Register Cache System not for Latency Reduction Purpose

Ryota Shioya*†, Kazuo Horio‡, Masahiro Goshima* and Shuichi Sakai*
* *Department of Information and Communication Engineering, University of Tokyo*
† *Research fellow of the Japan Society for the Promotion of Science*
‡ *Fujitsu Laboratories Ltd.*
Email: {shioya, horio, goshima, sakai}@mtl.t.u-tokyo.ac.jp

***Abstract*—A register cache has been proposed to solve the problems of the huge register files of recent superscalar processors. The register cache reduces the effective access latency of the register file for IPC improvement, simplifies the bypass network, and reduces the ports of the main register file. Though the primary purpose of the previous works is to improve IPC, the misses on the register cache may degrade the IPC. We propose Non-Latency-Oriented Register Cache System (NORCS). Though the effects of NORCS are the same as the conventional systems, it is free from register cache miss penalties that the conventional systems suffer from. In NORCS, the register cache itself is not different from that of the conventional systems. The difference is that the instruction pipeline has stages to read the main register file, which all instructions go through regardless of register cache hit / miss. Therefore, the instruction pipeline of NORCS is not immediately disturbed by the register cache misses. For a realistic 4-way superscalar processor, NORCS can simplify the bypass network to the same complexity as a 1-cycle-latency register file, and reduce the ports of the main register file from 12 to 4. CACTI simulation shows that the area and power consumption are reduced to 24.9% and 31.9% compared to the baseline model without register cache. Though these results are not different from the conventional systems, IPCs differ greatly. IPC of the conventional system decreases to 83.1% because of the cache miss penalties, while that of NORCS is retained at 98.0%.**

***Keywords*-Register file, Register Cache, Instruction pipeline, Instruction level parallelism, Low-energy Technologies**

I. Introduction

Programs essentially contain fragments that cannot be parallelized. As Amdahl's law suggests, speed-up of a single core which executes sequential portion of programs will remain to be important even in the era of many-core processors. In fact, the major processor vendors are maintaining the widths of each core of mainstream multi-core processors.

The physical register file and the bypass network have been ones of the most serious bottlenecks of recent superscalar processor cores.

In order to achieve high IPC, superscalar processor cores require a physical register file of high capacity to meet the number of in-flight instructions. In addition, SMT cores require a register file whose capacity is proportional to the number of threads.

A 4-issue superscalar processor core generally requires a register file with 8 read and 4 write ports. The register file consists of a multi-port RAM, and its circuit area is proportional to the square of the number of ports [1], [2]. **Figure 1**, a chip photograph of an Intel Pentium 4 processor, shows that the circuit area of the register file is comparable to that of the level-1 data cache despite that the capacities are different by more than an order of magnitude. This holds true for more recent processor cores.

The circuit area of the register file is comparable to that of the level-1 data cache, and the access latency of the register file is nearly equal to that of the level-1 data cache. Just like the level-1 data cache, the register file cannot be accessed in 1 cycle, and about 2 or 3 pipeline stages are assigned to it [3], [4]. This pipelining of the register file access causes the following two problems.

The first, IPC degradation, is a general problem with deeper pipelines. Increase in the latency of the register file increases miss penalties of speculations such as branch prediction. Moreover, deeper pipelines are susceptible to resource shortage and it results in pipeline stalls [5], [6]. However, these effects are indirect, and don't degrade the IPC so seriously. The IPC degradation for every 1 stage increase is no more than 2% even in the worst cases [7].

The second problem is specific to the register file latency, and is more serious than the IPC problem. The pipelining of the register file access requires a large bypass network. Instruction results produced in the last $2l$ cycles have to be forwarded through the bypass where l is the register file latency. The bypass network consists of long wires and series of wide multiplexers. A larger bypass network could limit the clock frequency, and consumes a larger amount of energy.

In addition to that of the bypass network, energy consumption of the register file itself is also serious. Energy consumption of a RAM is proportional to the circuit area and the access frequency [1], [2]. Only load/store instructions access the level-1 data cache just once. On the other hand, almost all the instructions access the register file more than twice (once for read and once for write). Thus, the register file, which has the circuit area comparable to that of the level-1 data cache, consumes much more energy than the level-1 data cache. In addition, SMT increases the access frequency in proportion to increase in throughput.

The energy consumption and resulting heat are ones of

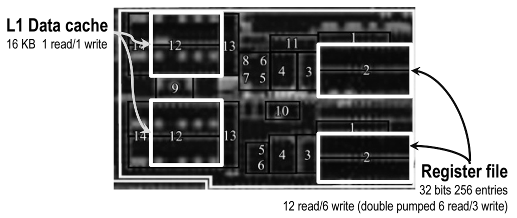


Figure 1. Integer execution core of Intel Pentium 4 Processor [11]. Pentium 4 processor uses a double-pumped 6-read/3-write register file as a 12-read/ 6-write register file [3].

the most serious problems of recent processor cores. The region including the register file and the bypass network is a ***hot spot*** in a core, and limits the clock frequency.

Naïve Methods to Reduce Complexity: To simply reduce the complexity of the register file or the bypass network, some naïve methods to reduce the number of the resources can be considered.

As for the bypass network, techniques of incomplete bypass have been proposed [8], [9], [10]. An incomplete bypass only provides results in the last $m < 2l$ cycles. After m cycles have passed from the cycle when a producer instruction is executed, the consumer have to wait to be issued so that it can obtain the result from the register file.

If the ports of the register file is reduced, the latency of the register file is reduced, and then the complexity of the bypass network is also reduced. In this case, the pipeline is disturbed when the ports fall short.

These techniques result in a trade-off between complexity and IPC. Section VI shows that the IPC degradation is as high as 20% in the worst cases.

A ***register cache*** is a better solution for these problems [12], [9], [10]. As its name suggests, a register cache is a cache to the ***main register file***. The capacity of the register cache is small enough to limit the access latency generally to 1 cycle. As far as the register cache hits, it is equivalent to a conventional register file with 1-cycle latency. This brings in the following two effects.

Firstly, the register cache is free from the IPC degradation stated before because the register cache is equivalent to a 1-cycle-latency register file. However, this effect is minor because this IPC degradation is small as stated before.

Secondly, the register cache reduces the complexity, energy consumption, and heat in the following ways:

1) **Simplifying bypass network:** The bypass is necessary not for the main register file but for the register cache. Thus, the bypass network is almost the same as that of a 1-cycle-latency register file (Section II).
2) **Reducing main register file ports:** The number of ports of the main register file can be reduced because only operands that miss the register cache access the main register file. Section VI shows that the number of the ports can be reduced from 12 to 4.

Though the register cache can solve these problems, it suffers from IPC degradation for yet another reason than stated before. As described in Section III, out-of-order scheduling of superscalar processor cores does not have capability to hide the main register file access latency on register cache misses. To make matters worse, the ***effective miss rate***, which is given by the probability of the pipeline disturbance, is much worse than the miss rate of the register cache itself. The instruction pipeline is disturbed if any one of the operands that simultaneously access the register cache cause a miss. For example, the evaluation result of 456.hmmer of SPEC2006 in Section VI shows that the register cache hit rate is as high as 94.2%, but an average number of operands that access the register cache a cycles is 2.49, then theoretical effective miss rate is $1 - 0.942^{2.49} \approx 13.9\%$, and the IPC degradation is as high as 10.2%.

This paper proposes a **Non-latency-Oriented Register Cache System** (**NORCS**). The **register cache system** refers to a system of accessing the register cache, and consists of the register cache, the main register file, and the instruction pipeline stages to access them. For simplicity, we refer to a conventional register cache system as a **Latency-Oriented Register Cache System** (**LORCS**). As its name suggests, NORCS is a register cache system that does not reduce the latency. The register cache and the main register file of NORCS are almost the same as those of LORCS. NORCS is characterized by its unique instruction pipeline.

LORCS has a *pipeline that assumes hit*. The pipeline has a stage to access the register cache but no stages to access the main register file. This is the same as the level-1 cache. A conventional instruction pipeline has the stages to access the level-1 cache but not the level-2 cache. On level-1 cache misses, pipeline stall or flush is usually introduced to make time to access the level-2 cache. It is because the relative positions of instructions in the pipeline cannot be changed after they are issued. This holds true for LORCS. Since LORCS reduces the latency, it results in considerable performance degradation caused by pipeline disturbance.

In contrast, NORCS has a *pipeline that assumes miss*. The pipeline has stages to access the main register file, and all instructions pass through the main register file access stages regardless of register cache hit or miss. On a miss, the main register file provides a value in the end of the main register file access stages. While on a hit, the register cache provides the value also in the end of the main register file access stages. Thus, NORCS doesn't reduce the access latency of the register file, and the IPC is not improved by nature. Instead, NORCS is free from such pipeline disturbance as LORCS does, because time to access the main register file is provided as stages in the pipeline. That is, the IPC is not necessarily improved, but nor considerably degraded. While maintaining IPC, NORCS reduces the complexity in basically the same way as LORCS.

The idea of a cache that doesn't reduce the latency might sound weird. For example, a level-1 cache that has the same latency as a level-2 cache is perfectly meaningless. One leading textbook defines a cache as "a small, *fast* memory lo-

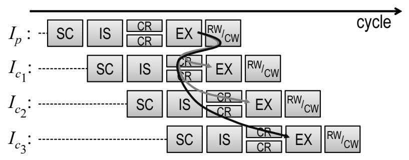


Figure 2. Backend Pipeline of LORCS (Hit)

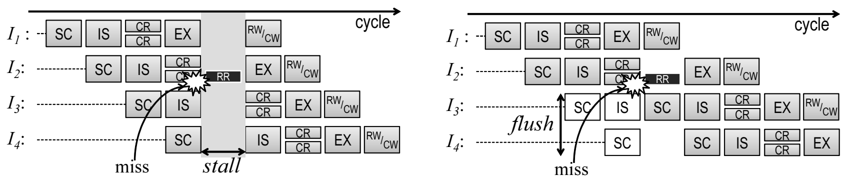

(a) Stall (b) Flush

Figure 3. Backend Pipeline of LORCS (Miss)

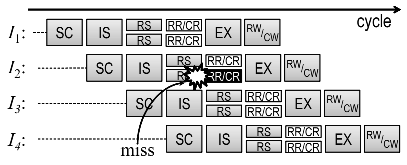


Figure 4. Backend Pipeline of NORCS

cated close to the CPU that holds the most recently accessed code or data"[13]. According to this definition, NORCS isn't a cache because NORCS is not *fast*. Non-latency-oriented cache system is useful only if applied to the register file, and NORCS is the only cache system that doesn't reduce latency. The precise reason why a non-latency-oriented cache system is only applicable to the register file is complicated, and discussed in Section V.

The rest of the paper is organized as follows. Section II and III introduces LORCS. After Section IV describes detailed design of NORCS, Section V gives considerations on NORCS. Then, Section VI presents evaluation results.

II. LORCS

This section describes LORCS (Latency-Oriented Register Cache System) focusing on its behavior on the register cache hit, in order to explain how it solve the problems caused by a large register file, especially increase in the complexity, energy consumption, and heat of the register file and the bypass network [12], [9], [10]. The behavior on the register cache miss is described in the next section.

A. Instruction Pipeline of LORCS

Figure 2 shows the behavior of the backend pipeline of LORCS. The LORCS in this figure has a register cache and a main register file with 1-cycle latency, respectively. SC, IS, EX, CR, CW, and RW indicate the stages for instruction scheduling, issue, execute, register cache read, register cache write, and, main register file write, respectively. RW/CW indicates that main register file write and cache write is performed at the same stage. CR stages are *horizontally* divided into 2 to indicate that each instruction has 2 source operands, which access the register cache in parallel.

In so far as the accesses hit the register cache, it behaves as a register file with 1-cycle latency, and cycles to read the main register file don't appear in the figure. The instruction pipeline is disturbed on a register cache miss for the same reason as a usual level-1 data cache. We describe details of the cache miss in the next section.

B. Register Write

Instruction results written to the main register file are temporarily held in a write buffer before they are moved to the main register file.

This write to the main register file through the write buffer uses the *write-through policy*; that is, the results are written both to the register cache and to the write buffer in parallel on the RW/CW stage right after the execution stage.

Unlike usual data caches, the use of the *write-back policy* does not reduce the number of accesses to the main register file. In usual data caches, stores to the same addresses often occur, making the number of write-backs less than the number of the stores. While in the register cache system, writes to the same entry seldom occurs because the register renaming of out-of-order superscalar processors eliminates such overwrites to the same entries.

C. Simplifying Bypass Network

The results of instructions are written both to the register cache and to the main register file (through the write buffer) in the RW/CW stage. Thus, the results of recently executed instructions are most likely to be found in the register cache, which means that it is not necessary for the main register file, the write buffer or the bypass network to provide such recent results.

In Figure 2, the instructions from I_{c_1} to I_{c_3} depend on I_p. I_{c_3} can receive the result through the register cache because I_{c_3} reads the register cache after I_p writes the result to the register cache. Therefore, only I_{c_1} and I_{c_2} have to receive the result of I_p through the bypass network. The bypass network is required to provide the results executed in the last 2 cycles only, which is the same as a register file with 1-cycle latency.

D. Reducing Main Register File Ports

The register cache in LORCS can reduce the number of read ports of main register file by allowing only the missed operands to access the main register file. The register cache can also reduce the number of write ports of main register file by using a write buffer without any additional forwarding paths. The use of the write buffer reduces the number of the write ports to the average throughput of instruction execution. This method only works on register writes and does not work on reads, because writes are not critical and can be delayed like a general store buffer in memory access, however reads are critical and cannot be delayed by using such buffer.

The evaluation results in Section VI show that 2 read and 2 write ports of the main register file are sufficient to meet the requirement.

The area of a main register file with a few ports is considerably small, because the area of a RAM is proportional to the square of the number of the ports [1], [2]. The evaluation results (Section VI) show that the area of the main register file is almost the same as that of the register cache, and the sum of the areas of the register cache and main register file is reduced to 30.7% of a full-port register file.

The reduction in area consequently reduces the latency of the main register file. The area of the main register file is almost the same as that of the register cache, thus the main register file can also be accessed in 1 cycle.

III. Register Cache Miss in LORCS

As Butts et al. pointed out, there are only two pragmatic options to read the main register file on a register cache miss, that is, ***stall*** and ***flush*** [10].

A. Stall or Flush

Figure 3(a) shows the behavior of the backend pipeline on the register cache with stall. In this model, execution of instructions is delayed by the main register file latency, which is 1 cycle in the figure.

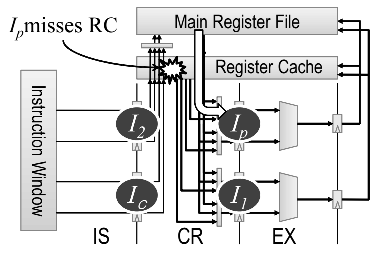
Figure 3(b) shows the behavior with flush. The flush model flushes and re-issues all the instructions that have been issued in the same or later cycles as the instruction that cause a register cache miss. In this case, execution is delayed by the *issue latency*, which is given by the number of cycles from the schedule stage to the stage where the flush occurs minus 1, because flushed instructions need to be replayed from the schedule stage. In this figure, 1 cycle is assigned to the schedule, issue, and the register cache read stages, and the issue latency adds up to $3 - 1 = 2$ cycles.

In general, the main register file latency is shorter than the issue latency, and the stall model works better than the flush model as evaluation results in Section VI show[1].

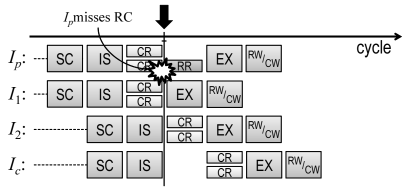
As Butts et al. pointed out [10], stall and flush are only pragmatic options on the register cache miss. We want to stress this point, because we have been asked so frequently why different options cannot be used to improve LORCS performance, which are ***selective stall*** and ***hit/miss prediction***. Though they might seem to reduce the miss penalty of LORCS easily, they turn out to be either very difficult to implement or have serious side effects because of resource conflicts.

B. Selective Stall

Figure 5 (a) and **(b)** show the block diagram and the pipeline chart of LORCS, respectively. In these figures, I_p, I_1, I_2, and I_c have been issued to the backend pipeline, and



(a) Block diagram



(b) pipeline chart

Figure 5. Selective stall

I_c is dependent on I_p. Figure 5 (a) is a snapshot of the point in time indicated by the down arrow in (b). At this point, I_p misses the register cache.

Only from Figure 5 (b), the *selective stall*, which would mean stalling I_p and I_c while continuing the execution of I_1 and I_2, might seem possible. This, however, is because this kind of chart doesn't have the capability to represent resource conflict. **Figure 5** (a) shows I_2 actually cannot proceed because it is blocked by I_p in the same *lane* of the pipeline. I_c, which must be stalled because it is dependent on I_p, blocks the other lane of any further issue. Thus, no instruction can be issued until I_p finally moves on. In this situation, only I_1 may avoid being stalled actually.

For an instruction to proceed, it is necessary to check not only that it is independent on the cache-missed instruction but also that the cache-missed instruction and its dependent instructions don't exist ahead in the same lane. This requires a recursive search of dependent instructions every cycle.

After all, *selective stall* is difficult because the backend of a out-of-order superscalar processor core is also a pipeline, and it does not have the capability to re-schedule instructions. Instructions are primarily scheduled in the instruction window so that they can flow through the backend pipeline.

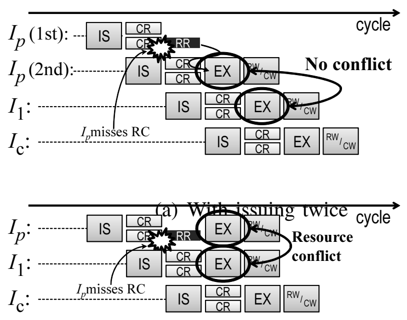
C. Hit/Miss Prediction

There have been proposal on hit/miss prediction for a usual data cache [14]. In these proposals, when a producer instruction is predicted to miss the cache, the issue of the consumer is delayed to avoid pipeline disturbance.

Hit/miss prediction for the register cache is slightly different from that for the data cache. **Figure 6 (a)** and **(b)** show pipeline charts of LORCS. As shown in Figure 6 (a), instructions predicted to miss the register cache should be issued twice as follows:

1) **First issue:** I_p, which is predicted to miss the register cache, is issued at regular timing. I_p will miss the register cache as predicted, and start the main register file access in the same way as usual.

[1]Usual level-1 data caches work better with flush. This is because the stall penalties, given by the level-2 cache latency in this case, are longer than the flush penalties, given by the issue latencies.



(a) With issuing twice

(b) Without issuing twice

Figure 6. Hit/miss prediction with and without issuing twice

2) **Second issue:** I_p is issued again just after the latency of the main register file. Then, the data arrives from the main register file just in time for the instruction to be used for execution.

The purpose of the first issue is to start main register file access. In order to start the main register file access, the physical register number of the source operand is needed. Issuing the instruction is the most reasonable way to obtain the physical register number.

The purpose of the second issue is to actually execute the instruction and to resolve resource conflict.

Figure 6 (b) shows the pipeline without issuing twice. In this figure, I_p, which is predicted to miss, is issued and starts the main register file access, and then waits there (by some means) for the data from the main register file. In this case, I_p and I_2 cause conflict for the functional unit (if they try to use the same type of functional unit).

In Figure 6 (a), the instruction scheduler selects I_p for the second issue and the issue of I_1 is delayed to the next cycle, resolving the resource conflict between I_p and I_1 in the usual way. Issuing the instruction is the most reasonable way to resolve the resource conflict.

In the case of hit/miss prediction for the data cache, the second issue can be avoided. Suppose a load instruction I_m is predicted to miss the level-1 cache, and another load instruction I_h is issued later. I_m and I_h do not use the same functional unit, that is, I_m obtains its own data from the level-2 cache, while I_h from the level-1 cache. Both of the data items must be written to the register file (and passed to the consumer instructions through the bypass), and this may cause a conflict on the register file write port (and an input port of the bypass). This conflict, however, can be resolved by adding a write port (and an input port of the bypass) for the data from the level-2 cache [14]. This, however, is not applicable for the register cache, because it is not realistic to add dedicated functional units for all the instructions that are predicted to miss the register cache.

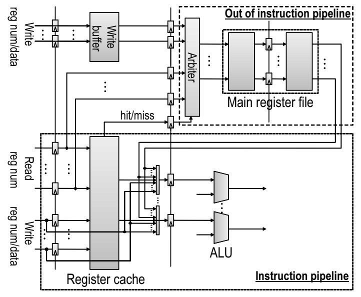


Figure 7. Block diagram of LORCS

As a result, issuing twice is the only practical way to implement hit/miss prediction in the register cache. This, however, consumes the issue width twice, and results in worse IPC than stall model, as is shown in Section VI.

IV. NORCS

We propose a ***NORCS*** (Non-latency-Oriented Register Cache System). Since NORCS doesn't reduce the access latency of the register file, and the IPC is not improved by nature. Instead, NORCS is free from such pipeline disturbance as LORCS does. That is, the IPC is not necessarily improved, but nor considerably degraded. While maintaining IPC, NORCS reduces the complexity in basically the same way as LORCS.

A. Structure of NORCS

Figures 7 and **8** show block diagrams of LORCS and NORCS, respectively. In these figures, 2 pipeline stages is assigned to accessing the main register file. NORCS has a similar structure to LORCS; the register cache and the main register file themselves are almost the same as those of LORCS.

The pipeline latches highlighted in Figure 8 are small but the most important difference between them. These latches are placed between the tag and the data arrays of the register cache, and the tag and the data arrays are accessed on different stages.

As shown in these figures, the difference in the logic is quite small, but it makes a great difference to the performance as described in the following sections.

B. Instruction Pipeline of NORCS

NORCS has a *pipeline that assumes miss*, and all instructions go through the stage(s) to read the main register file regardless of hit/miss to the register cache.

Figure 4 shows the pipeline chart of the NORCS. In this figure, the main register file, the tag and the data arrays

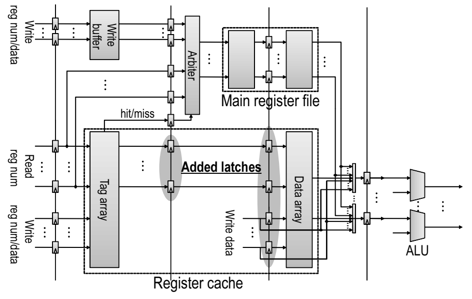


Figure 8. Block diagram of NORCS

of the register cache have 1-cycle latency, respectively. RS (***Register Scheduling***) indicates a stage for hit/miss detection of the register cache. In this figure, only the *lower* operand of the instruction I_2 causes the register cache miss. RR/CR indicates the stage to read either the main register file or the data array of the register cache depending on the result of the hit/miss detection on RS stage.

An issued instruction accesses the tag array of the register cache on the RS stage only for hit/miss detection. If its operands are found on the register cache (e.g., I_1) the instruction simply sends the result of the tag-matching to the following stages. On the non-painted RR/CR stage right before the execution stage, the data array is accessed depending on the tag-match result. If an operand is not found in the register cache (e.g., the *lower* operand I_2), the main register file is read on RR/CR stages painted in Figure 4 .

Pipeline Stall: Since the instruction pipeline of NORCS has RR stage to read the main register file, it is not disturbed only on a single register cache miss.

When the read ports of main register file fall short, that is, more operands than the number of the read ports of the main register file miss the register cache in a single cycle, the pipeline is disturbed. The pipeline must produce enough cycles to read the main register file by stalling the pipeline.

The conditions for the pipeline disturbance of LORCS and NORCS are summarized as follows:

- **LORCS:** If a single register cache miss occurs.
- **NORCS:** If register cache misses more than main register file read ports occur at a single cycle.

C. Simplifying Bypass Network

As described in Section II, LORCS can simplify the bypass network because the register cache of LORCS is equivalent to a 1-cycle-latency register file as far as the register cache hits. NORCS do so not in the same ways as LORCS because it doesn't reduce the latency of the register file, but by delaying the data array access of the register cache until the end of the main register file access stages. We describe this by comparing our technique with usual implementation of NORCS.

In usual implementation of a cache, the tag and the data arrays are accessed in parallel in order to obtain the data faster. **Figure 9** shows the pipeline chart of the usual implementation of NORCS, in which the data array and the tag array are accessed in parallel on CR stage. The stages are denoted by the same labels used in Figure 4. In this figure, I_c is dependent on I_p. I_c can receive the result of I_p through the register cache, because the CR stage comes a cycle after CW stage of I_p. If I_c had been issued in the earlier cycles, it must receive the result through the bypass. That is, the bypass must provide the result of I_p for 3 cycles after its issue.

Figure 10 shows the pipeline chart of proposed technique, in which the data array access of the register cache is delayed

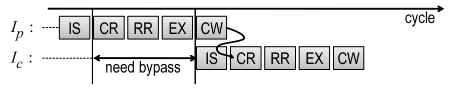


Figure 9. Bypass of NORCS (Naive implementation)

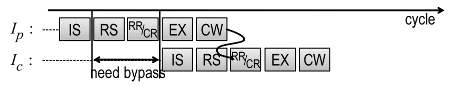


Figure 10. Bypass of NORCS

until the end of the main register file access stages. On the RR/CR stage right before the execution stage, the data array is accessed depending on the tag-match result detected on the RS stage. The bypass network only provides the result of I_p for the period of 2 cycles after its issue, because the RR/CR stage comes the cycle after the CW stage of I_p. This is shorter than the usual implementation by 1 cycle, and is the same as the register file with 1-cycle latency.

V. Considerations

This section gives consideration on NORCS. Section V-A explains why a cache system that doesn't a reduce latency works for a register cache but not for a usual data cache. Section V-B explains why NORCS outperforms LORCS.

A. Cache System That Doesn't Reduce Latency

The pipeline of NORCS works only for a register cache but not for a usual data cache. We try to explain this by using data dependency and the data flow graph (DFG). A cache system that doesn't reduce latency works if the latency doesn't affect the height of the graph.

Data dependencies can be categorized into "*value–value*" and "*via–index*". In a data cache, an *index* refers to a memory address; while in a register cache, an *index* refers to a register number. Data dependencies and the height of the DFG are summarized as follows:

- **Data Cache**
  - *Value–value*: *Value–value* dependency refers to one from a store to a dependent load instruction. In this case, increase in the latency of the data cache possibly heightens the DFG, because the data cache usually doesn't bypass values.
  - *Via–index*: *Via–index* dependency refers to one from load instruction to another instruction. Also in this case, increase in the latency of the data cache possibly heightens the DFG. One example of "via-index" dependency is pointer chasing. On the pointer chasing of the linked list, increase in the latency of the data cache can heightens the DFG.
- **Register Cache**
  - *Value–value*: Since the bypass network can immediately pass the data, the register cache latency is independent of the DFG height.
  - *Via–index* An *index* — register number is statically determined, that is, not calculated by other instructions. Therefore, there are no *value–index* dependencies in a register cache.

As a result, a cache system that doesn't reduce the latency works only for a register cache but not for a usual data cache.

B. Pipeline That Assumes Miss

NORCS achieves better performance than LORCS. The reason is, in short, this is a pipeline.

In a modern instruction pipeline with speculations, the latencies only degrade IPC indirectly through miss penalties. If there were no prediction misses or pipeline disturbance, longer latencies do not affect the IPC.

In the rest of this section, for simplicity, only branch prediction is taken into account, but the same is true for other predictions. On a branch prediction miss, the number of execution cycles is extended (approximately) by the miss penalty of the branch prediction. On a register cache miss, the number of execution cycles is extended (approximately) by the miss penalty of the register cache.

The total penalty cycles of LORCS are:

$$penalty_{bpred} \times \beta_{bpred} + latency_{\mathrm{MRF}} \times \beta_{\mathrm{RC}}, \tag{1}$$

where $penalty_{bpred}$ is the miss penalty of the branch prediction, $latency_{\mathrm{MRF}}$ is the latency of the main register file, β_{bpred} and β_{RC} are the *effective* miss rate of a branch prediction and register cache, respectively[2]. Note that β_{bpred} is not "branch prediction miss rate" for a branch instruction, but the *effective* miss rate given by the probability of the branch miss in each cycle. The same is true for β_{RC}.

The upper half of **Figure 11** shows the behavior on the register cache miss of LORCS. The stages are denoted by the same labels used in Figure 2 and 4. In the figure, the *lower* operand of the instruction I_3 misses the register cache. In the upper half of Figure 11, the lighter- and the darker-shaded regions correspond to $latency_{\mathrm{MRF}}$ and $penalty_{bpred}$ in the expression, respectively.

The branch miss penalty of NORCS is longer than that of LORCS by the latency of the main register file. Thus, the total penalty of NORCS is:

$$(penalty_{bpred} + latency_{\mathrm{MRF}}) \times \beta_{bpred}. \tag{2}$$

In the lower half of Figure 11, the surrounding lighter-shaded region and the surrounded darker-shadowed region correspond, respectively, to $latency_{\mathrm{MRF}}$ and $penalty_{bpred}$ in the expression.

Therefore, the difference between the total penalty cycles of LORCS and that of NORCS is:

$$(1) - (2) = latency_{\mathrm{MRF}}(\beta_{\mathrm{RC}} - \beta_{bpred}). \tag{3}$$

Thus, if $\beta_{\mathrm{RC}} > \beta_{bpred}$, NORCS outperforms LORCS. This expression can be interpreted as follows: NORCS moves the register cache miss penalty of LORCS ($latency_{\mathrm{MRF}}$) into the miss penalty of the branch prediction. β_{RC} is much higher than β_{bpred}, and this is the reason why NORCS can outperform LORCS.

[2]In the flush model, the register cache miss penalty is given by the issue latency, which is usually not shorter than $latency_{\mathrm{MRF}}$.

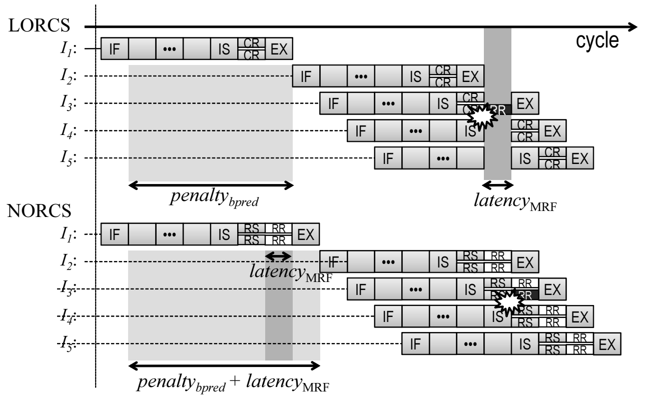


Figure 11. Pipeline of LORCS and NORCS

VI. Evaluation

A. Evaluation environment

We evaluated LORCS and NORCS using a cycle accurate processor simulator *Onikiri 2*[15] developed in our laboratory. Unlike SimpleScalar Tool set [16], which is used widely for researches on processor architecture, Onikiri 2 replays execution of instruction in the exact cycle when it is on the execution stage. Thus, Onikiri 2 can simulate more precisely than SimpleScalar Tool set, when adopting data predictions such as address match/mismatch prediction.

We used 29 programs from the SPEC CPU2006 [17] benchmark with *ref* data sets. The programs were compiled using gcc 4.2.2 with the "−O3" option. We skipped the first 1G instructions and evaluated the next 100M instructions. The left column of **Table I**, labeled "Baseline", gives the configuration of the baseline processor. This basic structure is based on MIPS R10000 out-of-order superscalar processor, which can issue up to 6 instructions per cycle. Some configurations such as the branch predictor and the caches are set to match modern processors. We implemented both LORCS and NORCS on the baseline processor.

1) Evaluation models: We evaluated the following models:

- **PRF:** The baseline model with Pipelined Register Files with complete bypass networks.
- **PRF-IB:** Pipelined Register Files with Incomplete Bypass networks. If operands are produced within 2 cycles, they are sent through the bypass network. Otherwise, the backend pipeline is stalled until the operands can be obtained through the register files [8]. The bypass networks have the same complexity as those for LORCS, NORCS, and register files with 1-cycle latency.
- **NORCS:** A model of NORCS. The register caches are adopted to the integer register files. Each register cache in the "infinite" model has the same number of entries as the corresponding register file.
- **LORCS:** A model of LORCS. The configuration of the register caches is the same as in NORCS model. Several

options on register cache miss are implemented; details of these are described later.

The left column of **Table II**, labeled "Baseline", summarizes the other parameters for the register file system for each model. "PRF", "RC", and "MRF" in the table mean the pipelined register file, the register cache, and the main register file, respectively.

2) Replacement Policy of Register Cache: In addition to LRU, we evaluated **USE-B** model that is a *use-based* replacement policy proposed by Butts et al. [10]. They used a *use-predictor* that predicts how many times a register will be referenced before its release [18] for replacement. The configuration of the use-predictor is shown in Table II. These parameters are the same as those given in [10].

3) Miss Model of LORCS: Regarding the behavior on a register cache miss in LORCS, STALL and FLUSH models implement backend stall and backend flush, respectively, as described in Section III-A. In addition to STALL and FLUSH, we evaluated the following idealized models:

- **SELECTIVE-FLUSH:** This model selectively flushes and re-issues only those instructions that miss the register cache and its dependent instructions.
- **PRED-PERFECT:** This model is extremely idealized. It uses hit/miss prediction described in Section III-C with a 100% hit rate. This removes absolutely any possible pipeline disturbance caused by register cache misses. Thus, the performance degradation from a model with "infinite" register cache only comes from consuming twice the issue width and delaying the execution of instructions that caused the register cache misses (Section III).

B. Evaluation Result

As described in Section V-B, Both NORCS and LORCS have the same benefit that is to reduce the circuit area and the energy consumption of the register file. However, performance of both model is significantly different. Therefore, it is important to compare areas and energy consumption per performance of each model. First, we show evaluation results of performance, area and energy consumption, and then we show relationship among them.

1) Register Cache Hit Rate: In addition to LRU and the use-based policy, we show evaluation results of pseudo optimal replacement policy in this section. OPT replacement policy is known as an ideal replacement policy, which replaces the entry that will not be referred to for the longest time by any instructions. POPT replaces the entry that will not be referred until the furthest future only by the in-flight instructions.

Figure 12 shows the average register cache hit rates in LORCS over all the benchmark programs. POPT and USE-B indicate models with the pseudo optimal replacement policy and the use-based policy, respectively. The number of the main register file ports is fixed at 2-read/2-write and the register cache miss model is fixed to STALL, because the results are strongly influenced only by the number of entries and replacement policies of the register cache. We also evaluated register cache hit rates in NORCS, but we only show the results for LORCS because there are no significant differences between these 2 models.

USE-B models show considerably high hit rates. The hit rates of USE-B models are higher than those of LRU models by about 3–4%, and as high as those of ideal POPT models. Hereafter, we use USE-B for LORCS while LRU for NORCS. NORCS shows better performance in spite of the lower hit rates of LRU.

2) Configuration of LORCS: LORCS has more parameters than NORCS. For the remainder of this section, we limit the search space for LORCS.

The Number of the main register file Ports: First, we evaluated the IPC of LORCS and NORCS models changing the number of the ports of the main register file. **Figures 13(a)** and **13(b)** show average relative IPCs to the model with a full-port main register file, that is 8 read /4 write. Figure 13(a) changes the number of the write ports while fixing the number of the read ports at 2. On the contrary, Figure 13(b) changes the number of the read ports fixing the number of the write ports at 2. We only show the

Table I
SIMULATION CONFIGURATIONS

Name	Baseline	Ultra-wide
ISA	Alpha	←
pipeline stages	fetch:3, rename:2, dispatch:2, issue:2	fetch:4, rename:5, dispatch:2, issue:1
fetch width	4 inst.	8 inst.
execution unit	int:2, fp:2, mem:2.	int:6, fp:4, mem:2.
inst. window	int:32, fp:16, mem:16	unified:128
ROB	128 entries	512 entries
branch pred.	8 KB g-share	16 KB g-share
miss penalty	11∼12 cycles	14∼15 cycles
BTB	2 K entries, 4 way	4 K entries, 4 way
RAS	8entries	64entries
L1C	32 KB, 4 way, 64 B/line,3 cycles	←
L2C	4 MB, 8 way, 64 B/line, 10 cycles	←
main memory	200 cycles	←

Table II
PARAMETERS OF REGISTER FILE SYSTEMS

Name	Baseline	Ultra-wide
PRF latency	2 cycles	←
PRF capacity	int:128, fp:128	int:512, fp:512
PRF ports	12 ports	24 ports
RC latency	1 cycle	←
RC capacity	4, 8, 16, 32, 64 entries	16, 32, 64 entries
RC associativity	full	2-way
MRF latency	1 cycle	←
MRF capacity	int:128, fp:128	int:512, fp:512
MRF ports	read:1-3 write:1-3 ports	read:4 write:4 ports
write buffer	8 entries	←
use predictor	4 K entries, 4 way, 4 bits/pred, 2 bits/conf, 6 bits/tag, 6 bits/future ctl	←

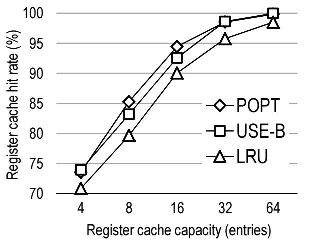
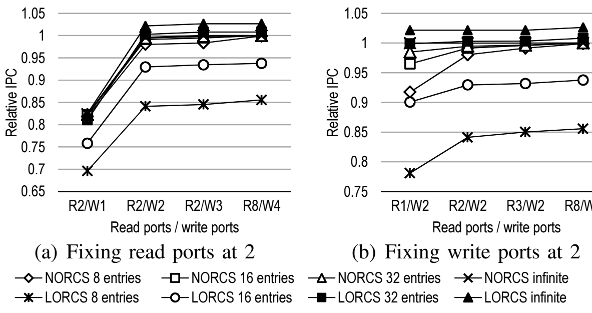


Figure 12. Register cache hit rate (LORCS)



(a) Fixing read ports at 2

(b) Fixing write ports at 2

Figure 13. Avg. relative IPC (fixing MRF ports)

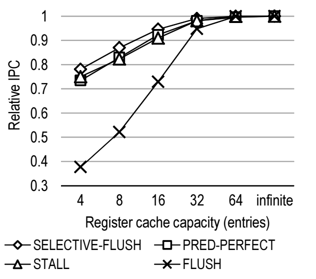


Figure 14. Avg. relative IPC (LORCS USE-B)

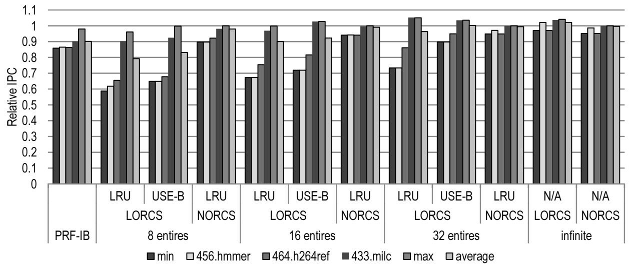


Figure 15. Average relative IPC

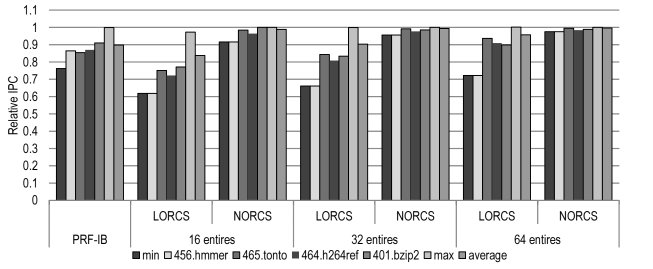


Figure 16. Average relative IPC of ultra-wide superscalar processor

results for models with for an 8-, 16-, 32- and "infinite"-entry register cache because models with a 4- and 64 entry register cache show the same tendency.

These graphs show that 2 read ports and 2 write ports in the main register file are sufficient to maintain the performance of the full-port main register file. Hereafter, we fix the number of the ports of the main register file at 2-read/2-write.

Behavior on a Register Cache Miss of LORCS: **Figure 14** shows the average relative IPCs of the models with different behaviors on a register cache miss, while changing the number of entries in each register cache. The IPCs are given relative to a model with "infinite" register caches.

As we explained in Section III, the graph shows that FLUSH models have the worst performance. On the other hand, the realistic STALL models have almost the same performance as the ideal SELECTIVE-FLUSH and PRED-PERFECT models. The ideal SELECTIVE-FLUSH model selectively flushes dependent instructions, however, a register cache miss penalty of each flushed instruction is much larger than that of STALL model, as described in Section III. Therefore, there are no significant differences between these 2 models.

Thus, hereafter, we fix the behavior on the register cache miss to STALL.

3) IPC: **Figure 15** shows the IPCs for each model relative to the baseline PRF model for an 8-, 16-, 32- and "infinite"- entry register cache. We only show the results for these models because models with a 4- and 64 entry register cache show the same tendency. We evaluated PRF-IB model, LORCS model with LRU and USE-B policy, and NORCS model with LRU policy. The "min", "max" and "average" bars are the minimum, maximum, and average of the programs in the benchmark, respectively. The other bars such as "456.hmmer" are specific programs that show the lowest and the highest performance.

The IPC degradation of NORCS models is considerably small; it is only 2.0%, even with an 8-entry register cache. Furthermore, the variations of the IPC of the various programs are also small. The performance of NORCS is not sensitive to the register cache hit rates (Section V-B).

The LORCS models with an "infinite"-entry register cache and of 32-entry register cache with USE-B policy show the performance improvement relative to the baseline model due to the shortened pipeline, although this is only 2.1% and 0.2%, respectively.

On the other hand, LORCS models with the other configurations show significant performance degradation. Even with USE-B policy, the average degradation is 16.9% and 7.3% for an 8- and 16-entry register cache, respectively. With LRU policy, the degradation is more marked — the average degradation is 20.8%, 10.0%, and 3.6% for an 8-, 16-, and 32-entry register cache, respectively.

In addition, the degradations of LORCS largely vary from program to program. In particular, the IPC degradation of LORCS with LRU for 456.hmmer is no less than 25% even with a 32-entry register cache.

This is because the performance of LORCS is more sensitive to the register cache hit rates (Section V-B). The register cache hit rates depends highly on the replacement policy, and largely varies from program to program. It is undesirable to show such low performances for specific programs even if the average performance is acceptable.

On the whole, in order to overachieve the PRF-IB model, only 8-entry register cache is necessary for NORCS, while 32-entry register cache and USE-B policy is necessary for LORCS. Moreover, these configurations are adequate

Table III
EFFECTIVE MISS RATE

	LORCS with 32 entry-RC (USE-B)					NORCS with 8 entry-RC (LRU)				
	Issued	Read	RC Hit(%)	Effc Miss(%)	IPC	Issued	Read	RC Hit(%)	Effc Miss(%)	IPC
429.mcf	0.44	0.53	92.2	3.4	0.99	0.49	0.60	82.83	0.98	1.00
456.hmmer	1.88	2.49	94.2	15.7	0.90	1.90	2.53	62.95	11.7	0.90
464.h264ref	1.91	2.57	99.0	8.8	0.95	1.87	2.51	73.14	8.7	0.92
average	1.48	1.90	98.6	2.7	1.00	1.46	1.89	79.91	2.3	0.98

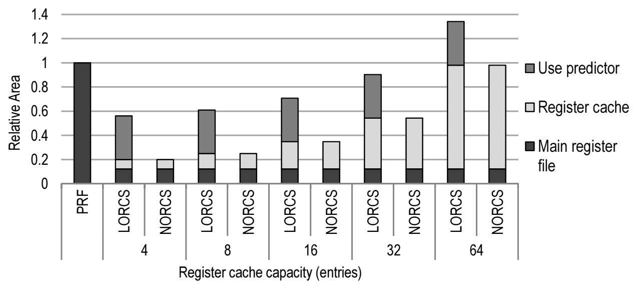


Figure 17. Relative areas

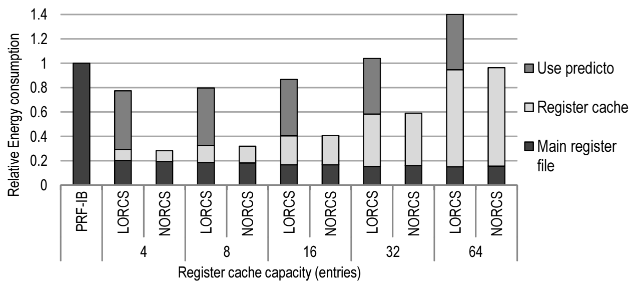


Figure 18. Relative energy consumption

because they achieve almost the same performance as the models with "infinite"-entry register cache.

4) Effective miss rate: **Table III** shows the effective miss rates, which is given by the probability of the pipeline disturbance. This table shows the results for LORCS with 32 entry USE-B register cache and NORCS with 8 entry LRU register cache. These 2 models show almost same performance in Section VI-B3. This table shows the results of characteristic benchmark programs and average of the all benchmark programs. "Issued " and "Read" are the number of issued instructions per cycle and the number of operands reading the register cache per cycle, respectively. "RC Hit" and "Effec Miss" are register cache hit rate per access and effective miss rate, respectively.

As described in Section V-B, effective miss rate in LORCS is much worse than the miss rate of the register cache itself. 464.h264ref shows that the register cache hit rate is as high as 99.0%, but an average number of operands that access the register cache a cycles is 2.57 and the effective miss rate is 8.8%. As a result, the IPC degradation for 464.h264ref is 5.1%.

As described in Section V-B, effective miss rate in NORCS is not sensitive to the register cache hit rates. the table show that the register cache hit rates for the NORCS model is much worse than those of the LORCS model, because register cache capacities of these models are different. However, the effective miss rates and IPC degradations are not much different from those of the LORCS model.

5) Circuit Area and Energy Consumption: We evaluated the area and the energy consumption of each model using CACTI 5.3 model [2]. We evaluated ITRS 45nm and 32nm technology nodes [19], but we only show the results for 32nm node because both show the same tendency.

Circuit Area: **Figure 17** shows the circuit areas relative to that of PRF model. The areas include those of the register cache, main register file and use-predictor.

The areas of the register cache and the main register file are the same in both NORCS and LORCS because they have the same number of entries and ports. The area of the main register file is reduced to 12.2%, because the number of the ports of the main register file is reduced from 12 to 4, and the area of a register file is proportional to the square of the number of the ports.

The area of use-predictor is 36.1% of that of the register file in PRF-IB model. The area of use-predictor is relatively large compared with that of the register cache and the main register file, because the use-predictor has many ports. Generally speaking, a predictor like the use-predictor requires the same number of read ports as the fetch width and the same number of write ports as the retirement width. The read ports are used to get predicted data on the frontend stages, while the write ports are used to write training data on the retirement stages [18]. In the evaluated configuration, the use-predictor requires 4-read/4-write ports.

In NORCS, the total area of the main register file and the register cache is reduced to 19.9%, 24.9%, 34.7%, 42.0%, and 98.0% of the register file of PRF model for a 4-, 8-, 16-, 32- and 64-entry register cache, respectively. In LORCS, the total area of the main register file and the register cache is the same as that of NORCS, but the total area including use-predictor is much larger than that of NORCS. The total areas including use-predictors are 56.1%, 61.0%, 70.8%, 90.3%, and 134.1% of the register file of PRF model for a 4-, 8-, 16-, 32-, and 64-entry register cache, respectively.

Energy Consumption: **Figure 18** shows the energy consumption of the register cache, main register file and use-predictor in each model relative to the register file in PRF model. The energy consumption is evaluated for each benchmark programs, and then averaged over all the programs.

The total energy consumption of the register cache and the main register file is reduced to about 28.2%, 31.9%, 40.6%, 59.0%, 96.3% for a 4-, 8-, 16-, 32-, and 64-entry register caches. In LORCS and NORCS, the total energy consumption of the register cache and the main register file

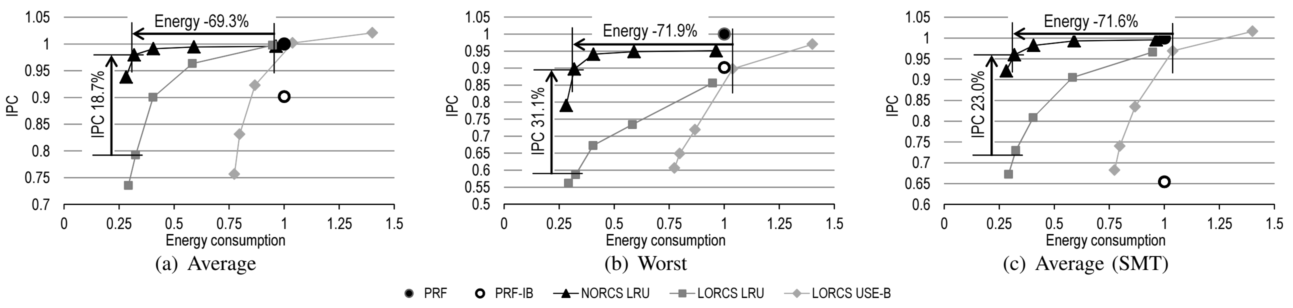


Figure 19. Trade-off between IPC and energy

differs slightly, because the number of accesses and the time to execute the programs differ slightly.

The energy consumption of the use-predictor is 48.1% of that of the register file in PRF-IB model. In LORCS, the total energy consumption of the register cache, main register file and use-predictor is 77.4%, 79.8%, 86.7%, 103.8%, and 140.1% for a 4-, 8-, 16-, 32- ,and 64-entry register caches.

6) Trade-off between IPC and Energy Consumption: **Figure 19(a)** shows the relative IPC per energy consumption in each model. Each IPC and energy consumption is relative to the register file in PRF model and is the same as those in Figures 15 and 18. Each data point of the line in this figure indicates a value of the model with a 4-, 8-, 16-, 32-, and 64- entries register cache from left to right.

The figure shows that NORCS reduces energy consumption without large performance degradation. On the other hand, in LORCS with LRU and USE-B, performance and energy consumption are trade-offs. LORCS with fewer entries of a register cache reduces energy consumption, but performance also degrades.

There are no significant difference of IPCs between NORCS with 8-entry LRU register cache and LORCS with 64-entry register cache of LRU policy, and the difference is 1.8%. However, the energy consumption of the model of NORCS is reduced by 69.3% from the model of LORCS. NORCS and LORCS with 8 entries LRU register cache show almost the same energy consumption. However, the model of NORCS improves IPC by 18.7% from the model of LORCS.

Figure 19(b) shows the relative IPC per energy consumption in the benchmark with the worst IPC in Figure 15. This figure shows the properties of NORCS and LORCS more clearly. NORCS with a 8-entry LRU register cache and LORCS with a 32-entry USE-B register cache show almost the same IPCs, but the former reduces the energy consumption by 71.9% from the latter. NORCS with the same configuration and LORCS with 8-entries LRU register cache show almost the same energy consumption, but the model of NORCS improves IPC by 31.1% from the model of LORCS.

C. Evaluation on Wide Superscalar Processor

Previous researchers have targeted ultra-wide superscalar processors with 8-wide and 512-entry register files [10]. In this subsection, we evaluated the register cache system on these ultra-wide superscalar processors.

The right columns of Tables I and II, labeled "Ultra-wide", show the configurations of an 8-wide superscalar processor, used as the baseline model in this section. This configuration is almost the same as that used in [10]. The models of the register cache systems are almost the same as those described before, but some parameters are changed to match the evaluation of Butts et al. Specifically, the number of the ports of the main register file is 4-read/4-write, and the register cache is 2-way set-associative that uses a unique indexing policy proposed by Butts et al. [10].

Figure 16 shows the IPCs for each model relative to the baseline PRF. The labels are the same as the ones in Figure 15. The results show a similar tendency to those in Section VI-B3. The IPC degradation of NORCS models is small — 0.12%, 0.6%, and 0.03% for a 16-, 32-, and 64-entry register cache, respectively. On the other hand, the LORCS show significant performance degradation — 16%, 9.7%, and 4.3% for a 16, 32, and 64-entry register cache, respectively.

Butts et al. evaluated the performance improvement of LORCS compared to PRF-IB models, and showed that a 64-entry register cache with USE-B policy outperforms PRF-IB [10]. Our evaluation results show that LORCS with a 64-entry register cache and the same parameters outperforms PRF-IB by 6.6%, and this closely matches their result of 6%. NORCS with a 16-entry register cache, on the other hand, outperforms PRF-IB by 10.1%.

These results show that NORCS also works well on ultra-wide superscalar processors. Specifically, NORCS with a 16-entry register cache with LRU policy outperforms LORCS with a 64-entry register cache with USE-B policy in this configuration.

D. Evaluation on SMT Processor

As described in Section I, SMT suffers from a need for a larger register file. We evaluate our proposal on a SMT with the same configuration given in Table I and a 2-way SMT feature. **Figure 19(c)** shows the relative IPC per energy consumption in each model. We executed 2 threads for all the combinations from 29 programs in SPEC CPU2006.

The IPC degradations in whole models are worse than those in single thread execution (Section VI-B3). However, the IPC degradation in NORCS models is still small; the average degradation is 4.1% and 1.8% for an 8- and 16-entry register cache. On the other hand, LORCS models show significant performance degradation. Even with USE-B policy, the average degradation is 26.0%, 16.5%, and 3.1% for an 8-, 16-, and 32-entry register cache.

There are no significant difference of IPCs between NORCS with 8-entry LRU register cache and LORCS with 32-entry register cache of USE-B policy, and the difference is 0.94%. However, the energy consumption of the NORCS model is reduced by 71.6% from the LORCS model. NORCS and LORCS with 8 entry LRU register cache show almost same energy consumption. However, the NORCS improves IPC by 23.0% from the LORCS.

VII. Conclusion

A register file is one of the most costly units in the recent superscalar processors. A large register file causes many problems as described in Section I. This paper introduces NORCS, a new technique of the register cache system that does not reduce the latency. NORCS is characterized by a unique pipeline that assumes miss of the register cache. We just add a set of pipeline latches to a conventional register cache system in order to add pipeline stages to read the main register file. This minimum modification causes a great difference in performance, because the pipeline of NORCS is not disturbed only on register cache misses.

We evaluated the IPC, area, and energy consumption of NORCS, LORCS, and pipelined register file on a practical 4-way and an ultra-wide 8-way superscalar processor. The evaluation results show the area and the energy consumption of NORCS is reduced to 24.9% and 31.9% of the pipelined register file, respectively, at a minimum cost of 2.1% IPC degradation. NORCS using the small 8-entry register cache with LRU policy achieves the same level performance as conventional LORCS using a 32-entry register cache with use-based policy, which shows very high hit rates that are as high as the ideal model, and in this configuration, the area and the energy consumption of NORCS is reduced to 27.6% and 31.9% of LORCS, respectively.

Acknowledgment

This research is partially supported by Grant-in-Aid for Scientific Research B #20300015 from Ministry of Education, Culture, Sports, Science and Technology Japan, Grant-in-Aid for JSPS Fellows #219259, from the Japan Society for the Promotion of Science, and Semiconductor Technology Academic Research Center (STARC) Japan.

References

[1] S. Rixner *et al.*, "Register organization for media processing," in *Proceedings of the International Symposium on High-Performance Computer Architecture*, 2000, pp. 375–386.

[2] S. Thoziyoor, N. Muralimanohar, J. Ahn, and N. Jouppi, "Cacti 5.1." HP Laboratories, Tech. Rep., 2008.

[3] S. Wijeratne *et al.*, "A 9GHz 65nm Intel Pentium 4 processor integer execution core," in *Proceedings of the Ineternational Solid-State Circuits Conference*, 2006, pp. 353–365.

[4] R. P. Preston *et al.*, "Design of an 8-wide superscalar RISC microprocessor with simultaneous multithreading," in *Proceedings of the Ineternational Solid-State Circuits Conference*, 2002, pp. 334–335.

[5] T. Monreal, A. Gonzalez, M. Valero, J. Gonzalez, and V. Vinals, "Delaying physical register allocation through virtual-physical registers," in *Proceedings of the International Symposium on Microarchitecture*, 1999, pp. 186–192.

[6] D. Balkan, J. Sharkey, D. Ponomarev, and A. Aggarwal, "Address-value decoupling for early register deallocation," in *Proceedings of the International Conference on Parallel Processing*, 2006, pp. 337–346.

[7] E. Sprangle and D. Carmean, "Increasing processor performance by implementing deeper pipelines," in *Proceedings of the International Symposium on Computer Architecture*, 2002, pp. 25–34.

[8] P. Ahuja, D. Clark, and A. Rogers, "The performance impact of incomplete bypassing in processor pipelines," in *Proceedings of the International Symposium on Microarchitecture*, 1995, pp. 36–45.

[9] J. Cruz, A. Gonzalez, M. Valero, and N. Topham, "Multiple-Banked Register File Architecture," in *Proceedings of the International Symposium on Computer Architecture*, 2000, pp. 316–325.

[10] J. A. Butts and G. S. Sohi, "Use-Based Register Caching with Decoupled Indexing," in *Proceedings of the International Symposium on Computer Architecture*, 2004, pp. 302–313.

[11] *The photograph is an excerpt from http://www.chip-architect.com/*, Chip Architect.

[12] R. Yung and N. C. Wilhelm, "Caching processor general registers," *Proceedings of the International Conference on Computer Design*, pp. 307–312, 1995.

[13] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, 1996.

[14] R. Kessler, "The alpha 21264 microprocessor," *IEEE micro*, vol. 19, no. 2, pp. 24–36, Mar/Apr 1999.

[15] *Processor Simulator Onikiri 2*. http://www.mtl.t.u-tokyo.ac.jp/~onikiri2/

[16] D. Burger and T. M. Austin, "The simplescalar tool set, version 2.0," University of Wisconsin-Madison, Computer Sciences Department, Tech. Rep., 1997.

[17] *SPEC CPU2006 suite*, The Standard Performance Evaluation Corporation. http://www.spec.org/cpu2006/

[18] J. A. Butts and G. S. Sohi, "Characterizing and predicting value degree of use," in *Proceedings of the International Symposium on Microarchitecture*, 2002, pp. 15–16.

[19] *International Technology Roadmap for Semiconductors http://www.itrs.net/*, Semiconductor Industries Association, 2005.