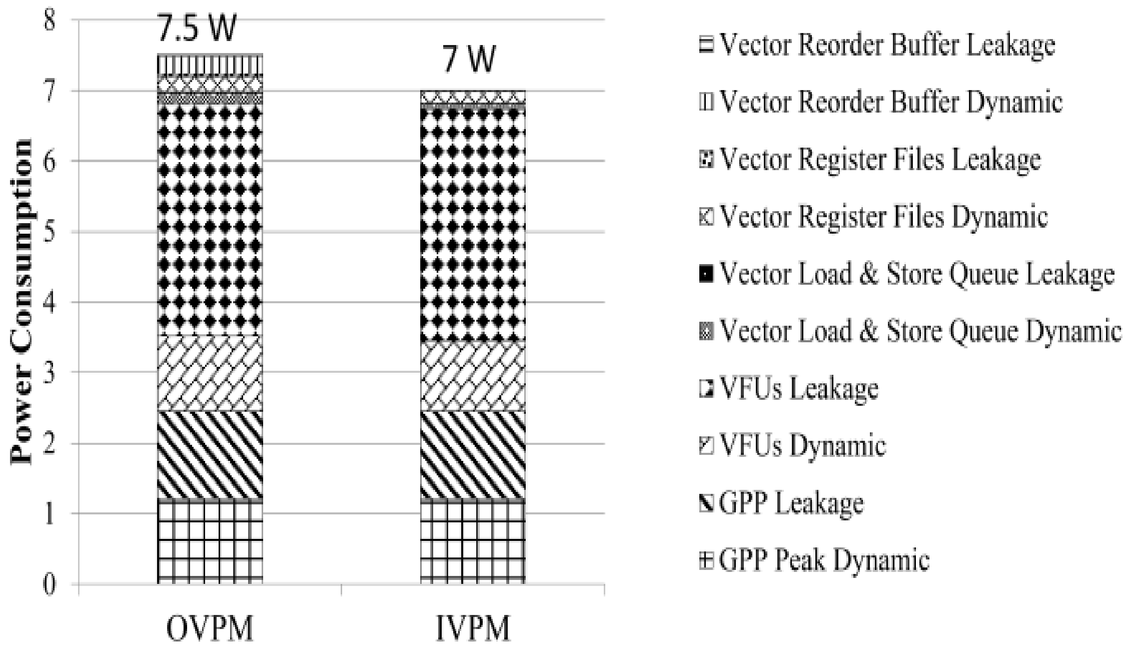


Figure 2.7: *Power Consumption Breakdown.*

with the efficiency improvement, VMBCs decrease as the speedup ratio increases. This proves that the main factor of performance improvements by the proposed architecture is an efficient use of the memory bandwidth. One exception is the benchmark program *ray*. This is because most of the pipeline stall happened in VFUs in the case of IVPM. Consequently, these evaluations clarify that OVPM could improve the computational efficiency on short vector processing by reduce the pipeline stalls.

Breakdown of Power Consumption

Figure 2.7 shows the power breakdowns of OVPM and IVPM. OVPM can be implemented with only a 7% higher power consumption than IVPM because it shares the renaming unit and commit unit with the scalar unit. The additional power consumption of OVPM comes from vector register files, vector reorder buffers, and load and store queues. OVPM adopts a larger number of physical vector register files than IVPM. Those vector register files are used for register renaming in order to omit name dependencies. Moreover, OVPM

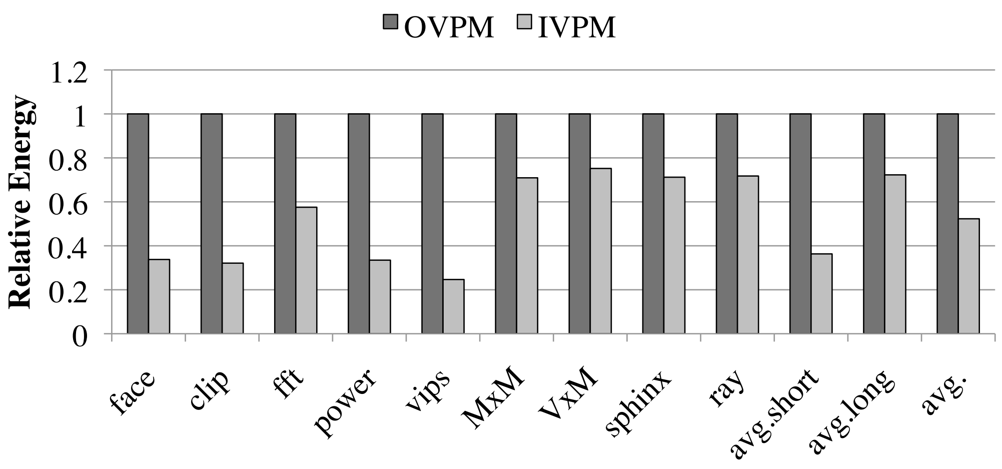


Figure 2.8: *Energy Consumption of OVPM and IVPM.*

consumes additional power for realizing the out-of-order execution of vector instructions by using the vector reorder buffers. The load and store queues of OVPM also achieve a higher power consumption than those of IVPM because they install new hardware to detect the load-store coherency.

Figure 2.8 shows the energy consumption of IVPM and OVPM. OVPM achieves a lower energy consumption than IVPM, especially for the MMAs with short vectors. This is because OVPM could significantly shorten the execution times of MMAs with short vectors at a low additional power cost. Consequently, this evaluation confirms that it is worth to invest some power budget to OVPM for media processing from the point of view of power efficiency.

2.6 Conclusions

Aiming to enhance the potential of GPPs on MMAs, a vector extension with OVPM has been proposed. OVPM overcomes the inefficiencies in executing MMAs by conventional vector architectures, which obey an in-order instruction issue policy. OVPM is implemented by using two instruction buffers. The two instruction buffers ensure that vector instructions can be issued as long as their operands are ready. The evaluation results show that OVPM obtains 3.25x speedup on average with only 7% additional power consumption. The results indicate that OVPM could improve the power efficiency of MVPX for all MMAs.

Chapter 3

A Multi-banked Cache Memory Associating one Tag with Multiple Data Array

3.1 Introduction

Vector processors, used in high-end computing systems are famous for their high data transfer performance [26]. However, the high data transfer performance leads to enormous costs in terms of chip area, power consumption and energy consumption. For instance, the vector processor of an SX-9 supercomputer system employs 8960 I/O pins for simultaneous data transfers from/to main memory units [42]. The large number of I/O pins is the main reason to make the chip area reach nearly 400 mm^2 and consume up to 240 W [43]. Moreover, a lot of energy is consumed on off-chip memory accesses, which consumes 70% of the energy consumption of an SX-9 node including 16 processors and 1 TB shared memories [44]. These costs are too expensive for desktop computers, which is the main execution platform of MMAs. Therefore, the high bandwidth memory sub-systems cannot straightforwardly transplant to desktop computers.

In the previous researches of high performance computing system domains, it has been shown that a multi-banked cache memory could effectively reduce the energy consumption caused by high data transfer performance of vector processors [45] [46] [26] [47] [48]. At the same time, several literatures have also reported that MMAs have high data reusability [49] [50]. Therefore, the multi-banked cache memory also has a high potential for MMAs.

However, conventional multi-banked cache memories of vector processors cannot transfer data efficiently for MMAs. In order to adopt multi-banked cache memories to MMAs, there are at least two requirements that should be considered. There are vectors of various lengths involved in MMAs. Hence, the first requirement is that the multi-banked cache memory should efficiently transfer vectors of various lengths. Moreover, as the main execution environment of MMAs, desktop computers invest so much energy consumption in a cache memory. Hence, the other requirement is that the multi-banked cache memory should achieve a low energy consumption. However, conventional multi-banked cache memories either have a low data transfer efficiency for short vectors or cost high energy consumption on their tag arrays. Therefore, the purpose of this chapter is to improve the data transfer performance of multi-banked cache memories for MMAs with short vectors at a low energy consumption on tag arrays.

To this end, this chapter proposes a multi-banked cache memory for vector processors called MVP-cache. Unlike conventional multi-banked cache memories that consist of one data array and one tag array, MVP-cache associates one tag array with multiple independent data arrays of small-sized cache lines. As a result, MVP-cache consumes less static power on its tag arrays. At the same time, MVP-cache can also achieve a high efficiency on short vector data transfers because the flexibility of data transfers can be improved by controlling data transfers of each data array.

The rest of the chapter is organized as follows. Section 3.2 clarifies the challenges for designing a multi-banked cache memory for MMAs. Section 3.3 describes the details of MVP-cache, and Section 3.4 evaluates the performance of MVP-cache. Section 3.5 gives the conclusions of this chapter.

3.2 Challenges for Designing a Multi-banked Cache Memory for MMAs

This section aims to clarify the problems of conventional multi-banked cache memories and obtain some ideas to the design of MVP-cache.

Multi-banked cache memories have a considerable potential to improve data transfer performance. Those cache memories consist of multiple independent cache banks, called sub-caches, and each sub-cache is connected to vector register files via interconnection fabric. When a cache hit occurs in a multi-banked cache memory, data can be simultaneously transferred from/to multiple independent sub-caches. In this way, multi-banked cache memories enable to improve the data transfer performance. Although the multi-banked cache memories bring additional energy consumption for vector processors, it is smaller than the energy saving induced by reducing the number of off-chip memory accesses.

Conventional multi-banked cache memories can be classified into the multi-banked cache memories of a large cache lines (MBC-L) [51] and of a small cache lines (MBC-S) [47]. In this chapter, a multi-banked cache memory is classified into MBC-S if its cache line size is equal to or smaller than 8 bytes, otherwise classified into MBC-L. The problems of MBC-L and MBC-S are described as follows.

3.2.1 Problems of MBC-L

MBC-L allocates several data elements with consecutive addresses in the same sub-cache. Figure 3.1 shows an example of MBC-L. There are four sub-caches, and each of them can transfer 64-bit data per cycle. The cache line size of this MBC-L is assumed at 32 bytes. A data layout of a vector V is shown in Figure 3.1. V represents a vector of double-precision floating-point values. It is assumed that V contains eight elements. In this case, since each cache line can store four data elements, the consecutive elements V[0] to V[3]

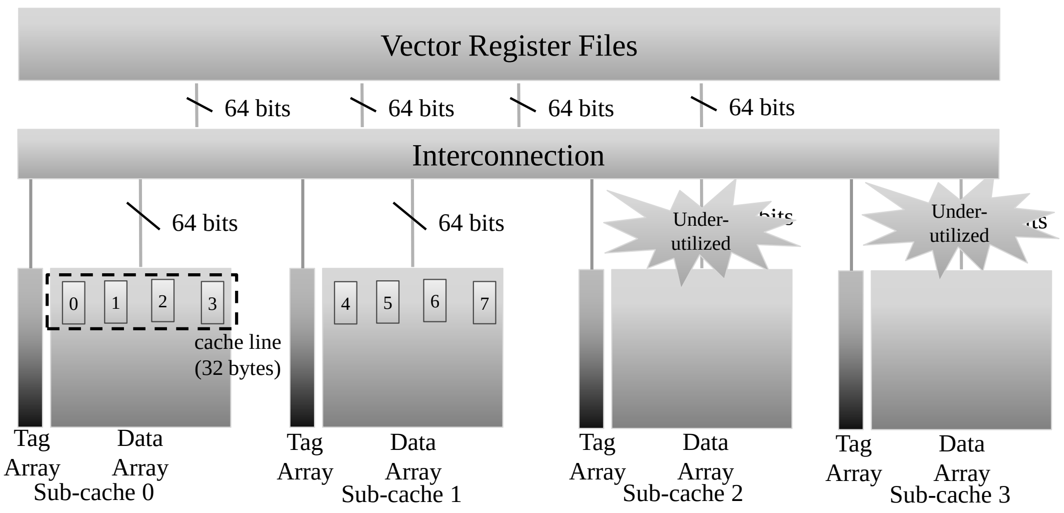


Figure 3.1: *Short Vector Data Transfer in MBC-L.*

are stored in the *sub-cache*0, and elements V[4] to V[7] are stored in the *sub-cache*1.

The problem of MBC-L is that MMAs with short vectors cannot make a good use of its high cache bandwidth. Figure 3.1 illustrates the problem of using MBC-L for MMAs with short vectors. The vector V only has eight elements. Those eight elements are stored in the cache lines of *sub-cache*0 and *sub-cache*1, and there are no data stored in the cache lines of *sub-cache*2 and *sub-cache*3. Therefore, the data can be only transferred from *sub-cache*0 and *sub-cache*1 can transfer data in parallel. Since two of four sub-caches are underutilized, the sustained bandwidth only reaches half of the peak bandwidth.

In practical uses, the underutilization of sub-caches due to short vectors will be more serious than the example. The vector V is assumed to be double-precision floating-point values in Figure 3.1. In a realistic situation, single-precision floating-point values or integer values are also commonly used in MMAs. Since the size of these data is smaller than those of double-precision floating-point values, there are more data that are concentrated on one sub-cache. This will potentially lead to more underutilized sub-caches. Moreover, to obtain a high cache bandwidth, a large number of sub-caches would be employed. Espasa et al. [47] have proposed a MBC-L for a vector ISA extension. Their MBC-L adopts 16 sub-caches and the cache line size of each sub-cache is 64 bytes equaling the size of eight

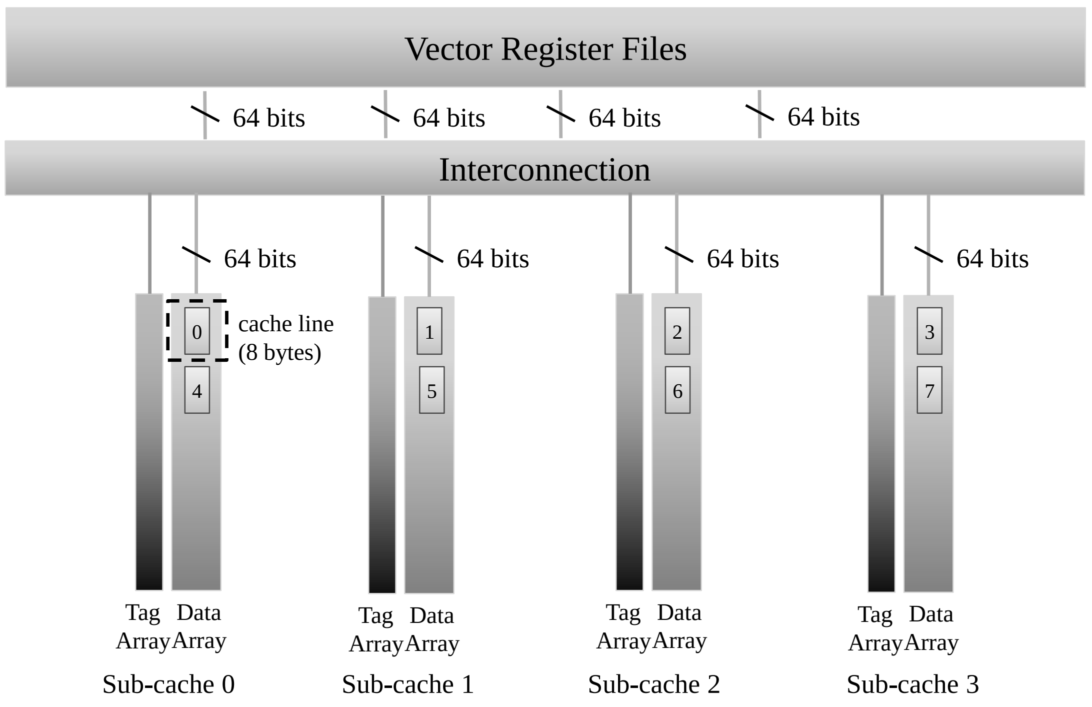


Figure 3.2: *Short Vector Data Transfer in MBC-S.*

double-precision floating-point elements. This configuration means that, if the length of a double-precision floating-point vector is shorter than 128, its data transfer potential would be underutilized.

3.2.2 Problems of MBC-S

In contrast to MBC-L, MBC-S enables to efficiently transfer short vector data. Since each cache line of MBC-S can store one or two double-precision floating-point data, consecutive elements could be dispersed to the different sub-caches. Figure 3.2 illustrates a data layout in the case of MBC-S of eight-byte cache lines. In this case, consecutive elements V[0] to V[3] are dispersed in the *sub-cache*0 to *sub-cache*3, respectively because each cache line can only store one double-precision floating-point element. By the full use of sub-caches, MBC-S enables to transfer short vector data at a high bandwidth.

However, the smaller cache line size of MBC-S means more cache lines and thus more cache tags. As a result, MBC-S needs larger tag arrays and a higher energy consumption than MBC-L. Because each cache line should have a cache tag, MBC-S has to install larger tag arrays than MBC-L. The increase in tag array size leads to high energy consumption. Although many researchers have proposed several techniques to reduce the energy consumption of cache memories [52], most of them reduce the energy consumption of data arrays. Therefore, the energy consumption of tag arrays will become more and more serious.

3.2.3 Related Work

Espasa et al. have proposed an out-of-order vector architecture [53] and have applied it to Tarantula [47], a vector extension to an Alpha processor. The cache mechanism of Tarantula, which is categorized into MBC-L, employs 16 sub-caches and an address re-ordering algorithm [54] to reduce the bank conflictions. However, performance degradation of MMAs with short vectors could occur due to inefficient short vector transfers for MBC-L.

Batten et al. have noted that not only access latency of memory sub-systems but also their bandwidth is very important to improve the application performance [48]. They have proposed an inexpensive non-blocking cache memory for vector architectures to improve the bandwidth and reduce the access latency of memory sub-systems. Musa et al. have designed a vector cache for vector architectures [51]. The vector cache introduces a bypass mechanism and miss status handling registers to improve the sustain memory bandwidth to next generation vector supercomputers. However, both of the two proposals adopt MBC-S and cause high energy consumption in the tag arrays.

To sum up the above discussion, when considering the design of MVPX for desktop computers, neither of the two configurations of multi-banked cache memories can satisfy requirements of a high data transfer performance at low energy consumption. MBC-L cannot transfer short vectors efficiently, and MBC-S consumes a large energy on tag arrays. Therefore, a new organization of a multi-banked cache memory is needed to overcome the drawbacks of MBC-L and MBC-S.

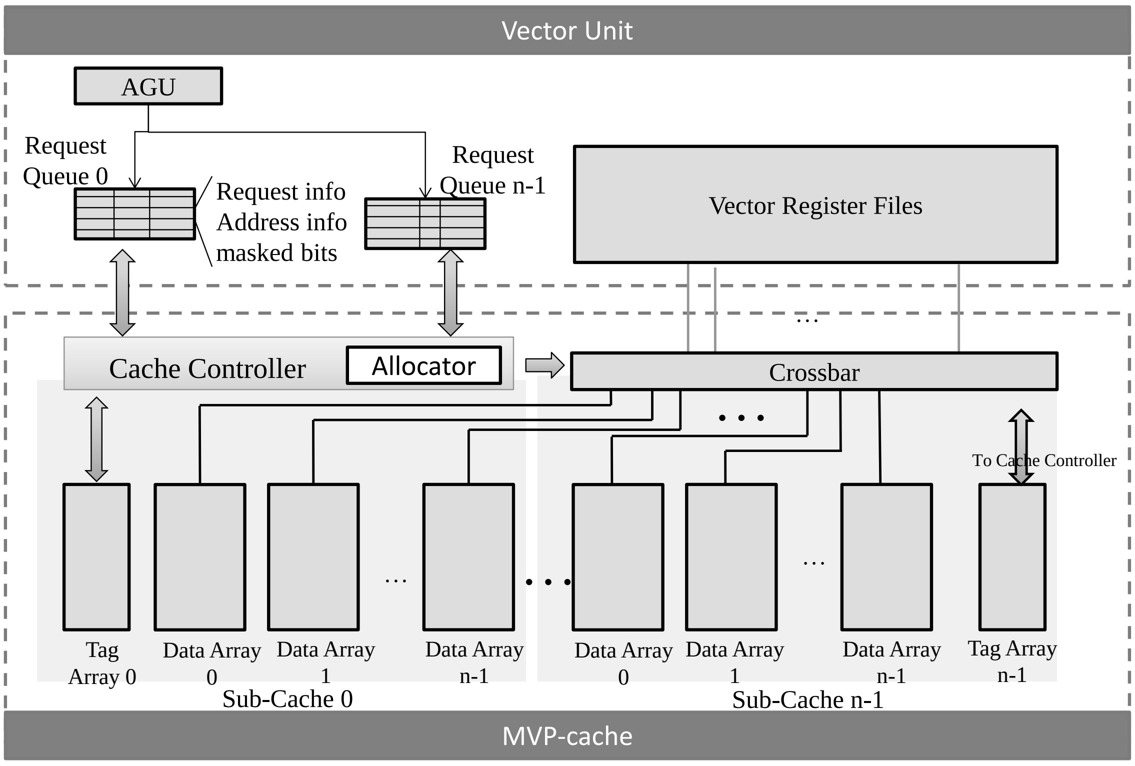


Figure 3.3: *Block Diagram of MVP-cache.*

3.3 MVP-cache

The purpose of MVP-cache is to increase the data transfer performance for short vectors at a low energy consumption. The design of MVP-cache makes a full use of the advantages of MBC-L and MBC-S. Essentially, MBC-L costs less energy on tag arrays than MBC-S because one cache tag is associated with a large cache line. Meanwhile, MBC-S is more efficient than MBC-L for short vector data transfers because consecutive vector elements are dispersed across different sub-caches due to the small-sized cache line. Therefore, the key idea of this approach is to associate multiple independent data arrays of the same size with one tag array.

3.3.1 Organization of MVP-cache

Figure 3.3 shows the organization of MVP-cache. As mentioned above, MVP-cache associates one tag array with multiple data arrays with small-sized cache lines in a sub-cache.

Each of data arrays adopts an eight-byte cache line, which is the size of double-precision floating-point values, and independently connects to cache ports of the vector unit via a crossbar. The cache lines that are associated with the same tag act as an atomic unit of data management and miss handling.

For data transfers of MVP-cache, if a cache hit occurs, cache lines that are associated with the same tag will be transferred. However, sometimes only a portion of data arrays in a sub-cache need to transfer their data. Transferring the unnecessary data to vector register files means a waste of energy consumption and bandwidth of MVP-cache. Therefore, MVP-cache controls which data should be transferred, by adding a new field in a cache request, called *Masked Bits*. If a masked bit is set to one, data should be transferred from the corresponding data array to the vector register file, otherwise the data transfer is disabled. By using masked bits, MVP-cache avoids transferring the unnecessary data and improves its energy efficiency.

MVP-cache could potentially satisfy the requirement of high data transfer performance for short vectors at a low energy consumption. First, because one tag array manages multiple data arrays in a sub-cache, the size of tag arrays is much smaller than that of MBC-S. It is assumed that the cache capacity and cache line size of MVP-cache are the same as those of MBC-S. In the case where MVP-cache associates one tag array with n data arrays, the size of tag arrays of MVP-cache is n times smaller than that of MBC-S. Second, MVP-cache can tolerate even shorter vectors than MBC-L. It is assumed that, in a typical case, the cache line size of data array of MVP-cache is 8 bytes, and that of MBC-L is 64 bytes. The number of data arrays of MVP-cache is N_{MVP}, and the number of data arrays of MBC-L is N_{MBCL}. For a unit stride access of double-precision floating-point data, at least N_{MVP} elements are needed for the full use of cache bandwidth. As analyzed in Section 3.2.1, for MBC-L, in order to make a full use of cache bandwidth, the vector length of applications should be longer than $8 \times N_{MBCL}$. When achieving the same cache bandwidth, the value of N_{MVP} is equal to that of N_{MBCL}. Thus, MVP-cache needs eight times less data elements than MBC-L to fully use the cache bandwidth. In

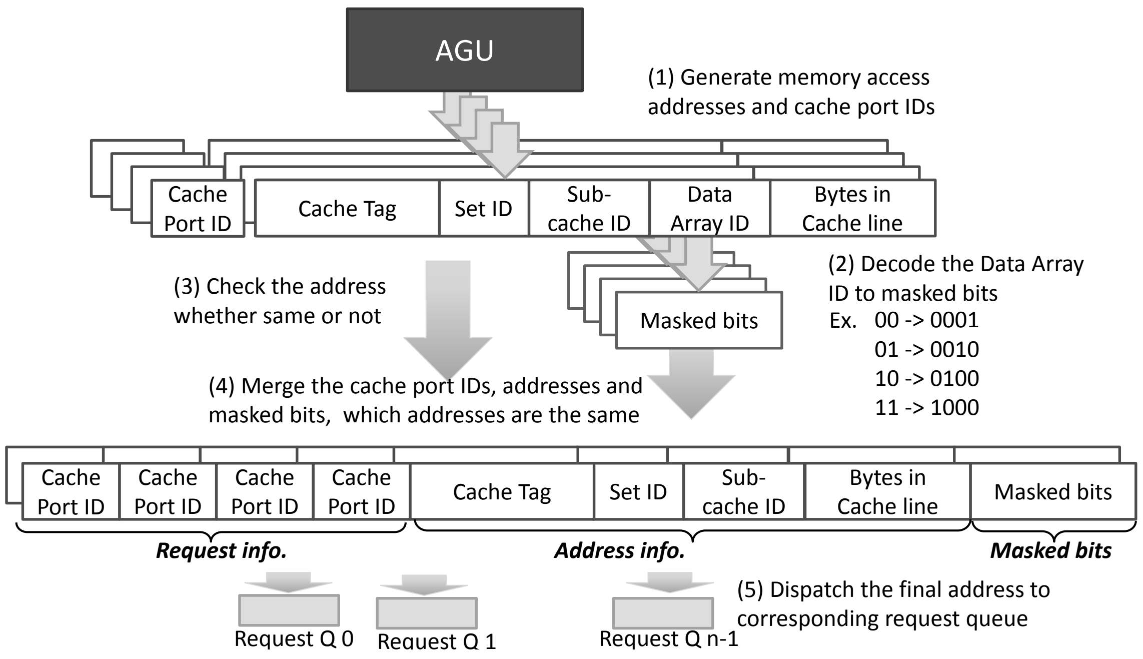


Figure 3.4: *Generation of* Request Info*,* Address Info *and* Masked Bits*.*

other words, MVP-cache is potentially more efficient for MMAs with short vectors than MBC-L.

3.3.2 Data Transfer Control Information of MVP-cache

In order to control the data transfer of MVP-cache, its cache controller needs some cache access information including *Request Info*, *Address Info* and *Masked Bits*. *Request info* stores information of the cache port IDs requested by a cache access. *Address Info* is the target address of the cache access. *Masked Bits* own the hints that which data arrays in a sub-cache should carry out data transfers. *Address Info* is used to judge if a cache access hits or misses, while *Request Info* and *Masked Bits* are used by the crossbar allocators to generate the connections between MVP-cache and vector register files.

The cache access information is generated according to memory addresses and stored in the data request queues. The generation steps of *Request Info*, *Address Info* and *Masked Bits* are shown in Figure 3.4 and described as follows.

Step 1 Generate the memory addresses accessed by a vector load/store instruction, and determine the cache port IDs used for the accesses. For each address, `cache tag`, `set ID`, `sub-cache ID`, `data Array ID` and `bytes in a cache line` can be obtained according to the configuration of MVP-cache.

Step 2 Decode `data array IDs` to set masked bits. If the `data array ID` of a memory access is n, the n th masked bit should be set as one, because the `data array ID` means that the specified data array owns the data that should be transferred.

Step 3 Check whether there are the same memory addresses in the generated addresses. The `cache tag`, `set ID` and `sub-cache ID` of all access addresses are compared each other. If they are the same, it means that they would access the same set of the data arrays in the same sub-cache.

Step 4 If there are the same memory addresses, their `cache port IDs`, memory addresses, and `masked bits` are merged to generate *Request Info*, *Address Info*, and *Masked Bits*, respectively. For *Masked Bits*, they are merged by using the bit-wise OR operation with the other masked bits. For `cache port IDs`, they are merged by using logical SHIFT and bit-wise AND operations to hold the whole information.

Step5 Dispatch the final results to the corresponding request queues according to the `Sub-cache ID`.

3.3.3 MVP-cache Crossbar Allocator

The MVP-cache controller adopts an $n \times m$ allocator to control the crossbar, where n denotes the number of data arrays of all sub-caches, and m means the number of cache ports. The base design of the allocator and crossbar is described in [55] [56]. The input of the allocator is a request matrix $R = (r_{i,j})_{n \times m}$, where the element $r_{i,j} = 1$ means that data array i needs to use cache port j. The output of the allocator is a grant matrix $G = (g_{i,j})_{n \times m}$, where the element $g_{i,j} = 1$ means that data array i is allowed to use cache port j. $G = (g_{i,j})_{n \times m}$ is used to configure the crossbar of MVP-cache.

Algorithm 1 Generation of Request Matrix $R = (r_{i,j})_{n\times m}$ Using Masked Bits and Request Info.

```
1: for $i = 0 \to n-1$ do
2:     for $j = 0 \to m-1$ do
3:         if $globalMaskBit[i] == 1$ and $j == globalRequestInfo[i]$ then
4:             $r[i][j] \leftarrow 1$
5:         else
6:             $r[i][j] \leftarrow 0$
7:         end if
8:     end for
9: end for
```

MVP-cache uses *Masked Bits* to control which data array transfers the data to vector registers. Therefore, it is also necessary to designs an algorithm that generates the request matrix $R = (r_{i,j})_{n\times m}$ with the concern of *Masked Bits.* The algorithm is shown in Algorithm 1. The $globalMaskBit[i]$ and $globalRequestInfo[i]$ mean the *Masked Bits* and *Request Info* of all sub-caches, respectively. They are generated by combining the local *Masked Bits* and *Request Info* of each sub-caches together. By using Algorithm 1, *Masked Bits* can be used to control data transfers that are really necessary for the vector unit as mentioned in Section 3.3.1.

3.3.4 Tag Array Conflicts of MVP-cache

The memory accesses to the different sets in the same sub-cache cannot be performed simultaneously because a sub-cache can only judge whether a cache access hits or misses once per cycle. In this chapter, such a situation is called a tag array conflict. Tag array conflicts frequently occur in the case of stride accesses. The wider the stride width is, the more tag array conflicts occur.

In this chapter, two parameters of MVP-cache are tuned in order to reduce the number of tag array conflicts in stride memory accesses. One is the numbers of sub-caches. The other one is the numbers of data arrays in one sub-cache. Increasing the number of sub-caches increases the probability that memory accesses are dispersed across different sub-caches, while increasing the number of data arrays increases the probability that

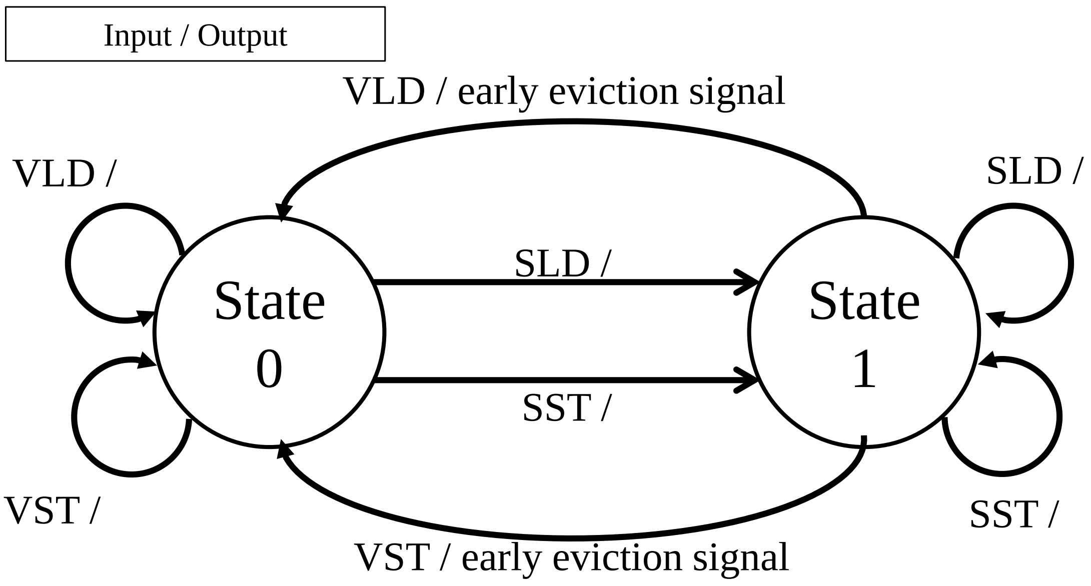


Figure 3.5: *State Transmission Diagram.*

memory accesses are concentrated on the same set in the same sub-cache. Increasing the numbers of data arrays or sub-caches also brings extra energy consumption in MVP-cache. However, this chapter considers that the overhead is small because the stride width of an MMA is less than three in most cases [54].

3.3.5 Coherency of MVP-cache with L1 cache

As shown in Figure 3.3, both the L1 data cache memory and the vector unit can read and write data from/to MVP-cache. Therefore, the coherency between the L1 data cache memory and MVP-cache should be considered. Essentially, this problem equals to which cache memory owns the latest data. Therefore, this section describes a state machine to record the ownership of latest data.

From the view point of a memory sub-system, the L1 data cache memory and MVP-cache can be directly accessed. Hence, only two internal states are needed for the state machine. It is defined that *State*0 means that MVP-cache owns the latest data, and *State*1 means that the L1 data cache memory also owns the latest data. The varieties of data requirement are considered as the inputs of the state machine. There are four inputs including combinations of data load and store requirement from the vector unit and the L1 data cache memory. They are vector load (VLD), vector store (VST) directly to the MVP-cache, and scalar load (SLD), scalar store (SST) to the L1 data cache memory. The output signals are used to control the data transfers between the L1 data cache memory and MVP-cache.

Figure 3.5 shows inputs, outputs, and the state transmissions of the state machine. For $State0$, in the cases of VLD and VST, the state would not be changed because the ownership of the data is not changed. Meanwhile, in the cases of SLD and SST, the cache line would be transferred from/to the L1 data cache memory, and the ownership is changed. Therefore, in these cases, the state transits to $State1$. For $State1$, in the cases of VLD and VST, it is necessary to update cache lines because the latest data are stored in the L1 data cache memory. Therefore, a signal would be send to the L1 data cache memory to invalidate the corresponding cache line and update MVP-cache. Such a behavior is so-called early eviction, and its mechanism has been proposed in many literatures [57] [58]. Then, data are transferred to/from the vector unit and the state transits to $State0$. In this state, if the input is the SLD and SST, the state does not transit, due to no change in ownership. In this way, the cache coherency can be guaranteed.

Table 3.1: *Configuration of MVPX and Memory Sub-System.*

Parameters	Value
Process Technology	32 nm
Processor Frequency	1GHz
Vector ALU Pipeline Latency	10 cycles
Vector Multiplier Pipeline Latency	15 cycles
Vector Division Pipeline Latency	20 cycles
Number of Vector Lanes	8
Number of Architectural Vector Registers	16
Number of Physical Vector Registers	96
Entries per Vector Register	128 entries
The size of VMIB and VAIB	128 entries
The size of Vector Load and Store Queue	512 entries
Number of Cache Ports	8
Cache Capacity	2 MB
Cache Bandwidth	64 bytes/cycle
Cache Line Size of MVP-cache	8 bytes
MVP-cache Access Latency	20 cycles
Number of Sub-caches	4
Number of Data Arrays in a Sub-cache	8
Memory Bandwidth	32 bytes/cycle
Main Memory Latency	100 cycles
Cache Line Size of MBC-S	8 bytes
Cache Line Size of MBC-L	64 bytes
MBC-L Access Latency	10 cycles
MBC-S Access Latency	10 cycles
Number of Sub-caches of MBC-S and MBC-L	32

3.4 Performance Evaluations

3.4.1 Experimental Methodology

A simulator of MVP-cache has been developed based on the SimpleScalar toolset [39]. The simulator also adopts OVPM proposed in Chapter 2 to estimate the performance of MVP-cache. McPAT0.8 [40] and CACTI6.5 [59] are used to evaluate the energy consumption of MVP-cache. The OVPM specification of MVP-cache MBC-S, and MBC-L are listed in Table 3.1. The cache line sizes of MVP-cache, MBC-S, and MBC-L are 8 bytes, 8 bytes

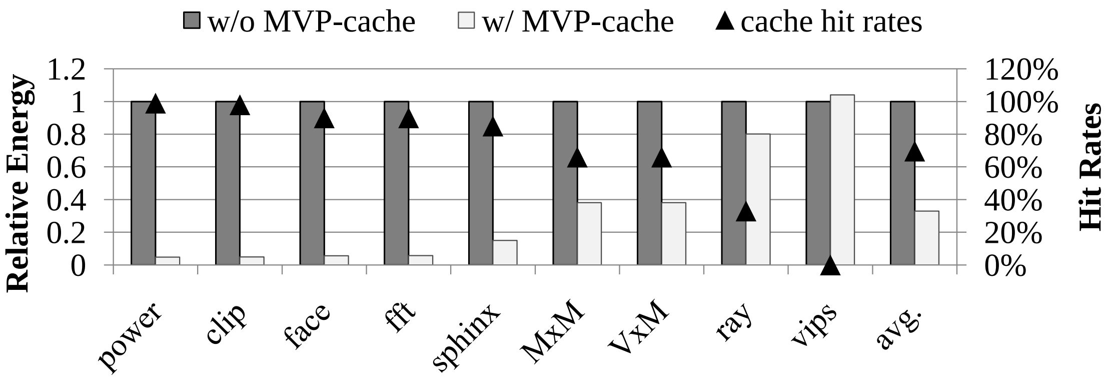


Figure 3.6: *Energy Reduction Effect of MVP-cache.*

and 64 bytes, respectively. To fairly compare performances of the three kinds of cache memories, their cache capacities and cache bandwidths are set to the same values. All the three multi-banked memories support the interleaved memory access to hide the access latencies and avoid bank conflicts. The benchmark programs used in this chapter are the same as Chapter 2.

3.4.2 Evaluation Results

Energy Reduction of MVP-cache

Since data transfers from MVP-cache do not need to go through I/O pins, they consume less energy than those from an off-chip memory. However, MVP-cache itself also consumes a certain energy. Therefore, the energy consumption with an off-chip memory is evaluated in order to clarify energy reduction effects by introducing MVP-cache.

Figure 3.6 illustrates the evaluation results. The energy consumption of VPM with MVP-cache is compared to that without MVP-cache. For all benchmark programs except *vips*, the memory sub-system with MVP-cache consumes less energy. The main factor of energy reduction is that MVP-cache reduces the number of off-chip memory accesses. Therefore, the energy saving effects of MVP-cache are proportional to the cache hits of each MMA. In the case of *vips*, since there are no data that can be reused, the memory

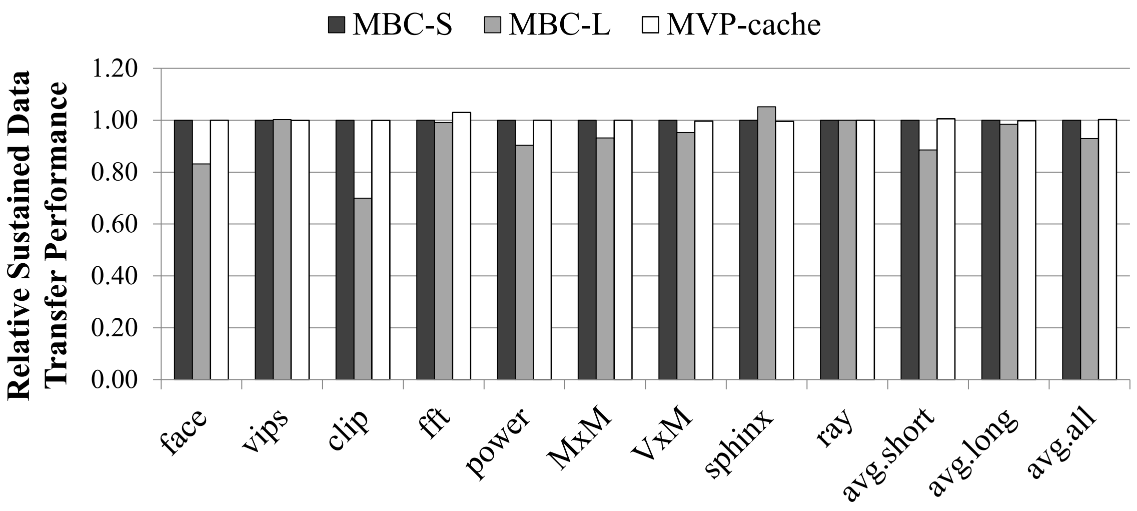


Figure 3.7: *Data Transfer Performance of MVP-cache.*

sub-system with MVP-cache consumes a bit higher energy than that without MVP-cache. The results show that energy consumption saved by MVP-cache is much larger than that it costs. Therefore, adding MVP-cache is an effective way to save the energy consumption of VPM for desktop computers.

Data Transfer Performance of MVP-cache

Figure 3.7 compares sustained bandwidths of MBC-S, MBC-L and MVP-cache, which are normalized by the bandwidth of MBC-S. MVP-cache attains almost the same performance as MBC-S, and a higher performance than MBC-L. The performance improvement against MBC-L is significant for MMAs in the short vector category. Hence, memory intensive MMAs with short vectors such as *clip* can achieve the greatest performance improvement in all the benchmark programs. Note that there is no performance improvement by MVP-cache in the benchmark *vips* even though it belongs to the short vector category. This is because there is no data reusability in this benchmark program. These results suggest that MVP-cache can transfer short vector data more efficiently than MBC-L.

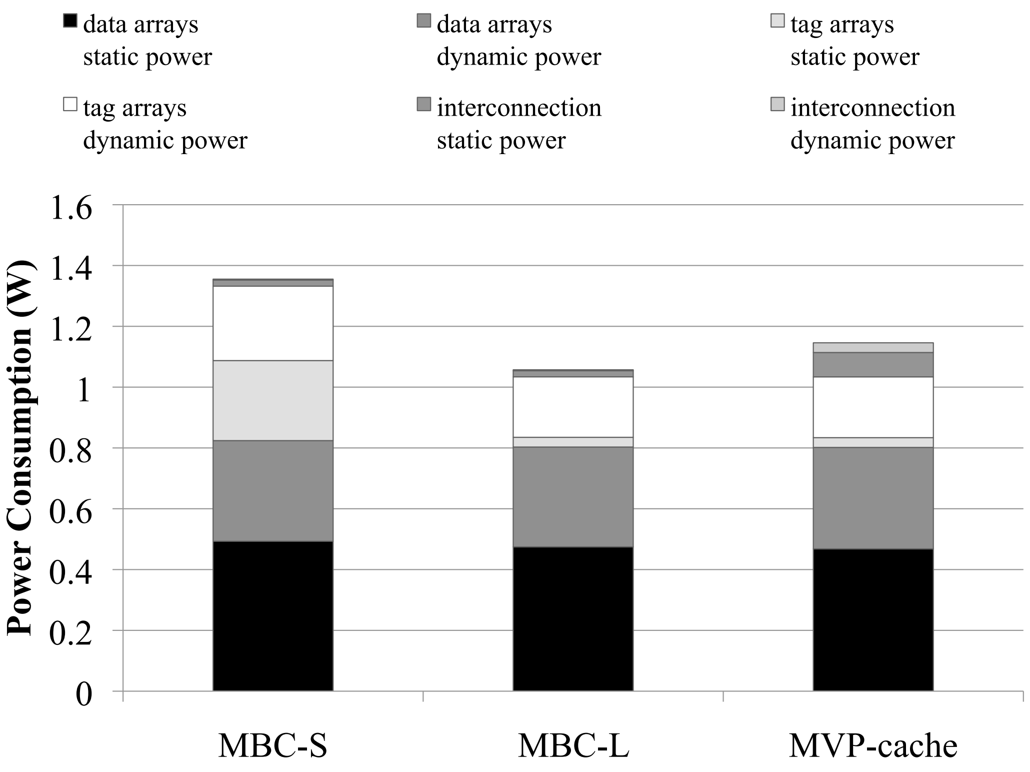


Figure 3.8: *Power Consumption Breakdown of MVP-cache.*

Power Consumption and Area Breakdown

Another key factor to affect the energy is power consumption. Figure 3.8 illustrates the power consumption breakdowns of MVP-cache, MBC-S, and MBC-L. Compared with MBC-S, MVP-cache has less power consumption on tag arrays because multiple data array associated with the same tag array. In this way, the size of tag array could be reduced. The reduction of tag size lead to smaller area for tag arrays as shown in Figure 3.9. As a result of small area of tag arrays in MVP-cache, the static power consumption of tag arrays of MVP-cache is reduced. Moreover, the dynamic power consumption of tag arrays of MVP-cache can be also reduced compared with that of MBC-S. This is because sharing tag array with multiple data arrays could reduce the size of each tag. Therefore, the power consumption on driving tag arrays become lower than MVP-S. As a result, the dynamic power consumption of tag array of MVP-cache is also reduced. Meanwhile, MVP-cache

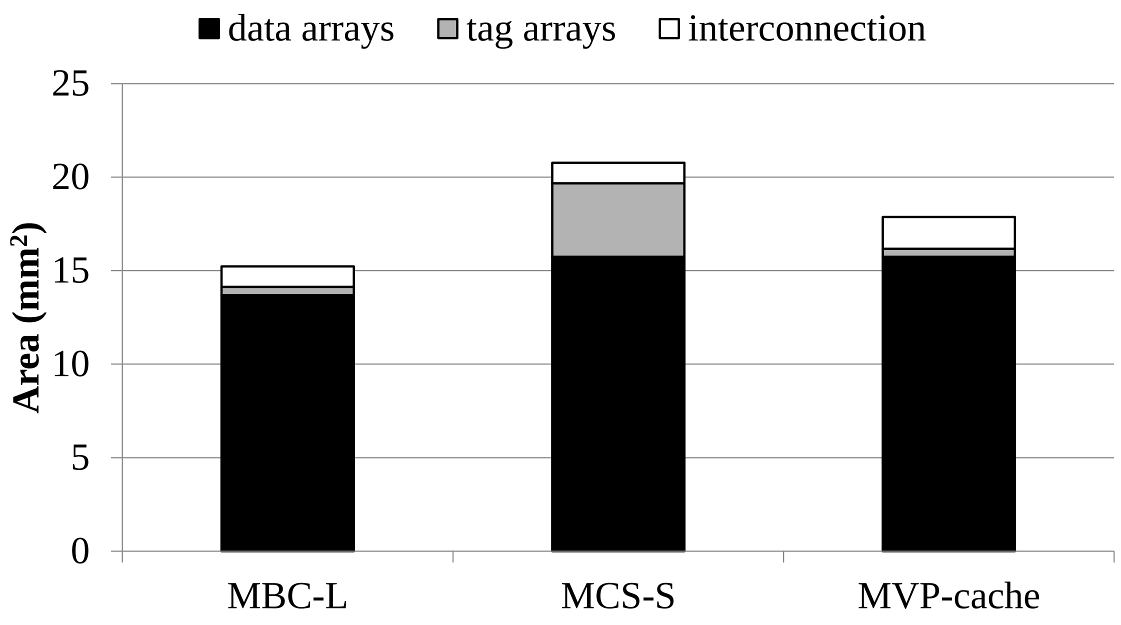


Figure 3.9: *Chip Area of MVP-cache.*

has to adopt more sub-caches to reduce the tag array conflicts. Hence, it costs more power on data arrays and the interconnection between cache and vector register files.

Figure 3.9 compares chip area of MVP-cache with that of MBC-S and MBC-L. The evaluation results show that the tag array area of MVP-cache is smaller than MBC-S. Although the complicated interconnection of MVP-cache takes a larger chip area, the total size of MVP-cache is still smaller than MBC-S. This results confirm that MVP-cache could effectively reduce the tag array size by associating one tag array with multiple data arrays.

Energy Reduction by MVP-cache

Energy consumption of MVP-cache is compared with that of MBC-S and MBC-L. Figure 3.10 shows the energy consumptions normalized by that of MBC-S. Except for the benchmark programs *vips* and *fft*, MVP-cache achieves a lower energy consumption for MMAs in the short vector category because it can transfer short vector data more efficiently than MBC-L and cost less power on tag arrays than MBC-S. *vips* is an exception

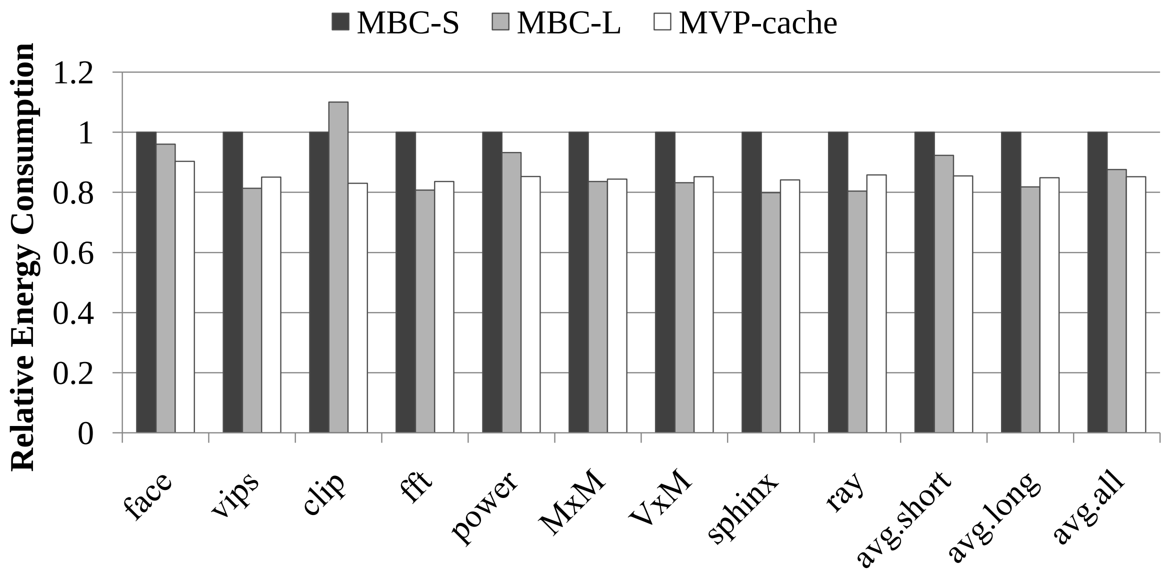


Figure 3.10: *Comparison of Energy Reduction among MBC-S, MBC-L and MVP-cache.*

because there is no data reusability in this benchmark program. For the benchmark program *fft*, MVP-cache consumes a higher energy than MBC-L because it only obtains a slight performance improvement from MVP-cache. On the other hand, for MMAs in long vector categories, MVP-cache achieves a lower energy consumption than MBC-S but a slightly higher energy consumption than MBC-L because these MMAs obtain little performance improvement from MVP-cache. Since the energy reduction effect of MVP-cache in the short vector category is larger than its energy consumption overhead in the long vector category, MVP-cache obtains the lowest energy consumption on average. Therefore, MVP-cache is a low energy approach to the improvement of the data transfer performance.

Design Space Exploration of MVP-cache

As mentioned in Section 3.4, it is necessary to clarify the trade-off between the performance and energy consumption when adjusting the numbers of data arrays and sub-caches. Increasing the numbers of data arrays and sub-caches would effectively reduce the performance degradation caused by stride accesses. Meanwhile, a large number of

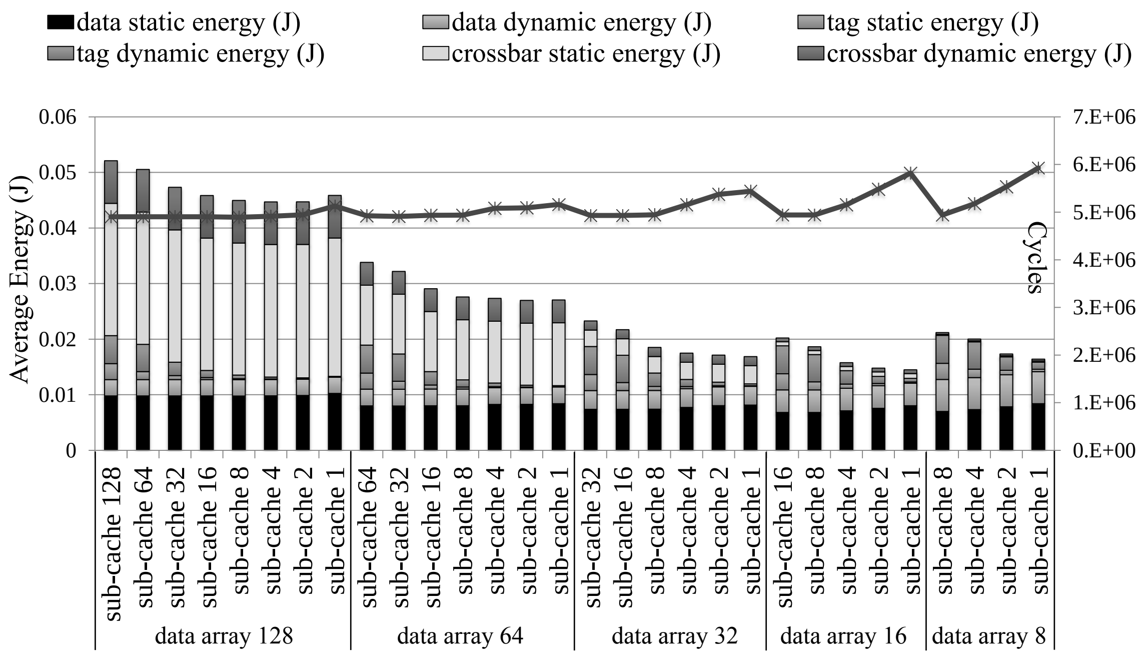
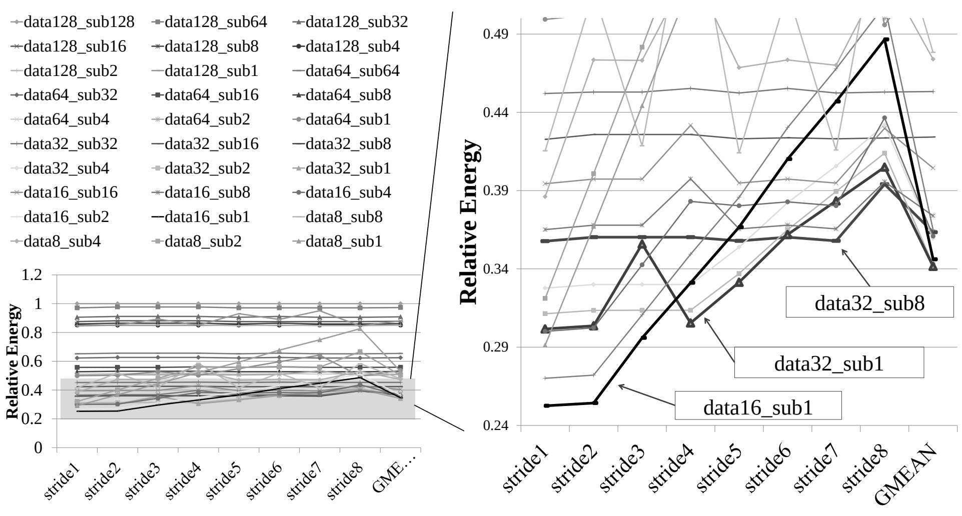


Figure 3.11: *Average Energy Consumption and Execution Cycles of all Benchmark Programs at Various Configuration of MVP-cache.*

data arrays would lead to a high power consumption on the crossbar between data arrays and cache ports of the vector unit. Moreover, increasing the number of sub-caches would increase the energy consumption of tag arrays.

Figure 3.11 shows average energy consumption breakdowns and execution cycles of all benchmark programs with the charge of the numbers of sub-caches and data arrays. Sub-cache n (n=8, 16, 32, 64, 128) and data array m (m=1, 2, 4, 8, 16, 32, 64, 128) represent that the numbers of sub-caches and data arrays are n and m, respectively. The division of n and m denotes the number of data arrays that are associated with one tag array.

With the decrease in the number of data arrays, the energy consumption of the crossbar is also reduced. It achieves a reasonable value in the case of 16 data arrays. At the same time, when the 16 data arrays share one tag array, the energy consumption of the tag array is reduced significantly. Although the average execution cycles of MMAs increase due to stride accesses, the decrease of performance is much smaller than the reduction in energy consumption. Therefore, one tag array associated with 16 data arrays is the most



Stride Width	Lowest Energy Configuration
Stride 1	data 16 sub 1
Stride 2	data 16 sub 1
Stride 3	data 16 sub 1
Stride 4	data 32 sub 1
Stride 5	data 32 sub 1
Stride 6	data 32 sub 1
Stride 7	data 32 sub 8
Stride 8	data 32 sub 8

Figure 3.12: *Energy Consumption for Different Stride Width.*

energy efficient configurations for the MMAs.

Energy Consumption for Different Stride Widths

In order to investigate energy consumptions in the cases of different stride widths, MVP-cache is evaluated by using some micro benchmark programs. The micro benchmark programs are designed as an addition of two vectors with different stride widths. Figure 3.12 shows the energy consumption of different MVP-cache configurations. Sub n (n=8, 16, 32, 64, 128) and data array m (m=1, 2, 4, 8, 16, 32, 64, 128) represent that the numbers of sub-caches and data arrays are n and m, respectively. The results are normalized against the energy consumption in the case of 128 data arrays and 128 sub-caches.

With increase of stride widths, the impacts from stride accesses would also increase, which leads to further performance degradation. Hence, in the cases of stride widths ranged from 1 to 8, the lowest energy configurations of MVP-cache is different. The configuration of 16 data arrays sharing one tag array, 32 data arrays sharing one tag array and 32 data array sharing four tag arrays achieve the lowest energy consumptions for stride widths ranging from 1 to 3, 4 to 6, and 7 to 8, respectively. On average, these three configurations achieve almost the same energy consumption. However, since the stride widths of most of the MMAs is less than three [54], the configuration of 16 data arrays sharing one tag array is considered as the most efficient for MMAs. This result is also in accord with the result in Section 3.4.2.

3.5 Conclusions

In order to match the demands of high data transfer performance and low energy consumption, MVP-cache is designed and evaluated. It associates one tag array with multiple data arrays to reduce the energy consumption of tag arrays and improve the efficiency on short vector data transfers. Based on the performance evaluations with MMAs, the effects of MVP-cache are discussed in terms of data transfer performance improvement and energy reduction. The evaluation results show that MVP-cache can achieve a comparable performance with the other competitive cache organizations, while the energy consumption of MVP-cache is smaller than those of the other. It is also found that the configuration of 16 data arrays associated with one tag array is a reasonable configuration for media benchmark programs used in this chapter. When the ranges of MMAs extend, it is also possible to use the same methodology mentioned in this chapter to find a reasonable configuration.

As the future work of this chapter, the implantation of MVP-cache will be considered by using the 3D integration technologies in order to further reduce the energy consumption.

Chapter 4

A Performance-Power Optimization Method for MVPX

4.1 Introduction

This chapter aims to improve the power efficiency of MVPX by satisfying the hardware requirements of each MMA. Chapters 2 and 3 introduced the mechanisms to improve the computing performance and data transfer performance of MVPX. In those two chapters, the number of parallel pipelines in each VFU and the number of cache ports are fixed for all MMAs. However, each MMA has its own performance requirement. For some memory-intensive MMAs, they require a large number of cache ports to improve the data transfer performance. Meanwhile, the computationally-intensive MMAs require a large number of parallel pipelines to improve the computing performance. If the numbers of hardware resources are fixed, MVPX cannot always achieve power-efficient for all MMAs. When MVPX employs a large amount of hardware resources to fit the maximum hardware requirement, MVPX wastes its power consumption for the MMAs which do not need so much hardware. On the other hand, if MVPX employs a small amount of hardware, it loses performance for the MMAs, which need more hardware resources to improve their performance.

A common approach to this problem is to employ a large amount of hardware resources and turn off the unnecessary ones for each application by using power gating techniques [60] [61]. Many kinds of hardware resources could be the power gating candidates, such as cache memories [62], arithmetic function units [63] [64] and cache ports [65] [66]. This chapter focuses on the number of parallel pipelines in each VFU and the number of cache ports because they are the most important parameters to affect the performance and power consumption of MVPX. Therefore, in order to achieve a power-efficient execution for any MMAs, it is important to find the proper numbers of the two parameters.

In the previous researches, either a simulation approach [67] [68] or an analytic approach [69] [70] has been used to find the most power-efficient configuration of processors. The simulation approach uses performance and power simulators of processors to obtain the execution cycles and power consumptions for all possible combinations of hardware resources. Then, the most power-efficient configuration could be found by sorting the energy consumption of all possible configurations. Since the most power-efficient configuration for an MMA may vary during the execution, the configuration should be dynamically optimized at runtime. Therefore, it is necessary to find the most power-efficient configuration as quickly as possible. However, the simulation approach takes a long time to find the most power-efficient configuration due to a long simulation time to obtain the execution cycles and power consumption for a configuration and exhaustively simulating all possible configurations.

Meanwhile, the analytic approach estimates the performance and power consumption of a processor by using an analytic model established for a processor. Analytical approaches cost a shorter time than simulation approaches because of the quicker estimations of execution cycles and power consumption by using an analytical model. However, the problem of the analytic approach is that it still exhaustively estimates the execution cycles and power consumption for all possible configurations, which is the waste of time on a large amount of unnecessary estimation.

In order to find the most power-efficient configuration as quickly as possible, this chapter proposes a performance-power optimization method (PPoM) for MVPX. Motivated by the avoidance of unnecessary estimation, PPoM adopts the greedy searching method to find the most power-efficient configuration. PPoM examines the performance bottleneck of a certain configuration, and only estimates the execution cycles and power consumption of the configuration, which could remove the current performance bottleneck. In this way, a large amount of unnecessary estimation could be avoided. Moreover, for the sake of a quick examination of the performance bottleneck and estimation of execute cycles and power consumption, this chapter proposes an analytic model by using the enqueue and dequeue throughputs of issue queues. This is because the enqueue and dequeue throughputs could reflect the utilization of hardware resources.

4.2 Importance of Finding the Most Power-Efficient Configuration

In this section, the requirements for finding the most power efficient configuration are clarified by the observation of the performance of MMAs on MVPX at various configurations. Then, this section discusses related work in order to show the problems of conventional approaches.

4.2.1 Various Most Power-efficient Configurations for each MMAs

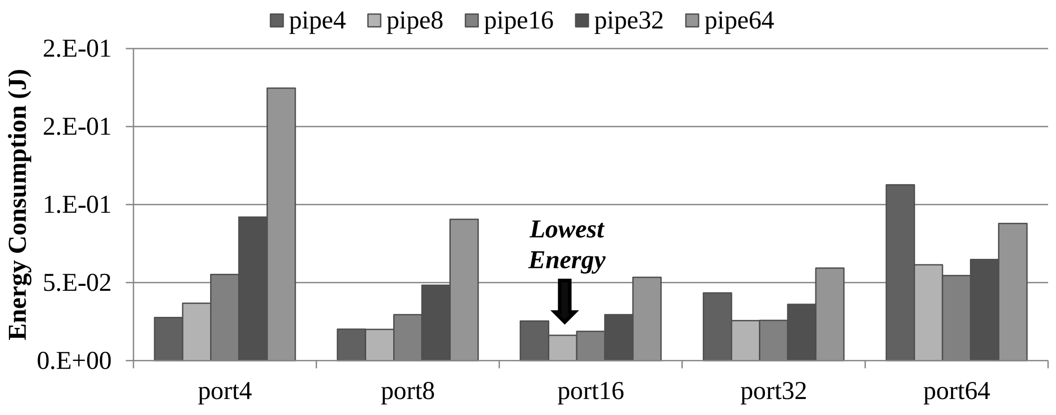


Figure 4.1: *Energy Consumption of* clip *at Different Numbers of VFUs and Cache Ports.*

Figures 4.1 to 4.9 show the energy consumption at various numbers of parallel pipelines in each VFU and cache ports. The arrows in the figures show the most power-efficient configuration. Each benchmark program requires a different configuration to achieve the highest power efficiency due to various hardware requirements. Computationally-intensive MMAs, such as *ray*, require a large number of parallel pipelines in each VFU to improve the computing performance, while memory-intensive MMAs such as *M*x*M* require a large number of cache ports to improve the data transfer performance. On the other hand, the

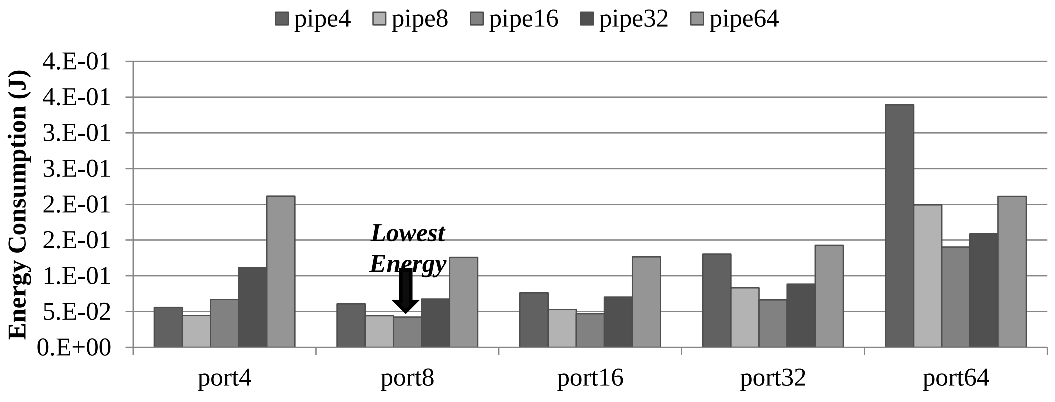


Figure 4.2: *Energy Consumption of* power *at Different Numbers of VFUs and Cache Ports.*

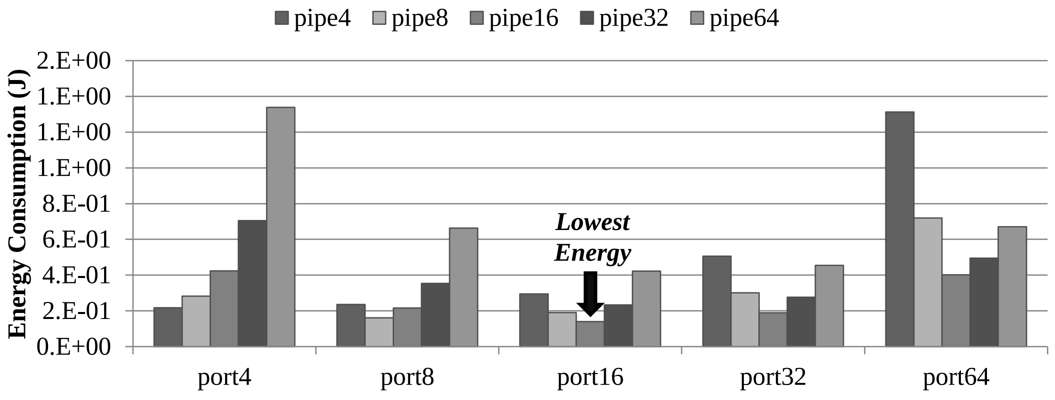


Figure 4.3: *Energy Consumption of* fft *at Different Numbers of VFUs and Cache Ports.*

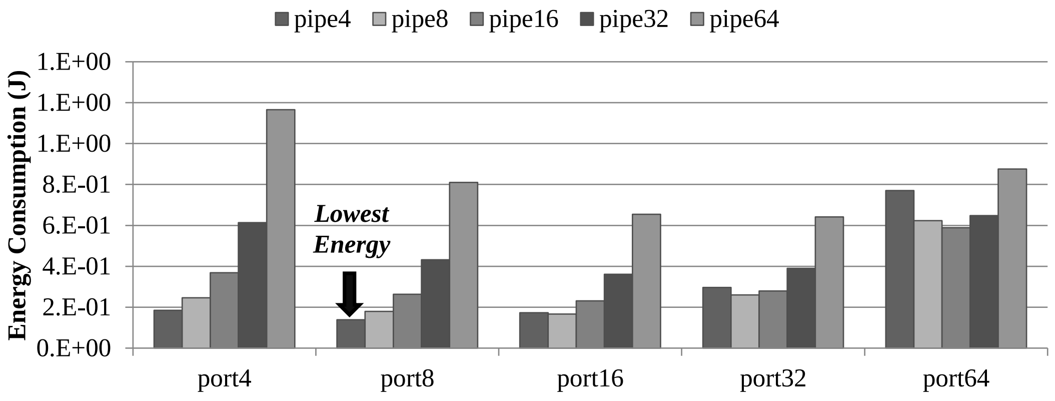


Figure 4.4: *Energy Consumption of* face *at Different Numbers of VFUs and Cache Ports.*

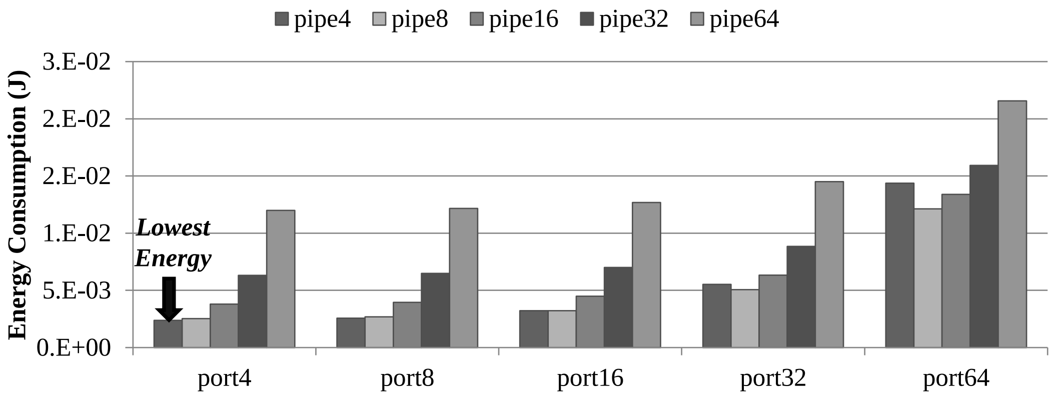


Figure 4.5: *Energy Consumption of* vips *at Different Numbers of VFUs and Cache Ports.*

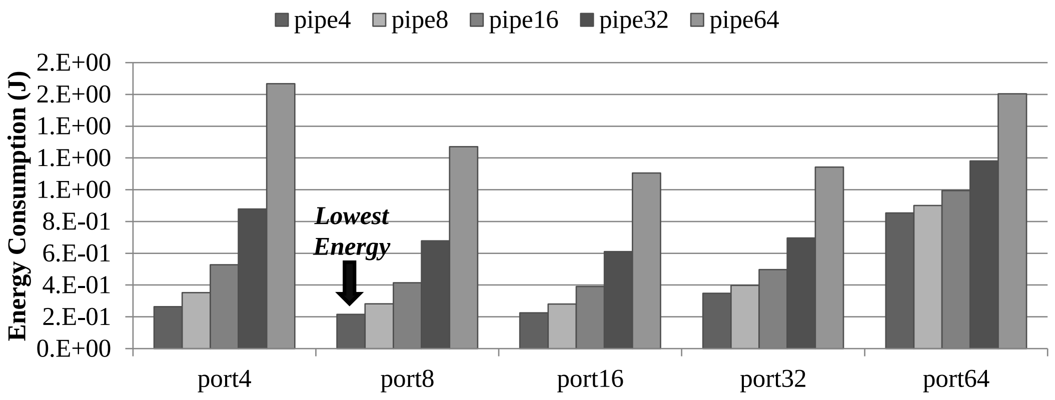


Figure 4.6: *Energy Consumption of* MxM *at Different Numbers of VFUs and Cache Ports.*

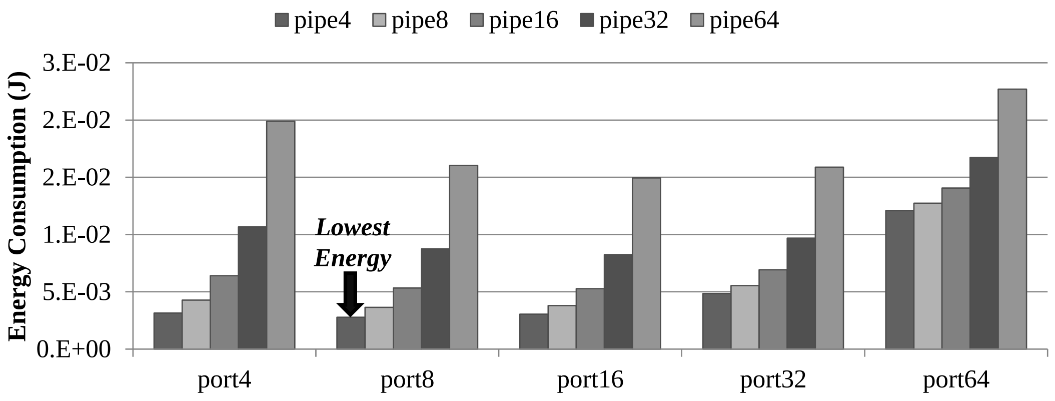


Figure 4.7: *Energy Consumption of* VxM *at Different Numbers of VFUs and Cache Ports.*

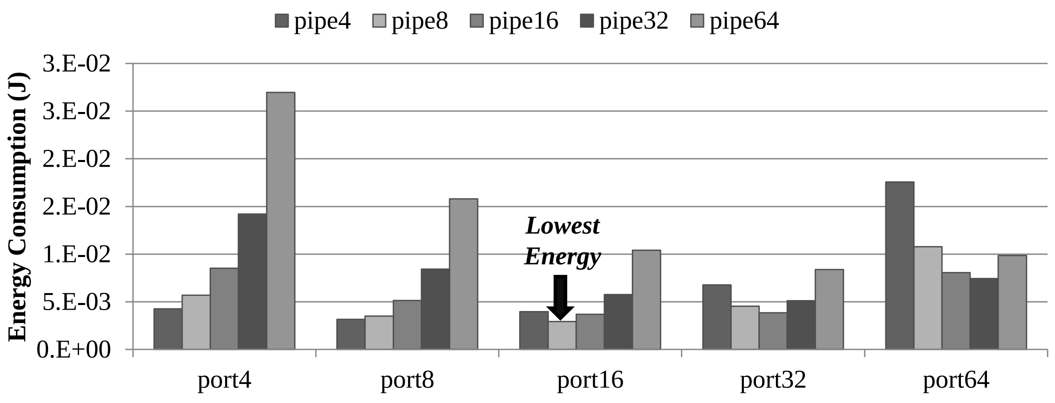


Figure 4.8: *Energy Consumption of* sphinx *at Different Numbers of VFUs and Cache Ports.*

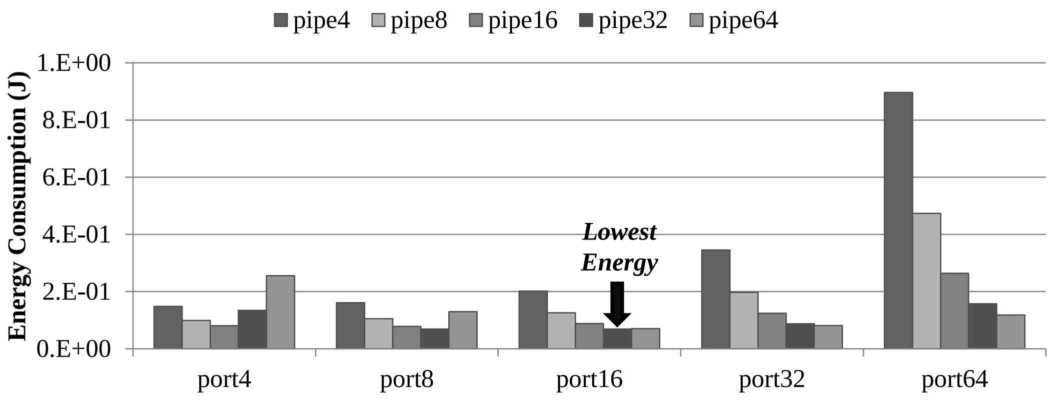


Figure 4.9: *Energy Consumption of* ray *at Different Numbers of VFUs and Cache Ports.*

performance improvement cannot always match the overheads on power consumption. As results, the most power-efficient configurations of each MMA are various. Therefore, MVPX should adjust the numbers of parallel pipelines in each VFU and cache ports for each MMA, and the accuracy of finding the most power-efficient configuration determines the benefit of the reconfiguration capability.

4.2.2 Requirements for Finding the Most Power-Efficient Configuration

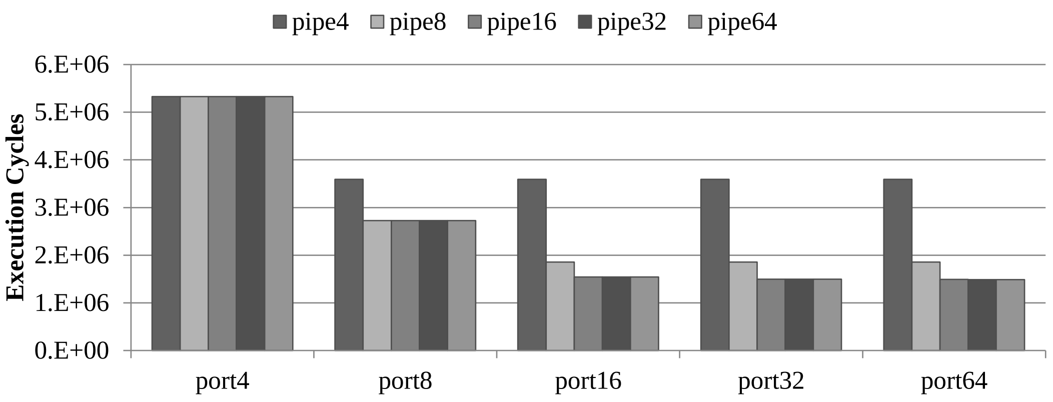


Figure 4.10: *Execution Cycles of* clip *at Different Numbers of VFUs and Cache Ports.*

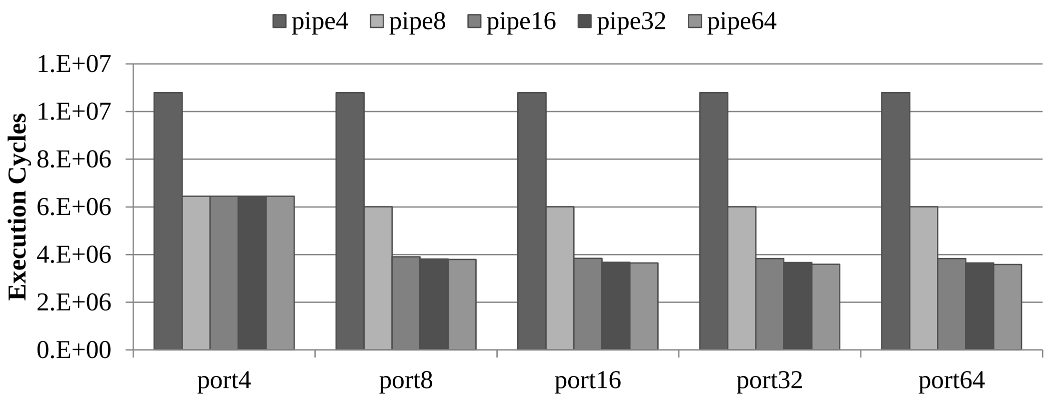


Figure 4.11: *Execution Cycles of* power *at Different Numbers of VFUs and Cache Ports.*

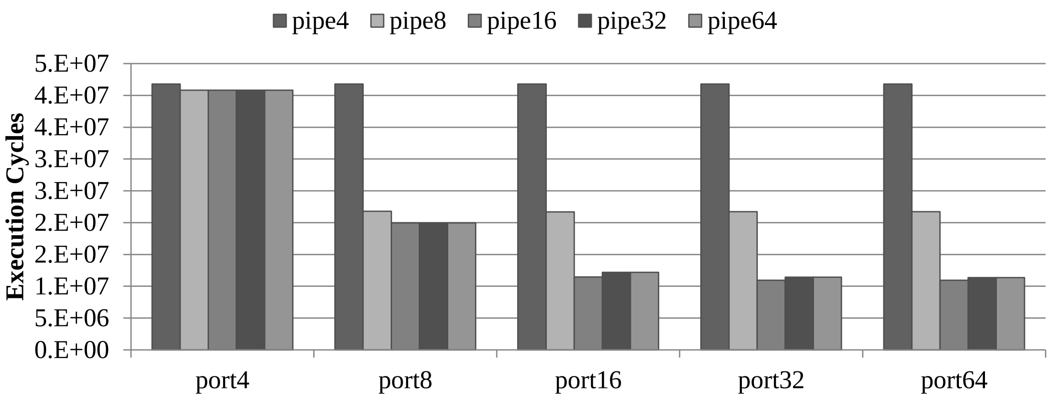


Figure 4.12: *Execution Cycles of* fft *at Different Numbers of VFUs and Cache Ports.*

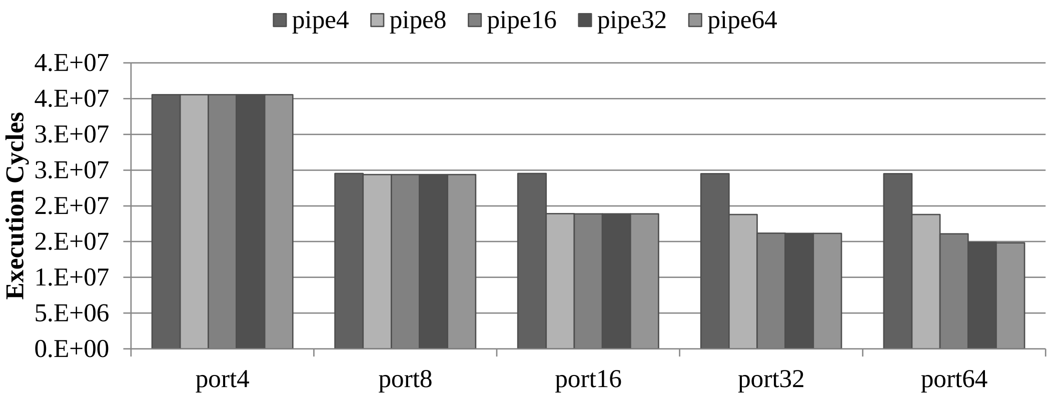


Figure 4.13: *Execution Cycles of* face *at Different Numbers of VFUs and Cache Ports.*

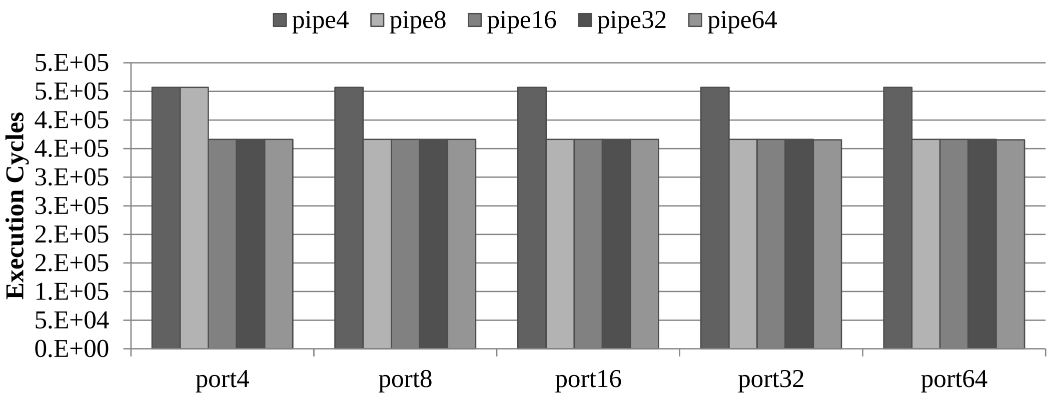


Figure 4.14: *Execution Cycles of* vips *at Different Numbers of VFUs and Cache Ports.*

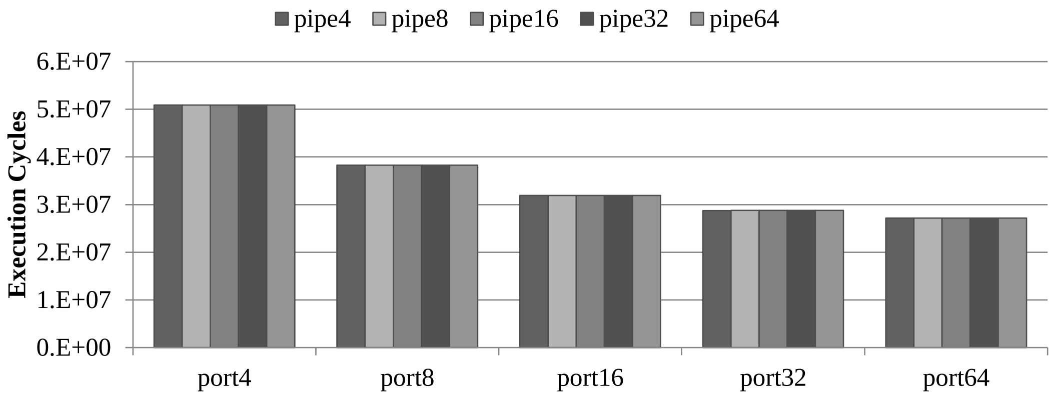


Figure 4.15: *Execution Cycles of* MxM *at Different Numbers of VFUs and Cache Ports.*

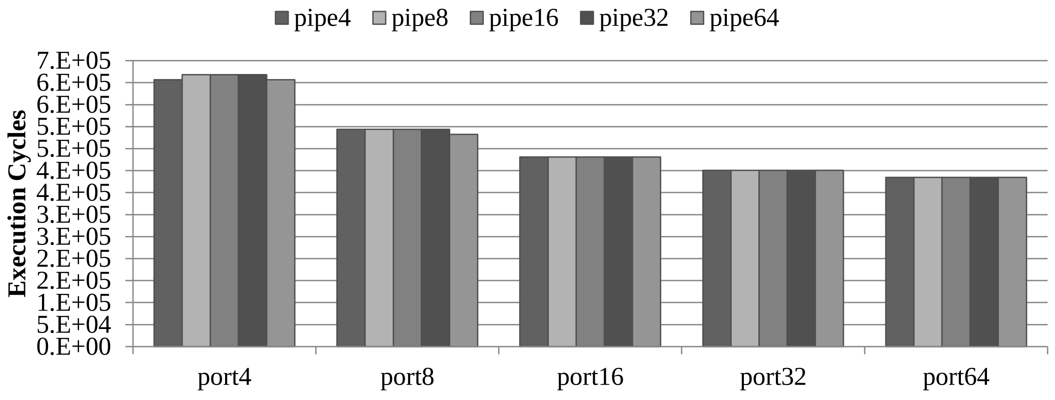


Figure 4.16: *Execution Cycles of* VxM *at Different Numbers of VFUs and Cache Ports.*

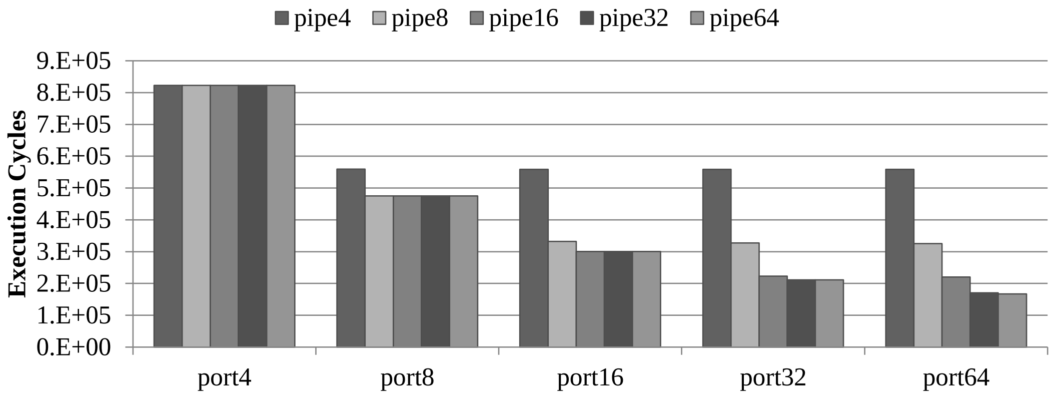


Figure 4.17: *Execution Cycles of* sphinx *at Different Numbers of VFUs and Cache Ports.*

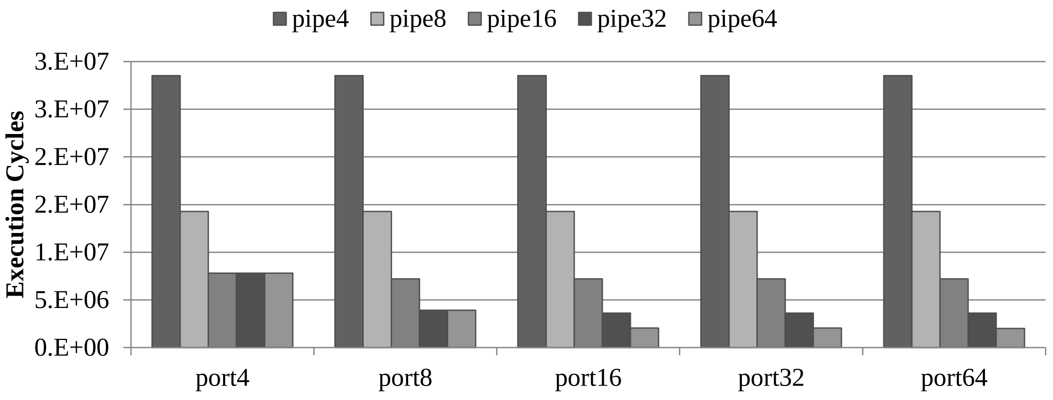


Figure 4.18: *Execution Cycles of* ray *at Different Numbers of VFUs and Cache Ports.*

The execution time and power consumption are two important factors to affect the power efficiency. Regarding the power consumption, the peak power consumption of MVXP in each configuration is used, which is independent to MMAs. This is means that the most power-efficient configuration of MMAs is different with each others because the numbers of hardware resources have different impacts on different MMAs. Therefore, it is possible to obtain the requirements to PPoM by investigating the execution cycles of each MMA on MVPX with various configurations.

Figures 4.10 to 4.18 show the execution cycles of MMAs on MVPX with various configurations. The results show that the performance of MMAs does not change across different configurations in some cases. For example, in Figure 4.10, increasing the number of parallel pipelines of VFUs in the case of four cache ports does not reduce the execution cycles of the benchmark program *clip*. This is because data transfer performance is the bottleneck at that time. Therefore, the increase in the number of parallel pipeline makes no sense, while increasing the number of cache ports is the right way to improve the performance of the benchmark program *clip* in the case of four cache ports. Such a result means that it is necessary to estimate the performance bottleneck in order to obtain a hint that is used to detect the shortage of hardware resources for a certain MMA. In this way, the unnecessary estimation can be avoided.

Furthermore, Figures 4.10 to 4.18 also show that, in some cases, even though the hardware resources are increased, the execution time is ideally reduced. In addition to computing and data transfer performances, the time for waiting for dependent data is a key factor that significantly affects the performance on executing MMAs. There are a lot of reasons that lead to a long waiting time due to dependent data, such as cache misses of MVP-cache, the exposure of the latency of VFUs due to short vectors, and so on. The long waiting time for data dependency will restrict the performance improvement even with the increase in the number of hardware resources. Therefore, it is necessary to estimate the execution cycles with the consideration of the data dependency waiting time.

4.2.3 Related Work

Several researches have shown how to find the most power-efficient configuration on a vector processor. Shoji [68] has obtained the most power-efficient configuration by establishing Pareto frontiers for some MMAs and finding the most power-efficient configuration on the Pareto frontiers. Sato [67] has proposed a greedy algorithm based on the roofline model in order to find the proper number of vector cores for for highly power-efficient execution. Both of the two researches obtain the most power-efficient configuration by executing the MMAs several times by using various configurations. Although both of the researches could find the most power-efficient configuration, it is quite time-consuming on the simulation. Hence, they cannot be used at runtime.

To reduce the execution time for configuration searching, performance and power consumption need to be estimated without executing the program [69] [71]. In recent researches, many performance anlytical models are based on machine learning or regression analysis. Martinez et al. [72] have proposed a coordinated hardware resource management framework for chip multiprocessors by using a performance estimation model based on machine learning. Matthew et al. [73] have proposed an analytical model by using regression analysis in order to find the minimized number of threads, which have no performance degradation. The advantage of these approaches is that they can coordinate a large variety of parameters together and anticipate the performance when they changed. These approaches need a training stage in order to improve the prediction accuracy. The training stage still takes a long time to achieve a high accuracy. Moreover, since these approaches cannot predict what is the performance bottleneck of an application, they have to exhaustively search the possible combinations of all parameters.

Michaud et al. [70] have observed that the issue IPC and the number of instructions in the instruction issue queue follow a Power-Law relationship [74]. Tejas et al. [75] have exploited this relationship and proposed an analytical model with concerns about penalties of data cache misses, instruction cache misses and branch prediction misses. This model is considered under the assumption that the dominant factor for the processor performance is the issue IPC. This is true in the case where a superscalar processor executes an application with massive instruction-level parallelism. However, in the case of a vector processor, the performance is often determined by a performance bottleneck due to the shortage of a certain kind of hardware resources, such as the number of parallel pipelines in each VFU. Therefore, this model is not appropriate for MVPX because it cannot predict the performance bottleneck of MVPX for a given application. Hence, the exhaustive search of possible combinations of all parameters takes a long time to find a power-efficient configuration.

4.3 A Performance-Power Optimization Method for MVPX

4.3.1 Searching Method of PPoM

In order to find the most power-efficient configuration as quickly as possible, PPoM adopts the greedy searching method instead of exhaustive searching method to find the most power-efficient configuration in order to reduce the unnecessary estimations. PPoM examines the performance bottleneck of a certain configuration, and only estimates the execution cycles and power consumption of the configuration, which could remove the current performance bottleneck. In this way, a large amount of unnecessary estimation could be avoided.

PPoM consists of three main steps. In the first step, the performance bottleneck of an MMA is examined for the current configuration, and the number of the hardware resource is increased, which enables to remove the performance bottleneck. In the second step, the performance bottleneck and the execution cycles is revised by using an analytical model of VPMs, and the power consumption of the revised configuration is obtained by using a power consumption matrix. In the third step, the energy consumption of the current configuration is compared to that of the revised configuration. If the energy consumption of the revised configuration is lower than that of the current configuration, the revised configuration would be used for the next evaluation steps. These three steps are iterated until a most power-efficient configuration is obtained and hence the revised configuration is expected to consume more energy than the current one.

4.3.2 Analytical Model of MVPX based on the Issue Queues

Issue Queues of MVPX

There are two issue queues in the MVPX as shown in Figure 2.5. One is used for vector memory instructions called VMIQ. The other one is used for vector arithmetic instructions called VAIQ. When all the operands of a vector instruction are ready, the vector instruction will be dispatched into VMIQ or VAIQ. The number of instructions dispatched into the issue queue per cycle is called *enqueue throughput*. The *enqueue throughput* reflects the operands waiting time of a vector instruction. The longer the operand waiting time is, the lower the *enqueue throughput* achieves. Meanwhile, vector instructions in VMIQ and VAIQ wait for available cache ports and VFUs, respectively. If the hardware resources become available, the the instruction will be issued. The number of instructions issued from the issue queue per cycle is called *dequeue throughput*. The *dequeue throughput* implies the throughput to operate a vector instruction. The more hardware resources MVPX employs, the higher *dequeue throughput* achieves.

Estimation of Performance Bottleneck

The estimation of performance bottleneck for a certain MMA means to estimate what kind of hardware resources, in terms of VFUs and cache ports, are deficient for the MMA. If a deficiency occurs, the corresponding issue queue will be filled up because *dequeue throughput* cannot catch up with the *enqueue throughput*. Therefore, the performance bottleneck can be estimated by comparing the *enqueue* and *dequeue* throughputs. If the *dequeue throughput* of VMIQ is higher than the *enqueue throughput* of VMIQ, the data transfer performance is the performance bottleneck. If the *dequeue throughput* of VAIQ is higher than the *enqueue throughput* of VAIQ, the computing performance is the performance bottleneck. In this way, the performance bottleneck can be estimated by using *enqueue* and *dequeue throughputs* of issue queues of MVPX.

Estimation of Execution Cycles

The estimation method of execution cycles by using dequeue is described as follows. As mentioned above, the performance bottleneck makes a certain issue queue filled up. This means that the corresponding hardware resource is used all the time during the execution. Therefore, the execution cycles are approximated with the utilization cycles of the hardware resource, which has the performance bottleneck.

The utilization cycles of the hardware resource which is the bottleneck for the MMA can be Equation (4.1).

$$Cycles_{utilization} = I_{issued} \times CPI_{issued}, \tag{4.1}$$

where $Cycles_{utilization}$ denotes utilization cycles of the hardware resource which is the bottleneck, and I_{issued} and CPI_{issue} denote the number of instructions and cycles per instruction (issue CPI) for the hardware resource, respectively. Since the dequeue throughput is the number of instructions that can be issued to hardware resources in each cycle (issue IPC), CPI_{issue} is the reciprocal of the dequeue throughput as expressed in Equation 4.2.

$$CPI_{issued} = \frac{1}{IPC_{Issued}} = \frac{1}{Throughput_{dequeue}}, \tag{4.2}$$

By using Equations (4.1) and (4.2), the execution cycles of a MMA can be calculated by using dequeue throughpurt.

Changes of Enqueue and Dequeue Throughputs the New Hardware Configuration.

In order to estimate the performance bottleneck and execution cycles after increasing the amount of hardware resources, it is necessary to estimate the enqueue and dequeue throughputs of the new hardware configuration. When the amount of hardware resources is increased, the throughput to operate a vector instruction will be increased. Therefore, *dequeue throughput* will be increased as follows. In the case of the increase in the number of parallel pipelines in each VFU,

$$DA_{revised} = \frac{P_{revised}}{P_{revised}} \times DA_{before}, \tag{4.3}$$

$$DM_{revised} = DM_{current}, \tag{4.4}$$

where $DA_{revised}$, $DM_{revised}$ and $P_{revised}$ mean *dequeue throughput* of VAIQ, *dequeue throughput* of VMIQ, the number of parallel pipelines after the increase in parallel pipelines, respectively. $DA_{current}$, $DM_{current}$ and $P_{current}$ mean *dequeue throughput* of VAIQ, *dequeue throughput* of VMIQ, the number of parallel pipelines before the increase in parallel pipelines, respectively.

In the case of the increase in the number of cache ports,

$$DA_{revised} = DA_{current}, \tag{4.5}$$

$$DM_{revised} = \frac{CP_{current} \times (1 - HR) + CP_{revised} \times HR}{CP_{current}} \times DM_{before}, \tag{4.6}$$

where $CP_{revised}$ and $CP_{current}$ denote the numbers of parallel pipelines after and before the increase in the number of cache ports, respectively. HR means the cache hit rate of the MMA.

Moreover, the increase in the number of hardware resources also means the increase in the throughput to remove data dependency. Therefore, the *enqueue throughput* will be increased as follows.

In the case of the increase in the number of parallel pipelines in each VFU,

$$EA_{revised} = \frac{P_{current} \times (1 - K) + P_{revised} \times K}{P_{current}} \times EA_{current}, \tag{4.7}$$

$$EM_{revised} = \frac{P_{current} \times (1 - S) + P_{revised} \times S}{P_{current}} \times EM_{current}, \tag{4.8}$$

where $EA_{revised}$, $EM_{revised}$ and $EA_{current}$, $EM_{current}$ mean *enqueue throughput* of VAIQ, *enqueue throughput* of VMIQ after and before the increase in number of parallel pipelines, respectively. K denotes a ratio of the number of vector arithmetic instructions that are data dependent on the vector arithmetic instruction to the total number of vector arithmetic instructions. S denotes a ratio of vector store instructions to total vector memory instructions.

In the case of the increase in the number of cache ports,

$$EA_{revised} = \frac{CP_{current} \times (1 - AL \times HR) + CP_{revised} \times AL \times HR}{CP_{current}} \times EA_{current}, \quad (4.9)$$

$$EM_{revised} = \frac{CP_{current} \times (1 - ML \times HR) + CP_{revised} \times ML \times HR}{CP_{current}} \times EM_{current}, \quad (4.10)$$

where AL denotes a ratio of the number of vector arithmetic instructions that are data dependent on the vector load instruction to the total number of vector arithmetic instructions. ML denotes a ratio of the number of vector memory instructions that are data dependent on the vector load instruction to the total number of vector memory instructions.

4.3.3 Estimation of Power Consumption

In order to find the most power-efficient configuration for each MMA, the power consumption of MVPX in different configurations should be discussed. In this chapter, peak power consumption is considered as the power consumption of hardware resources. peak power consumption refers to the maximum amount of heat generated by the processor, processor, for which the cooling system in a computer is required. peak power consumption is used for the evaluation in this chapter because it is necessary to avoid the overheat due to a wrong estimation. peak power consumption can be approximated by the sum of peak dynamic power consumption and static power consumption. The values of peak power consumption of various parameters can be known at the processor design stage.

Therefore, it can be assumed that such a table is stored in the MVPX.

It can be expressed as a 2-D array named *TDP[M][N]*, where M and N are the maximum number of parallel pipelines and cache ports, respectively. The power consumption of a certain configuration can be obtained by specifying the numbers of parallel pipelines and cache ports as shown in Equation (4.11).

$$Power = TDP[P][CP], \tag{4.11}$$

where P and CP are the specified numbers of parallel pipelines and cache ports.

4.3.4 Summary of PPoM

Figure 4.19 summarises the flow of PPoM. Overall, PPoM will iterate several times to estimate the performance bottleneck and execution cycles. This is because the performance bottleneck may be changed when the amount of hardware resource is increased. In order to start up PPoM, a MMA should be executed once to collect the initial information about the MMA. The initial information includes cache hit rate, enqueue throughput and dequeue throughput at the base configuration which is the configuration at the lowest power consumption. The initial information also contains K, S, ML and AL, which are the dependency ratios of instructions used in Equations (4.3) to (4.10).

In detail, firstly, the amount of hardware resources should be increased according to the performance bottleneck. Then, the enqueue throughput and dequeue throughput are estimated using Equations (4.3) to (4.10). After that, power consumption is estimated according to Equations (4.11). Then, the performance bottleneck after the increase in the number of hardware resources should be estimated. According to the bottleneck, the execution cycles can be generated by using Equations (4.1) and (4.2). Since PPoM aims to find the most power-efficient configuration, the performance improvement is compared with the increase of power consumption. If the performance improvement is higher than the increase of power consumption, PPoM will try the configuration so as to remove

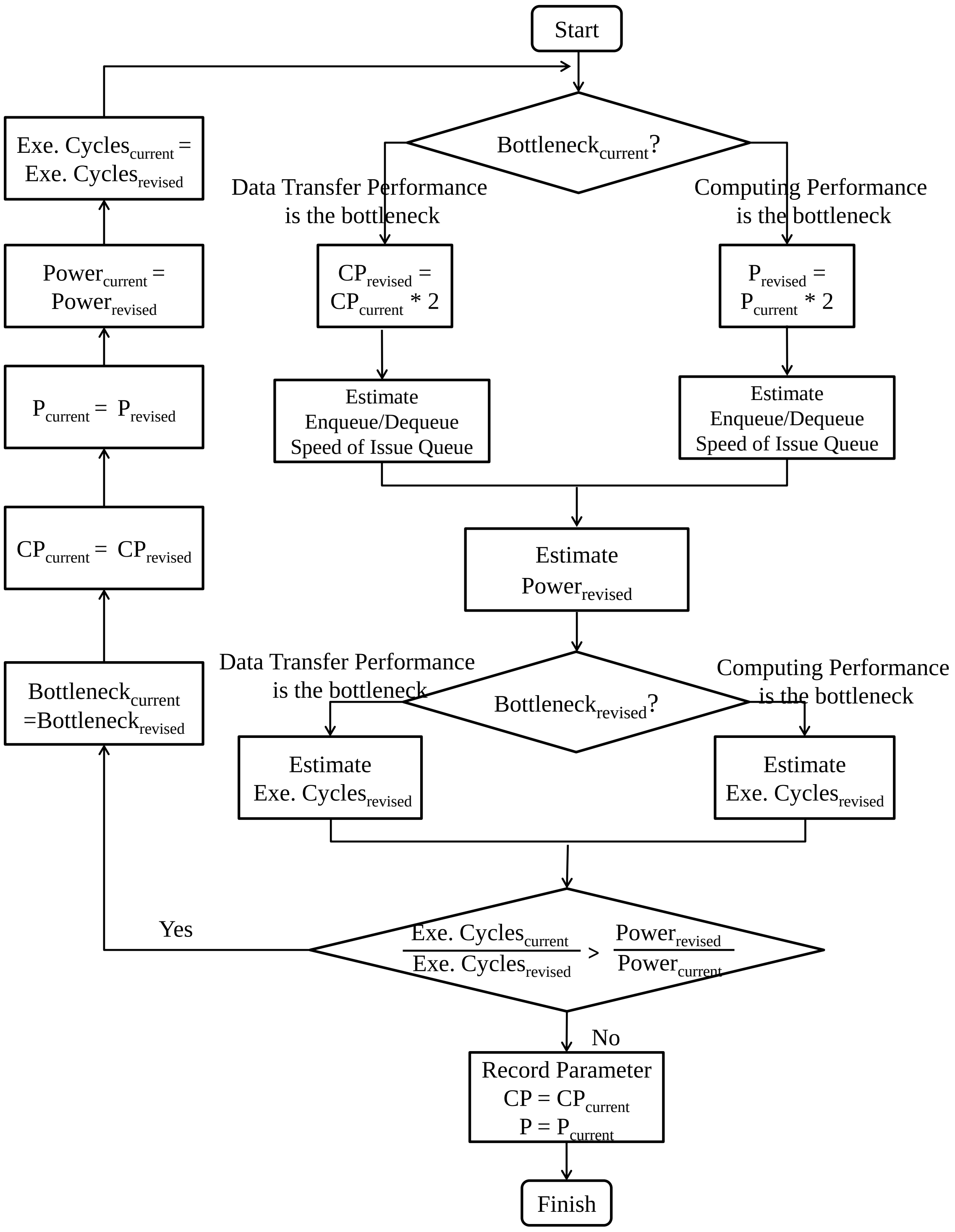

Start
Bottleneck$_{current}$?
Data Transfer Performance is the bottleneck
Computing Performance is the bottleneck
CP$_{revised}$ = CP$_{current}$ * 2
P$_{revised}$ = P$_{current}$ * 2
Estimate Enqueue/Dequeue Speed of Issue Queue
Estimate Enqueue/Dequeue Speed of Issue Queue
Estimate Power$_{revised}$
Bottleneck$_{revised}$?
Data Transfer Performance is the bottleneck
Computing Performance is the bottleneck
Estimate Exe. Cycles$_{revised}$
Estimate Exe. Cycles$_{revised}$
Exe. Cycles$_{current}$ / Exe. Cycles$_{revised}$ > Power$_{revised}$ / Power$_{current}$
Yes
No
Bottleneck$_{current}$ =Bottleneck$_{revised}$
CP$_{current}$ = CP$_{revised}$
P$_{current}$ = P$_{revised}$
Power$_{current}$ = Power$_{revised}$
Exe. Cycles$_{current}$ = Exe. Cycles$_{revised}$
Record Parameter CP = CP$_{current}$ P = P$_{current}$
Finish

Figure 4.19: *Flowchart of PPoM.*

the current bottleneck. Otherwise, PPoM records the configuration before increasing the amount of hardware resources as the most power-efficient configuration.

4.4 Performance Evaluations

4.4.1 Experimental Methodology

Table 4.1: *Baseline Configuration of MVPX for PPoM.*

Parameters	Value
Process Technology	32 nm
Processor Frequency	1GHz
Vector ALU Pipeline Latency	10 cycles
Vector Multiplier Pipeline Latency	15 cycles
Vector Division Pipeline Latency	20 cycles
Number of Vector Lanes	4
Number of Architectural Vector Registers	16
Number of Physical Vector Registers	96
Entries per Vector Register	256 entries
The size of VMIB and VAIB	128 entries
The size of Vector Load and Store Queue	512 entries
Number of Cache Ports	4
Cache Capacity	2 MB
Cache Line Size of MVP-cache	8 bytes
MVP-cache Access Latency	20 cycles
Number of Sub-caches	8
Number of Data Arrays in a Sub-cache	32
Memory Bandwidth	32 bytes/cycle
Main Memory Latency	100 cycles

PPoM is integrated into the simulator of MVPX. As mentioned in Section 4.3.4, PPoM should use the execution information which needs to be collected using a baseline configuration of MVPX. The baseline configuration of MVPX is summarized in Table 4.1. The numbers of cache ports and parallel pipelines in each VFU are set to be 4. For PPoM, when the amount of hardware resources needs to be increased, the number of cache ports or parallel pipelines is doubled. The numbers of cache ports and parallel pipelines in each VFU range from 4 to 64. This is because when the amount of hardware resources is increased from 64 to 128, the peak power consumption of would become more than two times higher as shown in Table 4.2, while the performance improvement is twofold. The

Table 4.2: *Table of peak power consumption (W) of MVPX in Different Configurations.*

	port4	port8	port16	port32	port64
pipe4	5.18	5.63	7.06	12.09	31.41
pipe8	6.91	7.36	8.78	13.82	33.14
pipe16	10.36	10.81	12.23	17.27	36.59
pipe32	17.26	17.71	19.14	24.17	43.49
pipe64	32.78	33.23	34.66	39.69	59.01

benchmark programs used in this chapter are the same as those in Chapters 2 and 3.

4.4.2 Evaluation Results

In order to evaluate the effectiveness of PPoM, this chapter first compares the most power-efficient configuration found by PPoM with that of the best configuration. Then, this chapter compares the energy consumptions of various configurations including the configuration found by the proposed PPoM, the lowest power consumption configuration, the highest performance configuration, the most power-efficient configuration for each MMA and the most power-efficient configuration averaged in all MMAs, in order to show that the configuration found by PPoM achieves a high power efficiency. At last, this chapter compares the numbers of simulation and estimation of PPoM with those of conventional approaches to confirm that PPoM could find the most power-efficient configuration more quickly.

Accuracy of PPoM

Table 4.3 shows the most power-efficient configurations found by PPoM and by exhaustively simulating all possible combinations. The results from exhaustively simulating all possible combinations can be considered as the correct answer of the best configuration. In this evaluation, seven out of nine benchmark programs can obtain the best configuration by using PPoM. For the benchmark programs of *fft* and *sphinx*, PPoM is failed

Table 4.3: *Comparison of the Most Power-Efficient Configurations Found by PPoM and Exhaustively Simulating the All Possible Combination.*

Benchmarks	Proposal	Simulation	Same Configuration
face	pipe4 port8	pipe4 port8	Yes
vips	pipe4 port4	pipe4 port4	Yes
clip	pipe8 port16	pipe8 port16	Yes
power	pipe16 port8	pipe16 port8	Yes
MxM	pipe4 port8	pipe4 port8	Yes
VxM	pipe4 port8	pipe4 port8	Yes
ray	pipe32 port16	pipe32 port16	Yes
sphinx	pipe4 port8	pipe8 port16	No
fft	pipe4 port4	pipe16 port16	No

because there is no significant performance improvement unless the numbers of parallel pipelines and cache ports are increased at the same time. Such a situation occurs because the computing performance and data transfer performance are well balanced in a certain configuration. However, since PPoM can only increase one kind of hardware resources each time, it fails to find the lowest energy consumption for the benchmark programs of *fft* and *sphinx*.

Energy Consumption of the Configuration Found by PPoM

In order to show that MVPX can obtain a high power efficiency by using the configuration found by PPoM, the energy consumptions of MVPX using various configurations are evaluated. The evaluated configurations include the configuration found by PPoM along with the configurations with the lowest power consumption, the highest power consumption and the lowest energy consumption.

Figure 4.20 shows the evaluation results normalized by the energy consumption of the lowest energy consumption. Averagely, the energy consumption of the configuration found by the proposal is only 7% higher than that of the lowest energy consumption. This is because PPoM can find the optimal configurations of most benchmark programs except for *fft* and *sphinx*. Even in the case of the benchmark programs *fft* and *sphinx*,

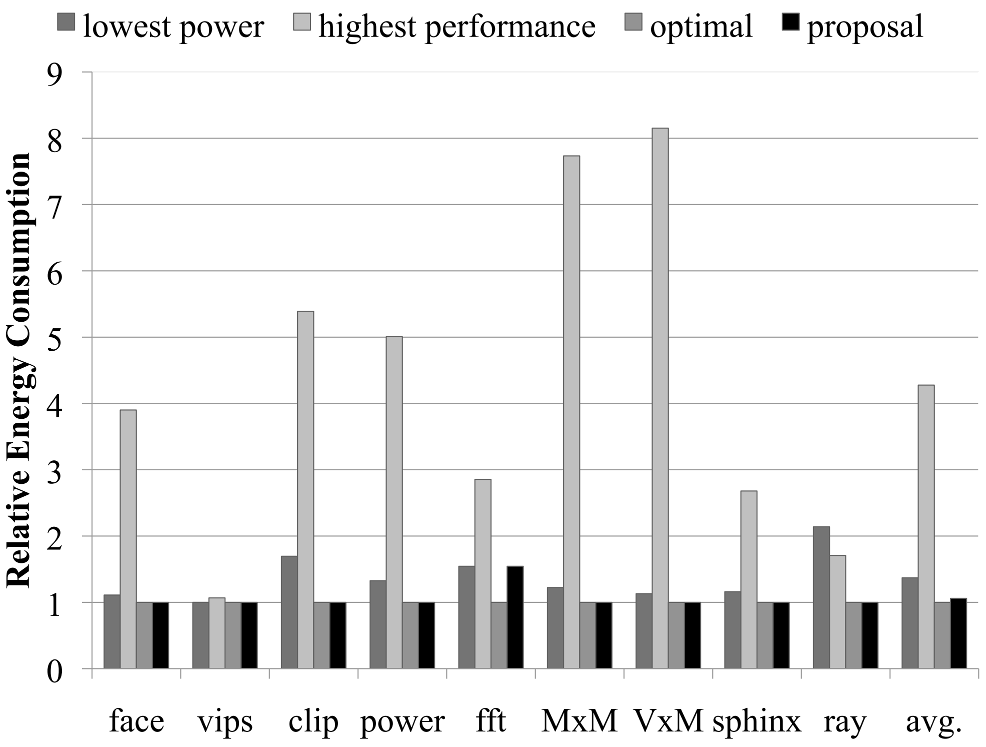


Figure 4.20: *Comparison of Energy Consumption of the Configuration Found by PPoM.*

configurations found by the proposed PPoM can still achieves the second lowest energy consumption. Therefore, the configuration found by the proposed PPoM enables MVPX to obtain a high power efficiency for each MMA.

The number of Simulation and Estimation

In order to show the quickness of finding the high power efficiency configuration, the numbers of simulation and estimation of the proposal are compared with those of conventional approaches mentioned in Section4.2.3. Table 4.4 summarizes the comparison results. The exhaustive method is to execute or simulate all possible combinations of hardware resources [68]. Greedy with simulation method is to use the greedy algorithm and roofline model to find the configuration [67]. Since there are five parameters for cache ports and five parameters for parallel pipelines, the number of all possible combinations of

Table 4.4: *Comparison of the number of Simulation Times and Estimation Times.*

Approaches	The Number of Estimation Times	The Number of Simulation Times
Proposal	1 to 10	1
Simulation Method [68]	0	25
Greedy with Roofline [67]	0	1 to 10
Analysis Model [70]	25	1

Table 4.5: *The number of Estimation Times for each MMA.*

Benchmarks	The Number of Estimation Times (Proposal) Configuration
face	2
vips	1
clip	4
power	4
MxM	2
VxM	2
sphinx	2
ray	6
fft	1

parameters and the maximum number of searching times by using greedy algorithms are 25 and 10, respectively. The analysis model [70] is to establish a performance estimation model to substitute for the real execution of a certain application. Therefore, the number of estimation times of analysis method is the same as that of simulation times of the exhaustive method. For PPoM, it needs to obtain some application information in advance of searching an appropriate configuration using the greedy algorithm. Therefore, PPoM needs to simulate the MMA once, and perform the estimation at most 10 times. The numbers of the estimation are listed in Table 4.5. Through the comparison, it is shown that PPoM achieves the least simulation times and estimation times. Consequently, PPoM can fast find the high power efficiency configuration.

4.5 Conclusions

This chapter proposes and evaluates PPoM in order to improve the power efficiency by matching the different hardware requirement of MMAs. PPoM contains a analytical model of MVPX by using enqueue throughput and dequeue throughput to estimate the performance and power consumption of MVPX in different configurations. The analytical model could be also used to estimate the performance bottleneck of MMAs. Base on the thought of the greedy algorithm, PPoM increases the amount of hardware resource so as to remove the performance bottleneck. The evaluation results show that PPoM could obtain the most power-efficient configuration for seven out of nine MMAs by uses less estimation and simulation times than conventional approaches.

Although the numbers of cache ports and parallel pipelines are two most important parameters to influence performance and power consumption of MVPX, the importance of other parameters such as MVL, MVP-cache capacity and so on cannot be ignored. Therefore, as the future work of this chapter, it is necessary to establish a PPoM that enables to consider various kinds of parameters at the same time.

Chapter 5

Conclusions

This dissertation explores the design space of vector architectures in order to design a vector extension to acceleration a wide range of MMAs at a high power efficiency. Although the vector architecture is considered as a potential high power efficiency approach to media processing, there are still several issues for vector architectures to efficiently execute MMAs.

First, the conventional vector architecture cannot execute MMAs with short vectors efficiently. Second, conventional multi-banked cache memories for vector architectures cannot achieve a high data transfer performance and low power consumption at the same time for MMAs. Third, conventional methods to find the most power-efficient configuration cost too much time to find the most power efficient configuration of the vector architecture for MMAs.

In order to overcome these problems, the design space of the vector architecture is explored to find the power efficient solution for a wide range of MMAs.

Several solutions have been proposed in this dissertation to address these issues, including:

- OVPM: OVPM could effectively reduce the stalls due to short vectors.
- MVP-cache: MVP-cache adopts a single tag array associated with multiple data

arrays. Such an organization could effectively reduce the power consumption of tag array, and improve the short vector data transfers.

- PPoM: enables to reduce the simulation times and found the optimal configuration in low power consumption

In Chapter 2, the instruction issue policies of vector architecture have been explored, and OVPM has been proposed to improve the computational efficient by reducing the pipeline stalls. The objective of Chapter 2 is to improve the computational efficiency of MMAs with short vectors. In order to achieve the objective, OVPM has been proposed. By using OVPM, even though a certain vector instruction is stalled, the subsequent vector instruction enables to be executed, overtaking the stalled vector instruction. In this way, the pipeline stalls can be reduced, which leads to the improvement of computational efficiency. As a result, OVPM could achieve 3.25x higher computational efficiency than IVPM only with a 7% power increase. Therefore, MVPX should adopt an out-of-order vector issue policy in order to improve the power efficiency.

In Chapter 3, the cache line sizes of multi-banked cache memory have been explored, and a multi-banked cache memory for MVPX, called MVP-cache has been proposed. The objective of Chapter 3 is to enable multi-banked cache memory to efficiently transfer the short vector with low power consumption on tag arrays. MVP-cache associates single tag array with multiple data array. It is possible to achieve a high data transfer performance for short vectors and a low power consumption on tag array. This is because the relationship between the size of cache lines and the number of tags is decoupled. As a result, MVP-cache can achieve a comparable performance with the other competitive cache organizations, while the energy consumption of MVP-cache is smaller than those of the other. MVP-cache can also improve the power efficiency of MVPX

In Chapter 4, the numbers of parallel pipelines in each VFU and cache ports have been explored, and a PPoM has been proposed in order to find the power-efficient configuration with a short estimation time. PPoM contains a performance estimation model of the vector architecture to estimate the execution cycles and performance bottleneck of a certain MMA. The numbers of parallel pipelines and cache ports have been searched base on the greedy algorithm. By using the proposed PPoM, the optimal or sub-optimal configuration can be found in a short time. As a result, PPoM could obtain the most power-efficient configuration for seven of nine MMAs, which takes less estimation and simulation time than conventional approaches. Therefore, the proposed PPoM can also improve the power efficiency of MVPX.

These proposed solutions have solved the issues on different layers of the existing vector architecture for MMAs. Enhanced with these proposed solutions, it is possible for MVPX to accelerate a wide range of MMAs with a high power efficiency.

There are two main future works for this dissertation. First, 3D integrated MVPX should be considered. The implementation of MVPX is not considered in this dissertation. As shown in Chapter 3, the crossbar between OVPM and MVP-cache consume a large power. In order to further improve the power efficiency of MVPX, a 3D stacked implementation of crossbars, MVP-cache and OVPM should be considered. Second, a PPoM that can consider more amounts of hardware resources should be considered. Although the numbers of cache ports and parallel pipelines are two most important parameters to influence performance and power consumption of MVPX, the importance of other parameters such as MVL, MVP-cache capacity and so on cannot be ignored. Therefore, it is necessary to establish a PPoM that enables to consider various kinds of parameters at the same time.

References

[1] N. Sei and M. Shuichi, "Development of Real-time Encoder for 8K Ultra-high Definition Ttelevision ," in *Proceedings of Intelligent Signal Processing and Communication Systems, 2010 International Symposium*, ser. ISPACS '10, 2010, pp. 1–4.

[2] International Telecommunication Union: ITU, "BT.2020 : Parameter Values for Ultra-high Definition Television Systems for Production and International Programme Exchange," http://www.itu.int/rec/R-REC-BT.2020-0-201208-I, 2012.

[3] M. Miura, K. Fudano, K. Ito, K. Aoki, H. Takizawa, and H. Kobayashi, "GPU Implementation of Phase-Based Stereo Correspondence and Its Application," in *2012 IEEE International Conference on Image Processing*, ser. ICIP2012, 2012, pp. 1697–1700.

[4] H. Park, Y. Park, and S. Mahlke, "Polymorphic Pipeline Array: A Flexible Multicore Accelerator With Virtualized Execution For Mobile Multimedia Applications," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 42, 2009, pp. 370–380.

[5] Y. Lin, H. Lee, M. Woh, Y. Harel, S. Mahlke, T. Mudge, C. Chakrabarti, and K. Flautner, "SODA: A Low-power Architecture for Software Radio," in *Proceedings of the 33rd annual international symposium on Computer Architecture*, ser. ISCA '06, 2006, pp. 89–101.

[6] Y. Lee, R. Avizienis, A. Bishara, R. Xia, D. Lockhart, C. Batten, and K. Asanović, “Exploring the tradeoffs between programmability and efficiency in data-parallel accelerators,” in *Proceedings of the 38th annual international symposium on Computer architecture*, 2011, pp. 129–140.

[7] N. Wu, M. Wen, W. Wu, J. Ren, H. Su, C. Xun, and C. Zhang, “Streaming HD H.264 Encoder on Programmable Processors,” in *Proceedings of the 17th ACM international conference on Multimedia*, ser. MM ’09, 2009, pp. 371–380.

[8] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin, R. Espasa, E. Grochowski, T. Juan, and P. Hanrahan, “Larrabee: A Many-core x86 Architecture for Visual Computing,” in *SIGGRAPH ’08: ACM SIGGRAPH 2008 Papers*, 2008, pp. 1–15.

[9] A. Lefohn, M. Houston, J. Andersson, U. Assarsson, C. Everitt, K. Fatahalian, T. Foley, J. Hensley, P. Lalonde, and D. Luebke, “Beyond Programmable Shading (Parts I and II),” in *SIGGRAPH ’09: ACM SIGGRAPH 2009 Courses*, 2009, pp. 1–312.

[10] J. Park, G. Bikshandi, K. Vaidyanathan, P. T. P. Tang, P. Dubey, and D. Kim, “Tera-scale 1d fft with low-communication algorithm and intel&reg; xeon phi&trade; coprocessors,” in *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC ’13. New York, NY, USA: ACM, 2013, pp. 34:1–34:12. [Online]. Available: http://doi.acm.org/10.1145/2503210.2503242

[11] J.-C. Chiu, Y.-L. Chou, and H.-Y. Tzeng, “A Multi-Streaming SIMD Architecture for Multimedia Applications,” in *Proceedings of the 6th ACM Conference on Computing frontiers*, 2009, pp. 51–60.

[12] Intel, “Intel 64 and IA-32 Architectures Optimization Reference Manual,” http://software.intel.com/en-us/avx/, 2011.

[13] K. Asanovic, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. Kubiatowicz, N. Morgan, D. Patterson, K. Sen, J. Wawrzynek, D. Wessel, and K. Yelick, "A View of the Parallel Computing Landscape," *Commun. ACM*, vol. 52, no. 10, pp. 56–67, 2009.

[14] D. Talla, L. K. John, and D. Burger, "Bottlenecks in Multimedia Processing with SIMD Style Extensions and Architectural Enhancements," *IEEE Transactions on Computers*, vol. 52, pp. 35–46, 2003.

[15] J. Gebis and D. Patterson, "Embracing and Extending 20th-Century Instruction Set Architectures," *IEEE Computer*, vol. 40, no. 4, pp. 68–75, April 2007.

[16] P. Yongjun, J. K. P. Jason, P. Hyunchul, and M. Scott, "Libra: Tailoring SIMD Execution using Heterogeneous Hardware and Dynamic Configurability," in *Proceedings of the 45nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 45, dec. 2012, pp. 84–95.

[17] A. Peleg, S. Wilkie, and U. Weiser, "Intel MMX for Multimedia PCs," *Comm. ACM*, 1997.

[18] S. Thakkar and T. Huff, "The Internet Streaming SIMD Extensions," *IntelTechnology Journal*, pp. 26–34, May 1999.

[19] J. Reinders and J. Reinders, "Avx-512 instructions," 2013.

[20] R. Dennard, F. Gaensslen, H.-N. Yu, V. LEO RIDEOVT, E. Bassous, and A. R. Leblanc, "Design of ion-implanted mosfet's with very small physical dimensions," *Solid-State Circuits Society Newsletter, IEEE*, vol. 12, no. 1, pp. 38–50, 2007.

[21] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proceedings of the 38th Annual International Symposium on Computer Architecture*, 2011, pp. 365–376.

[22] M. Taylor, "A landscape of the new dark silicon design regime," *Micro, IEEE*, vol. 33, no. 5, pp. 8–19, 2013.

[23] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach Fifth Edition.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.

[24] T. Watanabe, "Instruction Set Architecture for a Series of Vector Processors and Their Performance Evaluations," Ph.D. dissertation, University of Tohoku, 2005.

[25] T. Soga, A. Musa, Y. Shimomura, R. Egawa, K. Itakura, H. Takizawa, K. Okabe, and H. Kobayashi, "Performance evaluation of NEC SX-9 Using Real Science and Engineering Applications," in *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, 2009, pp. 1–12.

[26] D. Abts, A. Bataineh, S. Scott, G. Faanes, J. Schwarzmeier, E. Lundberg, T. Johnson, M. Bye, and G. Schwoerer, "The Cray BlackWidow: a Highly Scalable Vector Multiprocessor," in *SC '07: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing*, 2007, pp. 1–12.

[27] Y. Sato, R. Nagaoka, A. Musa, R. Egawa, H. Takizawa, K. Okabe, and H. Kobayashi, "Performance Tuning and Analysis of Future Vector Processors Based on the Roofline Model," in *Proceedings of the 10th workshop on MEmory performance: DEaling with Applications, systems and architecture*, ser. MEDEA '09, 2009, pp. 7–14.

[28] A. Musa, "High Performance Memory Architecture for Vector Processors," Ph.D. dissertation, University of Tohoku, 2009.

[29] S. Mohanty, V. K. Prasanna, S. Neema, and J. Davis, "Rapid design space exploration of heterogeneous embedded systems using symbolic search and multi-granular simulation," in *Proceedings of the Joint Conference on Languages, Compilers and Tools for Embedded Systems: Software and Compilers for Embedded Systems*, ser. LCTES/SCOPES '02. New York, NY, USA: ACM, 2002, pp. 18–27. [Online]. Available: http://doi.acm.org/10.1145/513829.513835

[30] K. Diefendorff and P. Dubey, “How Multimedia Workloads Will Change Processor Design,” *IEEE Computer*, vol. 30, pp. 43–45, 1997.

[31] G. M. Amdahl, “Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities,” in *AFIPS ’67 (Spring): Proceedings of the April 18-20, 1967, spring joint computer conference*, 1967, pp. 483–485.

[32] C. Bienia, S. Kumar, J. P. Singh, and K. Li, “The PARSEC Benchmark Suite: Characterization and Architectural Implications,” Princeton University, Tech. Rep., January 2008.

[33] M. Li, R. Sasanka, S. V. Adve, Y. Chen, and E. Debes, “The ALPBench Benchmark Suite for Complex Multimedia Applications,” in *Proceedings of the IEEE International Workload Characterization Symposium*, 2005.

[34] C. Kozyrakis, “Vector vs. Superscalar and VLIW Architectures for Embedded Multimedia Benchmarks,” in *Proceedings of the 35th Annual IEEE/ACM International Symposium on Microarchitecture*, 2002, pp. 283–293.

[35] C. Kozyrakis and D. Patterson, “Overcoming the limitations of conventional vector processors,” in *Proceedings of the 30th annual international symposium on Computer architecture.* ACM, 2003, pp. 399–409.

[36] R. Espasa and M. Valero, “Decoupled vector architectures,” in *High-Performance Computer Architecture, 1996. Proceedings., Second International Symposium*, 1996, pp. 281–290.

[37] Intel, “P6 Family of Processors Hardware Developer’s Manual,” Intel White Paper, 1998.

[38] R. E. Kessler, “The Alpha 21264 Microprocessor,” *IEEE Micro*, vol. 19, no. 2, pp. 24–36, Mar. 1999.

[39] T. Austin, E. Larson, and D. Ernst, “SimpleScalar: An Infrastructure for Computer System Modeling,” *Computer*, vol. 35, no. 2, pp. 59–67, 2002.

[40] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, “McPAT: an Integrated Power, Area, and Timing Modeling Framework for Multi-core and Manycore Architectures,” in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 42, 2009, pp. 469–480.

[41] Y. Yokoya, Y. Kudoh, T. Hayasaka, J. Traeff, H. Ritzdorf, and Y. Hayashi, “The Compilers and MPI Library for SX-9,” NEC Corporation, Tech. Rep., 2008.

[42] S. Nakazato, S. Tagaya, N. Nakagomi, T. Watai, and A. Sawamura, “Hardware Technology of the SX-9 (1): Main System,” *NEC TECHNICAL JOURNAL*, vol. 3, pp. 15–18, 2008.

[43] K. Umezawa, H. Hamaguchi, T. Takeda, T. Hosaka, M. Natori, and T. Nagata, “Packaging Technology of the SX-9,” *NEC TECHNICAL JOURNAL*, vol. 3, pp. 29–33, 2008.

[44] F. Rudolf, in *SC ’10: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, 2010.

[45] Y. Funaya, R. Egawa, H. Takizawa, and H. Kobayashi, “Cache Partitioning Strategies For 3-D Stacked Vector Processors ,” in *Proceedings of 3D Systems Integration Conference (3DIC), 2010 IEEE International*, 2010, pp. 1– 6.

[46] A. Musa, Y. Sato, T. Soga, K. Okabe, R. Egawa, H. Takizawa, and H. Kobayashi, “A Shared Cache for a Chip Multi Vector Processor,” in *MEDEA ’08: Proceedings of the 9th workshop on MEmory performance*, 2008, pp. 24–29.

[47] R. Espasa, F. Ardanaz, J. Emer, S. Felix, J. Gago, R. Gramunt, I. Hernandez, T. Juan, G. Lowney, M. Mattina, and A. Seznec, “Tarantula: a vector extension to

the alpha architecture," in *Proceedings of the 29th annual international symposium on Computer architecture*, 2002, pp. 281–292.

[48] C. Batten, R. Krashinsky, S. Gerding, and K. Asanovic, "Cache Refill/Access Decoupling for Vector Machines," in *MICRO 37: Proceedings of the 37th annual IEEE/ACM International Symposium on Microarchitecture*, 2004, pp. 331–342.

[49] A. Shahbahrami, B. Juurlink, and S. Vassiliadis, "A Comparison Between Processor Architectures for Multimedia Applications," in *Proc. 15th Annual Workshop on Circuits, Systems and Signal Processing*, 2004, pp. 138–152.

[50] C. Bienia, "Benchmarking modern multiprocessors," Ph.D. dissertation, Princeton University, January 2011.

[51] A. Musa, Y. Sato, T. Soga, R. Egawa, H. Takizawa, K. Okabe, and H. Kobayashi, "Effects of MSHR and Prefetch Mechanisms on an On-Chip Cache of the Vector Architecture," in *Parallel and Distributed Processing with Applications, 2008. ISPA '08. International Symposium on*, 2008, pp. 335–342.

[52] I. Kotera, K. Abe, R. Egawa, H. Takizawa, and H. Kobayashi, "Power-aware Dynamic Cache Partitioning for CMPs," *Transactions on high-performance embedded architectures and compilers III*, pp. 135–153, 2011.

[53] R. Espasa, M. Valero, and J. E. Smith, "Out-of-order vector architectures," in *Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture*, 1997, pp. 160–170.

[54] A. Seznec and R. Espasa, "Conflict-free accesses to strided vectors on a banked cache," *IEEE Trans. Comput.*, vol. 54, no. 7, pp. 913–916, Jul. 2005.

[55] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.

[56] K. Sewell, R. Dreslinski, T. Manville, S. Satpathy, N. Pinckney, G. Blake, M. Cieslak, R. Das, T. Wenisch, D. Sylvester, D. Blaauw, and T. Mudge, "Swizzle-switch networks for many-core systems," *Emerging and Selected Topics in Circuits and Systems, IEEE Journal on*, vol. 2, no. 2, pp. 278–294, 2012.

[57] H.-S. Lee, G. Tyson, and M. Farrens, "Eager writeback-a technique for improving bandwidth utilization," in *Microarchitecture, 2000. MICRO-33. Proceedings. 33rd Annual IEEE/ACM International Symposium on*, 2000, pp. 11–21.

[58] J. Stuecheli, D. Kaseridis, D. Daly, H. Hunter, and L. John, "Coordinating dram and last-level-cache policies with the virtual write queue," *Micro, IEEE*, vol. 31, no. 1, pp. 90–98, 2011.

[59] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Optimizing nuca organizations and wiring alternatives for large caches with cacti 6.0," in *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, 2007, pp. 3–14.

[60] M. B. Henry and L. Nazhandali, "From transistors to nems: Highly efficient power-gating of cmos circuits," *J. Emerg. Technol. Comput. Syst.*, vol. 8, no. 1, pp. 2:1–2:18, Feb. 2012. [Online]. Available: http://doi.acm.org/10.1145/2093145.2093147

[61] A. B. Kahng, S. Kang, T. Rosing, and R. Strong, "Tap: Token-based adaptive power gating," in *Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design*, ser. ISLPED '12. New York, NY, USA: ACM, 2012, pp. 203–208. [Online]. Available: http://doi.acm.org/10.1145/2333660.2333711

[62] M. Sato, Y. Tobo, R. Egawa, H. Takizawa, and H. Kobayashi, "A Flexible Insertion Policy for Dynamic Cache Resizing Mechanisms," in *Cool Chips XV (COOL Chips), 2013 IEEE*, 2013, pp. 1–3.

[63] C.-C. Yeh, K.-C. Chang, T.-F. Chen, and C. Yeh, "Maintaining performance on power gating of microprocessor functional units by using a predictive pre-wakeup strategy," *ACM Trans. Archit. Code Optim.*, vol. 8, no. 3, pp. 16:1–16:27, Oct. 2011. [Online]. Available: http://doi.acm.org/10.1145/2019608.2019615

[64] P.-H. Wang, C.-L. Yang, Y.-M. Chen, and Y.-J. Cheng, "Power gating strategies on gpus," *ACM Trans. Archit. Code Optim.*, vol. 8, no. 3, pp. 13:1–13:25, Oct. 2011. [Online]. Available: http://doi.acm.org/10.1145/2019608.2019612

[65] H. Matsutani, M. Koibuchi, D. Ikebuchi, K. Usami, H. Nakamura, and H. Amano, "Ultra fine-grained run-time power gating of on-chip routers for cmps," in *Proceedings of the 2010 Fourth ACM/IEEE International Symposium on Networks-on-Chip*, ser. NOCS '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 61–68. [Online]. Available: http://dx.doi.org/10.1109/NOCS.2010.16

[66] R. Das, S. Narayanasamy, S. K. Satpathy, and R. G. Dreslinski, "Catnap: Energy proportional multiple network-on-chip," *SIGARCH Comput. Archit. News*, vol. 41, no. 3, pp. 320–331, Jun. 2013. [Online]. Available: http://doi.acm.org/10.1145/2508148.2485950

[67] Y. Sato, "A program optimization strategy for vector architectures with cache mechanisms," Ph.D. dissertation, Tohoku University, 2012.

[68] N. Shoji, "Power-performance model for a media-oriented vector architecture," Master's thesis, Tohoku University, 2013.

[69] C. Ykman-Couvreur, P. A. Hartmann, G. Palermo, F. Colas-Bigey, and L. San, "Run-time resource management based on design space exploration," in *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, ser. CODES+ISSS '12. New York, NY, USA: ACM, 2012, pp. 557–566. [Online]. Available: http://doi.acm.org/10.1145/2380445.2380530

[70] T. S. Karkhanis and J. E. Smith, "A first-order superscalar processor model," in *Proceedings of the 31st Annual International Symposium on Computer Architecture*, ser. ISCA '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 338–. [Online]. Available: http://dl.acm.org/citation.cfm?id=998680.1006729

[71] G. Mariani, G. Palermo, V. Zaccaria, and C. Silvano, "Design-space exploration and runtime resource management for multicores," *ACM Trans. Embedded Comput. Syst.*, vol. 13, no. 2, p. 20, 2013.

[72] J. Martinez and E. Ipek, "Dynamic multicore resource management: A machine learning approach," *Micro, IEEE*, vol. 29, no. 5, pp. 8–17, 2009.

[73] M. Curtis-Maury, F. Blagojevic, C. D. Antonopoulos, and D. S. Nikolopoulos, "Prediction-based power-performance adaptation of multithreaded scientific codes," *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 10, pp. 1396–1410, Oct. 2008. [Online]. Available: http://dx.doi.org/10.1109/TPDS.2007.70804

[74] P. Michaud, A. Seznec, and S. Jourdan, "An exploration of instruction fetch requirement in out-of-order superscalar processors," *Int. J. Parallel Program.*, vol. 29, no. 1, pp. 35–58, Feb. 2001. [Online]. Available: http://dx.doi.org/10.1023/A:1026431920605

[75] S. Eyerman, L. Eeckhout, T. Karkhanis, and J. E. Smith, "A performance counter architecture for computing accurate cpi components," *SIGPLAN Not.*, vol. 41, no. 11, pp. 175–184, Oct. 2006. [Online]. Available: http://doi.acm.org/10.1145/1168918.1168880

Publications

Journal

1. Ye Gao, Masayuki Sato, Ryusuke Egawa, Hiroyuki Takizawa, and Hiroaki Kobayashi: MVP-cache: A Multi-Banked Cache Memory for Energy-Efficient Vector Processing of Multimedia Applications. *IEICE Transaction on Information and Systems)* [Conditional Acception] [Chapter 3]

Conference Papers

1. Ye Gao, Masayuki Sato, Ryusuke Egawa, Hiroyuki Takizawa, and Hiroaki Kobayashi: A Performance-Power Optimization Method for Vector Processing Mechanisms. *Cool Chips XV (COOL Chips)*, pp.1-3, 14-16 April 2014.] [Under Review] [Chapter 4]

2. Ye Gao, Naoki Shoji, Ryusuke Egawa, Hiroyuki Takizawa, and Hiroaki Kobayashi: Design and Evaluation of a Media-oriented Vector Processor with a Multi-banked Cache Memory. *The 11th IEEE Symposium on Embedded Systems for Real-Time Multimedia (ESTIMedia2013)*, 3-4 October 2012. [Chapter 3]

3. Ye Gao, Ryusuke Egawa, Hiroyuki Takizawa, and Hiroaki Kobayashi: An out-of-order vector processing mechanism for multimedia applications. *In Proceedings of the 9th conference on Computing Frontiers (CF '12)*, pp. 233-236, 15-17 April 2012. [Chapter 2]

4. Ye Gao, Naoki Shoji, Ryusuke Egawa, Hiroyuki Takizawa, and Hiroaki Kobayashi: A media-oriented vector architectural extension with a high bandwidth cache system. *Cool Chips XV (COOL Chips)*, pp.1-3, 18-20 April 2012. [Chapter 4]

5. Ye Gao, Ryusuke Egawa, Hiroyuki Takizawa, and Hiroaki Kobayashi: MVPX: A Media-oriented Vector Processing Mechanism. *The 7th internetional conference on High Performance and Embedded Architectures an Computers*, 2012. [Chapter 4]

6. Ye Gao, Ryusuke Egawa, Hiroyuki Takizawa, and Hiroaki Kobayashi: A Load-Forwarding Mechanism for the Vector Architecture in Multimedia Applications, *In the Proceeding of 13th EUROMICRO Conference on Digital System Design, September 1st to 3rd,* 2010 France. [Chapter 2]

7. Ye Gao, Ryusuke Egawa, Hiroyuki Takizawa, Hiroaki Kobayashi: An Out-of-order Vector Processing Mechanism for Multimedia Applications. *In the Proceeding of Summer United Workshops on Parallel, Distributed and Cooperative Processing 2010, August 3rd to 5th, 2010, Kanazawa, Japan* [Chapter 2]