博 士 論 文

Register Files of Superscalar Processors for Area and Energy Efficiency

（スーパスカラプロセッサのレジスタファイルの

面積・エネルギー効率向上に関する研究）

指 導 教 員

坂井 修一 教授

山田 淳二

Abstract

In the era of multicore processors, the area and energy efficiency of out-of-order superscalar processor cores is all the more important. It is because a multicore processor with more efficient cores can have a larger number of cores, and consequently more computational power. However, the region that includes the **register file** is one of the *hot spots*, and limits the computational power of the cores.

The area and energy consumption of the register file is proportional to the square of the number of ports. Thus, reducing its ports is effective to downscale the register file, and a number of techniques have been proposed to do so. This thesis mainly focuses on the two techniques, introducing a **register cache**, and **multi-banking the register file**.

First, the author designed a register cache system in detail. The **register cache** is a cache for the *main* register file. Compared with the original register file, the register cache is smaller because it has a smaller size[1]; the main register file is smaller because it has fewer ports. However, conventional register cache systems suffer from low **IPC** (Instructions Per Cycle) due to register cache misses. Shioya, et al. solved this problem with **Non-latency Oriented Register Cache System** (NORCS). Researchers in NVIDIA adopted this idea for their GPUs.

However, they did not show detailed design of NORCS. The original article evaluated NORCS from the viewpoint of microarchitecture, and used CACTI, a design space exploration tool for usual instruction/data caches (not for register caches). In contrast, the authors designed NORCS with **FreePDK45**, an open source process design kit for 45 nm technology, for detailed evaluation from the viewpoint of LSI design.

The results with FreePDK45 are consistent with that of the original article. The author also performed SPICE simulations with RC parasitics to precisely estimate

[1]counted in the number of bytes, or the number of entries because the word size can be considered as 8B (= 64b) in this area.

the latency of the register cache system.

Second, the author proposes the two architectural techniques for multibanked register files. **Multibanking** is the ultimate way to reduce the register file ports. Multibanking divides one n-port register file into n (or more) single-port banks while maintaining the throughput. Although multibanking achieves the minimum number of ports (i.e., 1), pipeline disturbance caused by **bank conflicts** can considerably degrade the IPC. To reduce the bank conflict probability of multibanked register files, this thesis shows the two microarchitectural techniques; one is **Bank-Aware Instruction Scheduler** (BAIS), and the other is **Skewed Multistaged Multibanked Register File** (MStage).

BAIS schedules the instructions so that no bank conflict occurs in the stages to read/write the register file. The idea of bank-aware scheduling itself is not new. Prior studies briefly mentioned the possibility of bank-aware scheduling, or rejected it because it could increase the latency. On the contrary, the author shows an implementation of BAIS and clarifies that the latency of the logic is not practically increased. Although bank-aware scheduler uses as many arbiters as the number of banks, they do not practically prolong the latency because they work in parallel.

In contrast, MStage is a totally new microarchitecture. MStage has two stages to read the bank of the multibanked register file, and an instruction that missed the bank because of a bank conflict still has a second chance to read the same bank in the second stage. As a result, MStage drastically reduces the pipeline disturbance caused by bank conflicts. This thesis also shows the analytic solutions for the pipeline disturbance probabilities of several multibanked register files.

The evaluation results show that, from NORCS, BAIS with 24 banks achieves a 23.6% and 61.8%, and MStage with 18 banks achieves a 40.6% and 68.9% reduction in area and in energy consumption, while maintaining a relative IPC of 97.2% and 97.3%, respectively. In summary, NORCS, BAIS, and MStage show higher efficiency in area and energy consumption in ascending order.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

According to the published results of SPEC CPU [3], in the 1990s, the best scores had been improved by approximately 40% per year, keeping pace with an increase in the clock frequency. However, this increase in the clock frequency has been almost stopped. Nevertheless, since the 2000s, the best scores improved by more than 20% per year even without the increase in the clock frequency [4]. The primary factor behind this improvement is an increase in the width of high-end superscalar cores. Recently, 8-issue cores, such as the IBM POWER8, and Intel Haswell and Skylake, have come onto the market [5–8]. In the papers only few years ago, 8-issue cores were called “ultra-wide” [2].

Register File Composed of a Multiport RAM

Such ultra-wide cores, however, suffer from increased area and energy consumption of the **register file**.

Wide cores require a large number of register entries proportional to the number of in-flight instructions. In addition, multithreaded cores require a number of registers proportional to the number of threads.

Besides, the circuit area of a register file composed of a multiported RAM is proportional to the square of the number of its ports [9–11]. An i-issue core generally requires a $2i$-read+i-write (i.e., $3i$-port) register file.

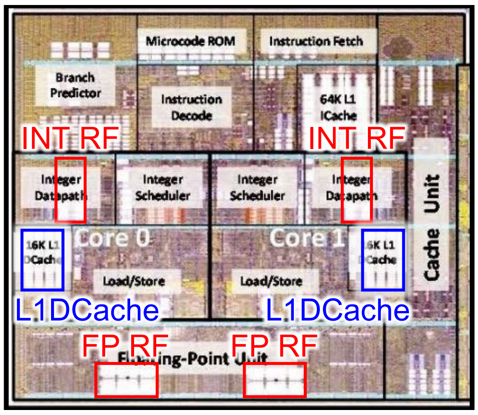


Figure 1.1: Bulldozer 2-Core Processor Module Die Photo (Cited from Fig. 4.5.7 of [1])

Example of Modern Processor Cores

Figure 1.1 shows a die photograph of the AMD Bulldozer 2-core processor module processor, which is the most documented in recent processors [1, 12, 13]. The integer core of the processor is a moderate-sized, non-multithreaded 4-issue one. Nevertheless, as shown in this figure, the 96-entry integer register file with 8-read+ 4-write (i.e., 12-port) is comparable with the 16KB level-1 data cache (**L1D**) in area, even though their sizes are different $16\text{K} \div (96 \times 8) \simeq 21.3$ times. This means that the register file cell is approximately 20 times larger than the L1D cell in area.

Energy Consumption of Register File

Energy consumption and its resulting heat are two of the most serious bottlenecks in recent cores [14]. The energy consumption of a RAM is proportional to its circuit area and accessed frequency [9–11]. While L1D is accessed only once by load/store instructions, the register file is accessed more than once for read and once for write by almost all instructions. In addition, multi-threading increases the accessed frequency in proportion to an increase in throughput. As a result, the register file with a comparable area with L1D consumes much more energy than L1D. The region that includes the register file is a **hot spot** in a core that limits the clock frequency and the scale of the core.

This hot spot problem is becoming more serious, because downscaling the operating voltage is becoming more difficult. In fact, the voltage range available for DVFS is shrinking with each new process. In addition, the register file is one of the most sensitive modules to low voltage in a core [15]. Therefore, architectural solution against energy and heat is more important than ever.

1.2 Reducing Register File Ports

Because its area and energy consumption is proportional to the square of the number of ports, reducing its ports is effective to downscale the register files. Therefore, some methods are proposed to reduce the ports of the register file. These methods are categorized into the following three types.

Replicated register file If the register file is divided into some parts, the number of ports is reduced in exchange for a replication. Replicated register file is already widely used for recent processors [5, 13, 16–19].

Register Cache Register cache is a drastic method of reducing the register file ports [2, 20–22]. Compared with the original register file, the register cache is smaller because it has only 4 to 8 entries; the *main* register file is smaller because it has only a few ports.

Multibanked Register File Multibanking is the ultimate method of reducing the register file ports. With multibanking, even a register file composed of minimum single-port cells has a potential to supply a sufficient throughput as a register file. However, a considerable IPC degradation caused by **bank conflicts** has never been solved.

Replicated Register File

The Bulldozer core shown in Figure 1.1 composes the 8-read + 4-write integer register file of a **replicated pair** of 4-read + 4-write RAMs to reduce the ports from 12 to 8 though the number of cells is doubled (Section 9.1). To be precise, not a 8-read+4-write cell, but the pair of 4-read+4-write cells is approximately 20 times larger than a cache cell. Such replication is widely used for recent processors [5, 13, 16–19].

Register Cache

A **register cache** is a drastic method of reducing the register file ports [2,20–22]. Compared with the original register file, the register cache is smaller because it has only 4 to 8 entries; the *main* register file is smaller because it has only a few ports.

However, conventional register cache systems suffer from low IPC due to register cache misses. The back-end pipeline is stalled when *any* of the register accesses in a cycle cause a register cache miss. If the register cache miss rate per access is 5% and the number of accesses per cycle is 3, the stall probability is as high as $1 - (1 - 0.05)^3 = 1 - 0.857 = 14.3\%$.

To reduce this probability, Shioya, et al. proposed the non-latency-oriented register cache system (**NORCS**) [2], which is the latest proposal on the register file for area and energy efficiency, and researchers in NVIDIA adopted this idea for their GPUs [21, 22].

The NORCS pipeline is disturbed when register cache misses in a single cycle exceed the main register file read ports. With the same number of accesses of 3 and register cache miss rate of 5%, the pipeline with 2-read-port main register file is disturbed if *all* the 3 accesses miss the register cache, and the stall probability is reduced from 14.3% to $0.05^3 = 0.0125\%$.

Multibanked Register File

Multibanking is the ultimate method of reducing the register file ports. In the Bulldozer core, the 96-entry register file will be divided, for example, into 16 banks of 6-entry RAMs composed of small cells such as of L1D. Because the original (replicated) register file cell is approximately 20 times larger than the L1D cell, multibanking can reduce the register file to ideally 1/20 in area and in energy consumption. Then, the hot spot problem will be drastically mitigated.

However, multibanking is a technique typically used for the main memory of vector processors, and not directly applicable to the register file of superscalar cores because the IPC will be considerably degraded by **bank conflicts**. The pipeline disturbance caused by bank conflicts is much higher than naïve intuition, because the pipeline is disturbed when *any* of the banks causes a bank conflict. The disturbance probability is $P = 1 - (1 - p)^b$, where p is the per-bank conflict probability and b is the number of banks. Increasing b does not decrease P very much. In the example in Section 7.4, increasing b to 100 decreases p to 0.0298%, but dose P only to $1 - (1 - 0.000298)^{100} \simeq 2.9\%$. Although an IPC degradation of 2.9% may be acceptable, $b = 100$ is unrealistic for the number of banks.

Although some techniques have been proposed to reduce bank conflicts without increasing the banks [23–27], none of them solved the problem efficiently. Prior studies cannot achieve feasible register file system composed of the minimum single-port cells.

1.3 Contribution of This Thesis

The contribution of this thesis is broadly divided into two parts. The first part is a design of NORCS. The second part is two proposals on feasible microarchitectures with multibanked register files. As far as the author knows, these two proposals are the only feasible register file system composed of the minimum single-port cells.

The detailed contribution of this thesis is as follows:

1. LSI design of conventional register file and register cache systems.
   Shioya, et al. used **CACTI** [11, 28, 29] to evaluate the area and energy consumption of register cache systems. CACTI is a design space exploration tool for caches. For a fast search for an optimal configuration in such a large design space, CACTI performs a rough estimation unfavorable for LSI design. The author shows more detailed design of register cache systems with FreePDK45, an open source process design kit for 45 nm technology.
2. Bank-aware instruction scheduler (BAIS).
   BAIS schedules the instructions so that no bank conflict occurs in the stages to read/write the register file. The idea of bank-aware scheduling itself is not new. Prior studies briefly mentioned the possibility of bank-aware scheduling, or rejected it because of the increased latency [23, 30]. However, an implementation in this study clarifies that the latency of the logic is not practically increased. However, the bank-aware scheduling is not as efficient as skewed multistaging described below, the evaluation clarified the performance and the limitations.
3. Skewed multistaging for a multibanked register file (MStage).
   This is the most important contribution of this thesis. In this technique, register file accesses that cannot obtain their operands because of bank conflicts still have second chances in the second stage of the pipeline.
4. Analytic solutions and IPC, Area and Enegy evaluation.
   The author presents analytic solutions for the stall probabilities of existing and proposed techniques.
   The author evaluated IPCs of the existing and proposed techniques by the Onikiri 2 [31] simulator. The author found that the IPC of these models is primarily determined by the number of register file accesses per cycle of a

program.
The author also evaluated the area and energy consumption. The author used the formula of CACTI for memory cell and performed logic synthesis for their control circuits with the library of FreePDK15, FreePDK for 15 nm technology.

1.4 Organization of This Thesis

This thesis is broadly divided into two parts. The first part is composed of Chapters 3 and 4, and shows design of NORCS. The second part is composed of Chapters 5, 6 and 7, and proposes two techniques on multibanked register files. The detailed organization is as follows:

A superscalar processor requires a large register file composed of a multiported RAM. Chapter 2 explains why such a large register file is necessary and describes other basic structures and behaviors of the register file. These characteristics are preliminary knowledge for an explanation of register cache and multibanked register file described in the following chapters.

Chapters 3 and 4 show design of NORCS.

Chapter 3 explains NORCS from the viewpoint of microarchitecture. This chapter basically follow the original article [2].

Chapter 4 shows the design of NORCS (Non-latency-Oriented Register Cache System).

This chapter also contains design of a conventional register file, and reveals the basic characteristics of the conventional register files. A heavily multiported RAM such as a conventional register file, generally needs a hierarchical bitline structure in order to reduce its latency. This chapter helps to understand hierarchical bitline structure and its characteristics.

Chapters 5, 6 and 7 propose two techniques on multibanked register files.

There is no *standard* implementation of a multibanked register file. Chapter 5 shows a possible *plain* multibanked register file. The author compares the proposed techniques with the *plain* multibanked register file. This chapter also reveals the basic structures and behaviors of the plain multibanked register file.

Chapter 6 describes the proposed **Bank-Aware Instruction Scheduler** (BAIS). An implementation in this chapter clarifies that the latency of the logic is not practically increased.

Chapter 7 describes the proposed **Skewed Multistaged Multibanked Reg-**

ister File (MStage), which is the most important contribution throughout this thesis. MStage has two stages to read the banks of the multibanked register file, and an instruction that missed the bank because of a bank conflict still has a second chance to read the same bank in the second stage. As a result, MStage drastically reduces the pipeline disturbance caused by bank conflicts. The analytic solutions are also presented for quantitative explanation of the drastic reduction of the pipeline disturbance.

Chapters 8 and 9 describes the evaluations of the conventional and proposed models. Chapter 8 shows the evaluation of IPC, and Chapter 9 shows the area and energy consumption.

Chapter 10 finally concludes this thesis.

Chapter 2

Basics of Register File Systems

This chapter describes the basic structure and behavior of the register file in a conventional superscalar core, which provide preliminary knowledge on a register file to understand register cache and multibanked register file systems described in the following chapters.

The register cache and multibanked register file systems reduce the register file ports in exchange for infrequent pipeline disturbance caused by the port shortage. Thus, this chapter details the operand bypass network because it has an extra role to provide operands in place of the expensive ports in these systems. Methods to resolve the pipeline disturbance is also discussed.

As mentioned in Section 1.1, a conventional register file is large because it is composed of a multiport RAM. This chapter also explains the reason from the viewpoint of LSI design.

This chapter is organized as follows: First of all, Section 2.1 explains the role of the register file in the instruction pipeline of a superscalar core. Among the stages in the pipeline, Sections 2.2 and 2.3 cover register renaming and instruction scheduling, because they are important to understand the register file systems. Then, Sections 2.5 and 2.6 explain the operand bypass network and pipeline disturbance. Section 2.7 shows physical layout of multiport RAM cells for register files, which provides a simple explanation for the reason why a register file becomes large. Lastly, Section 2.9 describes related work to reduce register file ports.

2.1 Register File in Pipeline

Figure 2.1 shows a block diagram of the instruction pipelines of a 2-issue out-of-order superscalar core.

Pipeline Stages

Instructions proceed in each stage of the pipeline as follows:

Fetch First, instructions are fetched from the instruction cache.

Rename The operands of the instructions are renamed from the logical to the physical register numbers. Section 2.2 gives a detailed explanation of this register renaming.

Dispatch The renamed instructions are dispatched into the instruction window.

Sched The instructions dispatched in the instruction window are scheduled to be issued. Section 2.3 gives a detailed explanation on this instruction scheduling.

Issue The scheduled instructions are read from the instruction window to be issued for execution.

Read The issued instructions read their source operands from the physical register file.

Exec The instructions are executed using the source operand values read from the physical register file.

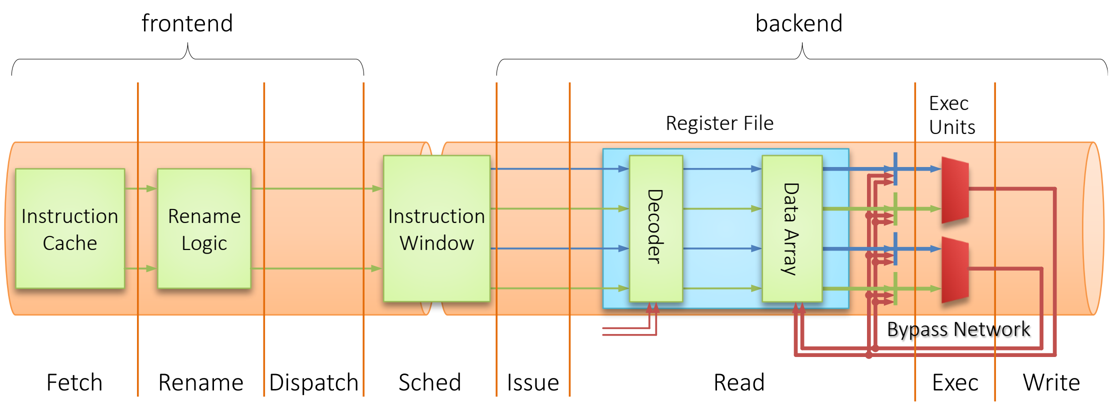


Figure 2.1: Frontend and Backend Pipeline of Superscalar Core

Write The results of the executed instructions are written back to the physical register file.

In recent cores, several cycles are allocated to the stages other than the execution stage.

Frontend and Backend Pipelines

In modern out-of-order cores, the instruction pipeline is decoupled into the **frontend** and **backend** pipelines by the instruction scheduling window. The stages from fetch to dispatch and ones from schedule to writeback belong to the frontend and backend pipelines, respectively.

Instructions flow through the frontend pipeline in order; are scheduled in the instruction window; then, flow through the backend pipeline out of order. The order of instructions is changed from in-order to out-of-order only once in the sched stage. As detailed in Section 2.6, the positional relationship among all the instructions in the backend pipeline must be preserved as issued because everything has been arranged so that the issued instructions flow through the backend pipeline in the issued order.

2.2 Register Renaming

Out-of-order cores resolve dependencies among instructions through the registers in the rename stage.

Basics of Register Renaming

The operands of an instruction are renamed from logical to physical register numbers as follows:

Allocation to Destination Operand A free physical register is allocated to the logical register of the destination operand. This mapping is stored to the register map table.

Resolution for Source Operands By reading the register map table, the physical registers currently allocated to the logical registers of the source operands are resolved.

After this renaming, the instruction can proceed using the physical registers only.

Allocation and Free of Physical Registers

The allocation of physical registers is simpler. The renaming logic picks up a physical register from the free list of the physical registers, and allocates it to the destination of each instruction in order.

In contrast, the free of physical registers is more complex, because the renaming logic needs to know the timing when a physical register will never be read.

Suppose a preceding and a succeeding instructions I_{pred} and I_{succ} have the same logical register as their destinations. That is, I_{succ} is to overwrite the result of I_{pred}. The physical register allocated to the destination of I_{pred} can be freed when I_{succ} is committed [32] for the following reason: When overwriting I_{succ} is committed, this overwrite becomes irreversible, and succeeding instructions that have the same logical register as their source operands will definitely read the result of I_{succ}. Otherwise, that is, before the commitment of I_{succ}, there remains a possibility that I_{succ} is cancelled (and not re-executed) because of a branch misprediction. In this case, succeeding instructions in the correct path can read the result of I_{pred}.

Figure 2.2 shows two instruction flows of the same program which contains a conditional branch (I_{bcc}). The left part shows the flow when I_{bcc} is untaken, and the right part shows the flow when I_{bcc} is taken. In the figure, $L_1(P_8)$ denotes a *physical register* 8 is allocated to a *logical register* 1. On the left part, I_q overwrites the result of I_p, and it is never referred after the branch. On the right part, I_q is not executed, and the result of I_p is referred by I_d.

When I_{bcc} is predicted as untaken by a branch predictor, $L_1(P_8)$ is overwritten

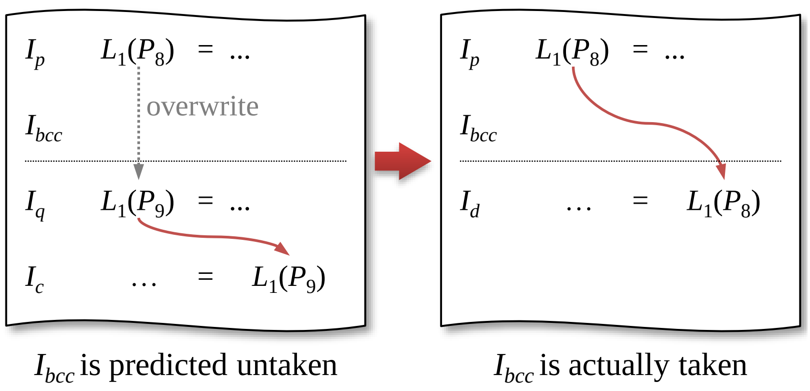


Figure 2.2: Instruction Flows when a Conditional Branch (I_{bcc}) is Untaken (left) and Taken (right).

by I_q and P_8 is needless as shown in the left part. However, specularively executed instructions can be cancelled before their commit stage. In the case of a branch misprediction, I_q is cancelled, and I_d is to read the result of I_p as shown in the right part. Therefore, P_8 cannot be freed until the commit stage of I_q.

In-Order Allocation and Out-of-Order Free of Physical Registers

If the physical register allocated to I_{pred} were freed when I_{pred} is committed, the physical register would be allocated and freed in order, because the commitment is also performed in order.

However, in reality, it is freed when not I_{pred} but I_{succ} is committed; and thus, it is freed out of order. It is because the commitment of I_{succ} can be after the commitment of I_{pred} by an arbitrary number of instructions. An extreme case when there is no overwriting instruction helps to understand the fact that the physical register is freed out of order. In this case, the physical register allocated to I_{pred} will never be freed.

In summary, physical registers are allocated in order and freed out of order.

Physical Register Number Randomization

As a result of this asymmetry between the allocation and free, the sequence of register numbers in the free list is randomly shuffled after sufficient cycles have passed since initialization.

The sequence of the addresses to the main memory by load/store instructions has some regularity. In contrast, the sequences of the register numbers to the physical register file can be considered as random.

This randomness of physical register numbers is essential for the bank conflict probability in multibanked register files described in Section 7.4.

2.3 Instruction Scheduling

Assume that a preceding and a succeeding instructions I_p and I_c are dependent, that is, a single source operand of I_c is the same as the destination operand of I_p. As described in Section 2.2, the same physical register is allocated to the destination operand of I_p and the source operand of I_c.

While maintaining this dependency, these instructions are scheduled in the instruction window through the wakeup and select phases as follows:

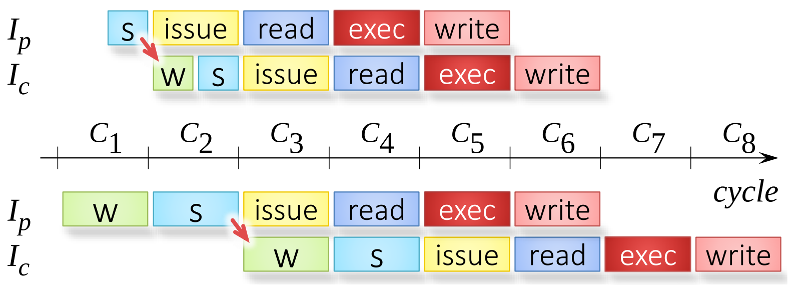


Figure 2.3: Wakeup and Select in one cycle (upper) and two cycles (lower), where the stages are W: wakeup, S: select, issue: issue, exec: execution, and read/write: register read/write.

The wakeup phase When I_p is selected to be issued and executed, the physical register allocated to its destination will be ready to be used in fixed cycles. The number of this fixed cycles is called **issue latency**, and given by the number of cycles in the issue, read, and exec stages. Because the physical register will be ready in this fixed cycles, the wakeup logic sets the ready flag of the source operand of I_c which has the same physical register.

If all the source operands of an instruction are set ready, the instruction is ready to be issued. In this case, the single source operand of I_c becomes ready to be used (in the issue latency), and I_c becomes ready to be issued.

The select phase The select logic selects instructions to be issued from ready instructions. The number of instructions to select is given by the amount of execution resources, such as the number of execution units.

Wakeup and Select Loop

The wakeup and select form a loop, that is, the wakeup and select kick off each other. In the above-described example, the select of I_p, wakeup of I_c, and select of I_c occur in series.

This loop of wakeup and select should turn round in one cycle for *back-to-back* execution of dependent instructions, as described below.

Figure 2.3 compares the pipelined behavior of I_p and I_c with the wakeup and select loops of one cycle (upper) and in two cycles (lower). In the lower half of the figure, the issue of I_c is delayed by two cycles because two cycles are taken for the wakeup and select of I_c after the select of I_p. In contrast, in the upper half of the figure, the issue of I_c is delayed only by one cycle, and I_p and I_c can be executed back-to-back.

2.4 Instruction Pipeline Depth

Some techniques of register cache and multibanked register file systems described in the following chapters reduce the pipeline disturbance probability in exchange for the deeper instruction pipeline.

This section describes the side effect of the instruction pipeline depth of the superscalar core. The instruction pipeline depth does not directly increase execution cycles.

Figure 2.4 shows pipelined behavior of seven instructions with two mispredictions. The register file read latencies are one and two cycles in the upper and lower pipelines, respectively, and the lower pipeline is one cycle deeper than the upper.

A misprediction causes a pipeline flush, that is, the pipeline restarts from the fetch stage after the detection of the misprediction. Comparing the two pipelines in the figure, the lower pipeline delays by one cycle per misprediction.

Thus, the increase in execution cycles is given by the product of the difference in the pipeline depths (one cycle in the figure) and the occurrence probability of misprediction. Thus, if the occurrence probability of misprediction is sufficiently

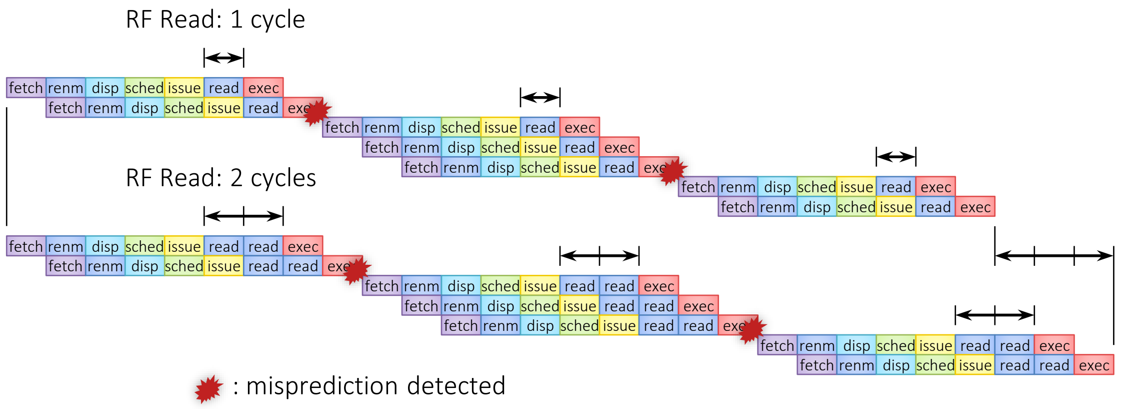


Figure 2.4: Effect of Pipeline Depths on Execution Cycles.
The register file latencies are one cycle (upper) and two cycles (lower).

low, the increase in execution cycles is also sufficiently low.

In the evaluation in Chapter 8, one-cycle increase in pipeline depth decreases IPC by 1.4% on average.

The Deeper Pipeline Outperforms The Short Pipeline

In a certain condition, the deeper pipeline with a less disturbance probability outperforms the short pipeline with a high disturbance probability. This paragraph clarifies the condition that the deeper pipeline outperforms the short pipeline.

The total penalty cycles are the total of the penalty of the prediction miss ($penalty_{pred} \times \beta_{pred}$) and the penalty of the pipeline disturbance ($penalty_{distb} \times \beta_{\mathrm{distb}}$). β_{pred} and β_{distb} are the *effective* miss rate of a prediction and disturbance of the pipeline, respectively. Note that the *effective* miss rate given by the probability of the miss or the disturbance in each cycle.

Therefore, the total penalty cycles of the pipeline are:

$$penalty_{pred} \times \beta_{pred} + penalty_{distb} \times \beta_{\mathrm{distb}}, \tag{2.1}$$

To simplify the problem, the author compares the base pipeline with a high disturbance probability ($\beta_{\text{distb-base}}$) and the one cycle deeper pipeline with a less disturbance probability ($\beta_{\text{distb-deep}}$). $penalty_{distb}$ is one cycle for both the pipelines.

The total penalty cycles of these two pipelines are:

$$\begin{aligned} penalty_{pred} &\times \beta_{pred} + 1 \times \beta_{\text{distb-base}} \quad \text{(base)} & (2.2)\\ (penalty_{pred} + 1) &\times \beta_{pred} + 1 \times \beta_{\text{distb-deep}} \quad \text{(one cycle deeper)} & (2.3) \end{aligned}$$

Therefore, (2.2) > (2.3) is the condition that the deeper pipeline outperforms the base pipeline.

This can be deformed as:

$$\begin{aligned} penalty_{pred} \times \beta_{pred} + 1 \times \beta_{\text{distb-base}} &> (penalty_{pred} + 1) \times \beta_{pred} + 1 \times \beta_{\text{distb-deep}} \\ \beta_{\text{distb-base}} &> \beta_{pred} + \beta_{\text{distb-deep}} \qquad (2.4) \end{aligned}$$

In the evaluation in Chapter 8, the total β_{pred} of the branch and latency prediction is sufficiently low, around 0.0022 per instruction.

Thus, if $\beta_{\text{distb-base}} > \beta_{\text{distb-deep}}$, the deeper pipeline outperforms the base pipeline.

2.5 Operand Bypass

In Figure 2.1, the bottom red lines denote the operand bypass network. The results can be passed through the operand bypass network from any output to any input of the execution units.

Role of Bypass for Conventional Register File

In conventional systems, the operand bypass network is needed for back-to-back execution. Figure 2.5 shows the pipelined behavior of two instructions I_p and I_c with and without the operand bypass. I_c depends on I_p; that is, I_c reads the same physical register as I_p writes. In the lower half of the figure, the issue of I_c is delayed for two cycles to wait until the result can be obtained through the register file. In contrast, if the value is passed through the operand bypass, I_p and I_c are executed back-to-back, as shown in the upper half of the figure.

The operand bypass network has to provide results that has produced in the number of cycles given by the sum of the write and read latencies of the register file. In the figure, the network has to provide results produce in $1 + 1$ cycles for the write and read. In recent cores, the latencies of the register file are longer, and the size of the bypass network is also problematic.

Role of Bypass for Register File Systems with Reduced Ports

It is needless to regard the ports of a conventional register file as resources, because a conventional register file is full-port, that is, there are always free ports to use.

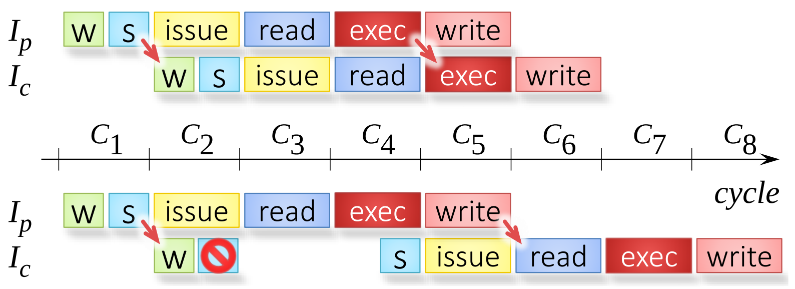


Figure 2.5: Scheduling w/(upper) and w/o (lower) operand bypass, where the stages are W: wakeup, S: select, exec: execution, and read/write: register read/write.

In contrast, the register file ports become expensive resources in the register cache and multibanked register file systems described in the following chapters. These systems reduce the ports in exchange for infrequent pipeline disturbance caused by the port shortage.

Accordingly, the operand bypass network has an extra role to provide operands in place of the expensive ports in these systems. The evaluation results in Section 8.2 show that almost half of the source operands are provided by the bypass network. That is, the operand bypass network reduces the effective number of register file accesses by half, to drastically reduce pipeline disturbance caused by the port shortage.

2.6 Pipeline Disturbance

A register cache miss in register cache systems or a bank conflict in the multibanked register files can cause pipeline disturbance. This situation is similar to but different from a Level-1 Data Cache (L1D) miss.

Selective Delay Problem

The author is sometimes asked if *selective delay* of the instructions that caused disturbance is possible without disturbing the entire pipeline. However, this is unrealistic even for L1D misses, as detailed below.

Positional relationship among all the instructions in the backend pipeline must be preserved as issued, because everything has been arranged so that the issued instructions can flow through the backend pipeline as issued. To implement selective delay, a kind of *post-scheduler* is required to rearrange the instructions flowing through the backend pipeline [2].

Rescheduling and Stalling

It is known that there are the following two ways to resolve disturbance in the backend pipeline while preserving positional relationship among instructions:

Rescheduling (also known as scheduler replay) The responsible instruction and its dependent instructions are canceled and rescheduled.

Stalling During the cycles when the entire backend pipeline is stalled, the missed data is obtained. Then, the pipeline can be restarted as if there had not been a miss.

In particular, the stalling of the backend pipeline must be *total* for the same reason why the selective delay is unrealistic. In a scalar or an in-order superscalar processor, stalling of the single instruction pipeline is *partial*, that is, the downstream stages go on while the upper stream stages are stalled until the hazard is resolved. In contrast, the stalling of the backend pipeline of an out-of-order superscalar processor must be total, i.e., all of the stages in the backend pipeline must be simultaneously stalled. Otherwise, the positional relationship among the instructions is broken.

L1D Miss

Most cores adopt rescheduling for L1D misses. Edmondson et al. also adopted rescheduling for the Alpha 21164 core at an early date after a detailed discussion on the trade-off between rescheduling and stalling [33].

They pointed out that stall logic could create critical paths because the write enable terminal of all the pipeline registers must be immediately turned off.

Register Cache Miss and Bank Conflict

However, almost all the studies on register caches and multibanked register files have adopted stalling [2, 20, 26, 27].

This difference primarily depends on the length of the miss latencies compared with the issue latency which gives the rescheduling penalty. The latency on an L1D miss is comparable to the issue latency, whereas that on a register cache miss is usually one cycle, much shorter than the issue latency. Shioya et al. evaluated both rescheduling and stalling, and concluded that stalling is advantageous for register cache misses [2].

Stalling is more advantageous for multibanked systems because the latency on a bank conflict is equal to or shorter than that on a register cache miss [26, 27].

2.7 Multiport RAM for Register File

An i-issue core generally requires a $2i$-read+i-write, i.e., $3i$-port register file directly connected to the input and output of i execution units, to execute any i instructions in each cycle without restriction. In Figure 2.1, the register file for the 2-issue core has four read and two write ports

In a conventional core, such a full-port register file is composed of a single multiport RAM.

Multiport RAM Circuit

Figure 2.6 shows a block diagram of a 64-bit × 128-entriy 4-read+2-write RAM of a register file with two execution units, and Figure 2.7 shows a circuit diagram of its 4-read+2-write RAM cell.

In this diagram, the vertical wires from the decoders to the RAM cells are the **wordlines**, and the horizontal wires between the execution unit and RAM cells are the **bitlines**. Each of the decoders asserts one wordline out of 128 to select the entry to which the wordline is connected, and the bitlines transmits 64-bit data between the selected entry and the execution units.

This RAM has four read and two write ports each of which is composed of a register number input (reg#_*[6:0] in the figure), a decoder, 128 wordlines, and 64 bitlines (, and read/write enable signals not shown in this figure).

This register file is full-port, that is, the four read and two write ports are directly connected to the four input and two output of the two execution units. Thus, each input and output of the execution units can access any of the entries independently via these dedicated ports.

Multiport RAM Cell

A multiport RAM cell is composed of a pair of cross-coupled inverters to store 1-bit data, and groups of transistors to access that data. The pair of cross-coupled inverters is also known as a **4T-cell**.

In Figure 2.7, the following types of access transistors are used:

Read A stack of two transistors is provided for a single-end bitline.

Write A group of three transistors is provided for a single-end bitline.

As described below, a single-end bitline is especially advantageous for heavily multiported RAMs.

A group of access transistors connects a wordline (vertical line in the figures) and a bitline (horizontal) to the 4T-cell in a cell. When the wordline is asserted, the transistors are turned on, the 4T-cell is electrically connected to the bitline, and the 4T-cell is read or written through the bitline.

As shown in Figure 2.7, a multiport RAM cell can basically be realized by simply

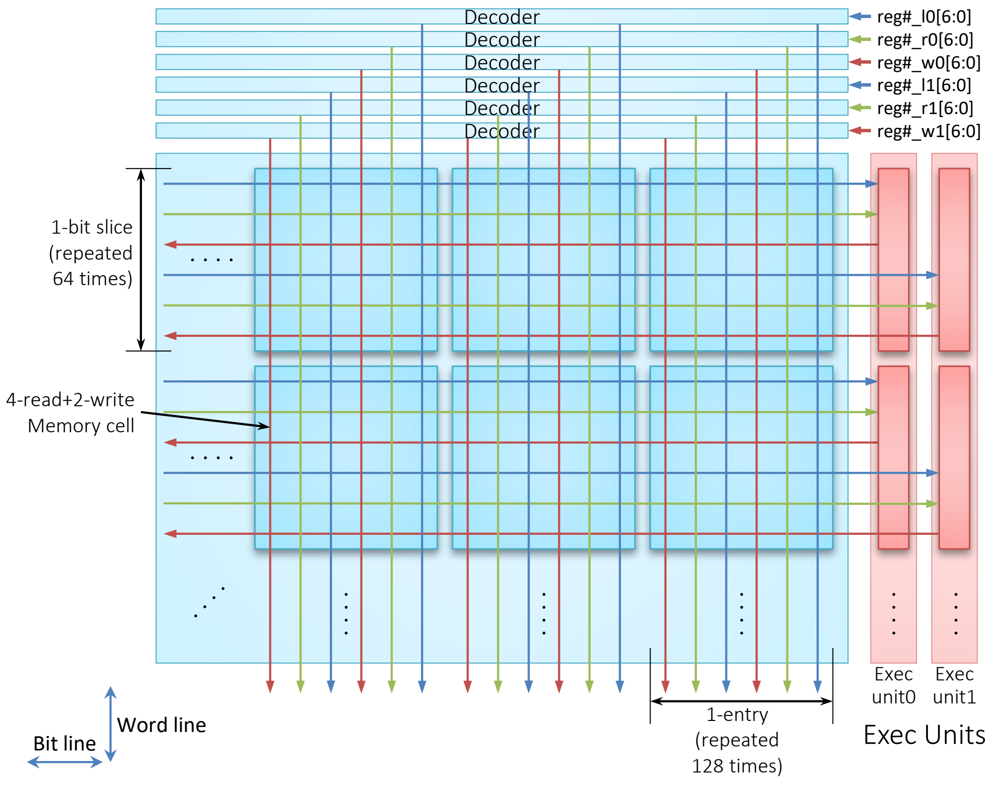


Figure 2.6: 128-entries 64-bits 4-read+2-write register file for a 2-issue core.

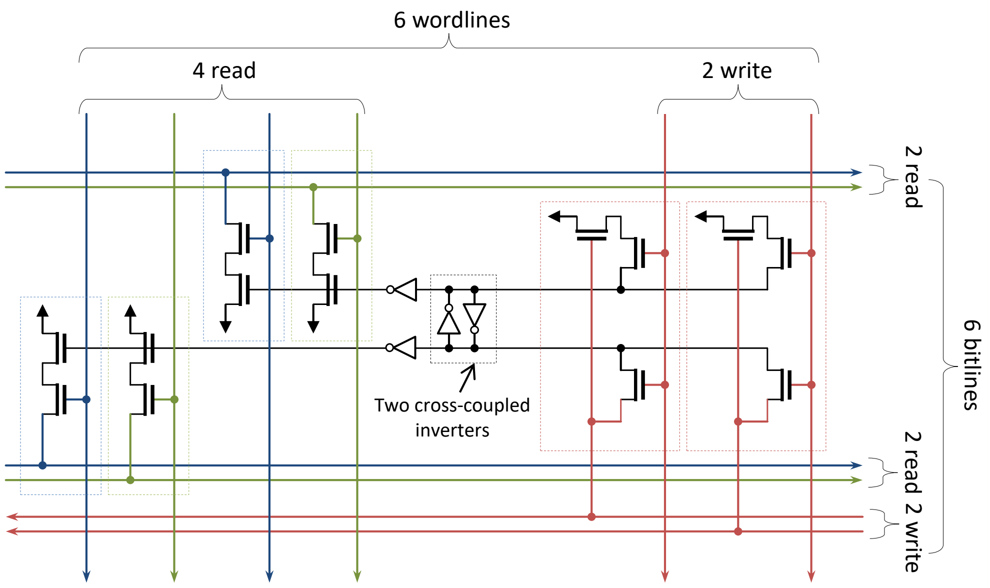


Figure 2.7: Circuit of 4-read+2-write RAM cell

adding multiple ports to the 4T-cell in parallel. In this figure, a decoupling inverter is added to reduce the load capacitance of the 4T-cell node.

Physical Layout of Multiport RAM Cell

Figure 2.8 shows physical layout of 2-read+2-write and 8-read+4-write RAM cells, which are part of the design of NORCS detailed in Chapter 4.

As shown in the figure, both the numbers of bitlines and wordlines are proportional to the number of ports, thus illustrating that the area of a multiport RAM is proportional to the square of the number of its ports [9–11].

In contrast, the number of access transistors is proportional to the number of its ports. Therefore, the area of transistors is not a bottleneck for heavily multiported RAMs.

Thus, single-end read/write is advantageous especially for heavily multiported RAMs to reduce their areas, and widely used for recent superscalar cores [12].

Practically, word- and bit-lines can be distributed to multiple wiring layers to mitigate their effect on the area. However, it is not very simple because a line in a

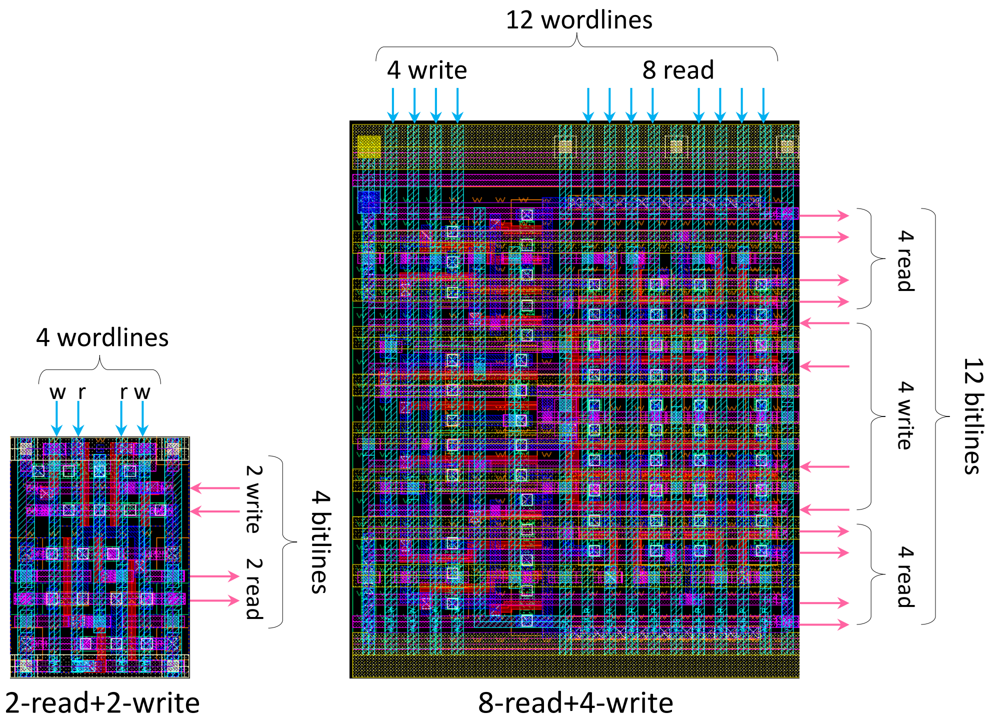


Figure 2.8: 2-read+2-write (left) and 8-read+4-write (right) RAM cells

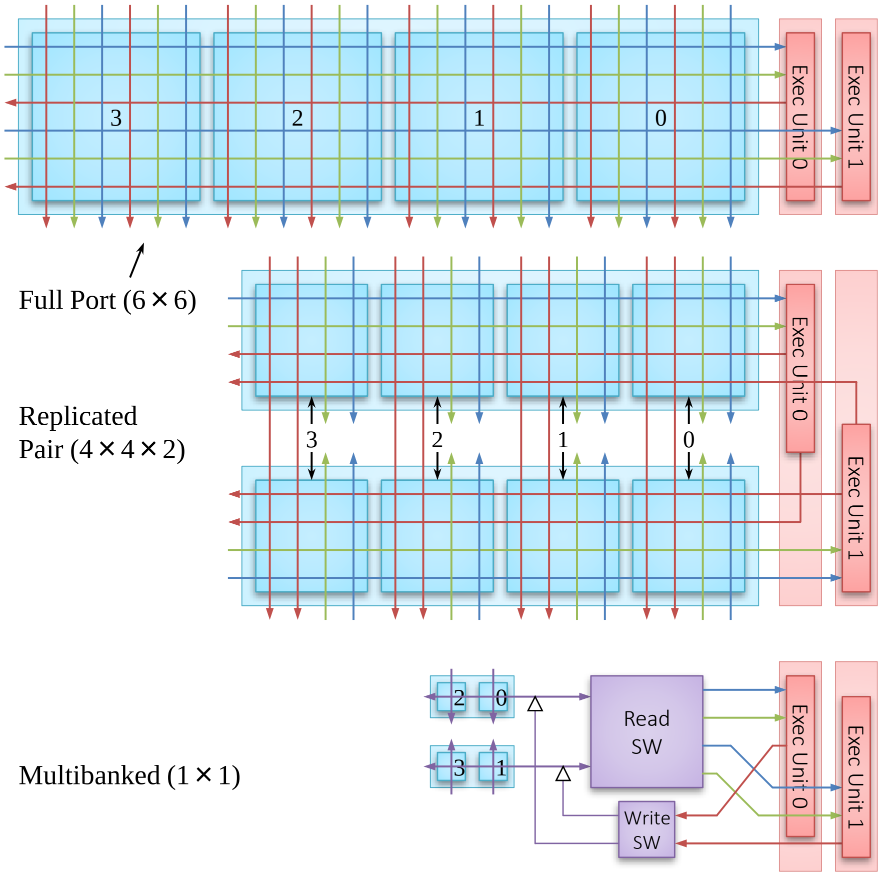


Figure 2.9: Three types of 1-bit × 4-entry, 4-read+2-write register files.

higher layer needs a via through each of the lower layers to the transistor layer.

2.8 Replicated and Multibanked Register Files

Because the area and energy consumption of a register file is proportional to the square of the number of its ports, several techniques try to decompose a register file to reduce the number of register file ports.

Figure 2.9 shows 1-bit × 4-entry, 4-read+2-write register files for two 1-bit execution units to illustrate the difference of the three types of implementation.

Replicated Pair

Many recent cores use a replicated pair of two w-read+w-write RAMs to implement $2w$-read+w-write register file [5, 13, 16–19].

In Figure 2.9, a full-port, i.e., 4-read+2-write register file shown in the upper part is decomposed into a replicated pair of two 2-read+2-write RAMs shown in the middle. As the entry numbers of the cells show, two cells vertically arranged have the same contents. To this end, the two write ports are directly connected to the output of the two execution units, and the same result is simultaneously written to pairs of two cells.

As a result, while the read ports of the cells are reduced by half, the write ports cannot be reduced. In the figure, the expressions 6×6 and $4 \times 4 \times 2$ denote the cell areas counted in the number of word- and bit-lines.

However, detailed evaluation in Section 9.1 shows, though this replication has a positive effect on the latency, but a slight negative effect on the area and energy consumption.

Multibanking

In Figure 2.9, a full-port, i.e., 4-read+2-write register file shown in the upper part is decomposed into two banks of 1-read/write RAMs shown in the lower. A read and a write switch are needed for any-to-any connection between the execution units and the banks.

Unlike replication described above, the contents are not replicated. In this figure, the register numbers are interleaved among the banks as with the address for usual multibanked main memory. However, the interleaving has no effect because the register numbers are randomized as described in Section 2.2.

The example in Figure 2.9 uses 1-read/write cells, which are the *minimum* with respect to the number of word- and bit-lines. Though the switches seem relatively bigger in this figure, they are smaller for realistic register files with more than hundred entries.

While the multibanking can use the minimum cells, it can be regarded as a *pseudo* implementation of a multiport RAM. Unlike full-port or replicated register files, it cannot perform accesses to arbitrary entries. A 1-read/write bank can perform only one access in a single cycle. In other words, if multiple accesses are to the same bank, they cannot be performed in the cycle. This situation is called a **bank conflict**. The example in Figure 2.9 will cause frequent bank conflicts because of too few banks for two execution units.

The two techniques proposed in Chapters 5, 6 and 7 try to solve this problem while using the minimum 1-read/write cells.

2.9 Related Work

This section reviews related work for area and energy efficiency of register files not detailed in the other chapters.

Utilization of Bypass

Tseng et al. proposed using the bypass to reduce bank conflicts [23, 24]. The register file ports are not accessed when the operands are provided by the bypass. Tseng et al. divided the register file into eight 2-read+2-write banks using this technique. The author also adopted this technique (see Section 8.2).

Number of Operands

Instructions that have two source operands are rare (see Section 8.2). Utilizing this fact, Kim et al. reduced the ports of the wakeup logic and register file [34], and Sangireddy did those of the register map table [35].

Distributed Register Files

Clustered or tile architectures have distributed register files [36–43]. If a register file is distributed to a group of i' execution units, the number of its ports is reduced to $3i' + \alpha$, where α is the number of additional ports for communication. Typically, $i' \geq 2$, while aggressive distribution of which $i' = 1$ incurs a certain level of IPC degradation depending on the accuracy of instruction steering.

In particular, clustered architectures can be regarded as an architectural reinforcement of replicated register files [5, 13, 16–19].

Clustered execution units and multibanked register file are contrasting. An execution result is located to the cluster where the instruction is steered in the former, to the bank in the latter.

Chapter 3

NORCS

This chapter explains NORCS from the viewpoint of microarchitecture before the design of NORCS in Chapter 4. Thus, the contents of this chapter basically follow the original article of Shioya, et al. [2]. The main difference from the original article is in the description of the main register file, which is pipelined and has 2-cycle latency in this study.

First, Section 3.1 summarizes conventional register cache systems and NORCS. A register cache can reduce the area and energy consumption of a register file. Compared with the original register file, the register cache is smaller because it has a smaller size; the main register file is smaller because it has fewer ports.

The earlier register cache systems suffered from IPC degradation caused by register cache misses, because their pipelines are stalled on a register cache miss. On the contrary, the NORCS pipeline has stages for reading the main register file. As a result, the NORCS pipeline is stalled not when a single register cache miss occurs but when the main register file read ports fall short.

Section 3.2 shows the difference of a register cache from a usual data cache, that significantly affects the design of NORCS. The difference is mainly because of register renaming. The cache entry of NORCS is smaller than that of a usual data cache, because spatial locality cannot be exploited because physical registers are randomly allocated by register renaming. Register renaming also avoids any overwrite on the same physical register. Therefore, NORCS adopted write-through policy.

Lastly, Section 3.3 shows the structure and pipelined behavior of NORCS in detail.

3.1 Register Cache System

A **register cache** can reduce the area and energy consumption of a register file. As the name suggests, a register cache is a cache for a register file. The register cache provides more than 90% of the source operands to the execution units taking the place of the **main register file**. Compared with the original register file, the register cache is smaller because it has a smaller size; the main register file is smaller because it has fewer ports.

However, the earlier register cache systems suffered from IPC degradation caused by register cache misses, because their pipeline is stalled on a register cache miss. Then, Shioya, et al. proposed **Non-Latency-Oriented Register Cache System** (**NORCS**) to solve this problem [2]. Researchers in NVIDIA adopted the idea of NORCS for their GPUs [21, 22].

A conventional pipeline has a stage for reading the register cache but not the main register file. As a result, the pipeline is stalled on a register cache miss in order to produce extra cycles to read the main register file. On the contrary, the NORCS pipeline has stages for reading the main register file. As a result, the NORCS pipeline is stalled not when a single register cache miss occurs but when the main register file read ports fall short, that is, the number of register cache misses in a single cycle exceeds the number of read ports of the main register file.

The difference of stall probabilities can be confirmed using following simple calculations. When the number of accesses per cycles is 3, a conventional pipeline is stalled when *any* of the 3 accesses miss the register cache. In contrast, if the number of read ports of the main register file is 2, the NORCS pipeline is stalled only when *all* of the 3 accesses miss the register cache. When the register cache miss rate per access is 5%, the stall probability of a conventional system is $1-(1-0.05)^3 = 14.3\%$, and that of NORCS is $0.05^3 = 0.0125\%$.

3.2 Difference from Usual Cache

A register cache caches registers in roughly the same manner as a usual data cache. However, they have some differences that affect design.

3.2.1 Difference in Basic Feature

Register Renaming

Out-of-order processors resolve dependency among registers using **register renaming**. Register renaming is one of the main factors in the differences described below.

A *new* physical register is randomly picked from the free list, and is allocated to the destination of each instruction.

A physical register repeats the following cycle: first, it is allocated and written once; then, it is read more than 0 times; and, finally, it is freed.

Entry Size

The entry of usual data caches is usually called a cache line whose size is typically 32 to 64 bytes, to reduce the management overhead, under the fact that it is also advantageous for IPC because of spatial locality of reference.

In contrast, a register cache entry is a physical register. Even if an entry has multiple registers, spatial locality cannot be exploited because physical registers are randomly allocated as described above.

This results in a difference in partial modification of an entry. A store instruction modifies only part of a data cache entry, and the modified entry still has valid data unmodified. Conversely, a register write modifies the entire register cache entry.

Lifetime of Cache Entry

A register cache entry has much shorter lifetime than a usual data cache entry. Shioya, et al. showed that a register cache with only 8 entries has sufficiently good IPC for a 4-issue processor [2]. If the IPC is 2, 2 entries are replaced in every cycle, and all the 8 entries will be replaced in 4 cycles.

3.2.2 Difference in Cache Design

The differences in the basic features described above result in differences in the cache design as described below.

Write Allocation

Because of the short lifetime, a register cache entry is almost always replaced before its physical register is re-allocated and re-written.

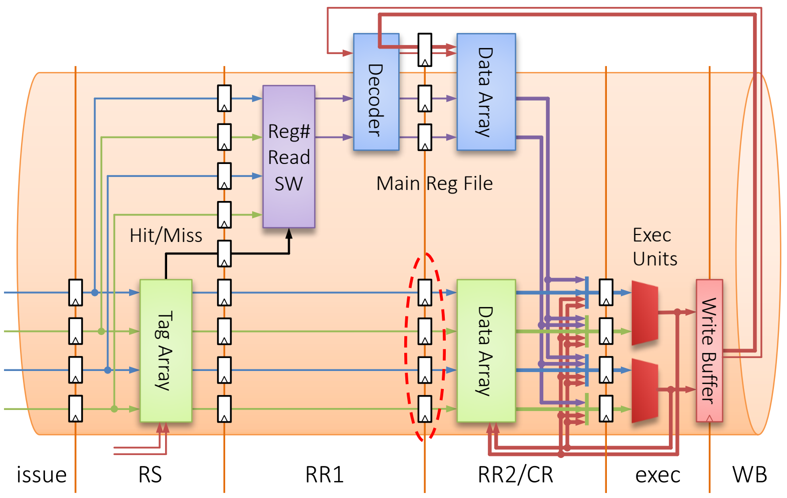


Figure 3.1: Backend Pipeline of NORCS

As a result, almost all of the register writes cause misses, and the new register cache entry is allocated on these write misses. This write allocation strongly affects the design of the tag array, as described in Section 4.2.3.2.

Write-through Policy

Unlike usual data caches, register caches should adopt not a write-back policy but a write-through policy.

In usual data caches, because stores to the same addresses often appear, a write-back policy reduces the number of write(-back)s to the main memory from the number of stores. Therefore, it is useful to adopt a relatively complex write-back mechanism for data caches.

Conversely, in the register caches, because an overwrite to the same entry never occurs because of register renaming, the write-back policy does not reduce the number of write(-back)s to the main register file. Consequently, a write-back policy is meaningless in the case of a register cache.

3.3 Structure and Pipelined Behavior

Figure 3.1 shows a block diagram of the backend pipeline of NORCS for a 2-issue processor. In a conventional processor, instructions read the register file in the stages after the instruction issue (labeled issue in the diagram) before execute (exec). In a register cache system, the **register cache** and the **main register file** are placed in the position of the register file.

In this diagram, instructions access the **tag array** in the register schedule (RS) stage; and then, read the **data array** in the cache read (RR2/CR) stage.

When an access to the register cache cause a miss, the instruction reads the main register file. In this diagram, the main register file is pipelined into the decoder (RR1) and the data array (RR2), and takes two cycles to read.

Although, the register cache needs only one cycle to read, the register cache in the NORCS pipeline has the same two cycle latency as the main register file, because of the additional pipeline latches, which are located between the tag array and the data array of the register cache. Therefore, the NORCS pipeline is not stalled immediately on a register cache miss.

The register cache is smaller than the conventional register file, because it has a smaller number of entries, whereas it has the same number of ports as the conventional register file. On the contrary, the main register file is smaller than the conventional register file, because it has a smaller number of ports; it has the same number of entries as the conventional register file.

In this diagram, the main register file has only two read ports; however, in a 2-issue processor, at most four source operands are read by the two instructions in a single cycle. Thus, the 4-to-2 register number switch (Reg# Read SW) is used to route two out of four register numbers (of two instructions) to the two read ports of the main register file, which is controlled by the hit/miss result from the tag array of the register cache.

The write buffer, placed between the execution units and the main register file, makes a difference in the writing method, as detailed in Section 3.4.

Figure 3.2 shows the pipelined behavior of NORCS for a 1-issue processor.

In this figure, the stages to read the source operands (CR and RR1/2) are divided into upper and lower halves to indicate that each instruction has the first and second source operands. The white boxes indicate that the instructions do not have a source operand and no register access is made. The gray bands denote cycles when the pipeline is stalled.

The author focuses on two instructions, I_1 and I_3, in the figure. I_1 has two source operands and only the *second* operand causes a register cache miss. On the other hand, both of the *first* and *second* source operands of I_3 cause register cache misses. These instructions flow through the pipelines as follows:

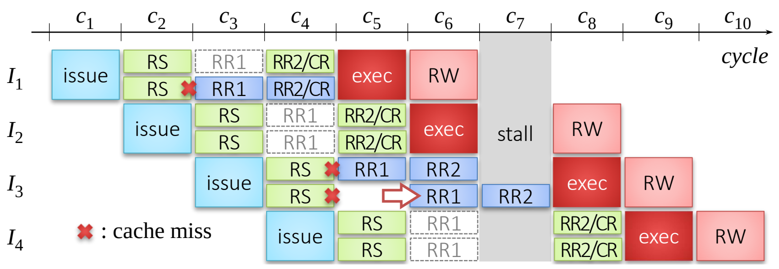


Figure 3.2: Instruction Pipelines of NORCS
issue, RS, CR/RR1/2, exec, and RW indicate the instruction issue, register schedule, register read, execute, and register write stages, respectively. The X symbols denote register cache misses.

I_1 I_1 accesses the tag array of the register cache in the RS stage and detects that the *second* operand causes a register cache miss in c_2.

The NORCS pipeline is not stalled for I_1. In spite of the register cache miss in c_2, the NORCS pipeline originally has the RR1 and RR2(/CR) stages for reading the main register file. Access to the *second* operand of I_1 reads the main register file in the RR1 and RR2(/CR) stages in c_3 and c_4. At the same time, access to the *first* operand passes through the RR1 stage and reads the register cache in the (RR2/)CR stage in c_4.

I_3 Both the *first* and *second* operands of I_3 cause register cache misses in c_5.

As described above, the NORCS pipeline is stalled when the main register file read ports fall short, and the main register file has only one read port in this figure. Thus, the NORCS pipeline is stalled for I_3, which causes two register cache misses in a single cycle. Because the NORCS pipeline has RR1 and RR2(/CR) stages for reading the main register file, I_3 uses the stages to read the main register file for the *first* operand. For the *second* operand, the pipeline is stalled for one cycle in c_7.

As shown in this figure, even when the instruction pipeline is stalled, the main register file works in a pipelined manner independently of the instruction pipeline.

3.4 Write Buffer

Unlike the read ports, the register cache does not reduce the write ports of the main register file. Note that, because it does not adopt write-back policy, the register cache itself does not reduce the number or write(-back)s to the main register file (Section 3.2). Instead, NORCS adopts a write buffer for this purpose.

The **write buffer** temporarily holds the results of instructions from the execution units; and then, write them to the write ports of the main register file. Because the register cache adopts a write-through policy, the results are written both to the register cache, and to the main register file via the write buffer, in parallel.

The number of write ports in the main register file is reduced by averaging the traffic from the execution units to the main register file. As described in the original article [2], the number of write ports is reduced from 4 to 2 for a 4-issue processor, at the risk that the reduced number of write ports gives the maximum IPC of the processor.

Chapter 4

Design of NORCS

Shioya, et al. used **CACTI** to evaluate the area and energy consumption of NORCS in their original article [2]. CACTI is a design space exploration tool for usual instruction/data caches (not for register caches). The evaluation with CACTI in the original article sufficiently proved the effectiveness of NORCS from the viewpoint of microarchitecture. However, approximations of CACTI are insufficient to show the practicality of NORCS from the viewpoint of LSI design. Therefore, in this study, the author shows more detailed design of NORCS with **FreePDK45**, an open source process design kit for 45 nm technology, for detailed evaluation from the viewpoint of LSI design.

Section 4.2 shows the detail design of NORCS. The author performed manual layout of the memory cells and arrays of NORCS. The memory cells of the main register file is smaller than that of the conventional register file, because of its fewer ports. Therefore, two memory cells were arranged in the width of a 1-bit slice of execution units.

Section 4.3 shows the evaluation results of the area, energy consumption and latency. The main register file adopted a hierarchical bitline architecture in the same way as the conventional register file.

The evaluation results were consistent with that of the original article, except for the static energy.

4.1 FreePDK45 and CACTI

In the original article, Shioya, et al. used **CACTI** to evaluate the area and energy consumption of NORCS [2]. CACTI is a design space exploration tool for usual instruction/data caches (not for register caches). A cache has a large design space; for example, as detailed in Section 4.2, there are a number of configurations for subarray partitioning. For a fast search for an optimal configuration in such a large design space, CACTI performs a rough estimation unfavorable for LSI design as follows:

Cell size The width and height of the memory cells are calculated using simple linear approximations based on the number of ports.

Layout NORCS consists of several kinds of memories, and they and the execution units must be aligned with each other. CACTI does not consider such inter-module constraints as this.

Latency CACTI is designed for cache memories as large as or larger than level-1 caches, and calculates their latency with a linear approximation especially for memories of such sizes. As a consequence, the divergence in this approximation becomes large for smaller memories such as register files and register caches.

Thus, in the original article, the latencies of the baseline register file are simply extrapolated, the register cache and main register file 2, 1 and 1 cycles from the ratio of their areas for IPC evaluation.

It is not rare to use CACTI in the microarchitecture field, and the evaluation with CACTI in the original article sufficiently proved the effectiveness of NORCS from the viewpoint of microarchitecture. However, these approximations of CACTI are insufficient to show the practicality of NORCS from the viewpoint of LSI design.

Design with FreePDK45

Therefore, in this study, the author shows more detailed design of NORCS with **FreePDK45**, an open source process design kit for 45 nm technology. First, the author manually designed the physical layout of the memories of NORCS, and measured the areas. Then, the author extracted the RC load from the layout, and performed SPICE simulation to calculate the energy consumption and latency.

Although the author focused on NORCS in this study, the results are partly applicable to conventional register cache because their structures are almost the same. the author also designed conventional register file, which has 4-read+2-write, as a baseline of register file.

The remainder of this chapter is organized as follows: Section 4.2 explains the circuit and layout of NORCS. Section 4.3 shows the evaluation methodology and the evaluation result in terms of area, energy consumption and latency.

4.2 Design

This section explains the design of NORCS in terms of of layout and circuit, and compares it to the conventional register file. NORCS has a complex structure, as compared to the conventional register file. In the design of NORCS, the bitline structure, the layout of the small size cell in main register file, and the tag array are important features. These are also the contribution of this thesis.

4.2.1 Outline

Figure 4.1 shows the floorplans of a conventional register file (upper) and that in NORCS (lower) for a 2-issue processor. Table 4.1 summarizes the main modules composed of the memory arrays that the author designed.

4.2.1.1 Conventional Register File

The upper diagram in Figure 4.1 shows a floorplan of a conventional register file (**CRF**). The CRF for a 2-issue processor is composed of 4-read+2-write cells for the 4 source and 2 destination operands of 2 instructions executed in parallel. The interconnection of the CRF and execution units is simple; the read and write ports of the register file are directly connected to the input and output ports of the execution units, respectively.

4.2.1.2 NORCS

The lower diagram in Figure 4.1 shows the floorplan of NORCS. NORCS consists of three modules: a main register file (**MRF**), a write buffer (**WB**), and a register cache (**RCD** and **RCT**).

Similar to the CRF, the read and write ports of the RCD are directly connected to the input and output ports of the execution units, respectively. RCT is placed

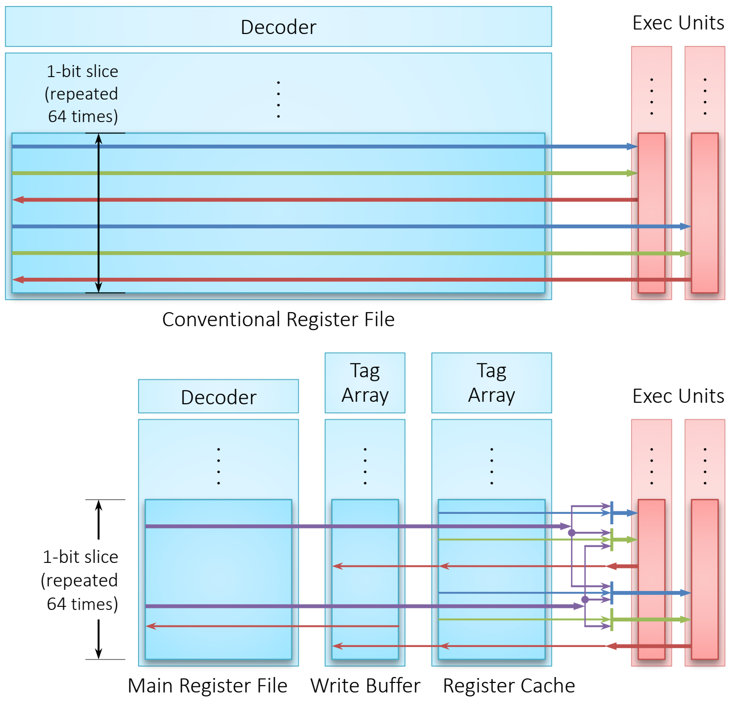


Figure 4.1: Floorplans of conventional register file (upper) and NORCS (lower)

in the position of the decoder of the CRF.

The WB is placed between the execution units and the MRF; it averages the traffic from the execution units to the MRF.

The MRF has a fewer ports than either the CRF or the RCD. The read and write ports of the MRF are connected to the execution units via the register number switch and the write buffer. Regarding the read ports, the register number switch is placed such that it routes the input ports of the execution units to the (fewer) read ports of the MRF.

Table 4.1: Designed Modules

Model	Description	Ports	Bits × Words
CRF	Conventional Register File	8-read+4-write	64 × 128
MRF	Main Register File	2-read+2-write	64 × 128
RCD	Register Cache Data array	8-read+4-write	64 × 8
RCT	Register Cache Tag array	8-search+4-write	7 × 8
WB	Write Buffer (tag and data)	2-read+4-write	64 × 4

4.2.2 Layout and Circuit

Figure 4.2 shows the circuits of the CRF, RCD, and MRF. In this figure, for the sake of simplicity, only 1-read+1-write, out of 8-read+4-write or 2-read+2-write ports, is shown.

All of these circuits have the same width (the vertical direction in the figure) of 64-bit data, which is based on the width of the execution units. Both the register files and the execution units adopt a bit-slice design. Therefore, the 64-bit width is arranged as the 64 repetitions of one-bit-slice width.

4.2.2.1 Conventional Register File

The upper diagram in Figure 4.2 shows the CRF.

The design in this figure adopts a hierarchical bitline architecture. The array is divided into 8 subarrays which consists of one local sense amp and 16 memory cells. As shown in the figure, the local sense amp is placed in the center of the 16 memory cells so as to reduce the length of the local bitlines.

This design uses single-ended bitlines. Single-ended bitlines are particularly effective at reducing the area of highly-multi-ported memory.

4.2.2.2 Register Cache of NORCS

The middle diagram in Figure 4.2 shows the RCD.

The RCD uses the same cells as the CRF. Shioya, et al. shows that 8 entries of a register cache with 128 entries of physical registers have sufficiently good performance in the original article [2]. In this case, the height (the horizontal direction in the figure) of the register cache is roughly $8/128 = 1/16$.

The tag arrays are placed in the position of the decoders of the CRF. The author explains the RCT in detail in Section 4.2.3.

The total number of entries is only half of one subarray of the CRF. Therefore, the register cache does not adopt a hierarchical bitline structure. The author evaluates the bitline structure in Section 4.3.2.2.

4.2.2.3 Main Register File of NORCS

The lower diagram in Figure 4.2 shows the circuit of the MRF.

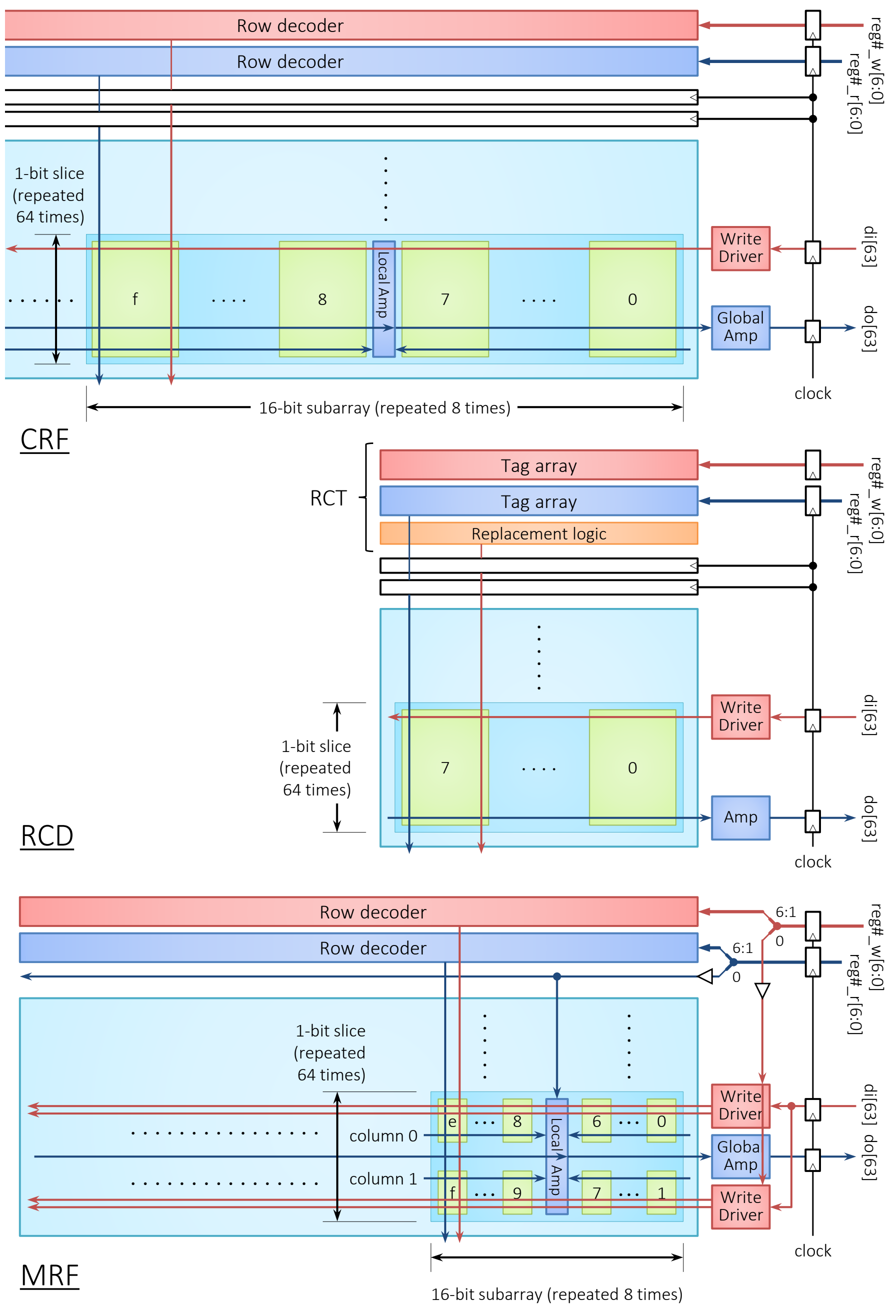


Figure 4.2: Circuits of CRF, RCD, and MRF

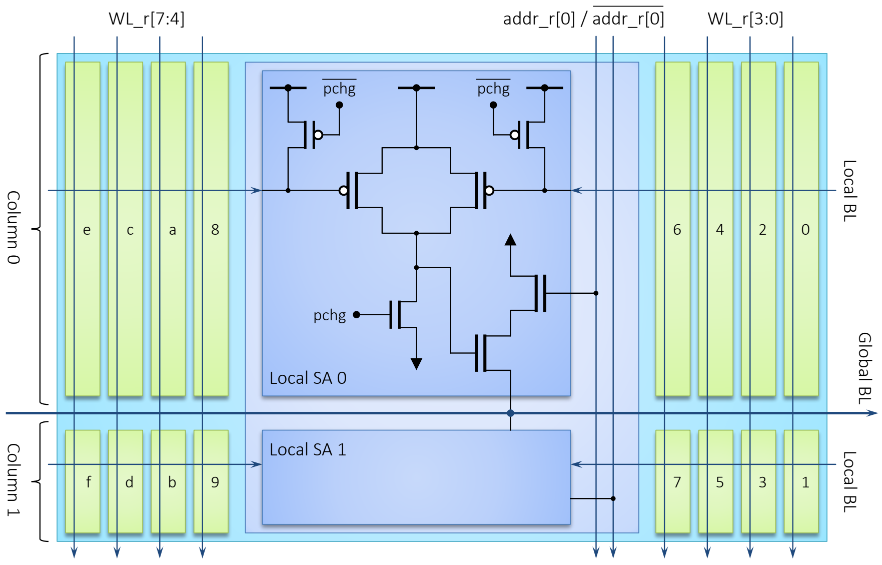


Figure 4.3: Local Sense Amp of main register file

The MRF uses a RAM cell with fewer ports. The height and width of the cell is smaller than that of the CRF, because there are fewer ports. As described above, the width (the vertical direction in the figure) of a 1-bit slice are determined by the width of the execution units. Therefore, the author arranged two memory cells in the width of a 1-bit slice.

Each subarray consists of 16 memory cells and a local sense amp. A wordline is shared by the two memory cells of different columns. One bit out of the addresses (reg#_r[0]/$\overline{\text{reg\#\_r[0]}}$ in the figure) selects column 0 or column 1.

Read Access of MRF

Figure 4.3 shows one subarray from Figure 4.2. The memory cells in the upper and lower half of Figure 4.3 belong to column 0 and column 1, respectively. When one wordline is activated, the two corresponding memory cells of column 0 and column 1 are selected and the two local bitlines are driven. The local sense amp selects one of two local bitlines and drives the global bitline.

Write Access of MRF

Unlike for read, the author adopted differential bitlines for write.

In Figure 4.2, reg#_w[0]/$\overline{\text{reg\#\_w[0]}}$ selects one write driver. The selected write driver drives its bitline pair, whereas the unselected write driver retains its bitline pair in a precharged state. As a result, the contents of the memory cell connected to the non-discharged bitline pair do not flip, even if its wordline is asserted.

This differential bitline design is smaller than one that uses two write wordlines for the two columns to select one of the two columns.

4.2.3 Tag Array of Register Cache

Figure 4.4 shows a circuit diagram of RCT and RCD for a 2-issue core. In this figure, the register cache is composed of three arrays. The lower array in the figure is the data array (RCD), which consists of 4-read+2-write RAM cells, as described in the previous subsection. The upper and middle arrays are the tag arrays (RCT) consisting of CAM cells.

The signals reg#_(01)L/R/D are the register numbers allocated to the left and right source and destination operands; where (01) is the issue port number, with values of either 0 or 1. The register numbers reg#_(01)D are written to the write ports of the CAMs. The register numbers reg#_(01)L/R/D are input to the search ports of the CAMs to associatively search the entries that have the same register numbers written to them.

The RCT is composed of a replicated pair of arrays for the destination (upper in the figure) and the source (middle) so as to reduce the effective number of search ports. The contents of the two arrays are replicated, that is, the same reg#(01)D are written to the same entries of both arrays at the same time.

The flip-flop between the tag and data arrays is the valid flag of each entry.

In the following two subsections, the author describes read, and write (and allocate) operations.

4.2.3.1 Read from the Register Cache

The read operation is the same as for a usual data cache. The tag array for read takes reg#_(01)L/R. If these register numbers match the contents, the corresponding match lines are asserted. Then, the results is AND-ed with the valid flags to produce the read_hit signals.

The values of the read_hit signals will be used as the values of the read wordlines of the RCD in the CR stage as they are. Therefore, RCD does not need a decoder.

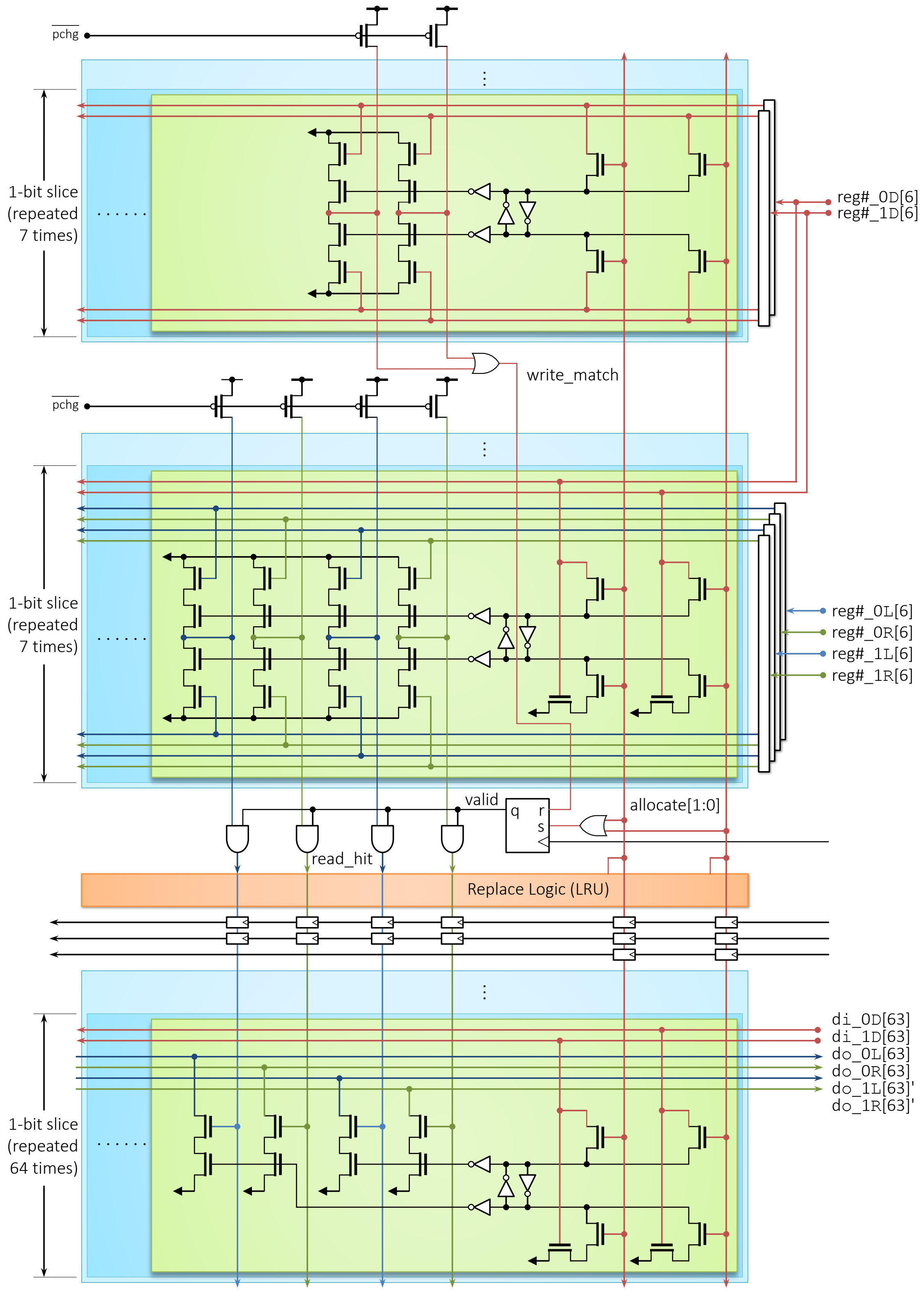


Figure 4.4: Circuit of RCT (upper and middle) and RCD (lower)

4.2.3.2 Write to the Register Cache

As described in Section 3.2, almost all of the writes cause misses. Therefore, it is not useful to exactly handle rare write hits.

In the usual data cache, allocation of a new entry must be started after write hit detection. A store instruction modifies only a part of the entry; this entry still has valid data that is unmodified (Section 3.2). Consequently, on a write hit, the hit entry must be used, and a new entry must not be allocated. However, this sequence of allocation after write hit detection will require one more cycle.

In the design in Figure 4.4, whether a write hits or misses the register cache, another entry is (newly) allocated to the write. The hit entry, if any, is invalidated in the same cycle. This technique works from the following reason: the newly allocated entry will have a completely new register value written, and the hit entry on the write can be regarded as obsolete.

Because of this blind write allocation, all of the write accesses write to RCD, regardless of whether the write accesses hit or miss the register cache.

4.3 Evaluation

The author designed NORCS and a conventional register file for a 4-issue core using **FreePDK45** [44], an open source process design kit (PDK) in 45 nm technology.

This section shows the evaluation results of these designs. First, the author summarizes the design methodology in Section 4.3.1. Next, the author optimizes the subarray configuration of CRF, RCD, and MRF in Section 4.3.2. Finally, Section 4.3.3, 4.3.4, and 4.3.5 show the detailed results on the area, energy consumption, and latency, respectively.

4.3.1 Design and Evaluation Methodology

4.3.1.1 Design Methodology

The author designed the layout of memory cells and arrays with Cadence Virtuoso. The author performed manual layout for the memory cells.

The author extracted RC parasitics with Mentor Calibre xACT 3D, and simulated with Synopsys HSIM.

The author verified the design with Mentor Calibre to comply with the DRC rules of FreePDK45. Although some of the DRC rules are too strict for memory cells, this design complies with all of the rules.

Table 4.2 summarizes the EDA tools that the author used.

4.3.1.2 Process Parameters

Table 4.3 summarizes the wire pitch in FreePDK45 to compare it with commercial Intel 45 nm technology [45]. The parameters shown strongly affect the size of memory arrays. As shown in the table, FreePDK45 is almost equivalent to Intel 45 nm technology. Both technologies have three lower metal layers with minimized pitches, and three upper metal layers with double pitches.

The author used the typical conditions of FreePDK45, i.e., a supply voltage of 1.1 V, a temperature of 55 °C, and the typical transistor model.

4.3.1.3 Architecture Simulation

To calculate energy consumption and energy efficiency, the author also performed architecture simulation to obtain several parameters such as the number of register reads and writes, the register cache hit rate, and the IPC.

The author used a very similar environment as the original article of NORCS [2]. The author used the Onikiri 2 simulator [31]. The author used all of the 29 programs of the SPEC CPU 2006 benchmark with the *ref* data sets [3]. The programs were compiled with gcc 4.2.2 −O3. The author evaluated the 100M instructions after the first 1G instructions.

Table 4.4 shows the processor configuration. The configuration is the same as in the original article.

Table 4.2: EDA Tools

Circuit and layout edit	Cadence Virtuoso IC6.1.5.500.15
LVS, DRC verification	Mentor Calibre v2012.3_31.26
RC Extraction	Mentor Calibre xACT 3D v2012.3_31.26
3D Field Solver	Synopsys Raphael E-2010.12
SPICE Simulation	Synopsys Hsim 2012

Table 4.3: Wire Pitch of Intel 45 nm Technology and FreePDK45

Layer		Intel 45nm	FreePDK45	Used for
Metal	M6	360	280	Global Bitline Power/GND
	M5	280		
	M4	240		
	M3	160	140	Wordline
	M2			Local Bitline
	M1		130	Cell
Contacted Gate		160	160	

Table 4.4: Configuration of Processor Core

ISA	Alpha w/ byte-word ext.
width	fetch, issue, commit: 4
inst. window	int:32, fp:16, mem:16
reorder buffer	128 entries
registers	int:128, fp:128
exec. units	int:2, fp:2, mem:2
pipeline stages	fetch:3, rename:2, dispatch:2, issue:2, register read:3
branch pred.	8K:gshare
miss penalty	13 cycles
BTB	2K-entries, 4-ways
L1C	32KB, 4-way, 64B/line, 3 cycles
L2C	4MB, 8-way, 64B/line, 10 cycles
main memory	200 cycles

4.3.2 Subarray Configuration

Before going into detailed evaluation in the next subsections, this subsection tries to fix the (sub)array configurations of the data arrays.

4.3.2.1 Subarray Configuration of MRF

The upper graph in Figure 4.5 shows the trade-off between latency and local bitline length for MRF. The y- and x-axes of the graph are the bitline latency and the data array area, respectively. The curves are plotted for the different numbers of cells per local bitline.

The x-axis of the graph, i.e., the area, also represents the total length of the bitline, because the width (vertical directions in the diagrams) of the CRF, RC, and MRF arrays are exactly the same as described in Section 4.2.2.3.

As shown in this graph, a shorter local bitline shortens the latency of the local bitline, while it lengthens the latency of the global bitline due to the increased number the local sense amplifiers (the detailed reason is described later). As a result, the total latency is minimized at the point of 4 cells per local bitline.

The lower graph in Figure 4.5 shows the total bitline latencies such as shown in the upper graph for different sizes of the driver transistors of the local sense amplifiers. In the graph, three curves are plotted for three relative sizes of the drivers: $\times 0.5$, $\times 1$ (1.1 µm in gate width), and $\times 2$. That is, the curve for $\times 1$ is precisely the same as the curve in the upper graph.

As shown in this graph, even if the drivers are doubled in size, the latency is not sufficiently improved for the increase in area. In both the cases, the capacitance of the global bitline is increased in the following two ways:

1. Its transistor capacitance is directly increased by the increased number or by the increased size of the driver transistors of the local sense amplifiers.
2. The data array area is increased by the increased number of the local sense amplifiers or by the increased size of the driver transistors. Then, its wire capacitance is indirectly increased by the increase in length.

From these results, The author choses 4 cells per bitline and $\times 1$ local sense amplifier for MRF.

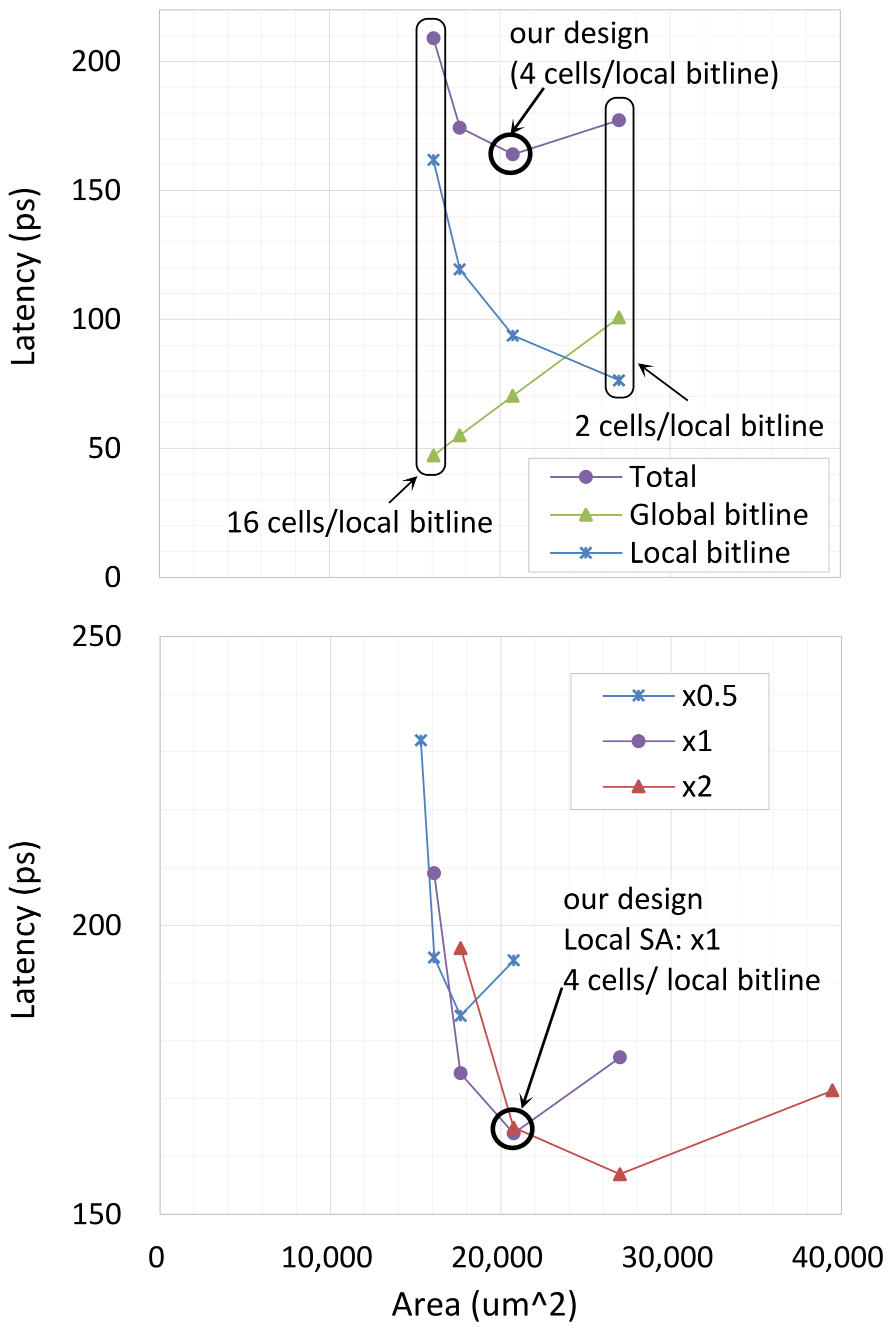
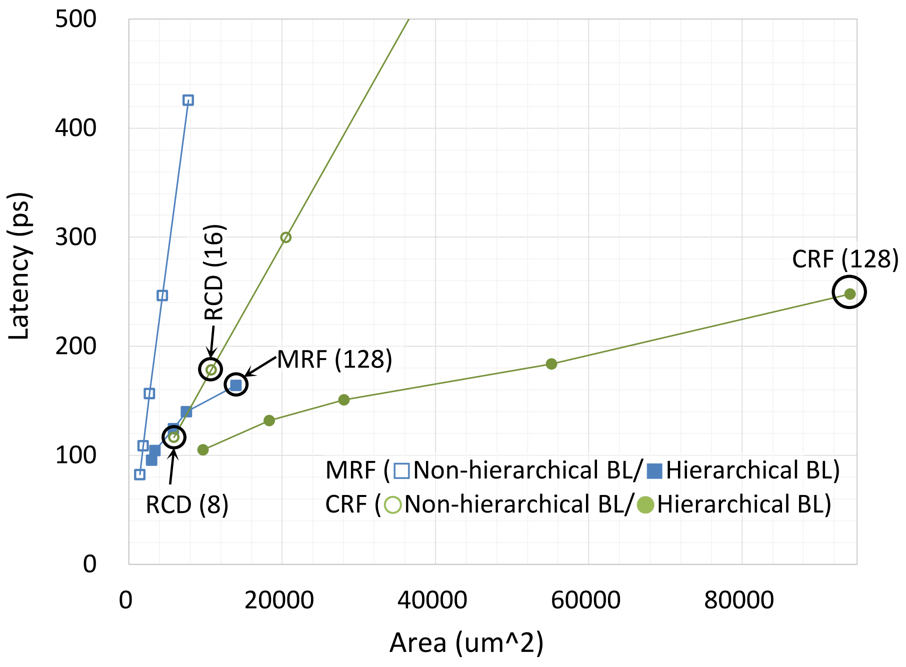


Figure 4.5: Bitline latency for different numbers of cells per local bitline (upper), and for different driver sizes (lower) for MRF.



Entries		MRF						CRF				
		Cells/ LBL		LBSs/ SA		SAs/ GBL		Cells/ LBL		LBSs/ SA		SAs/ GBL
8	=	1	×	4	×	2	=	2	×	2	×	2
16	=	2	×	4	×	2	=	2	×	2	×	4
32	=	2	×	4	×	4	=	4	×	2	×	4
64	=	4	×	4	×	4	=	4	×	2	×	8
128	=	4	×	4	×	8	=	8	×	2	×	8

LBL/GBL: Local/Global Bitline, SA: Subarray

Figure 4.6: Bitline latency for the number of entries.

4.3.2.2 Array Configurations of CRF and RCD

Figure 4.6 shows the total bitline latencies of MRF of 2-read+2-write cells, and those of CRF and RCD of 8-read+4-write cells. In this graph, the filled and non-filled markers are for hierarchical and non-hierarchical bitlines, respectively.

The tabular in the figure shows the subarray configurations for MRF and CRF obtained in Section 4.3.2.1. The ratios between the number of cells per local (Cells/LBL) to the number of subarrays per global bitline (SAs/GBL) are different for 8-, 32- and 128-entry; and 16- and 64-entry, because they are alternately doubled to double the entries. This difference is the cause of the fluctuations on the curves.

For the same area, i.e., for the same bitline length, the latency of MRF is longer than that of CRF both for hierarchical and for non-hierarchical bitlines, because of the difference in capacitance. As described above, MRF is composed of a larger number of smaller cells than CRF. Thus, the bitline of MRF of the same length as that of CRF has the same wire capacitance as and larger transistor capacitance due to a larger number of cells than that of CRF.

While CRF and MRF have 128, RCD has only 8 or 16 entries. As a result, while hierarchical bitline structure drastically reduces the latencies of CRF and MRF, it slightly reduces that of RCD for the increase in area.

4.3.3 Area

4.3.3.1 Layout of Cells and Arrays

Figure 4.7 shows the layout of two 2-read+2-write cells for the MRF (left) and one 8-read+4-write cell for the CRF and the RCD (right). The author used M2 and M3 for the local bitlines and wordlines, respectively.

As shown in Figure 4.7, two 2-read+2-write cells can be arranged within the width (the vertical direction in the figure) of a 1-bit slice.

Figure 4.8 shows the layout of the MRF subarray, which is composed of a pair of local SAs and 16 memory cells, based on the results of the optimization in Section 4.3.5. The local SA accounts for 31.6% of a subarray.

4.3.3.2 Area

Figure 4.9 shows the areas of the conventional register file and NORCS. Overall, NORCS with an 8-entry register cache achieves a 75.2% reduction in area. In this

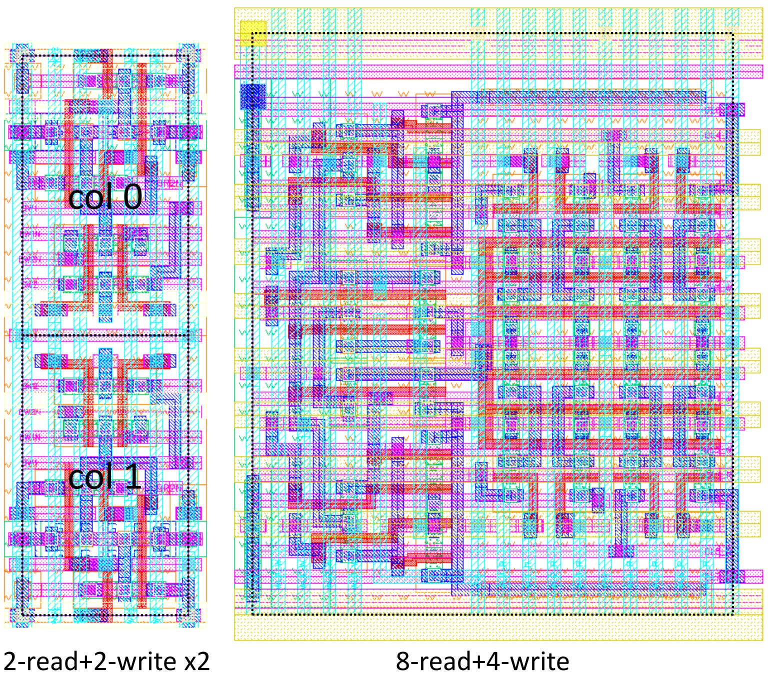


Figure 4.7: Layout of two 2-read+2-write cells for MRF (left) and one 8-read+4-write cell for CRF and RCD (right).

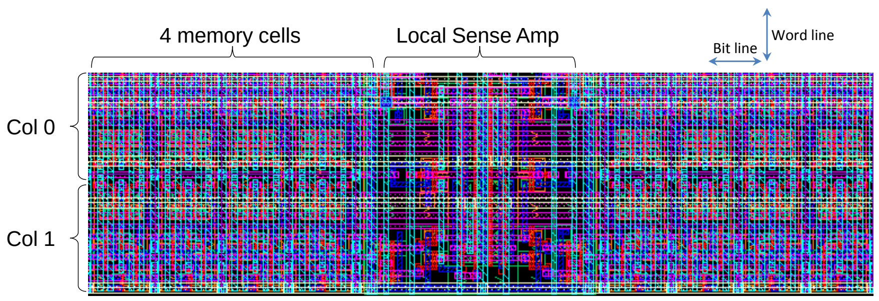


Figure 4.8: Layout of local SA and 16 memory cells.

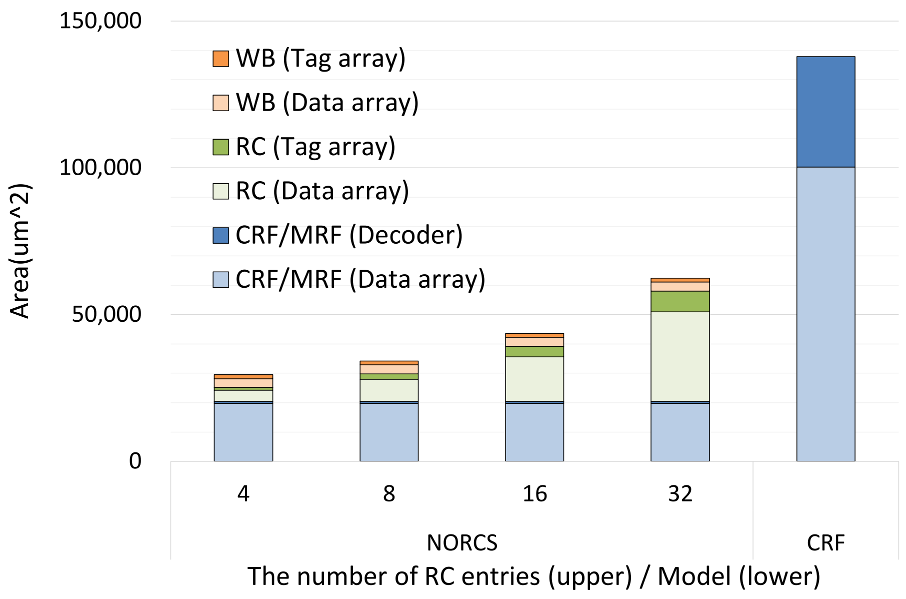


Figure 4.9: Total area of conventional register file and NORCS.

case, the MRF accounts for 59.6% of the total area.

4.3.4 Energy Consumption

The energy consumption of register cache systems strongly depends on both the register cache hit rate and the read-to-write ratio. The author calculates the energy consumption from these parameters obtained using the architecture simulation for the SPEC 2006 benchmark (Section 4.3.1.3).

4.3.4.1 Energy per Access

Figure 4.10 shows energy consumption of NORCS and CRF for one read access and for one write access.

For read access, the energy consumption of NORCS considerably differs as to whether the read access hits or misses the register cache. The relative read energy of NORCS to that of CRF is 18.2% on a register cache hit and 54.3% on a miss. It is mainly because of the smaller energy consumption of the RCD and the MRF as compared to the CRF.

In contrast, write energy does not vary based on whether the write access hits or misses the register cache, because all of the write accesses write to the RCD because of the blind write allocation described in Section 4.2.3. The relative energy consumption of NORCS to that of the CRF is 67.6%. The write energy of NORCS is increased by the WB.

4.3.4.2 Total Energy

Figure 4.11 shows the energy consumption for the SPEC 2006 benchmark averaged over one access, which is calculated as follows: first, energy per read and write accesses in Figure 4.10 is accumulated for 100M instructions (after the first 1G instructions) for all 29 programs in SPEC 2006. Then, the accumulated energy is divided by the total number of read and write accesses (Section 4.3.1.3).

As shown in Figure 4.11, a larger register cache affects the total energy consumption in the following two ways. On one hand, a larger register cache increase the register cache hit rate, and reduces the main register file read accesses which consumes more energy than the register cache. On the other hand, a larger register cache increases the energy consumption of the register cache itself.

As a result, the total energy consumption is minimized when the size of RC is 8. In this case, a 48.2% reduction in the total energy consumption of the CRF is achieved.

4.3.5 Latency

Figure 4.12 shows the latencies calculated by SPICE simulation for two types of pipeline stage allocation; the left and right halves are for two and three cycles, respectively. The allocation of each modules is as follows:

RCT RCT is always allocated to the first cycle.

RCD RCD is accessed on register cache *hit*. To reduce the complexity of the operand bypass network, RCD is allocated to the last cycle [2]. Thus, RCD is allocated to the second or third cycle in the two- or three-cycle allocations, respectively; and, nothing is allocated to the second cycle in the three-cycle allocation.

Here, the author should note that a decoder is not necessary for RCD, because RCT directly provides hit entry with the raed_hit signals for the wordlines (Section 4.2.3).

MRF MRF is accessed on register cache *miss*.

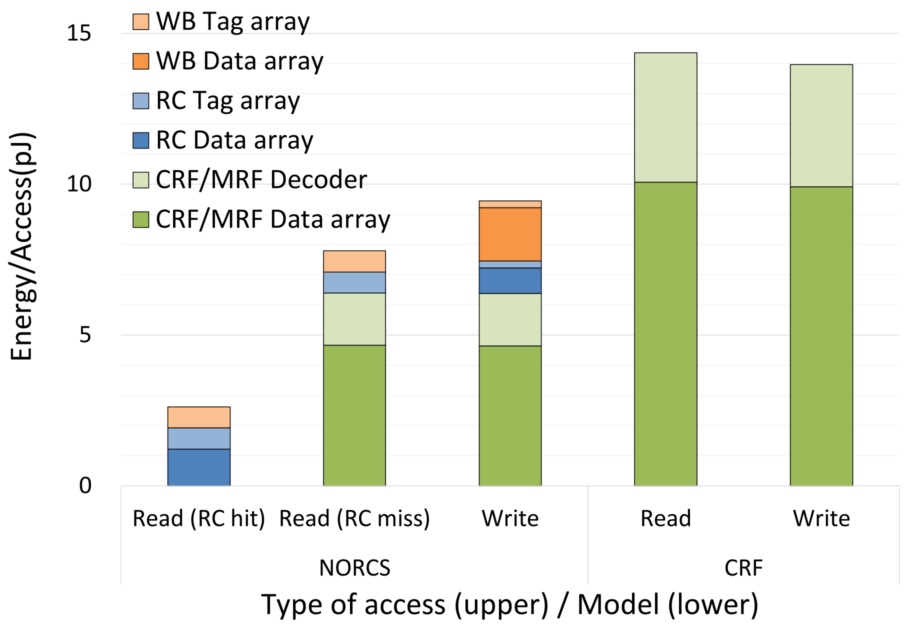


Figure 4.10: Energy consumption of NORCS and CRF per read access and per write access.

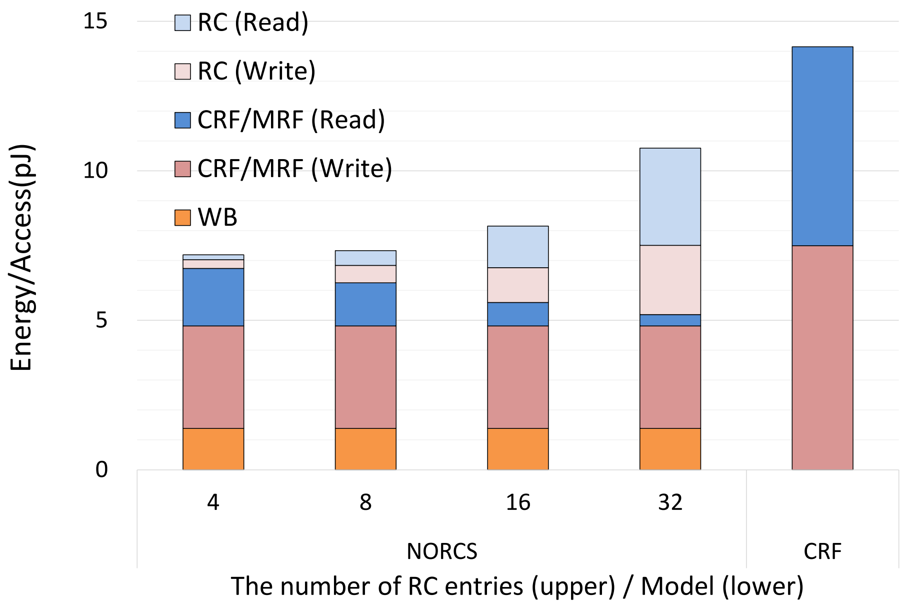


Figure 4.11: Energy consumption of NORCS and CRF for SPEC 2006.

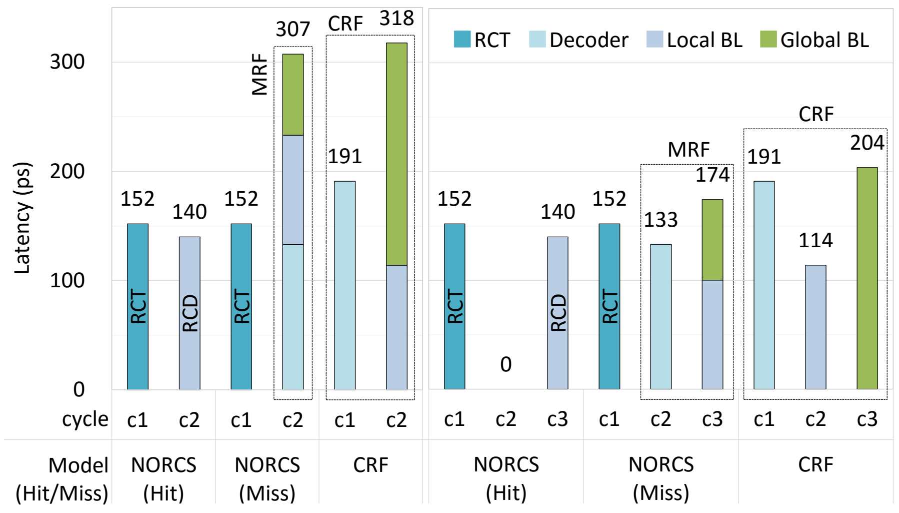


Figure 4.12: Read latency of NORCS and CRF. 2 stages (left) and 3 stages (right)

MRF is allocated to the second cycle in the two-cycle allocation. While in the three-cycle allocation, MRF is divided into two parts of the decoder and local/global bitlines, and allocated to the second and third cycles.

CRF CRF is divided into two parts of the decoder, and local/global bitlines, and allocated to the first and second cycles in the two-cycle allocation. While in the three-cycle allocation, CRF is divided into three parts of the decoder, local bitline, and global bitline; and allocated to the first, second, and third cycles.

The latencies of the critical paths of NORCS are 307 ps and 174 ps for the two- and three-cycle allocation, respectively. In both the cases, they are shorter than those of the CRF. Even in the two-cycle allocation, a clock frequency of 3 GHz can be achieved for the typical case, which is realistic for processors in 45 nm generation.

The three-cycle allocation provides more timing margin in return for IPC degradation caused by a one-cycle additional miss penalty of speculation such as branch prediction. As detailed in the later graph, this IPC degradation accounts for 1.9% for NORCS with 8-entry register cache.

Both in the two- and three-cycle allocations, the latencies of the critical paths of NORCS are comparable with (shorter than, in actual) those of CRF; and it is fair to allocate the same number of cycles both to NORCS and CRF.

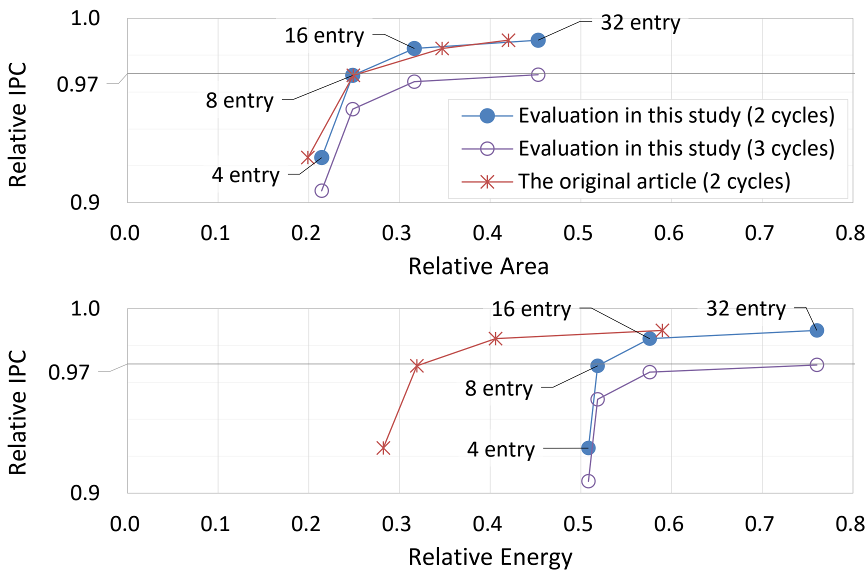


Figure 4.13: Relative IPC vs. area and energy.
The curves are plotted for different size of register cache: 4, 8, 16 and 32. The area and energy of the original article are cited from [2].

4.3.6 Area and Energy Efficiency

Figure 4.13 shows the relative IPC versus the relative area and energy consumption. The values of area and energy consumption are simply derived from Figures 4.9 and 4.11, respectively. The relative IPC is averaged for 29 programs using SPEC CPU 2006. The author shows the relative IPC for both two cycles and three cycles register file.

The graphs show the trade-off between IPC and area, and between IPC and energy consumption. In general, a technique will achieve higher IPC if it uses more area or energy. As a result, there will be upward curves in these graphs. When applying a technique to reduce area and energy while keeping the IPC, it is important to plot one point within the region close to the top of the graphs and as close to the y-axis as possible.

In each graph, the relative IPC is greater than 0.97 for a register cache size of greater than or equal to 8, and saturates above 16.

4.3.7 Difference from the Original Article

The author compared these evaluation results with those of the original article [2]. After the detailed design in this study, the author could confirm that the results in the original article were approximately correct. As summarized in Table 4.5, the results on area and latency in cycle are quite consistent, while those on energy is not.

Difference in Energy Consumption

The results on energy consumption considerably differs between the original article and this study. As for the reduction in energy consumption for an 8-entry register cache, the result of the original article and paper are 68.1% and 48.2%, respectively.

This difference is caused by the difference in the estimation of static energy. In the results of FreePDK45, static energy is considerably smaller than dynamic energy. In contrast, CACTI seems to overestimate static energy.

Although CACTI and FreePDK45 are both use 45 nm technology, their release dates are quite different. CACTI was released when static energy was supposed to be more serious than it is today.

Difference in Latency in Cycle

In the original article, they simply extrapolated the latencies in cycle from the ratio in the areas, and assumed that that of NORCS is the same as that of conventional register files. This latency assumption is quite important because it means

Table 4.5: Results of the original article and this study for 8-entry register cache

	Original	This study	
Relative Area	−75.1%		−75.2%
Relative Energy	−68.1%		−48.2%
Latency (cycles)	2-cycle	2-cycle	3-cycle
CRF	2	2	3
NORCS (RCT + MRF)	2	2	3
RCT	1	1	1
RCD	1	1	1
MRF	1	1	2

that the NORCS pipeline which is not for latency reduction purpose does not increase the pipeline depth from conventional register files.

In reality, as shown in Figure 4.6, it is not simple to estimate latencies from the ratio in bitline lengths or areas because of hierarchical bitline structure. However, as summarized in Table 4.5, this latency assumption and its resulting IPC in the original article are confirmed by the detailed design in this study.

Chapter 5

Multibanked Register File Systems

Multibanking is a technique typically used for the main memory of vector processors, there is no *standard* implementation of a multibanked register file. Therefore, this chapter devotes several pages to show a possible *plain* multibanked register file before going into the proposals in Chapters 6 and 7. The author compares the proposed techniques with the *plain* multibanked register file described in this chapter.

First, Section 5.1 shows a possible structure of plain multibanked register file. The most important part of the structure is a data switch. The interconnection of conventional multiport register file and execution units is simple; the read and write ports of the register file are directly connected to the input and output ports of the execution units, respectively. In contrast, a multibanked register file needs read data and write data switches for any-to-any routing between the execution units and banks, because any execution unit can read or write any bank.

Section 5.2 discusses an operand bypass network. The multibanked register file composed of single-port banks can bypass the values on its bank ports. These bank ports partially substitutes for the operand bypass network.

Section 5.3 shows a difference between a multibanked register file and a multibanked main memory. Unlike an address of a multibanked main memory, a physical register number of a multibanked register file is randomized by register renaming. Because of register renaming, a physical register number space does not need to be a linear space. The number of banks not a power of 2 is also applicable.

Lastly, Section 5.4 shows the related works.

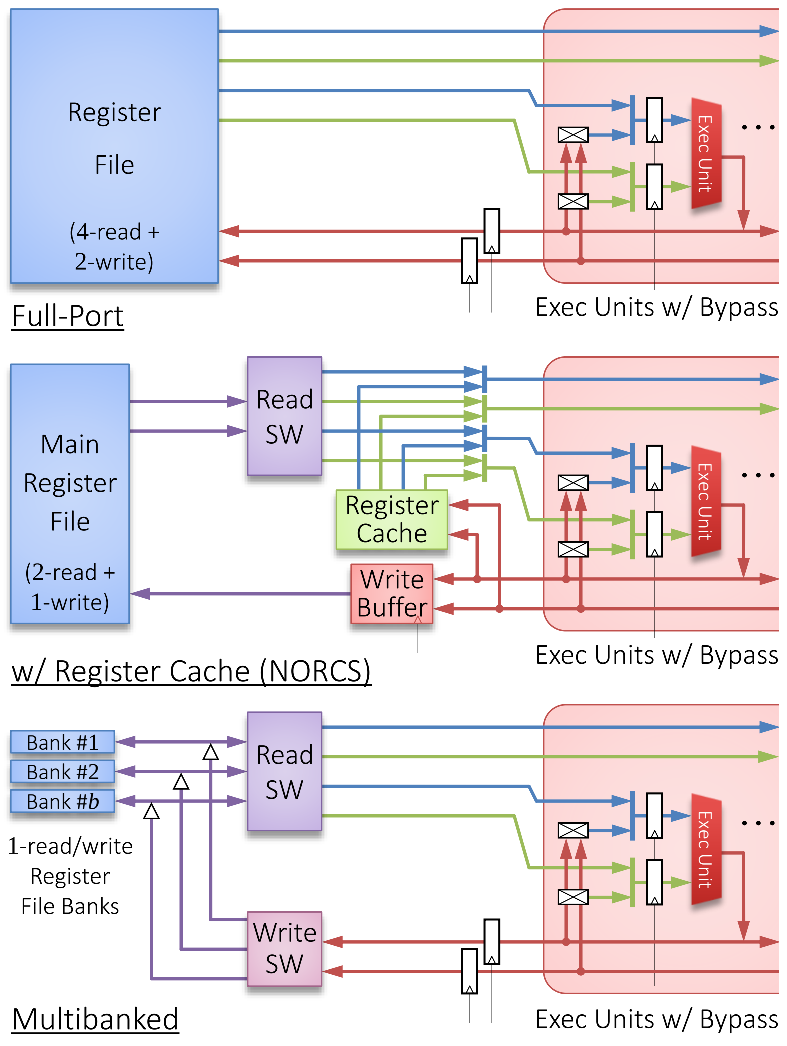


Figure 5.1: Datapaths of Full-port, Register Cache, and Multibanked Systems.

5.1 Structure

Figure 5.1 (lower) shows the datapath of a multibanked register file. Figure 5.2 adds the control to the left half.

5.1.1 Datapaths

A multibanked register file has read and write switches for any-to-any routing between the execution units and banks. As described in Chapter 1, the banks are more than an order of magnitude smaller than the original full-port register file in

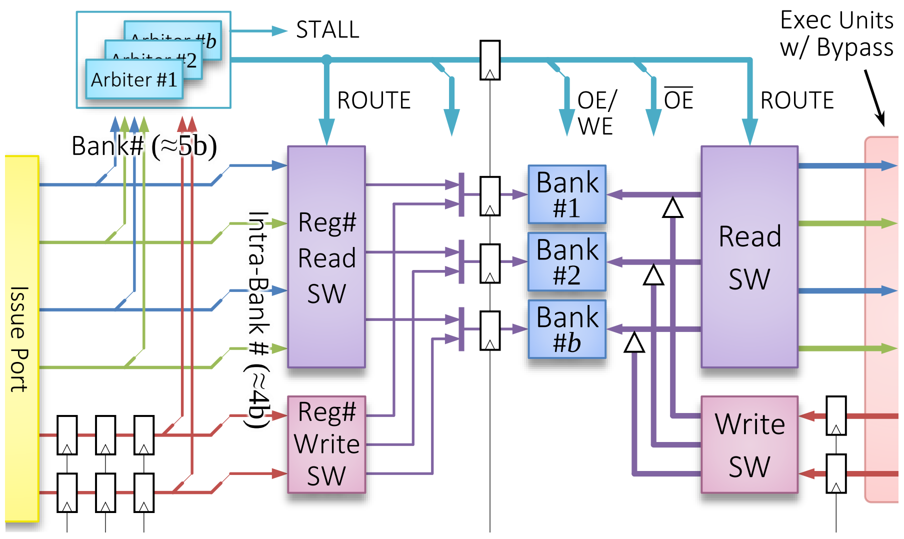


Figure 5.2: Control and Datapath of Multibanked Systems.

area.

Circuit Size of Data Switches

These switches are also more than an order of magnitude smaller contrary to expectation. Thus, the author gives an intuitive explanation on the circuit size of the switches before quantitative evaluation in Chapter 9.

The circuit size of these switches can be estimated via a 64-bit r-read+w-write memory with only 1-entry. This 1-entry memory works as a 64-bit any-to-any switch by writing a 64-bit word to any of the w write ports, and reading it from any of the r read ports. This 1-entry r-read+w-write memory is two orders of magnitude smaller than an r-read+w-write register file with hundred entries. Obviously, an actual switch can be optimized as a switch.

The read and write switches are a few times larger than this memory because they are not r-read+w-write, but r-read+b-write and b-read+w-write, respectively, where b is the number of banks and $b > r = 2w$. Finally, these switches are more than an order of magnitude smaller than the r-read+w-write register file.

The any-to-any routing and memory functions are integrated in a full-port, while distributed into the switches and banks in a multibanked register file. It is safe to say that a multibanked register file is smaller because of this function distribution at the risk of bank conflicts.

5.1.2 Control

The physical register number from the instruction issue port is used as the concatenation of the bank number and intra-bank number fields, which are 4- to 5-bit wide.

The system has the arbiters and the intra-bank number routing switches. The bank number field of the register number is decoded and distributed to the arbiters. Then, the intra-bank number field is routed to the bank through the switches controlled by the arbitration result.

Circuit Size of Arbiters and Register Number Switches

The arbiters and the register number switches are further smaller than the 64-bit datapath described above, mainly because they are around 4-bit wide.

The arbiter is equivalent to a select logic of an instruction scheduler that selects one out of the same number of instructions as the register file banks with fixed priority. Thus, its latency is a fraction of a half-cycle time usually allocated to the select logic that selects two or more out of 64 or more instructions. Note that the arbiters work in parallel with one another.

The intra-bank register number is 4-bit wide, and the register number routing switches are approximately 4/64 of the read/write switches for 64-bit data in area.

As shown by the pipeline registers in the middle of Figure 5.2, one cycle is assigned to the arbitration and register number routing throughout this thesis.

5.2 Reduction of Operand Bypass Network

The multibanked register file can reduce the operand bypass network only for instructions executed back-to-back as in a register file with half-cycle latency, because the bank ports partially substitutes for the original bypass. Figure 5.3 shows the pipeline with a multibanked register file to explain operand passing. In this figure, the result of I_p is passed to I_3, I_2, and I_1 as follows:

Bank Because of the short latency of the banks, I_3 can read the result of I_p through the bank.

Bank port Because the bank is 1-read/write, when the result appears at the bank port to be written, I_2 can read it through the read switch (Figure 5.1 (lower)).

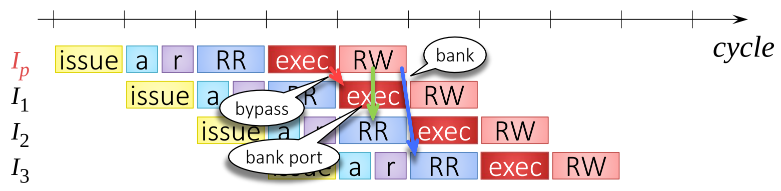


Figure 5.3: Operand Passing in Multibanked System.
The stages are: issue = instruction issue, a = arbitration, r = register number routing, RR = register read, exec = execute, and RW = register write.

Bypass Consequently, only I_1, which is executed back-to-back with I_p, must receive the result through the usual bypass in the execution unit.

Thus, the control logic for the original bypass can be used to control the passing via the bank ports.

5.3 Register Number for Multibanked Register File

The physical register number for a multibanked register file is similar to but different from the address for a multibanked main memory.

Randomness of Register Number

In a multibanked main memory, consecutive addresses reside in different banks to prevent bank conflicts in particular on continuous access. In contrast, it is meaningless to arrange the register numbers for the banks, because the register numbers are randomized by register renaming as shown in Section 2.2.

A cycle-accurate simulator in Chapter 8 reproduces that this shuffling actually randomizes the bank accesses.

Number of Banks

The number of banks of a multibanked main memory should be a power of 2 to avoid a complex operation such as division to obtain the bank number from the address [46]. A multibanked register file can more freely choose the number of banks. This is because the physical register number is not a consecutive sequence number but a unique identifier, i.e., a tag; thus, the physical register number can be the concatenation of the bank and intra-bank numbers. The number of banks not a power of 2 slightly decreases the utility of the bank number field.

From the same reason, a prime number of banks does not have a direct effect in reducing bank conflicts for multibanked register file.

5.4 Techniques to Reduce Bank Conflict

As mentioned in Chapter 1, some techniques have been proposed to reduce the bank conflict probability of a plain multibanked register file without increasing the banks but with complex mechanisms.

Register Access Queue [25]

Hironaka, et al. proposed scheduling queues for register accesses. An instruction is divided into one execution and two register read operations. They are dispatched to the separate queues, and scheduled so that no bank conflict occurs.

However, the total scheduling logic is tripled for one execution and two register read operations. Moreover, the register access queues are more complex than usual instruction queues. To schedule register read operations so that no bank conflict occurs, the register numbers of ready instructions must be associatively compared with one another in the queues.

Additionally, it is difficult to predict *safe* combinations of accesses which do not cause bank conflicts. It is not probable that learned safe combinations will not cause bank conflicts, because the register numbers are randomized as described in Section 5.3.

Register Multi-mapping [26]

Duong et al. proposed a technique to allocate to an instruction two registers in different banks. The instruction writes the result into both the registers. Even if the read access to the first register fails due to a bank conflict, there remains a chance to read the second in another bank. The pipeline is stalled if read accesses to both the registers fail because of double conflicts.

However, the overhead of multi-mapping is unacceptably large. A a naïve implementation of double mapping also doubles the registers. The register management logic, such as the register mapping table, the active and free lists, is also doubled.

Delayed Register Allocation [26, 27]

In addition to read, write accesses also cause bank conflicts. Write accesses have more alternatives to cope with because they have longer slack time to finish than read. Park et al. proposed the *delayed register allocation* [27]. The register multi-mapping described above adopted the same technique in combination [26].

This technique allocates a *virtual tag* to an instruction in the rename stage to resolve dependency among instructions; then, actually does a *physical register* immediately before the writeback stage so that no bank conflict occurs on writeback.

However, an extra mapping table is required to map the virtual tag to the register. This table is almost equivalent to a full-port register file with a 1/4 bitwidth, mainly because the tag is roughly 1/4 of the register in bitwidth. Park et al. added an extra cycle to read this table before reading the multibanked register file.

Chapter 6

Bank-Aware Instruction Scheduler

This chapter describes the proposed **bank-aware scheduler**. As mentioned in Chapter 1, prior studies briefly mentioned the possibility of bank-aware scheduling or rejected it because of the increased latency [23,30]. However, the detailed design clarifies that the latency of the logic is not practically increased.

In addition, the author made following observatios:

1. Accesses that obtain their operands from the operand bypass network can be excluded from the bank arbitration.
2. The two accesses to the two operands of an instruction can cause a bank conflict, and resolving this type of a bank conflict requires an additional cycle.
3. However, some of conflicts are caused by the two accesses to the same register value to calculate the square or double of the value, and can be excluded.

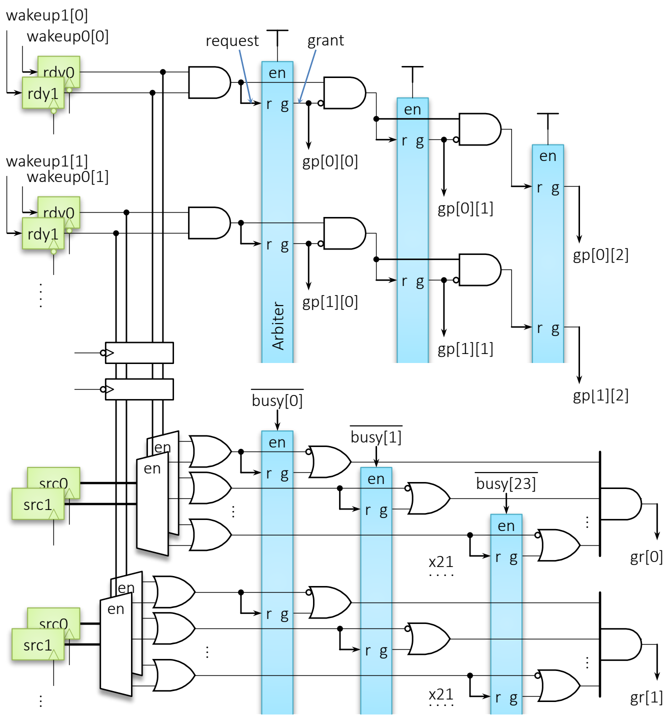


Figure 6.1: Proposed select logic (3 instructions to 24 banks).

6.1 Structure

Figure 6.1 shows the proposed select logic that selects three instructions to a 24-bank register file. The upper half of the figure represents conventional select logic composed of *cascaded* three cascaded arbiters; each arbiter selects at most one from at most W requests, where W is the instruction window size [47]. These arbiters work *in series* by withdrawing the requests to the next arbiter when granted. They produce gp[i][p] for the i-th instruction to be issued from the p-th issuing port.

The lower half is comprised of the **read arbiters** for the 24 banks added for the proposed logic. The physical register numbers allocated to the source operands are stored in the src0/src1 registers, which are parts of the instruction window entries. The bank numbers of these registers are decoded, bit-wise ORed, and distributed to the arbiters, which are identical to those in the conventional select logic stated above. When all of the read requests for i-th instruction, if any, is granted, gr[i] is asserted. The author should note that, unlike the conventional select logic stated above, these 24 arbiters work *in parallel.*

In typical instruction windows, src0/src1 are fields of the rows of the wakeup

CAM [47]. In this case, it is necessary to read the CAM in order to know the bank numbers of ready instructions for arbitration. Thus, the point of the proposed logic is to change these fields into the registers so that the bank numbers can be routed to the arbiters without reading the CAM.

Though not shown in the figure, the **write arbiters** exist that produce gw[i] in almost the same way as the read arbiters except that an instruction has only one destination operand. The read and write arbiters cannot be merged because the cycles when the target banks are used are different between read and write. As shown in the figure, the read arbiters are disabled by the busy signals when the bank is used for a write request of instructions issued in previous cycles.

Finally, the i-th instruction is selected to be issued from the p-th port when gp[i][p] && (gr[i] && gw[i]) == 1.

Size and Latency

The read/write arbiters are about $(24/w)$ times larger than the conventional select logic, where w is the issue width and is 3 in Figure 6.1. However, the latencies of the read/write arbiters are considerably shorter than that of the conventional logic. As stated before, the w arbiters work *in series*, while the 24 read/write arbiters work *in parallel*. Thus, the latencies of the read/write arbiters are basically $1/w$ of the conventional logic, and the critical path of the entire logic resides in the conventional logic. Therefore, the entire latency of the proposed select logic is longer than the conventional select logic by the latency of the 2-input AND gates that appear in the above expression.

6.2 Bypass-Aware Scheduling

As shown in Section 2.5, multibanked register file systems should properly handle operand bypasses.

Figure 6.2 shows the pipelined behaviors of two instructions I_p and I_c. I_c depends on I_p; that is, I_c reads the same physical register that I_p writes.

In the upper figure, I_c is issued immediately after I_p. In this case, the issue of I_c is not avoided by the bank-aware instruction scheduling, because I_c and I_p request the same bank in the different cycles (C_4 and C_5).

However, the select of I_c is delayed for one cycle for some reason, I_c is unexpectedly delayed for two cycles. In the lower of the figure, the issue of I_c is delayed

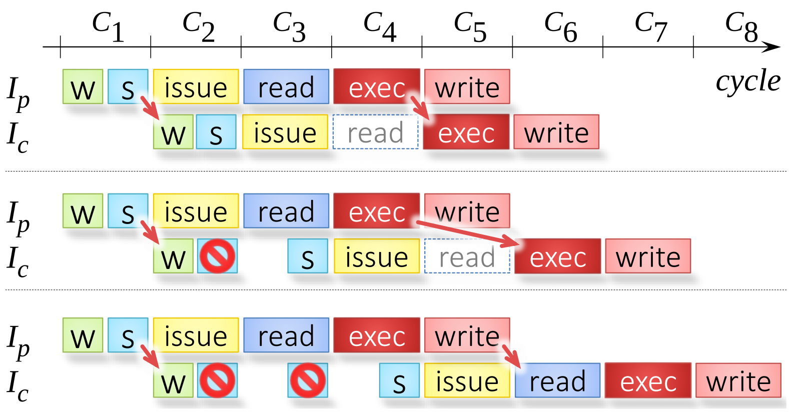


Figure 6.2: I_c is immediately issued (upper), delayed for one cycle for some reason (middle and lower). The stages are W: wakeup, S: select, exec: execution, and read/write: register read/write.

for one cycle. In this case, the value is usually passed through the operand bypass network. However, if left unhandled, I_c meaninglessly requests the same bank. This request is not granted because the bank is used by I_p in the cycle C_5, and the issue of I_c is delayed for another cycle, as shown in the lower of the figure.

To solve this problem for bank-aware instruction scheduler, the author adopts a trick. In Figure 6.1, the operand ready signals, which are set by the wakeup signals, are connected to the enable pins of the decoders for the read arbiters through the FFs shown in the middle. These FFs delay the requests for the read arbiters for two cycles after wakeup.

In the case of Figure 6.2, I_c does not request the bank in the cycles C_2 and C_3. This is the same as I_c does not have the source operands. As a result, I_c is selected in C_3 as shown in the middle of the figure.

6.3 Bank Conflict in One Instruction

A bank conflict can occur between the two source operands of an instruction. In this case, the backend pipeline must be invoked to make a cycle to read the second operand. The author should note that it is difficult to solve this problem with register renaming. Physical registers have already been allocated as destinations for the operands of the dependent instructions, and this mapping cannot be changed for the convenience of the source operands of the instruction to be scheduled.

The author found that some of these types of conflicts are caused when an instruction has two identical source operands. For example, sphinx3 in SPEC 2006 has a number of multiply instructions with two identical source operands to calculate the square. In this case, a bank conflict can be avoided by reading the bank once and duplicating the read value in the read switch.

Chapter 7

Skewed Multistaged Multibanked Register File System

The proposed **skewed multistaged multibanked register file** (MStage) reduces not the bank conflict probability but the pipeline disturbance probability by the second stage. In this chapter, MStage is detailed from Sections 7.1 to 7.3.

The main difference between MStage and the plain multibanked register file is pipelined behavior. Section 7.1 shows the difference between MStage and the plain multibanked register file in pipelined behavior.

The structures of MStage is basically the same as that of the plain multibanked register file except for the second stage. To implement the second stage, MStage adopted a skewed multistaged pipeline. Section 7.1 also shows the unique structure of the skewed multistaged pipeline.

Sections 7.2 and 7.3 show the basic structure of the control logic of MStage. Section 7.2 explains a request aggregation which is the technique to aggregate the requests to the same register file entry in the cycle. Section 7.3 shows the bank arbitration logic. The author also shows the detailed evaluation of the control logic in the latter chapter (Chapter 9), the results of the detailed design is consistent with the consideration in this chapter.

Two different mathematical models are also presented for quantitative explanation of a low disturbance probability of MStage in Section 7.4.

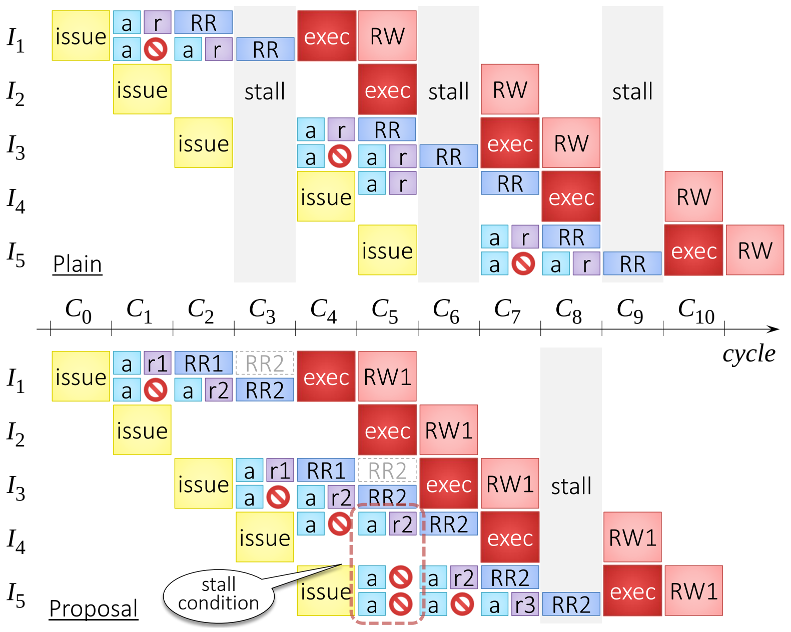


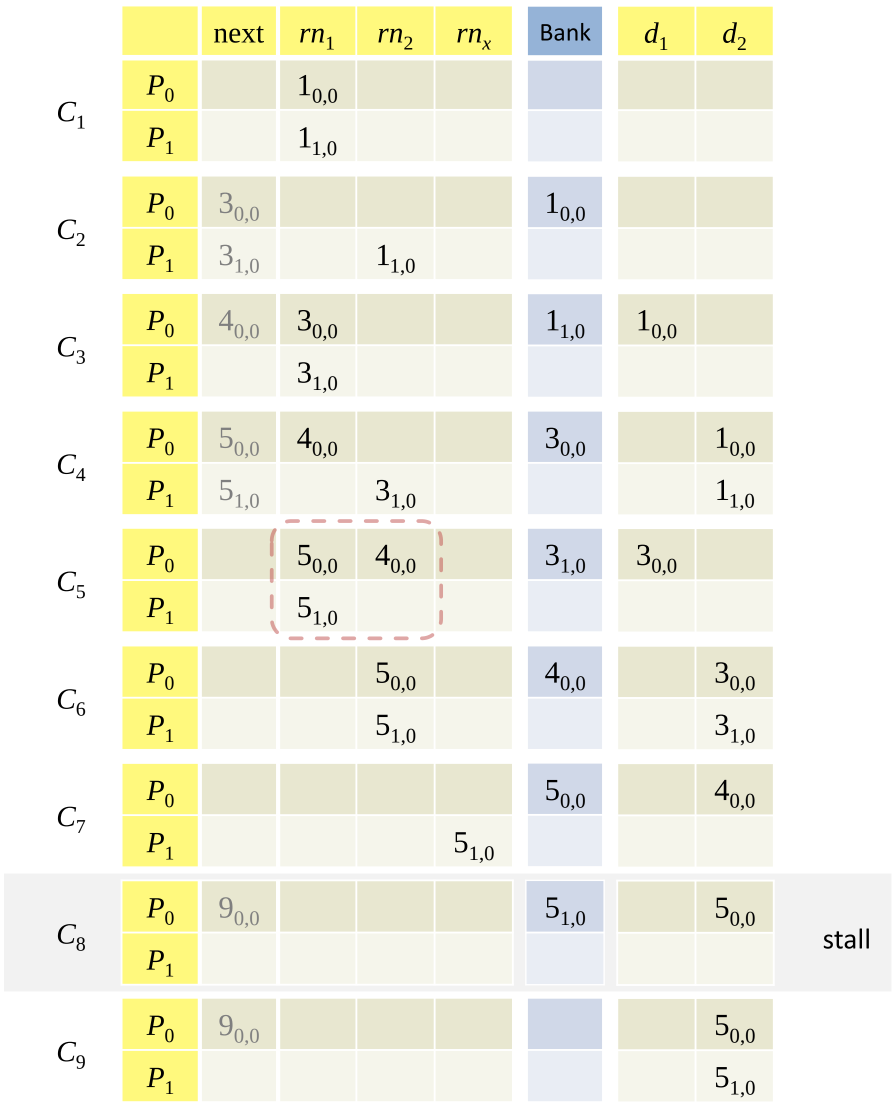
Figure 7.1: Plain and Multistaged Pipelines.
The stages are: issue = instruction issue, a = arbitration, r = register number routing, RR = register read, exec = execute, and RW = register write.

7.1 Skewed Multistaged Pipeline

Figure 7.1 shows the pipelined behavior of a plain and a multistaged pipelines. In this figure, the stages to read the register file are divided into *upper* and *lower* halves to indicate that an instruction has two source operands. The blank stages indicate that the instructions do not have the corresponding source operands. This figure shows the rarest case where all the accesses are to the bank 0 to prevent the diagram from being unnecessarily long.

In this section, the author denotes the accesses of the instruction I_i, from the p-th operand issue port, to the b-th bank as $i_{p,b}$. In Figure 7.1, I_1 has two source operands and they are denoted as $1_{0,0}$ and $1_{1,0}$.

Because all the accesses are to the bank 0, the plain pipeline shown in the upper half of the figure is stalled every time when an instruction has two source operands. For example, $1_{1,0}$ fails to read the bank due to the bank conflict with $1_{0,0}$ as denoted by the blocked sign. The pipeline is stalled in C_3, when $1_{1,0}$ reads the bank 0.



		next	rn_1	rn_2	rn_x	Bank	d_1	d_2	
C_1	P_0		$1_{0,0}$						
	P_1		$1_{1,0}$						
C_2	P_0	$3_{0,0}$				$1_{0,0}$			
	P_1	$3_{1,0}$		$1_{1,0}$					
C_3	P_0	$4_{0,0}$	$3_{0,0}$			$1_{1,0}$	$1_{0,0}$		
	P_1		$3_{1,0}$						
C_4	P_0	$5_{0,0}$	$4_{0,0}$			$3_{0,0}$		$1_{0,0}$	
	P_1	$5_{1,0}$		$3_{1,0}$				$1_{1,0}$	
C_5	P_0		$5_{0,0}$	$4_{0,0}$		$3_{1,0}$	$3_{0,0}$		
	P_1		$5_{1,0}$						
C_6	P_0			$5_{0,0}$		$4_{0,0}$		$3_{0,0}$	
	P_1			$5_{1,0}$				$3_{1,0}$	
C_7	P_0					$5_{0,0}$		$4_{0,0}$	
	P_1				$5_{1,0}$				
C_8	P_0	$9_{0,0}$				$5_{1,0}$		$5_{0,0}$	stall
	P_1								
C_9	P_0	$9_{0,0}$						$5_{0,0}$	
	P_1							$5_{1,0}$	

Figure 7.2: Cycle-by-Cycle Behavior of Skewed Pipeline in Figure 7.1.

7.1.1 Pipeline Behavior

In contrast, the multistaged pipeline has two stages of RR1 and RR2 for reading the register file banks, to reduce not bank conflicts but pipeline stalls as follows:

- $1_{0,0}$ wins the bank 0, and read it in RR1 in C_2.
- As in the plain pipeline, $1_{1,0}$ loses the bank 0 with $1_{0,0}$. However, in the multistaged pipeline, losing $1_{1,0}$ retries in C_2; then, it successfully reads it in RR2 in C_3 without stalling the pipeline.
  When losing $1_{1,0}$ reads the bank in C_3, winning $1_{0,0}$ passes through this stage with the source operand obtained in the previous cycle as denoted by the dashed box in the figure. As a result, both the source operands are provided at the same time for exec in C_4.
- The accesses to a bank are served in FCFS (FIFO) manner. $3_{1,0}$ lost in C_3 is given a higher priority than newly arriving $4_{0,0}$ in C_4, and hence never loses to the latter.
- In contrast, $5_{1,0}$ loses twice in C_5 and C_6, resulting in a pipeline stall in C_8.

In this figure, the multistaged pipeline finishes 1 cycle faster than the plain, owing to less stalled cycles, though the former is 1 stage deeper than the latter. Whereas stalls directly prolong the execution time, an extra pipeline stage only does so by prolonging the penalty on infrequent mispredictions.

7.1.2 Pipeline Structure

Figure 7.3 shows the unique structure of the skewed multistaged pipeline that realizes the above-described behavior. For simplicity, this figure extracts two read ports and one bank from r read and w write ports and b banks.

In the middle of this figure, there are two physical stages of Arbiter and Reg# SW; and Bank and Read SW. These **two physical stages** are skewed and shared by **three virtual stages**. The accesses from the issue ports P_0 and P_1 corresponds to $1_{0,0}$ and $1_{1,0}$ in Figure 7.1 (lower), and they follow the solid arrows as follows:

- In C_1, $1_{0,0}$ wins the bank 0, and the two physical stages are allocated to the first and second virtual stages.
  On the contrary, $1_{1,0}$ loses, and proceeds to the pipeline register denoted as rn_2.
- In C_2, because $1_{0,0}$ goes to the next stage, $1_{1,0}$ can win the bank, and the two

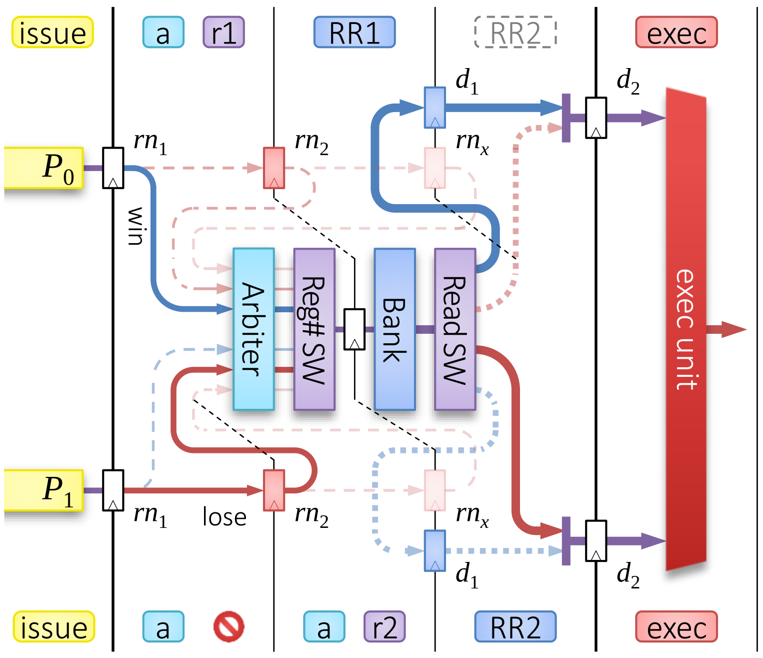


Figure 7.3: Skewed Multistaged Pipeline for 3 Read Ports and 1 Bank.

physical stages are allocated to the second and third virtual stages.

In this cycle, $1_{0,0}$ reads the bank 0, and the read operand is written to the pipeline latch denoted as d_1, to make a pair with the operand of $1_{1,0}$ that will be read in the next cycle, which preserves pipelined behavior.

In this manner, the two physical stages are dynamically allocated to the first and second, or to the second and third, of the three virtual stages depending on whether the accesses win or lose the bank. From the other perspective, in each of the three virtual stages, the physical partial circuit that realizes actual processing dynamically varies with winning or losing.

This is quite unusual for conventional pipelines. In general, a stage means a specific physical module. It is possible that a shared module appears in plural stages, for example, an I/D unified cache appears in the fetch and memory stages. However, it is unusual that a module moves to the neighboring stages.

Cycle-by-Cycle Behavior

Figure 7.2 shows cycle-by-cycle behavior of the skewed pipeline. In this figure, rn_1, rn_2, rn_x, d_1, and d_2 are the pipeline latches in Figure 7.3. The behavior in this figure is the same as Figure 7.1 (lower) and Figure 7.3, and the above explanation for the behavior can be exactly applied to that in this figure. Thus, the author does not repeat the same explanation, but the author should note that the following items with this figure:

- More than one accesses are serialized, and at most one access resides in the Bank column. This condition satisfies the resource restriction of the banks.
- All the accesses that appear in the rn_1 column in C_c also appear in the d_2 column in C_{c+3} (except for stalled cycles), where the difference from C_3 to C_{c+3} comes from the number of virtual stages. This condition ensures the pipelined behavior.

  In particular, the pair of $5_{0,0}$ and $5_{1,0}$ which appears in rn_1 in C_5 also appears in d_2 in C_9 because of the presence of the stalled cycle in C_8. In C_8, $5_{0,0}$ has already arrived at d_2; and then, it stays there until C_9 because of the stall, when $5_{1,0}$ catches up with $5_{0,0}$ in d_2.

7.1.3 Pipeline Stall

As explained in Section 2.6, stalling is more advantageous than rescheduling for multibanked systems.

Stall Condition

The multistaged pipeline is stalled if a bank has a total of 3 or more accesses in a cycle. In Figure 7.1 (lower) and Figure 7.2, this condition is met in C_5 as denoted by the dashed box with rounded corners. In this case, the bank 0 has one access being served and two newly arriving accesses. In Figure 7.2, $5_{1,0}$ proceeds to rn_x in C_7; then, reads the bank 0 in the stalled cycle of C_8.

Critical Paths for Stall

As mentioned in Section 2.6, in general, stall logic can make critical paths; fortunately, this does not hold true for the multistaged pipeline. In Figure 7.1, a stall condition is detected in the middle of C_5; then, the pipeline is actually stalled in C_8. Thus, the tree to distribute the stall signal can take longer than 2 cycles. Consequently, it is practically impossible for the logic to make critical paths.

7.2 Request Aggregation

If two or more instructions in the scheduler have the same source operand, it is probable that they are woken up and issued at the same time, and then, will cause a bank conflict for that operand.

This problem of accesses to the same register is specific to multibanked register files. Designers of multibanked *main memories* inevitably consider consecutive or stride accesses, but they have hardly considered accesses to the same address.

Tseng et al. proposed to share the read port among the read requests to the same register [23, 24]. The author also adopted **request aggregation** to the same register. As Tseng et al. called their technique as read sharing, they aggregate only read accesses. In contrast, the author aggregates both read and write accesses. Because the bank is 1-read/write, a data to be written can be read from the same port as shown in Section 5.2.

When two or more accesses to the same register are detected, the access with the highest priority requests the bank for the others. When this aggregated request is granted, all of the accesses receive the grant signals for the bank. After that, the following processes are automatically performed depending on whether the accesses are read or write:

Read and Read When two or more read requests are aggregated, the read switch duplicates the data read from the bank controlled by the grant signals.

Write and Read Aggregation of a write and a read request realizes the operand passing via the bank port described in Section 5.2.

These two processes are combined for one write and two or more read accesses.

Comparator Array

For this request aggregation, an array of comparators shown in Figure 7.4 is placed before the arbiters. As shown in this figure, the write accesses are given higher priority than the read accesses to ensure read-after-write dependency. Thus, the comparators can be omitted or added without affecting the correctness of the behavior. In this figure and the evaluation in Chapter 8 and 9, the comparators are provided only for the newly arriving accesses, and the match result is reused in the second stage when aggregated requests are not granted. Instead, extra comparators

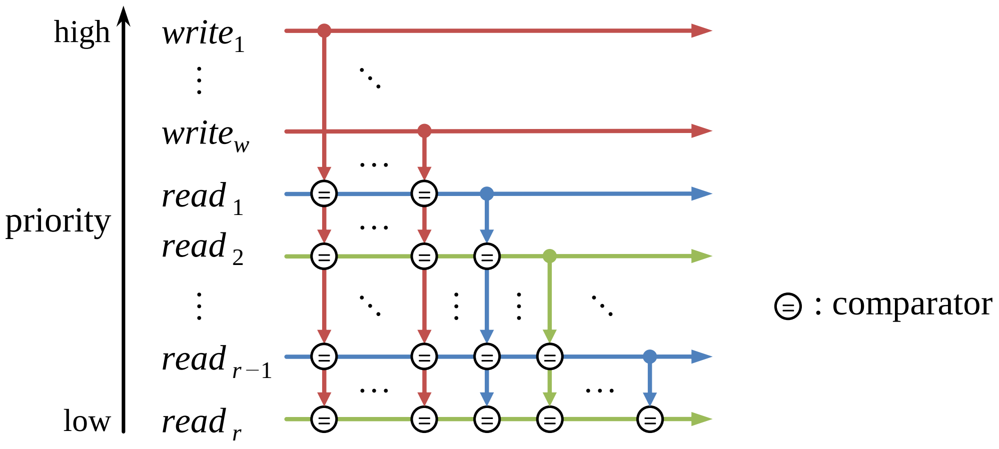


Figure 7.4: Comparator Array for Request Aggregation.

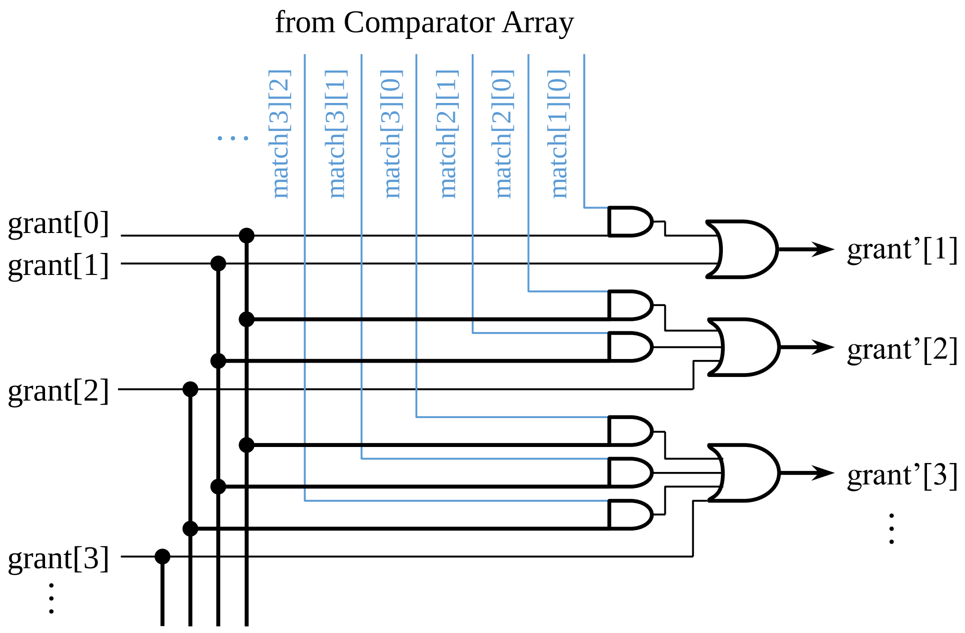


Figure 7.5: AND-OR Array for Request Aggregation.

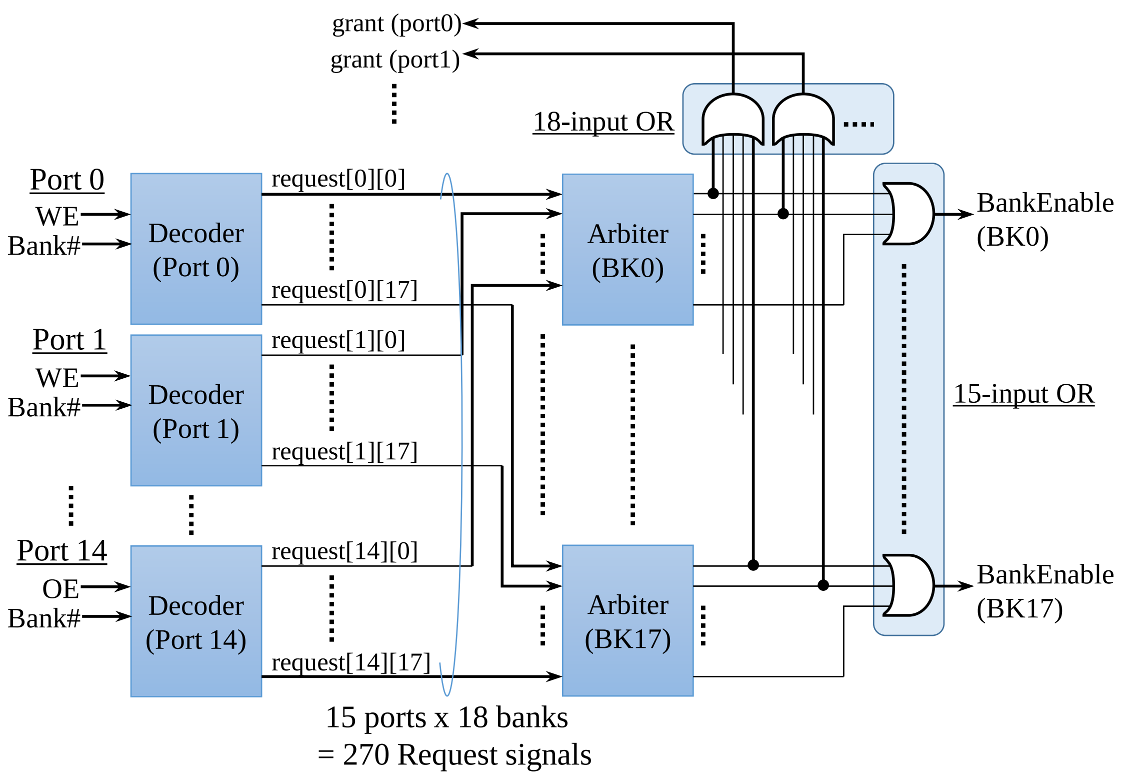


Figure 7.6: Decoder and Arbiter circuit for Multibanked Register File.

can be added between newly arriving and losing accesses with an IPC increase of 1% or less.

AND-OR Array

The comparator array is not enough to implement request aggregation. A request can be aggregated only if the request matches with the other request which has a *grant*. To implement this logic, the AND-OR array in Figure 7.5 is placed after the arbiters.

7.3 Decoder and Arbiter

Figure 7.6 shows the arbiter circuit which has 15 request ports (10-read+5-write) and 18-banked multibanked register file. Arbitration is proccessed for each bank. Therefore, the decoders are placed between the request inputs and the arbiters.

The each arbiter gives the grant signal of its bank, then the grant signals are gathered by multi-input OR circuit.

The number of the decoded request signals roughly determines the scale of the circuit. Therefore, the scale of the circuit is roughly proportional to the product of the number of requests and the number of banks. As shown in Section 9.2, the decoders and arbiters are the significant part of the control circuit of MStage.

7.4 Stall Probability

This section presents analytic solutions for the stall probabilities of existing and proposed techniques.

Unlike a multibanked main memory, the bank of each access is assumed to be randomly chosen, because the physical registers are randomly allocated to instructions after sufficient cycles have passed since initialization.

The author found two different approaches, a *queueing theory* approach and a *birthday problem* appraorch. The following two subsections explain each of the two different approaches.

Throughout this section, the number of banks is b, and the number of register accesses per cycle is n.

7.4.1 Queueing Theory Approach

The bank of a multibanked register file system can be modeled as a *waiting queue*, where the server is the bank itself, and the customers are accesses to the bank. This subsection presents analytic solutions for the stall probabilities based on queueing theory.

$M/D/1/1$ Queue

Each of the banks (not the whole system) of a plain multibanked system corresponds to the $M/D/1/1$ queue, as detailed as follows:

M The arrival process can be assumed to be Markovian because the bank accesses (customers) are randomized by register renaming (Section 5.3).

D A customer (access) is served by the server (bank) in a deterministic time of 1 cycle in a pipelined manner.

1 The number of servers (banks) is 1.

1 The capacity includes the places for a customer being served and those in the waiting room. That is, the $M/D/1/1$ queue has no waiting room.

Because of no waiting room, if two customers arrive at the $M/D/1/1$ queue in a single cycle, the lost customer leaves the queue. This corresponds to a bank conflict and the resulting pipeline stall.

$M/D/1/2$ **Queue** [48]

The $M/D/1/2$ queue, which has the waiting room for only one customer, largely reduces this leaving probability. Even if two customers arrive at the $M/D/1/2$ queue in a single cycle, the lost customer can wait in the waiting room and be served in the next cycle.

In this manner, a bank conflict for the $M/D/1/2$ queue does not necessarily result in a pipeline stall. The $M/D/1/2$ queue decreases the pipeline stall probability without decreasing the bank conflict probability. The bank conflict probability itself is contrarily increased in the $M/D/1/2$ queue.

Markov Chains

Figure 7.7 shows the Markov chains of the $M/D/1/1$ and $M/D/1/2$ queues. The circles represent the states corresponding to the number of customers in the system. The percentage numbers in the circles are the stationary probabilities of these states. The arrows labeled as "k (x_k%)" represent the state transition, where k is the number of arriving customers and x_k% is the transition probability.

For example, in the $M/D/1/2$ queue, if 2 customers arrive in the state 0 (i.e., 0 customers in the system), the first is served while the second waits in the waiting room; thus, the queue moves to the state 1 (1 customer). Then, if 1 customer arrives, the queue stays in the state 1, because the service time is 1 cycle.

Transition and Stationary Probabilities

For the ease of understanding the calculation, the probabilities are calculated assuming the number of banks $b = 10$ and the number of register accesses per cycle is fixed to $n = 3$.

In this case, the transition probability is given by the binomial distribution $x_k = {}_3C_k(1/10)^k(9/10)^{3-k}$. For example, $x_2 = {}_3C_2(1/10)^2(9/10)^1 = (3 \times 9)/10^3 = 2.7\%$,

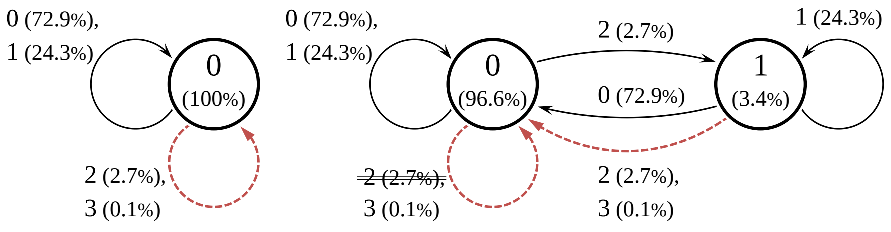


Figure 7.7: Markov chains of $\boldsymbol{M/D/1/1}$ and $\boldsymbol{M/D/1/2}$ queues.

resulting in the label “2 (2.7%)”.

The dashed arrows represent the state transitions with leaving customers, each results in a pipeline stall. The dashed arrows return to the state 0, which means stalled cycles are not counted in the stationary probabilities.

The stationary probabilities of the $M/D/1/2$ queues are calculated from the chain. Note that the stationary probability of the state 1 is as low as 3.4%, intuitively because the state 1 is hard to come (2.7%) and easy to go ($72.9 + 2.7 + 0.1 = 1 - 24.3 = 75.7\%$).

Pipeline Stall Probability

Compared with the $M/D/1/1$ queue, the $M/D/1/2$ queue decreases the pipeline stall probability per bank as follows:

State 0 If 2 accesses arrive, the $M/D/1/1$ queue lets one of them leave, while the $M/D/1/2$ queue does not and moves to the state 1. That is, the $M/D/1/2$ queue in the state 0 increases the least number of arriving accesses to cause a bank stall from 2 to 3. As a result, the stall probability in the state 0 is decreased from $2.7 + 0.1\%$ to 0.1%.

State 1 In the state 1, a stall occurs when 2 or more customers arrive as with the $M/D/1/1$ queue. However, the stall probability in the state 1 is multiplied by the stationary probability, and decreased from 2.8% to $2.8\% \times 3.4\% = 0.0952\%$.

In total, the stall probability pre bank p is decreased from 2.8% to $0.1 + 0.0952 = 0.1952\%$.

Then, the overall pipeline stall probability $P = (1 - p)^b$ is decreased from $1 - (1 - 0.028)^{10} \simeq 24.7\%$ to $1 - (1 - 0.001952)^{10} \simeq 1.93\%$.

A waiting room for only one customer has an equivalent effect to increasing the number of banks to b times. The simulation results in Chapter 8 show $P = 2.5\%$ when $b = 18$ for SPEC benchmark.

7.4.2 Birthday Problem Approach

This subsection presents analytic solutions for the stall probabilities based on birthday problem.

Multibanked Register File

The pipeline of a conventional multibanked register file with single-port cells is stalled on a bank conflict.

The stall probabilities of a conventional multibanked register file is equivalent to the *birthday problem* [49]. The probability that the bank of the second access is different to that of the first is $(b-1)/b$, and that the bank of the third access is different to both the first and second is $(b-2)/b$. In this way, the probability that all the n accesses are to different banks is $(b-1)/b \times (b-2)/b \times \cdots \times (b-n+1)/b$. Inversely, the stall probability, which is the probability that some of the n access are to the same bank, is as follows:

$$P_{\mathsf{Plain}}(b, n) = 1 - \frac{b-1}{b} \times \frac{b-2}{b} \times \cdots \times \frac{b-n+1}{b}. \tag{7.1}$$

When $n = 3$, the probability is:

$$P_{\mathsf{Plain}}(b, 3) = (3b-2)/b^2. \tag{7.2}$$

Multiported Multibanked Register File

The pipeline of the multiported system with 2-read/write cells is stalled when some banks have more than 2 accesses.

This is equivalent to an extended birthday problem that 3 or more people will have the same birthday. This probability is as follows [49]:

$$P_{\mathsf{mPort}}(b, n) = 1 - \sum_{i=0}^{\lfloor n/2 \rfloor} \frac{b!\, n!}{i!\,(n-2i)!\,(b-n+i)!\,2^i b^n}. \tag{7.3}$$

When $n = 3$, this equation is simplified as follows:

$$P_{\mathsf{mPort}}(b, 3) = 1/b^2. \tag{7.4}$$

Multistaged Multibanked Register File

The stall probability of the multistaged system is primarily the same as that of the multiported system because each bank of this system can *virtually* accept two accesses per cycle, by carrying over one of them to the next cycle. Conversely, the access carried over from the previous cycle increases the effective number of accesses in this cycle.

Rough Estimate

A carry-over occurs when a bank has 2 accesses in a single cycle, and this probability is the same as P_{Plain} (Equation 7.1). When $b = 16$ and $n = 3$, $P_{\mathsf{Plain}}(16, 3) \simeq 18.0\%$; thus, the effective number of accesses in the next cycles is increased from 3 to 3.18 on average. Then, the stall probability is obtained by assigning this increased number of accesses to the multiported stall probability P_{mPort} (7.3).

However, this discussion only holds true for the first cycles after the initial state. To be more precise, the multistaged stall probability is calculated as follows.

Number of Carry-Over

The situation can be divided based on m, the maximum number of accesses per bank:

1. When $m = 1$, the pipeline is never stalled.
2. When $m = 2$, the pipeline is conditionally stalled.
3. When $m \geq 3$, the pipeline is always stalled.

When $m \geq 3$, carrying over 2 (or more) accesses per bank is meaningless. Even if 2 accesses are carried over, one of the two will win but the other will also lose in the second stage. Consequently, the system can limit the number of carry over per bank to one. This is the exact reason why the pipeline is always stalled in item 3.

The single access carried over necessarily wins in the second stage because it is given higher priority. In Figure 7.1, the *lower* access of I_3 in the RR2 stage has the higher priority than the accesses of I_4 in the RR1 stage, and hence never loses to the latter.

Multistaged-Specific Stall

In items 1 and 3, the multistaged system behaves the same as the multiported system. In contrast in item 2, the pipeline of the multiported system is never stalled, while that of the multistaged system is conditionally stalled. Thus, stalls in item 2 are **multistaged-specific**.

In Figure 7.1, the bank has 3 accesses in cycle c_5, as denoted with a dashed box in the figure, causing a pipeline stall in cycle c_8. The situation in cycle c_5 is equivalent to item 3, where a bank has 3 (or more) accesses for the RR1 stage. Consequently, *the multistaged pipeline is stalled when a bank has a total of 3 (or more) accesses in a cycle whether they are in the RR1 or RR2 stages.*

In this figure, instructions I_3, I_4, and I_5 perform 2, 1, and 2 accesses, respectively. As a result, a carry-over is produced by the 2 accesses of I_3, then propagated by 1 access of I_4, and finally results in a stall by 2 accesses of I_5. These generation and propagation look like G and P functions in a carry-lookahead adder. Eventually, *a multistaged-specific stall occurs when a bank has* 2, (1,)$*$ 2 *accesses in consecutive cycles*, where "(1,)$*$" denotes zero or more repetition of "1,".

Stall Probability

The multistaged-specific stall probability is given by the sum of geometric series of the probabilities that a bank has 2, (1,)$*$ 2 accesses in consecutive cycles. Thus, it can be approximated by the first term, which is the probability that a bank has 2 and 2 accesses in two consecutive cycles.

The probability that a bank has 2 accesses in a single cycle is the same as P_{Plain}, the probability that the same bank has 2 accesses again in the next cycle is P_{Plain}/b. Thus, the multistaged-specific probability is $P_{\mathsf{Plain}} \times P_{\mathsf{Plain}}/b = (P_{\mathsf{Plain}})^2/b$.

Finally, the stall probability is given by the sum of the multiported stall probability (Equation 7.3) and the multistaged-specific probability, and is as follows:

$$P_{\mathsf{MStage}}(b, n) = P_{\mathsf{mPort}}(b, n) + P_{\mathsf{Plain}}(b, n)^2/b. \tag{7.5}$$

When $n = 3$, this probability is:

$$P_{\mathsf{MStage}}(b, 3) = 1/b^2 + (3b - 2)^2/b^5. \tag{7.6}$$

7.4.3 Comparison

Figure 7.8 shows the stall probabilities calculated by the two different approaches. In the graph, the x-axis is the number of banks. The graph shows the stall probabilities calculated by Section 7.4.1 and 7.4.2. The $x-$ and y-axis are log-scaledd. Four curves are plotted for Plain and MStage by the two different approaches, where the number of access are $n = 3$. The results of the two different approaches are quite consistent each other.

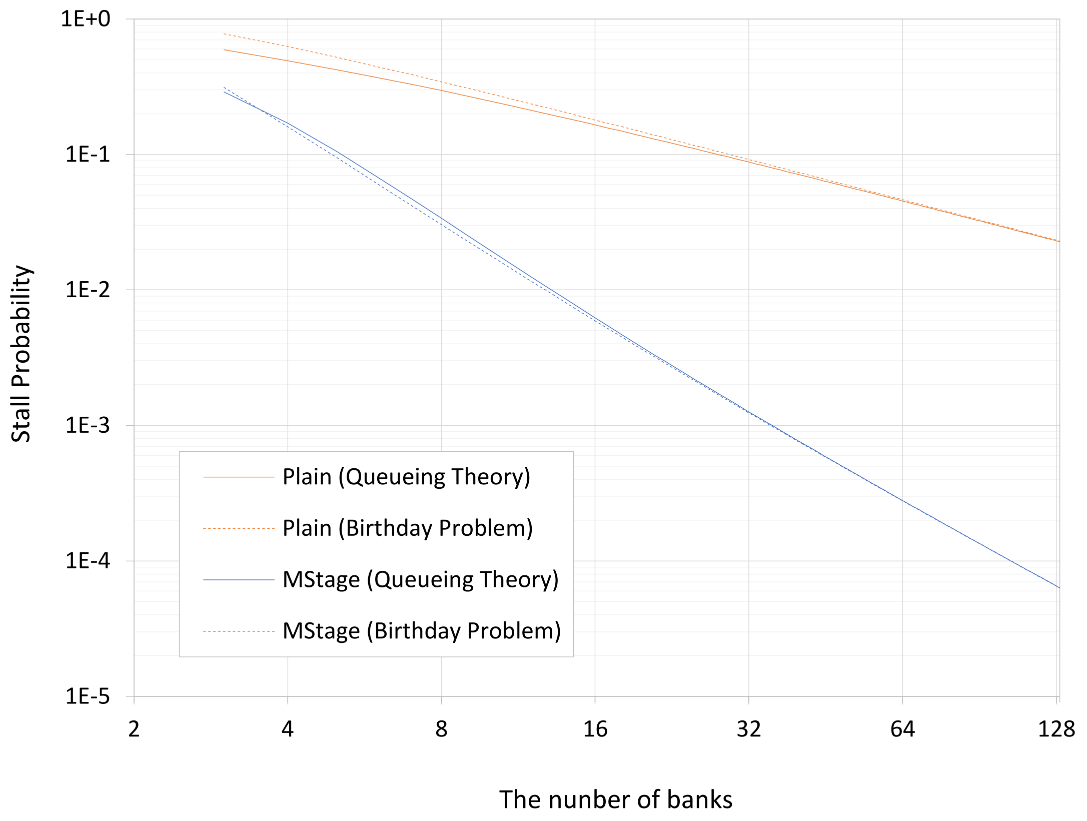


Figure 7.8: Stall Probabilities of Plain and MStage by two different approaches vs. The Number of Banks.

Chapter 8

Evaluation of IPC

This chapter shows the evaluation results of IPC. The author evaluated both of the conventional models (Multiported register file, NORCS, plain multibanked register file), and the proposed multibanked models (BAIS and MStage). Section 8.3 shows the detail of evaluated models. The author used SPEC CPU 2006 benchmark [3] and Onikiri 2 [31] simulator as shown in Section 8.1.

First, the author discusses the number of operands per cycle for the benchmark programs in Section 8.2. As shown in the Section 8.4 section, IPC of the most models including proposed MStage is primarily determined by the number of register file accesses per cycle of a program. For the same number of operands per cycle, the stall probability of MStage is smaller than that of Plain, because of the second stage. The evaluation results show that, MStage with 18 banks maintain a relative IPC of 97.3%.

The author also evaluated BAIS. BAIS with 24 banks maintain a relative IPC of 97.2%. MStage needs only 18 banks, while BAIS needs 24 banks to maintain a relative IPC more than 0.97. Therefore, MStage is more efficient in the area and energy consumption. Next Chapter 9 compares the area and energy consumption of MStage and BAIS in detail.

Lastly, Section 8.5 shows the effect of request aggregation and operand bypass. The results shows that both request aggregation and operand bypass were essential parts of the efficient multibanked register file.

8.1 Evaluation Methodology

Benchmark

The author used all 29 programs of the SPEC CPU 2006 benchmark with the *ref* data sets [3]. The programs were compiled with gcc 4.2.2 −O3. The author evaluated the 1G instructions after the first 10G instructions.

Simulator

The author used the Onikiri 2 [31] simulator, which was also used to evaluate NORCS [2]. This simulator is fully cycle accurate, that is, it reproduces the behavior of instructions in each stage in the correct cycles. The simulator executes instructions in the correct execute stages, and verifies the results with those of an on-line emulator in the commit stage. Thus, the behavior on mispredictions is also accurately reproduced. The simulator also reproduces the fact that register renaming actually randomizes the accessed registers (Section 5.3).

8.2 Number of Operands

First, the author counted the number of operands of each program to better understand the relationship between the numbers of operands and register file ports.

Some source operands are provided by the bypass or by the *request aggregation* without consuming the register file ports (Section 7.2). The other source operands consume the read ports, and become the cause of bank conflicts. Table 8.1 classifies these types of accesses[1].

Figure 8.1 shows the numbers of ports used by these types of accesses per cycle for all 29 programs and the average of them. Some programs such as h264ref and bzip2 consume approximately 2 read and 2 write ports per cycle; however, this does not mean that a 4-port register file is sufficient.

Figure 8.2 shows the cumulative distribution of port usage for h264ref, which is the program with the largest number of operands. In this graph, R(—) + W at the point of 4 ports is 52.4%, which means that only 52.4% of the execution cycles are covered by a 4-port register file. Conversely, this curve reaches 97.8% at the point

[1]Some technique such as delayed allocation excludes write accesses; however, the author did not show the evaluation result because the overhead is too large to compare with the others, as mentioned in Section 5.4.

Table 8.1: Classification of Accesses based on Exclusion.

Symbol	Read/Write	Excluded from the cause of bank conflict by
🟦 R(BP)	Read	Bypass
🟧 R(RA)		Request Aggregation
🟦 R(—)		(cannot be excluded)
🟥 W	Write	(can be excluded by Delayed Allocation, etc.)

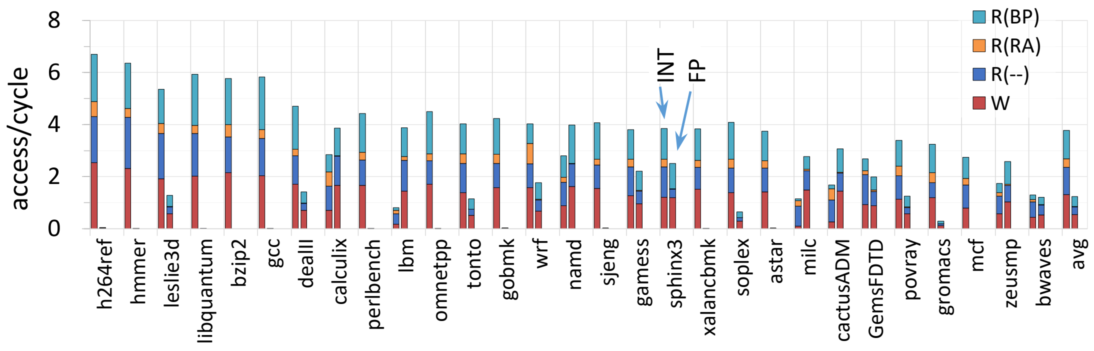


Figure 8.1: The number of Operands per Cycle for All 29 Programs

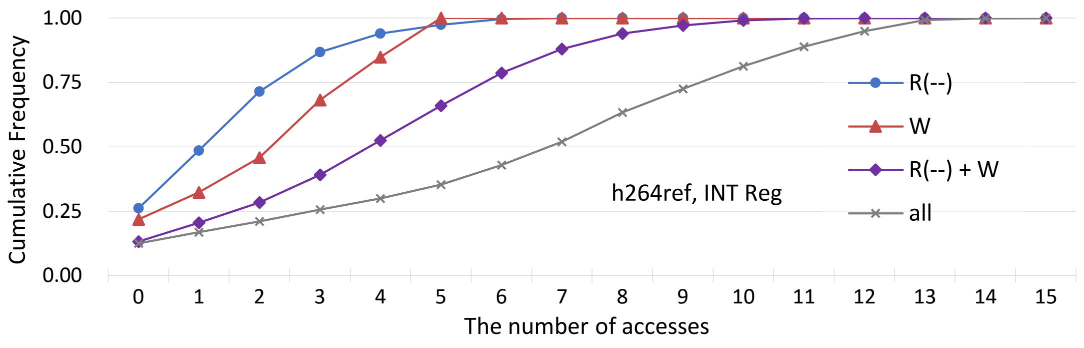


Figure 8.2: Cumulative Distribution of Accesses for h264ref.

of 9 ports, which means that it is not practically difficult to reduce the number of ports to 9 or more.

8.3 Evaluated Models

Table 8.2 shows the evaluated models of their default configurations. The author choses as the default the minimum configurations with which BAIS, MStage, NORCS and RPort show average relative IPC of more than 0.97.

Baseline Model

The baseline core has a full-port register file composed of a replicated pair of RAMs. This replication is widely used in recent cores such as the Bulldozer core in Chapter 1 [5, 13, 16–19].

Table 8.3 and Figure 8.3 give its configuration, which follows modern 8-issue cores such as the IBM POWER7/8, and Intel Haswell and Skylake processors [5–8].

Plain Multi-Bank Models

The author evaluated Plain model, which is the plain multibanked register file simply stalled on a bank conflict. For fair comparison with MStage, the author also evaluted PlainRA which is a Plain with request aggregation.

Reduced Port Model

As described in Section 8.2, it is not practically difficult to reduce the number of ports from 15 to 9 or less. In fact, commercial processors adopt ad-hoc methods to reduce the number of ports; however, they are difficult to generalize.

Table 8.2: Evaluated Models and Their Default Configurations

Name	Technique	ports	RF read stages			remarks
(baseline)	Replicated Pair	5/4r+5w (int/fp)	3		RF:3	
🟧 Plain	Plain Multibank	1rw (bank)	2	a/r:1	RF:1	w/o Request Aggregation
🟨 PlainRA						with Request Aggregation
🟦 BAIS	Bank-Aware Instruction Scheduler					w/o Request Aggregation
🟦 MStage	Multistage		3		RF:2	with Request Aggregation
🟥 RPort	Reduced Port	7rw (RF)			RF:2	
🟩 NORCS	NORCS	3r+3w (MRF)	2	RC:1	MRF:1	RC: 12 ent.

RF: reg. file, RC: reg. cache, MRF: main RF, a/r: arbitration/reg. number routing

Table 8.3: Configuration of Baseline Core

ISA	Alpha w/ byte-word ext.
width	fetch, issue, commit: 8
inst. window	64 (unified)
reorder buffer	256 entries
registers	int:180, fp:180
exec. units	int:3, fp:3, mem:2
pipeline stages	fetch to dispatch:9, schedule: 1, issue:2, register read: 3
branch pred.	16K:gshare + 8K:local
miss penalty	16 cycles
BTB	2K-entries, 4-ways
L1C	64KB, 8-way, 64B/line, 2 cycles
L2C	512KB, 8-way, 64B/line, 8 cycles
L3C	8MB, 8-way, 64B/line, 24 cycles
main memory	200 cycles

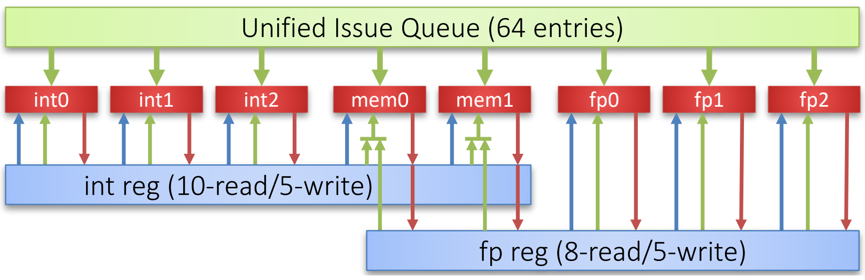


Figure 8.3: Execution units and Register Files of Baseline Core

Thus, the author evaluated the RPort model which has a b-port register file ($b < 15$) for reference. From the PlainRA model (Figure 5.1 (lower)), the b banks of 1-read/write RAM is replaced with one b-read/write RAM. The model is free from bank conflicts; instead, the backend pipeline is stalled when more than b operands are accessed in a single cycle. For fair comparison with MStage, the author applied the request aggregation (Section 7.2) to this model.

Register Latencies

As described in Section 5.1, the author assumed that the arbitration and register number routing of the multibanked models take one cycle, which is denoted as "a/r: 1" in Table 8.2.

Unfortunately, the register file latency is not documented for recent cores [50]. The author assumed that the latency of the baseline core is 3 cycles, and that of some models is reduced to 2 or 1, as shown in Table 8.2.

However, this difference of one cycle has an insignificant effect on the IPC of recent cores with highly accurate predictors. In this evaluation, the average IPC decreased by 1.4% because of this one-cycle increase.

These latencies are partially verified in Section 9.4.

8.4 Relative IPC and Stalled Cycles

Relative IPC

The graphs in Figure 8.4 show the averaged relative IPC of the models with different configurations averaged for the 29 programs in SPEC CPU 2006. In this graph, four bars are shown for multibanked models with different numbers of banks, for RPort with different numbers of ports, and for NORCS with different size of register cache. Regarding NORCS, the author evaluated many other configurations, e.g., a main register file with fewer write ports, and selected these four as representatives so that they can prove that the default configuration is the best.

The author evaluated the number of banks in multiples of 6 based on the layout constraint derived in Section 9.2. As described in Section 5.3, the number of banks not being a power of 2 does not have a significant impact.

While Plain and PlainRA cannot achieve sufficient IPC even with 30 banks, BAIS achieves a relative IPC as high as 97.2% with 24 banks, and MStage achieves a relative IPC of as high as 97.3% with 18 banks.

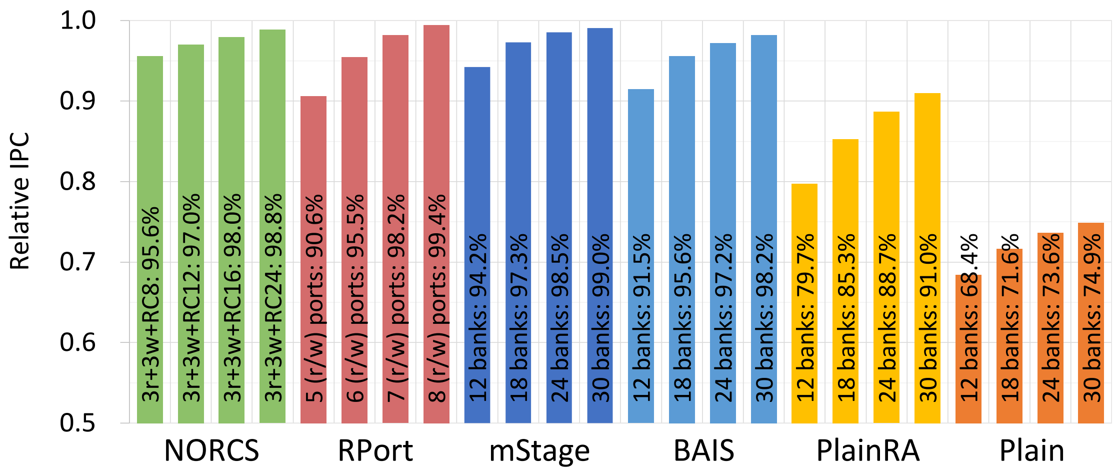


Figure 8.4: Averaged Relative IPC of Models with Different Configurations.

Figure 8.5 shows the relative IPC of the models with the default configurations shown in Table 8.2 for all the 29 programs in SPEC CPU 2006. The programs are arranged in descending order of the number of R(—) + W accesses shown as the curve in this graph, which is that to the integer register file or that to the floating-point register file, whichever is greater (Figure 8.1).

The author choses the default configurations so that MStage, BAIS, RPort, and NORCS show average relative IPC of more than 0.97. However,most of them show the relative IPC of as low as 0.9 for programs with a large number of register file accesses such as h264ref.

Relative IPC and Bank Conflicts

The first graph in Figure 8.6 shows the number of stalls per cycles caused by bank conflicts, the pipeline disturbance probability caused by bank conflicts, for 29 all programs and the average of all the 29 programs. The second graph is extracted from Figure 8.5 to compare with the first.

From PlainRA, MStage reduces this disturbance probability from 0.124 to 0.025, which is the actual effect of the $M/D/1/2$ queue.

Table 8.4 shows the correlation coefficent between (R(—)+W) and $(1-RelativeIPC)$. NORCS, RPort, Plain, PlainRA and MStage models have strong correlation between them, while BAIS does not.

As a whole, R(—)+W accesses per cycle, stalled cycles caused by bank conflicts,

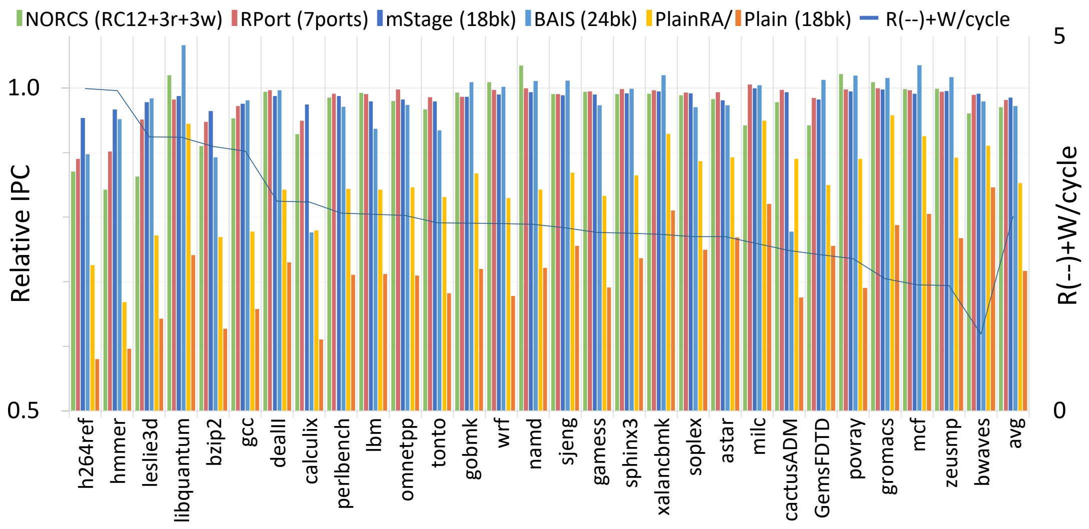


Figure 8.5: Relative IPC of Models with Default Configurations for All 29 Programs and Average of them.

and the IPC show strong correlation for NORCS, RPort, Plain, PlainRA and MStage models. It is safe to say that the IPC of these models is primarily determined by the number of register file accesses per cycle of a program.

8.5 Request Aggregation

Figure 8.7 shows the relative IPC of MStage with and without the bypass and request aggregation. In this graph, three bars are shown for each of the programs, and the differences between the first and second, and the second and third, show improvement by the bypass and by request aggregation, respectively.

As shown in this graph, these two techniques have the same level of impact on IPC. However, as shown in Figure 8.1, the accesses excluded by the request

Table 8.4: Correlation Coefficient between (R(—) + W) and (1 - Relative IPC).

NORCS	RPort	MStage	BAIS	Plain	PlainRA
0.64	0.79	0.78	0.21	0.75	0.78

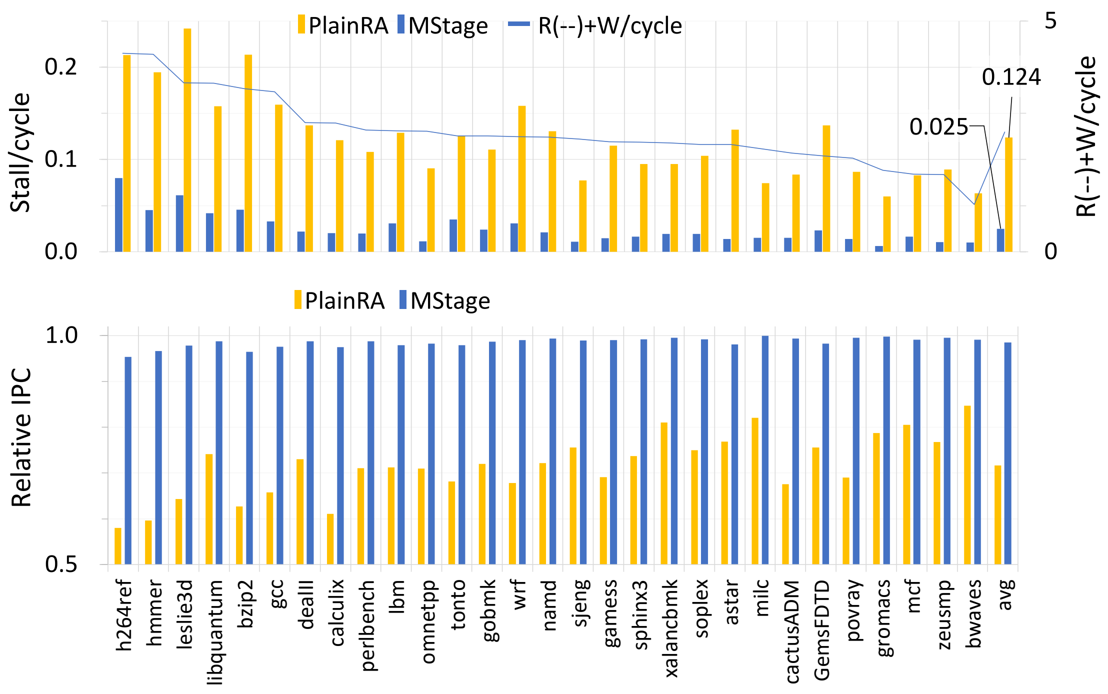


Figure 8.6: Stalled Cycles (upper) and Relative IPC (lower) for 29 All Programs and Average of them.

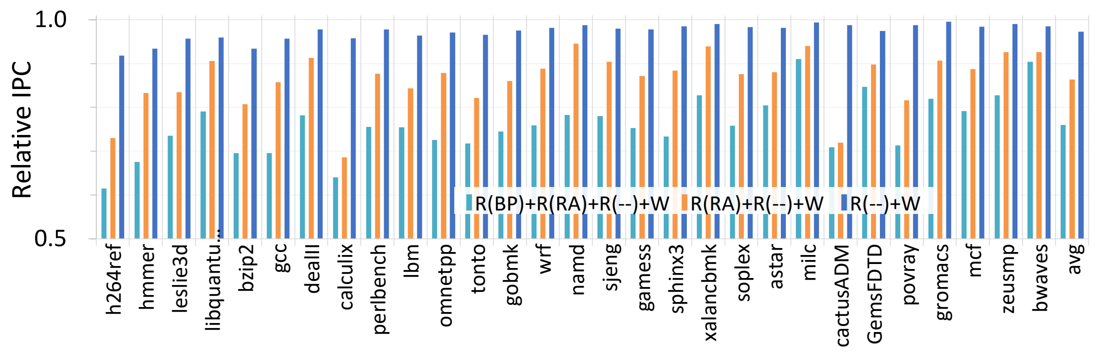


Figure 8.7: Relative IPC of MStage w/ and w/o the Bypass and Request Aggregation for 29 All Programs and Average of them.

aggregation are quite few. This phenomenon occurs because the bypass and request aggregation has an *indirect* and a *direct* effect, respectively, as described below.

Even if the accesses that can be excluded by the bypass are not excluded, they do not necessarily cause bank conflicts. They increase the number of accesses; and then, stochastically increase the stalled cycles.

On the contrary, if the accesses that can be excluded by request aggregation are not excluded, they increase the stalled cycles with probability 1.

Thus, the author can conclude that request aggregation is as important as the bypass for multibanked register files.

Chapter 9

Evaluation of Area and Energy Consumption

This chapter shows the evaluation results of the area and energy consumption of conventional and proposed multibanked models.The author evaluated the same models shown in the last chapter (Section 8.3).

The author described the entire system of the models in System Verilog. Then, the author performed a logic synthesis with the open source cell library of FreePDK-15. The memory cells and switches were modeled by CACTI, because FreePDK15 does not include RAMs or switches. Section 9.1 shows the detailed evaluation environment.

As shown in Chapter 4, the register file and the execution units must be aligned with each other. The author aligned the banks and data switches with the pitch of execution units. Section 9.2 shows the detailed layout.

The evaluation results show that, compared with NORCS [2], which is the latest research on a register file for area and energy efficiency, BAIS with 24 banks achieves a 23.6% and 61.8%, and MStage with 18 banks achieves a 40.6% and 68.9% reduction in circuit area and in energy consumption, while maintaining a relative IPC more than 0.97 as shown in the last Chapter 8.

9.1 Evaluation Methodology

FreePDK15 and CACTI

The author used FreePDK15, a predictive process design kit for 15nm FinFET technology [51], and NanGate FreePDK15 Open Cell Library [52].

Because this library does not include RAMs or switches, the author used CACTI [11, 28, 29] to evaluate them. CACTI is a design space exploration tool for usual instruction/data caches. CACTI calculates the RAM area from the numbers of vertical and horizontal wires, and the RAM energy from the capacitance of the transistors and wires charged and discharged in read and write operation.

CACTI is not designed to estimate switches. However, the circuit size of the switches can be estimated via a 64-bit r-read+w-write memory with only 1-entry. This 1-entry memory works as a 64-bit any-to-any switch by writing a 64-bit word to any of the w write ports, and reading it from any of the r read ports.

CACTI and FreePDK15 do not have the same semiconductor processes. Therefore, the results of them were converted with the minimum pitch of wires.

RAM and Switch Cells

Table 9.1 shows the areas of RAM cells calculated in this manner. Because the areas of small cells strongly depend on the designers' efforts, the author investigated recent researches on small-port memory cells [53, 54], and verified that the values are quite consistent.

Table 9.1: Estimated Cell Areas (μm^2).

Cell	area	relative	used for
1-read/write	0.096	1	RFB (Plain/MStage)
3-read+3-write	0.714	7.46	RF (NORCS)
7-read/write	1.169	12.22	RF (RPort)
3-read+5-write	1.219	12.74	WB (NORCS)
5-read+5-write×2	3.320	34.70	RF (base), RC (NORCS)
10-read+5-write	3.049	31.97	RF

RF: register file, RFB: RF bank, RC: register cache, WB: write buffer

Logic Synthesis and Place-and-Route

The author described the entire systems of Plain and MStage in System Verilog.

Then, the author synthesized, and placed-and-routed the description with Cadence Encounter v10.13 including RTL Compiler v10.10 with the standard cells in the FreePDK15 library. In the place-and-route, the RAMs and switches are treated as large cells which have parameters estimated with CACTI.

Regarding the baseline, NORCS, and RPort models, only RAMs and switches estimated with CACTI are simply added without consideration for layout constraint.

Baseline Model — Replicated Register Files

The area and energy consumption were normalized by a replicated pair of 5-read+5-write register file (int) and 4-read+5-write register file (fp) register files, because this replication is frequently used in recent cores such as the Bulldozer core mentioned in Chapter 1 [5, 13, 16–19].

Although this replication has a positive effect on the latency, but a slight negative effect on the area and energy consumption. As shown in Table 9.1, a pair of 5-read+5-write (10-port) cells is slightly larger than a 10-read+5-write (15-port) cell because of the overheads of the power lines.

As for the energy to read/write the register file, the former (10-port) is reduced to approximately 2/3 of that of the latter (15-port), because the lengths of the bit- and word-lines are reduced to $10/15 = 2/3$.

In contrast, the number of accesses is increased by the replication. As shown in Figure 8.1 (lower), the number of read accesses is equal to or less than that of write mainly because the bypass excludes approximately a half of read accesses. The replication doubles the write accesses. Thus, the total number of accesses is increased to more than $1/2 + 1/2 \times 2 = 3/2$ times.

Finally, the energy consumption is increased to more than $2/3 \times 3/2 = 1$ times depending on the ratio of the numbers of read and write accesses. The evaluation results show the energy consumption of the former is 1.01 times larger than those of the latter.

9.2 Layout

Figure 9.1 shows the place-and-route results of the BAIS, MStage, PlainRA and Plain integer register files. This figure also shows the shapes of the datapaths of

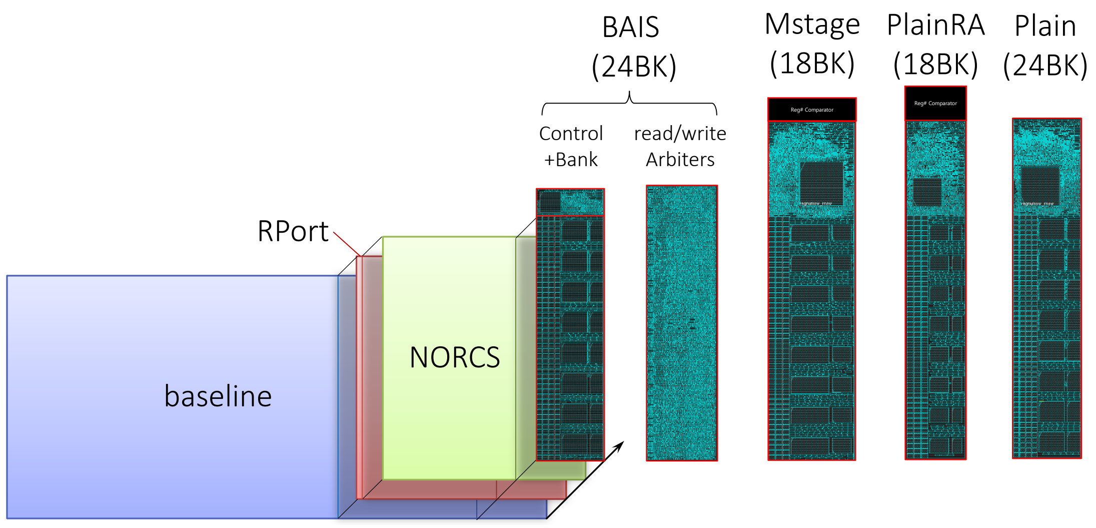


Figure 9.1: Layout of Integer Register Files of Each Model.

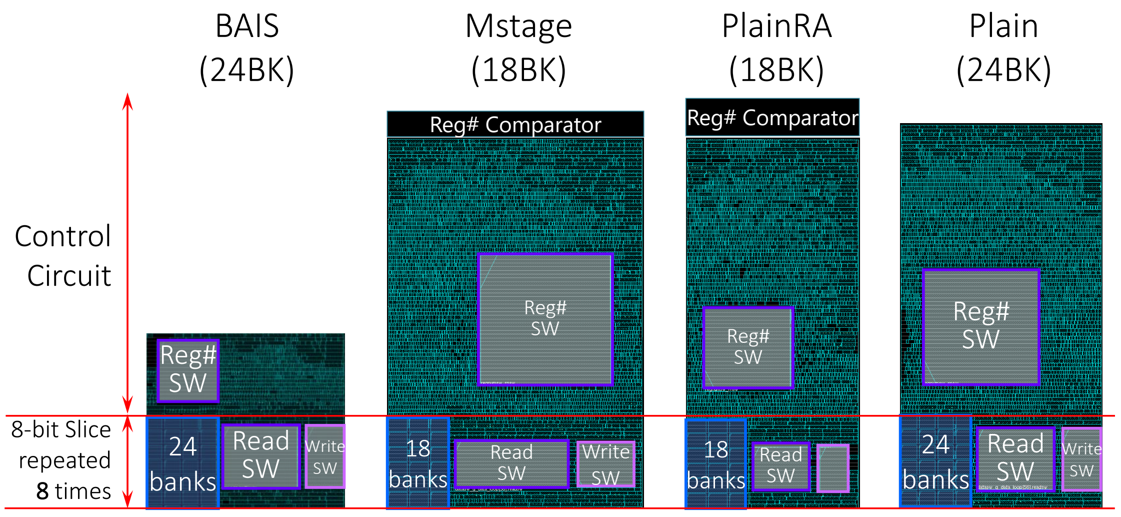


Figure 9.2: Enlarged View of the Bank Control Circuits of Each Model.

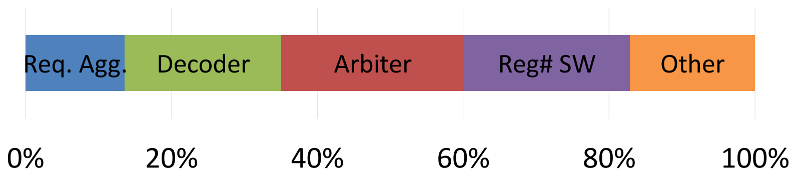


Figure 9.3: Breakdown of the Area of the Control circuit.

the baseline, NORCS and RPort integer register files for reference. Figure 9.2 shows the enlarged view of bank control circuits.

8-bit Slice

Because each of the banks requires a decoder and a buffer, the author adopted an 8-bit-slice design for the multibanked models to reduce the overhead to 1/8.

The width (the vertical direction in these figures) of the 8-bit slice is determined by the width of the operand buses between the execution units and the register file systems. These buses are $\{2\,(\text{read}) + 1\,(\text{write})\} \times 5\,(\text{unit}) = 15$ tracks of 8-bit wire bundles. These wires are routed in an upper layer, whose pitch is twice as thick as that in the lower layers used for RAMs [12] and switches. Thus, $15 \times 2 = 30$ tracks can be used within this width for the RAMs and switches. The width of the shapes of the datapaths of the three models in Figures 9.1 and 9.2 are also determined in this manner.

Banks

In this 30-track width, 6 register file banks are arranged. This is the reason why the number of banks of the multibanked models is the multiple of 6. The author cannot freely adjust the width and height of the RAM cell because they are almost completely determined by the number of bit- and word-lines.

Switches

The read and write switches are arranged so as to minimize the crossing wires. The heights (the horizontal direction) of the switches are determined by the number of routing control lines which run vertically through the eight 8-bit slices. Thus, the read and write switches cannot overlap with each other in the horizontal direction.

The height of the layout is thus determined by the sum of the heights of banks, and a read and a write switch.

Control

Figure 9.3 shows the breakdown of the area of the control circuit of MStage. The arbiters and the array of comparators for request aggregation occupies 25.0% and 13.6% of the control circuit, respectively. The significant part of the rest is decoder circuits to generate the request signals for the arbiters. As shown in Section 7.3, the number of decoder circuits is equal to the number of ports (15 ports in the desgin), and the number of arbiters is equal to the number of banks (18 banks in the design).

There remains much room for optimization. For example, the arbiters will be considerably minimized if implemented with dynamic logic.

In contrast, The bank control circuit of BAIS shown in Figure 9.2 are mostly pipeline latches, switches and bank decoders. However, BAIS also needs read/ write aribters in the instruction scheduler. Therefore, total area of control circuits including the arbiters in the instruction scheduler is larger than that of MStage.

9.3 Area

Figure 9.4 shows the relative area and energy consumption of the integer and floating-point register files of each model. The Plain and MStage areas include dead spaces produced by layout constraint. The energy is calculated using the access count produced by the simulation in Chapter 8.

The areas of the multibanked models are considerably smaller than those of the other models with the default configurations. As the register file bank areas are reduced, those of the switches and control logic become relatively large. In particular, the switch areas increase with the square of the number of banks. The area of MStage with 18 banks is approximately 8.3 times as large as that of the ideal register file.

As the number of registers increases, the register file areas become dominant. Thus, MStage with 1-read/write cells is more advantageous in heavily-multithread cores with several times more registers.

9.4 Latency

Because CACTI does not evaluate small RAMs, and FreePDK15 has not yet provided the HSPICE model, the author measured the length of the access paths to verify the assumption in Table 8.2. The author assumed that the latency to read one of the register file banks through the read switch is a third of that to read the baseline register file. As the area is reduced, the total length of the access paths of the former is reduced to 31.2% from that of the latter. Thus, it is safe to say that the latency assumption in Table 8.2 is not advantageous for multibanked models.

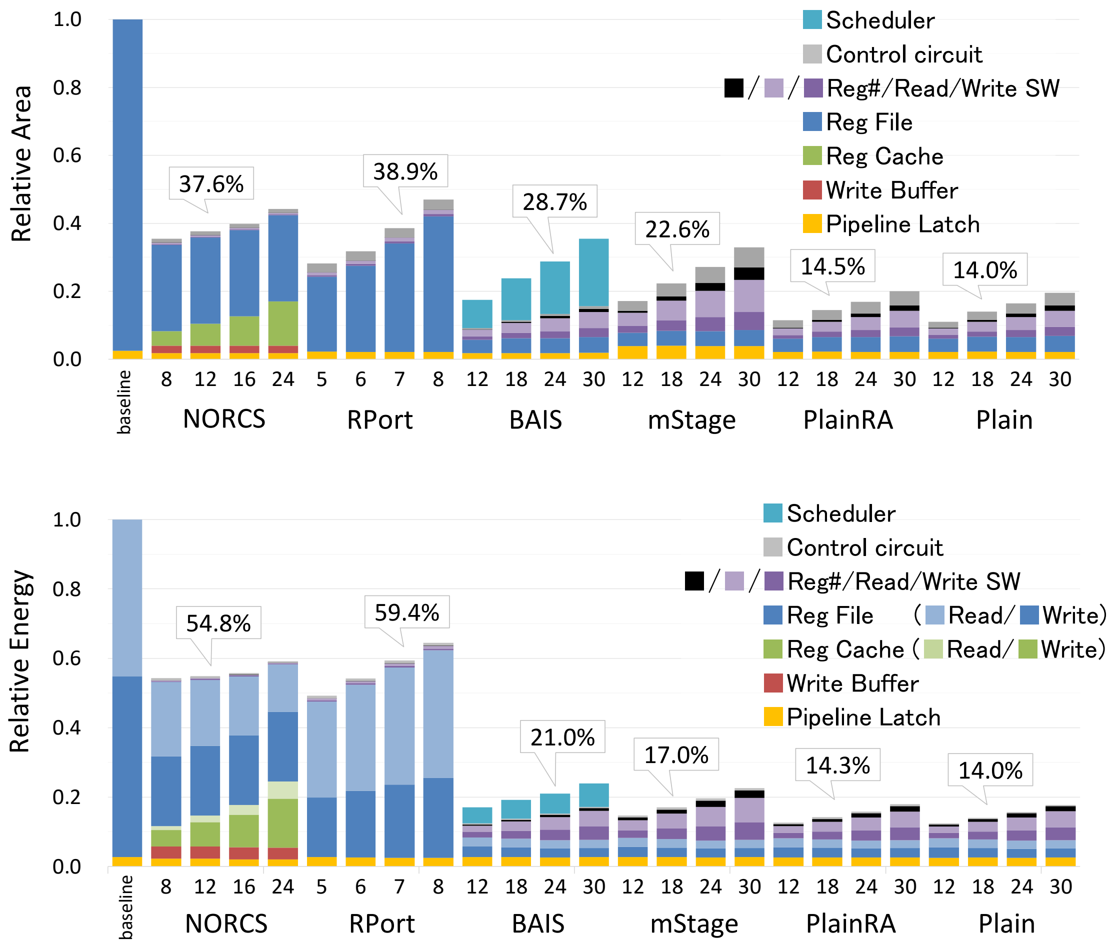


Figure 9.4: Relative Area (upper) and Relative Energy Consumption (lower).

9.5 Energy Consumption

As shown in Figure 9.4, the result of energy consumption is basically consistent with that of the area, except that the energy of the register file banks is reduced in inverse proportion to the number of banks; because only accessed banks consume dynamic energy. On the contrary, the energy of switches increases with the square of the number of banks.

9.6 Area and Energy Efficiency

Figures 9.5 and 9.6 show the relative IPC with respect to the relative area and energy consumption. The graphs are simply derived from the graphs in Figures 8.4 and 9.4 to show the trade-off between IPC and area, and between IPC and energy consumption. The curves for multibanked models plotted for the number of banks: 12, 18, 24, 30 and 36. The curve for RPort is plotted for the number of ports: 5 to 8. The curve for NORCS is plotted for the size of register cache: 8, 12, 16 and 24.

For technique to reduce area and energy while keeping IPC, it is important to plot one point within the region close to the top of the graphs as close to the y-axis as possible.

In each of the graphs, the points for MStage, BAIS, NORCS, and RPort with their default configurations (denoted by circles) are located within the region where the average relative IPC is more than 0.97, from left to right in this order, which proves that MStage reduces more area and energy than BAIS, NORCS and RPort while keeping the same level of IPC. Compared with NORCS (with a 12-entry register cache and a 3-read+3-write main register file), MStage (with 18 banks) achieves a 40.6% and 68.9% reduction in area and energy consumption, respectively.

Comparison with DVFS

For reference, the dashed curve in the right graph is plotted for DVFS assuming that the percentage of the register file to the whole core in energy consumption is 25% [14].

MStage, BAIS, RPort, and NORCS outperform DVFS. However, it should be noted that these techniques are not contradicting to DVFS, that is, a core that adopts these techniques can also utilize DVFS, and they reduces energy consumption and heat when the core is operating at the highest voltage. Additionally, as

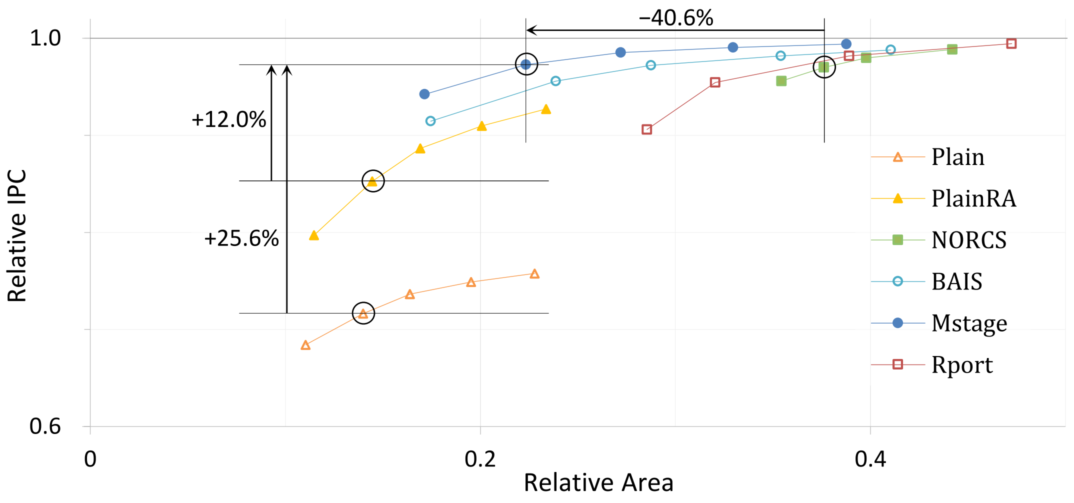


Figure 9.5: Relative IPC vs. Relative Area.

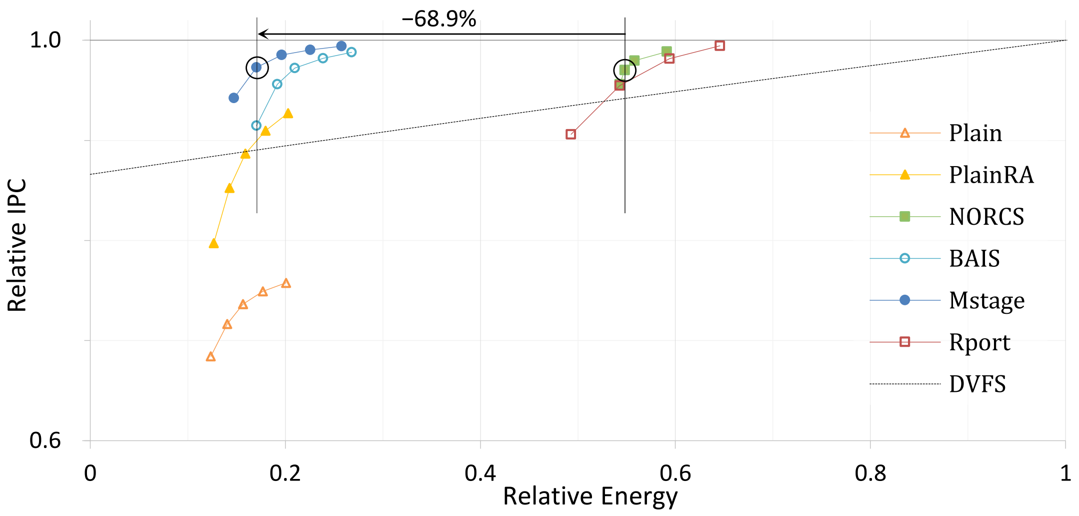


Figure 9.6: Relative IPC vs. Relative Energy Consumption.

mentioned in Chapter 1, downscaling the voltage, in particular of the register file, is becoming more difficult in recent technologies [15].

Chapter 10

Conclusion

10.1 Summary

The region that includes the register file is one of the hot spots in high-performance cores that limits the clock frequency. To reduce the area and energy consumption of the register file, this thesis mainly focused on the two techniques **register cache system** and **multibanked register file**. In this thesis, the author showed the practical design of register cache system, and proposed the two architectural techniques for multibanked register files, to reduce the possibility of the pipeline disturbances caused by bank conflicts.

For **register cache system**, the author designed NORCS with FreePDK45, an open source process design kit for 45 nm technology. NORCS is the latest research on a register file for area and energy efficiency [2]. Researchers in NVIDIA adopted this idea for their GPUs [21, 22]. Although the advantages of NORCS in the microarchitecture are accepted, no LSI design of NORCS has been published.

In this study, the author performed manual layout of the memory cells and arrays of NORCS, for detailed evaluation from the viewpoint of LSI design. The author also performed SPICE simulations with RC parasitics to precisely estimate the latency of the register cache system. The results with FreePDK45 were consistent with that of the original article.

For **multibanked register file**, the author proposed the two architectural techniques. **Multibanking** is the ultimate way to reduce the register file ports. Although multibanking achieves the minimum number of ports (i.e., 1), pipeline stall caused by bank conflicts can considerably degrade the IPC. To reduce the bank conflict probability of multibanked register files, this thesis showed the two microarchitectural techniques; one is **Bank-Aware Instruction Scheduler** (BAIS), and the other is **Skewed Multistaged Multibanked Register File** (MStage).

BAIS schedules the instructions so that no bank conflict occurs in the stages to read/write the register file. Prior studies briefly mentioned the possibility of bank-aware scheduling, or rejected it because of the increased latency [23,30]. The author showed an implemantation of BAIS and clarified that the latency of the logic was not practically increased.

The author also proposed MStage, which is a totaly new microachitecture. MStage has two stages to read the bank of the multibanked register file, and an instruction that missed the bank because of a bank conflict still has a second chance to read the same bank in the second stage. As a result, MStage drastically reduces the pipeline stall caused by bank conflicts. This thesis also showed the analytic solutions for the pipeline stall probabilities of several multibanked register files.

The evaluation results showed that, compared with NORCS, BAIS with 24 banks achieves a 23.6% and 61.8%, and MStage with 18 banks achieves a 40.6% and 68.9% reduction in circuit area and in energy consumption, while maintaining a relative IPC of 97.2% and 97.3%, respectively.

In summary, NORCS, BAIS, and MStage show higher efficiency in area and energy consumption in ascending order.

10.2 Future Direction

Application for Multithread Core

The areas of the switches and control logic become relatively large, because MStage drastically reduce the register file bank areas. Therefore, MStage is more advantageous in heavily-multithread cores with several times more registers.

Request Aggreagation

One of the advantage of MStage is its closedness. Unlike BAIS, almost no modification is needed for the other parts of the core. The control circuit of MStage sees only the access to the register file, is not aware of instructions. However, the request aggregation circuit has a possibility to be optimized if the control circuit was aware of instructions. The same entry accesses are at least partly caused by the instruction dependency, which is the plural source operands of instructions depend on the same one instruction.

VLSI design of MStage and BAIS

The author used FreePDK15 to evaluate area and energy consumption. However, FreePDK15 has the following limitations: Cell library is not sufficient, there is no RAM compilers, and HSPICE model has not released yet. Therefore, the other process design kit is needed to perform VLSI design of MStage and BAIS.

Application for the Other Part of Superscalar Processor

The proposed technique of MStage is also important for most architects because it is applicable to most of the other components of a superscalar processor core. This technique will greatly reduce the area and energy consumption of the entire superscalar processor core.

References

[1] Tim Fischer, Srikanth Arekapudi, Eric Busta, Carl Dietz, Michael Golden, Scott Hilker, Aaron Horiuchi, Kevin A. Hurd, Dave Johnson, Hugh McIntyre, Samuel Naffziger, James Vinh, Jonathan White, and Kathryn Wilcox, “Design solutions for the Bulldozer 32nm SOI 2-core processor module in an 8-core CPU,” in Proc. *IEEE International Solid-State Circuits Conference (ISSCC 2011), Digest of Technical Papers*, Feb. 2011, pp. 78–80.

[2] Ryota Shioya, Kazuo Horio, Masahiro Goshima, and Shuichi Sakai, “Register cache system not for latency reduction purpose,” in Proc. *the 43rd Annual International Symposium on Microarchitecture (MICRO)*, Dec. 2010, pp. 301–312.

[3] *SPEC CPU 2006*, The Standard Performance Evaluation Corporation. [Online]. Available: http://www.spec.org/cpu2006/

[4] John L. Hennessy and David A. Patterson, *Computer Architecture — A Quantitative Approach*, 5th ed. Morgan Kaufmann, Sep. 2011.

[5] B. Sinharoy, R. Kalla, W. J. Starke, H. Q. Le, R. Cargnoni, J. A. Van Norstrand, B. J. Ronchetti, J. Stuecheli, J. Leenstra, G. L. Guthrie, D. Q. Nguyen, B. Blaner, C. F. Marino, E. Retter, and P. Williams, “IBM POWER7 multicore server processor,” *IBM Journal of Research and Development*, vol. 55, no. 3, pp. 1:1–1:29, May 2011.

[6] B. Sinharoy, J. A. Van Norstrand, R. J. Eickemeyer, H. Q. Le, J. Leenstra, D. Q. Nguyen, B. Konigsburg, K. Ward, M. D. Brown, J. E. Moreira, D. Levitan, S. Tung, D. Hrusecky, J. W. Bishop, M. Gschwind, M. Boersma, M. Kroener, M. Kaltenbach, T. Karkhanis, and K. M. Fernsler, “IBM POWER8 processor core microarchitecture,” *IBM Journal of Research and Development*, vol. 59, no. 1, pp. 2:1–2:21, Jan. 2015.

[7] Kevin Krewell, "Intel's Haswell cuts core power," *Microprocessor Report*, Sep. 2012.

[8] Eyal Fayneh, Marcelo Yuffe, Ernest Knoll, Michael Zelikson, Muhammad Abozaed, Yair Talker, Ziv Shmuely, and Saher Abu Rahme, "14nm 6th-generation Core processor SoC with low power consumption and improved performance," in Proc. *IEEE International Solid-State Circuits Conference (ISSCC 2016), Digest of Technical Papers*, Feb. 2016, pp. 72–73.

[9] Neil H. E. Weste and David Money Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, 4th ed. Addison Wesley, 2011.

[10] Scott Rixner, William J. Dally, Brucek Khailany, Peter Mattson, Ujval J. Kapasi, and John D. Owens, "Register organization for media processing," in Proc. *International Symposium on High-Performance Computer Architecture (HPCA)*, 2000, pp. 375–386.

[11] Shyamkumar Thoziyoor, Naveen Muralimanohar, Jung Ho Ahn, and Norman P. Jouppi, "CACTI 5.1." HP Laboratories, Tech. Rep. HPL-2008-20, 2008.

[12] Michael Golden, Srikanth Arekapudi, and James Vinh, "40-entry unified out-of-order scheduler and integer execution unit for the AMD Bulldozer x86-64 core," in Proc. *IEEE International Solid-State Circuits Conference (ISSCC 2011), Digest of Technical Papers*, Feb. 2011, pp. 80–82.

[13] Hugh McIntyre, Srikanth Arekapudi, Eric Busta, Timothy Fischer, Michael Golden, Aaron Horiuchi, Tom Meneghini, Samuel Naffziger, and James Vinh, "Design of the two-core x86-64 AMD Bulldozer module in 32nm SOI CMOS," *IEEE J. Solid-State Circuits*, vol. 47, no. 1, pp. 164–176, Jan. 2012.

[14] Oguz Ergin, Deniz Balkan, Kanad Ghosea, and Dmitry Ponomarev, "Register packing: Exploiting narrow-width operands for reducing register file pressure," in Proc. *the 37th Annual International Symposium on Microarchitecture (MICRO)*, Dec. 2004, pp. 304–315.

[15] J. Abella, J. Carretero, P. Chaparro, and X. Vera, "The split register file," in Proc. *Design, Automation Test in Europe Conference Exhibition (DATE)*, Mar. 2010, pp. 945–948.

[16] Gary S. Ditlow, Robert K. Montoye, Salvatore N. Storino, Sherman M. Dance, Sebastian Ehrenreich, Bruce M. Fleischer, Thomas W. Fox, Kyle M. Holmes, Junichi Mihara, Yutaka Nakamura, Shohji Onishi, Robert Shearer, Dieter Wendel, and Leland Chang, "A 4R2W register file for a 2.3GHz wire-speed POWER processor with double-pumped write operation," in Proc. *IEEE International Solid-State Circuits Conference (ISSCC 2011), Digest of Technical Papers*, Feb. 2011, pp. 256–258.

[17] Jinuk Luke Shin, Robert Golla, Hongping Li, Sudesna Dash, Youngmoon Choi, Alan Smith, Harikaran Sathianathan, Mayur Joshi, Heechoul Park, Mohamed Elgebaly, Sebastian Turullols, Song Kim, Robert Masleid, Georgios K. Konstadinidis, Mary Jo Doherty, Greg Grohoski, and Curtis McAllister, "The next generation 64b SPARC core in a T4 SoC processor," *IEEE J. Solid-State Circuits*, vol. 48, no. 1, pp. 82–90, Jan. 2013.

[18] John Feehrer, Sumti Jairath, Paul Loewenstein, Ram Sivaramakrishnan, David Smentek, Sebastian Turullols, and Ali Vahidsafa, "The Oracle Sparc T5 16-core processor scales to eight sockets," *IEEE Micro*, vol. 33, no. 2, pp. 48–57, Mar. 2013.

[19] Jason M. Hart, Hoyeol Cho, Yuefei Ge, Gregory Gruber, Dawei Huang, Changku Hwang, Daisy Jian, Tim Johnson, Georgios K. Konstadinidis, Venkat Krishnaswamy, Lance Kwong, Robert P. Masleid, Umesh Nawathe Rakesh Mehta, Aparna Ramachandran, Hari Sathianathan, Yongning Sheng, Jinuk Luke Shin, Sebastian Turullols, Zuxu Qin, and King C. Yen, "A 3.6GHz 16-core SPARC SoC processor in 28 nm," *IEEE J. Solid-State Circuits*, vol. 49, no. 1, pp. 19–31, Jan. 2014.

[20] J. Adam Butts and Gurindar S. Sohi, "Use-based register caching with decoupled indexing," in Proc. *the 31st Annual International Symposium on Computer Architecture (ISCA)*, Jun. 2004, pp. 302–313.

[21] Mark Gebhart, Daniel R. Johnson, David Tarjan, Stephen W. Keckler, William J. Dally, Erik Lindhol, and Kevin Skadron, "Energy-efficient mechanisms for managing thread context in throughput processors," in Proc. *the 38th Annual International Symposium on Computer Architecture (ISCA)*, Jun. 2011, pp. 235–246.

[22] Mark Gebhart, Daniel R. Johnson, David Tarjan, Stephen W. Keckler, William J. Dally, Erik Lindholm, and Kevin Skadron, "A hierarchical thread scheduler and register file for energy-efficient throughput processors," *ACM Transactions on Computer Systems*, vol. 30, no. 2, pp. 8:1–8:38, Apr. 2012.

[23] Jessica H. Tseng and Krste Asanović, "Banked multiported register files for high-frequency superscalar microprocessors," in Proc. *the 30th Annual International Symposium on Computer Architecture (ISCA)*, Jun. 2003, pp. 62–71.

[24] ——, "A speculative control scheme for an energy-efficient banked register file," *IEEE Trans. Comput.*, vol. 54, no. 6, pp. 741–751, Jun. 2005.

[25] Tetsuo Hironaka, Moto Maeda, Kazuya Tanigawa, Tetsuya Sueyoshi, Kenichi Aoyama, Tetsushi Koide, Hans Juergen Mattausch, and Tadashi Saito, "Superscalar processor with multi-bank register file," in Proc. *Innovative Architecture for Future Generation High-Performance Processors and Systems (IWIA)*, Jan. 2005.

[26] Nam Duong and Rakesh Kumar, "Register multimapping: A technique for reducing register bank conflicts in processors with large register files," in Proc. *IEEE Symposium on Application Specific Processors (SASP '09)*, 2009, pp. 50–53.

[27] Il Park, Michael D. Powell, and T. N. Vijaykumar, "Reducing register ports for higher speed and lower energy," in Proc. *the 35th Annual International Symposium on Microarchitecture (MICRO)*, 2002, pp. 171–182.

[28] Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P. Jouppi, "CACTI 6.0: A tool to model large caches," HP Laboratories, Tech. Rep. HPL-2009-85, 2009.

[29] Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi, "The McPAT framework for multicore and manycore architectures: Simultaneously modeling power, area, and timing," *ACM Transactions on Architecture and Code Optimization*, vol. 10, no. 1, pp. 5:1–5:29, Apr. 2013.

[30] Rajeev Balasubramonian, Sandhya Dwarkadas, and David H. Albonesi, "Reducing the complexity of the register file in dynamic superscalar proces-

sors," in Proc. *the 34th Annual International Symposium on Microarchitecture (MICRO)*, Dec. 2001, pp. 237–248.

[31] "Onikiri 2." [Online]. Available: https://github.com/onikiri/onikiri2/

[32] Kenneth C. Yeager, "The MIPS R10000 superscalar microprocessor," *IEEE Micro*, vol. 16, no. 2, pp. 28–40, Apr. 1996.

[33] John H. Edmondson, Paul Rubinfeld, Ronald Preston, and Vidya Rajagopalan, "Superscalar instruction execution in the 21164 Alpha microprocessor," *IEEE Micro*, vol. 15, no. 2, pp. 33–43, Apr. 1995.

[34] Ilhyun Kim and Mikko H. Lipasti, "Half-price architecture," in Proc. *the 30th Annual International Symposium on Computer Architecture (ISCA)*, Jun. 2003, pp. 28–38.

[35] Rama Sangireddy, "Reducing rename logic complexity for high-speed and low-power front-end architectures," *IEEE Trans. Comput.*, vol. 55, no. 6, pp. 672–685, Jun. 2006.

[36] Gurindar S. Sohi, Scott E. Breach, and T.N. Vijaykumar, "Multiscalar processors," in Proc. *the 22nd Annual International Symposium on Computer Architecture (ISCA)*, Jun. 1995, pp. 414–425.

[37] Gregory A. Kemp and Manoj Franklin, "PEWs: a decentralized dynamic scheduler for ILP processing," in Proc. *International Conference on Parallel Processing*, vol. 3, Aug. 1996, pp. 239–246.

[38] Subbarao Palacharla, Norman P. Jouppi, and J. E. Smith, "Complexity-effective superscalar processors," in Proc. *the 24th Annual International Symposium on Computer Architecture (ISCA)*, Jun. 1997, pp. 206–218.

[39] Keith I. Farkas, Paul Chow, Norman P. Jouppi, and Zvonko Vranesic, "The multicluster architecture: reducing cycle time through partitioning," in Proc. *the 30th Annual International Symposium on Microarchitecture (MICRO)*, Dec. 1997, pp. 149–159.

[40] Richard E. Kessler, "The Alpha 21264 microprocessor," *IEEE Micro*, vol. 19, no. 2, pp. 24–36, Mar./Apr. 1999.

[41] Pierre Salverda and Craig Zilles, "A criticality analysis of clustering in superscalar processors," in Proc. *the 38th Annual International Symposium on Microarchitecture (MICRO)*, Nov. 2005, pp. 12–66.

[42] Francis Tseng and Yale N. Patt, "Achieving out-of-order performance with almost in-order complexity," in Proc. *the 35th Annual International Symposium on Computer Architecture (ISCA)*, Jun. 2008, pp. 3–12.

[43] Timo Stripf, Ralf Koenig, Patrick Rieder, and Juergen Becker, "A compiler back-end for reconfigurable, mixed-ISA processors with clustered register files," in Proc. *IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW)*, May 2012, pp. 462–469.

[44] James E. Stine, Ivan Castellanos, Michael Wood, Jeff Henson, Fred Love, W. Rhett Davis, Paul D. Franzon, Michael Bucher, Sunil Basavarajaiah, Julie Oh, and Ravi Jenkal, "FreePDK: An open-source variation-aware design kit," in Proc. *IEEE International Conference on Microelectronic Systems Education (MSE '07.)*, Jun. 2007, pp. 173–174.

[45] K. Mistry, C. Allen, C. Auth, B. Beattie, D. Bergstrom, M. Bost, M. Brazier, M. Buehler, A. Cappellani, R. Chau, C.-H. Choi, G. Ding, K. Fischer, T. Ghani, R. Grover, W. Han, D. Hanken, M. Hattendorf, J. He, J. Hicks, R. Huessner, D. Ingerly, P. Jain, R. James, L. Jong, S. Joshi, C. Kenyon, K. Kuhn, K. Lee, H. Liu, J. Maiz, B. Mcintyre, P. Moon, J. Neirynck, S. Pae, C. Parker, D. Parsons, C. Prasad, L. Pipes, M. Prince, P. Ranade, T. Reynolds, J. Sandford, L. Shifren, J. Sebastian, J. Seiple, D. Simon, S. Sivakumar, P. Smith, C. Thomas, T. Troeger, P. Vandervoorn, S. Williams, and K. Zawadzki, "A 45nm logic technology with high-k+metal gate transistors, strained silicon, 9 Cu interconnect layers, 193nm dry patterning, and 100% Pb-free packaging," in Proc. *IEEE International Electron Devices Meeting (IEDM 2007)*, 2007, pp. 247–250.

[46] Jeffrey R. Diamond, Donald S. Fussell, and Stephen W. Keckler, "Arbitrary modulus indexing," in Proc. *the 47th Annual International Symposium on Microarchitecture (MICRO)*, Dec. 2014.

[47] John Paul Shen and Mikko H. Lipasti, *Modern Processor Design: Fundamentals of Superscalar Processors*, 1st ed. Waveland Press, Inc., 2013.

[48] Oliver Brun and Jean-Marie Garcia, "Analytical solution of finite capacity M/D/1 queues," *Journal of Applied Probability*, vol. 37, no. 4, pp. 1092–1098, Dec. 2000.

[49] Anirban DasGupta, "The matching, birthday and the strong birthday problem: a contemporary review," *Journal of Statistical Planning and Inference*, vol. 130, no. 1, pp. 377–389, 2005.

[50] Arthur Perais, André Seznec, Pierre Michaud, Andreas Sembranty, and Erik Hagersten, "Cost-effective speculative scheduling in high performance processors," in Proc. *the 42nd Annual International Symposium on Computer Architecture (ISCA)*, Jun. 2015, pp. 247–259.

[51] Kirti Bhanushali and W. Rhett Davis, "FreePDK15: An open-source predictive process design kit for 15nm FinFET technology," in Proc. *International Symposium on Physical Design (ISPD)*, 2015, pp. 165–170.

[52] Mayler Martins, Jody Maick Matos, Renato P. Ribas, André Reis, Guilherme Schlinker, Lucio Rech, and Jens Michelsen, "Open cell library in 15nm FreePDK technology," in Proc. *International Symposium on Physical Design (ISPD)*, 2015, pp. 171–178.

[53] Shien-Yang Wu, J.J. Liaw, C.Y. Lin, M.C. Chiang, C.K. Yang, J.Y. Cheng, M.H. Tsai, M.Y. Liu, P.H. Wu, C.H. Chang, L.C. Hu, C.I. Lin, H.F. Chen, S.Y. Chang, S.H. Wang, P.Y. Tong, Y.L. Hsieh, K.H. Pan, C.H. Hsieh, C.H. Chen, C.H. Yao, C.C Chen, T.L. Lee, C.W. Chang, H.J. Lin, S.C. Chen, J.H. Shieh, M.H. Tsai, S.M. Jang, K.S. Chen, Y. Ku, Y.C. See, and W.J. Lo, "A highly manufacturable 28nm CMOS low power platform technology with fully functional 64Mb SRAM using dual/tripe gate oxide process," in Proc. *Symposium on VLSI Technology 2009*, Jun. 2009, pp. 210–211.

[54] M. Yabuuchi, H. Fujiwara, Y. Tsukamoto, M. Tanaka, S. Tanaka, and K. Nii, "A 28nm high density 1R/1W 8T-SRAM macro with screening circuitry against read disturb failure," in Proc. *IEEE Custom Integrated Circuits Conference (CICC)*, Sep. 2013, pp. 1–4.

Publications

Journal Papers

[1] Junji Yamada, Ushio Jimbo, Ryota Shioya, Masahiro Goshima, Shuichi Sakai: Skewed Multistaged Multibanked Register File for Area and Energy Efficiency, *IEICE Transactions on Information and Systems*, Vol. E100-D, No.4 (2017). (accepted).

[2] Junji Yamada, Ushio Jimbo, Ryota Shioya, Masahiro Goshima, Shuichi Sakai: Design of a Register Cache System with an Open Source Process Design Kit for 45nm Technology, *IEICE Transactions on Electronics, Special Section on Low-Power and High-Speed Chips*, Vol. E100-C, No. 3 (2017). (accepted).

[3] Ushio Jimbo, Junji Yamada, Ryota Shioya, Masahiro Goshima: Applying Razor Flip-Flops to SRAM Read Circuits, *IEICE Transactions on Electronics, Special Section on Low-Power and High-Speed Chips*, Vol. E100-C, No.3 (2017). (accepted).

[4] Masaya Kawano, Nobuaki Takahashi, Yoichiro Kurita, Koji Soejima, Masahiro Komuro, Satoshi Matsui, Shiro Uchiyama, Kayoko Shibata, Junji Yamada, Masakazu Ishino, Hiroaki Ikeda, Yoshimi Egawa, Yoshihiro Saeki, Osamu Kato, Hidekazu Kikuchi, Azusa Yanagisawa, Toshiro Mitsuhashi: Development of High-Density Package for Stacked DRAM Using Through-Silicon Vias, *IEICE Transactions on Electronics*, Vol. J90-C, No. 11, pp. 724–733 (2011). (in Japanese).

[5] Junji Yamada, Shoichiro Asano: A Study on Data Structures of Key-Value Store for SSD, *IEICE Transactions on Information and Systems*, Vol. J96-D, No. 3, pp. 519–530 (2013). (in Japanese).

Domestic Conferences

[6] Ushio Jimbo, Junji Yamada, Masahiro Goshima: Application of Clocking Scheme That Enables Dynamic Time Borrowing, *The 1st. cross-disciplinary Workshop on Computing Systems, Infrastructures, and Programming (xSIG 2017)* (2017). (in Japanese). (accepted).

[7] Junji Yamada, Ushio Jimbo, Ryota Shioya, Masahiro Goshima, Shuichi Sakai: Bank-Aware Instruction Scheduler for Multibanked Register File, *The 1st. cross-disciplinary Workshop on Computing Systems, Infrastructures, and Programming (xSIG 2017)* (2017). (accepted).

Oral Presentations

[8] 山田淳二, 倉田成己, 塩谷亮太, 五島正裕, 坂井修一: レジスタ・キャッシュ・システムにおけるレジスタ・ファイルのマルチバンク化, 情報処理学会研究報告 2014-ARC-212(18), pp. 1–14 (2014). (in Japanese).

[9] 神保潮, 山田淳二, 五島正裕, 坂井修一: ダイナミック・ロジックへのタイミング・フォールト検出手法の適用, 情報処理学会研究報告 2014-ARC-210(18), pp. 1–8 (2014). (in Japanese).

[10] 山田淳二, 倉田成己, 塩谷亮太, 五島正裕, 坂井修一: レジスタ・キャッシュ・システムにおけるレジスタ・ファイルへの書き込みの削減手法, 情報処理学会研究報告 2014-ARC-208(11), pp. 1–12 (2014). (in Japanese).

[11] 山田淳二, 倉田成己, 塩谷亮太, 五島正裕, 坂井修一: レジスタ・キャッシュ・システムの省電力化手法, 情報処理学会研究報告 2013-ARC-206(3), pp. 1–9 (2013). (in Japanese).

[12] Junji Yamada, Shoichiro Asano: Fast and compact index structure using Bloom Filter : For high speed database on NAND Flash SSD, *IEICE Technical Report. CPSY, Computer Systems* 111(255), pp. 31–36 (2011). (in Japanese).

Acknowledgments

First and foremost, I would like to express my profound gratitude to my advisor, Professor Shuichi Sakai and Professor Masahiro Goshima, who have been my supervisor during my doctor course period. They gave me helpful suggesions, invaluable advice, constructive comments and warm encouragements.

I would like to thank Associate Professor Hidetsugu Irie and Associate Professor Ryota Shioya for their insightful comments and suggestions during my doctor course period.

I would like to thank Professor Kenjiro Taura, Professor Fujita Masahiro, and Professor Makoto Ikeda who carefully read this thesis and give the various comments at my preliminary defense.

Special thanks to Ms. MinSeong Choi, Mr. Ushio Jimbo, Dr. Naruki Kurata, Mr. Mizuki Miyanaga, Mr. Hayato Nomura, and the rest of laboratory members. They have supported me from various aspects during everyday life.

I would like to thank Ms. Harumi Yagihara, Ms. Tamaki Hasebe, Ms. Noriko Katsu, Ms. Junko Kashimura and Ms. Saiko Akabane for their dedications to make administration affairs run smoothly.

I also would like to thank Mr. Yoshifumi Okamura and the rest of my former colleagues at Micron Memory Japan, Inc.

Finally, I would like to thank my family and friends in the University of Tokyo and elsewhere for their support.

This research is partially supported by Grant-in-Aid for Scientific Research No. 16H05855, 16H02797 and 26280012 from Ministry of Education, Culture, Sports, Science and Technology Japan. This work is supported by VLSI Design and Education Center (VDEC), the University of Tokyo with the collaboration with CADENCE Corporation, SYNOPSYS Corporation and Mentor Graphics Corporation.