Electronic Preprint for Journal of Information Processing Vol.26

Regular Paper

Bank-Aware Instruction Scheduler for a Multibanked Register File

Junji Yamada[1,†1,a)] Ushio Jimbo[2] Ryota Shioya[3,†2] Masahiro Goshima[4] Shuichi Sakai[1]

Received: December 25, 2017, Accepted: March 9, 2018

Abstract: The region that includes the register file is a hot spot in high-performance cores that limits the clock frequency. Although multibanking drastically reduces the area and energy consumption of the register files of superscalar processor cores, it suffers from low IPC due to bank conflicts. This paper proposes a bank-aware instruction scheduler which selects instructions so that no bank conflict occurs, except for a bank conflict in one instruction. The evaluation results show that, compared with NORCS, which is the latest research on a register file for area and energy efficiency, a proposed register file with 24 banks achieves a 20.9% and 56.0% reduction in circuit area and in energy consumption, while maintaining a relative IPC of 97.0%, and the latency of the instruction scheduler.

Keywords: superscalar processor, instruction scheduler, select logic, register file, multibanking

1. Introduction

Recently, 8-issue cores, such as the IBM POWER8, and Intel Haswell and Skylake, have come onto the market [1], [2], [3], [4]. Such wide cores, however, suffer from increased area and energy consumption of the register file. Wide cores require a large number of registers proportional to the number of in-flight instructions. Besides, the circuit area of a register file composed of a RAM is proportional to the square of the number of its ports [5], [6], [7].

Figure 1 shows a die photograph of the AMD Bulldozer processor, which is one of the most documented processors among recent ones [8]. 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 16 KB 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.

A **register cache** is a drastic method of reducing the register file ports [11], [12], [13], [14]. Compared with the original register file, the *main* register file is smaller because it needs only a few ports (Section 2.1).

Multibanking is the *ultimate* method in the sense that it can reduce the effective number of ports to one. 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 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. In this case, 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 naive intuition, because the pipeline is disturbed when *any* of the banks causes a bank conflict.

This paper shows **bank-aware instruction scheduler** design for a multibanked register file, which selects and issues instructions so that no bank conflict occurs in the multibanked register file. Although the idea of bank-aware scheduling itself is not new, no feasible implementation has been presented. Balasubramonian et al. briefly mentioned the idea of bank-aware scheduling while presenting their main proposal [15]; however, they did not show how to implement it. Tseng et al. even rejected bank-aware scheduling because the latency of the scheduling logic will be unacceptably increased [16].

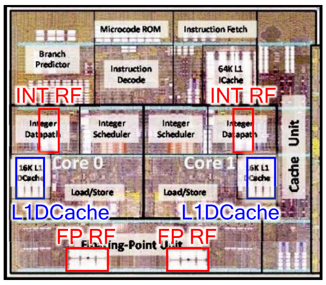


Fig. 1 Bulldozer 2-Core Processor (Ref. [8], Fig. 4.5.7). The boxes and texts for the register files (RF) and the level-1 data caches (L1DCache) are added by the authors on the basis of the articles [8], [9], [10].

[1] The University of Tokyo, Bunkyo, Tokyo 113–8656, Japan
[2] SOKENDAI (The Graduate University for Advanced Studies), Chiyoda, Tokyo 101–8430, Japan
[3] Nagoya University, Nagoya, Aichi 464–8603, Japan
[4] National Institute of Informatics, Chiyoda, Tokyo 101–8430, Japan
[†1] Presently with Toshiba Memory Corporation
[†2] Presently with the University of Tokyo
[a)] yamadaju@mtl.t.u-tokyo.ac.jp

Therefore, the main purpose of this paper is to show feasible design of a bank-aware scheduling logic. The rest of this paper is organized as follows: Section 2 introduces existing techniques including a *plain* multibanked register file. Then, Section 3 details our design. Sections 4 and 5 evaluate the area and energy efficiency of these systems. Surprisingly, the results show that our design overcomes the latest proposal introduced in Section 2.

2. Existing Techniques

This section introduces existing techniques. Section 2.1 introduces a register cache system as the latest proposal on a register file for area and energy efficiency, which is compared with our proposal in Sections 4 and 5. Section 2.2 shows a multibanked register file before going into our proposal in Section 3.

2.1 NORCS [12], [13]

A **register cache** can also reduce the register file area and energy consumption by reducing the number of ports [12]. 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 fewer ports.

However, conventional register cache systems suffer from low IPC due to register cache misses. The backend pipeline is stalled when *any* of the register accesses in a cycle causes 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**) [12], which is the latest proposal on the register file for area and energy efficiency, and researchers in NVIDIA adopted this idea for their GPUs [13].

As shown in **Fig. 2** (middle), NORCS has almost the same structure as conventional register cache systems. The main difference is their pipelined behavior as below.

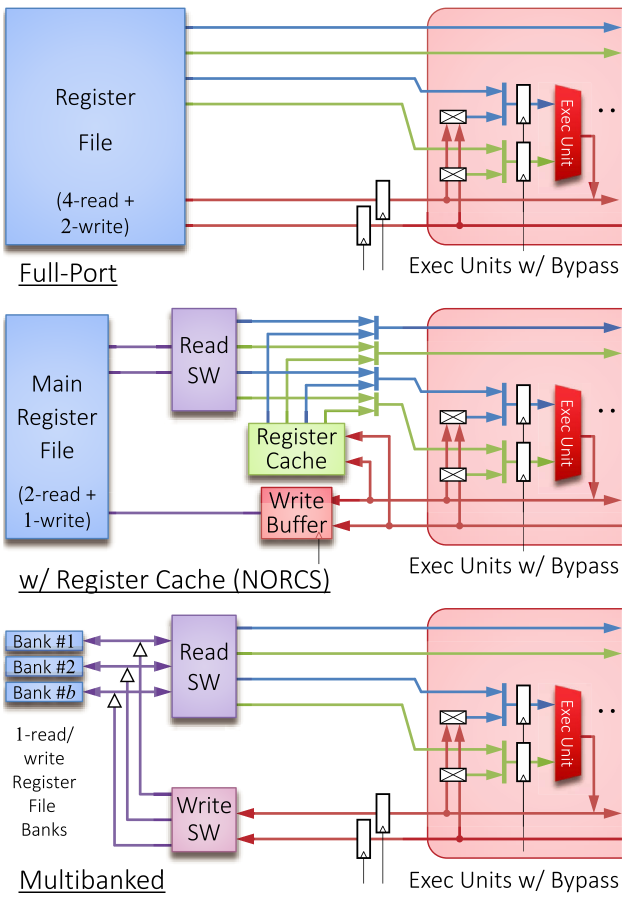
The pipeline of a conventional register cache system does not have a stage for reading the main register file, in the same manner that usual pipelines have stages to read L1D but not the main memory. Conversely, the NORCS pipeline provides dedicated stages to read the main register file, and all the instructions pass through these stages whether they hit or miss the register cache.

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 a 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 $0.05^3 = 0.0125\%$.

2.2 Multibanked Register File

Multibanking is a technique typically used for the main memory of vector processors; however, there is no *standard* implementation of a multibanked register file. Therefore, this subsection shows a typical *plain* multibanked register file before going into our proposal in Section 3.

Figure 2 (lower) shows the datapath of a multibanked register file. **Figure 3** adds the control to the left half.



Register File (4-read + 2-write)
Exec Unit
Exec Units w/ Bypass
Full-Port
Main Register File (2-read + 1-write)
Read SW
Register Cache
Write Buffer
w/ Register Cache (NORCS)
Bank #1
Bank #2
Bank #b
1-read/write Register File Banks
Write SW
Multibanked

Fig. 2 Datapaths of full-port, register cache, and multibanked systems.

A multibanked register file has read and write switches for any-to-any routing between the execution units and banks. As described in Section 1, the banks are sufficiently smaller than the original full-port register file in area.

These switches are also sufficiently small. We give an intuitive explanation on the circuit size of the switches before quantitative evaluation in Section 5.

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 a hundred entries.

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.

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.

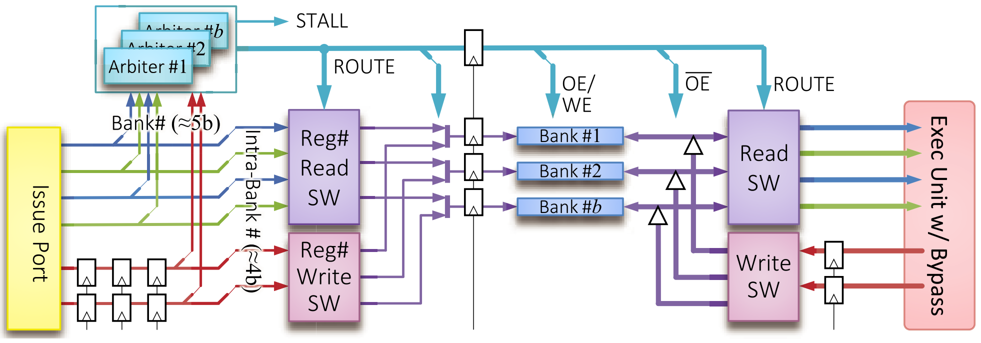


Fig. 3 Control and datapath of multibanked systems.

The system consists of 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.

The arbiters and the register number switches are even smaller than the 64-bit datapath described above.

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.

As shown by the pipeline registers in the middle of Fig. 3, one cycle is assigned to the arbitration and register number routing.

3. Bank-aware Scheduler

This section details our **bank-aware scheduler** design. As mentioned in Section 1, prior studies briefly mentioned the idea of bank-aware scheduling, or even rejected it because of the increased latency [15], [16]. However, our detailed design clarifies that the latency of the logic is not practically increased.

We found the following three points:

(1) Accesses that obtain their operands from the 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 this type of a bank conflict requires an additional cycle to be solved.
(3) However, some of them are caused by the two accesses to the same register value to calculate the square or double of the value, and can be excluded.

3.1 Structure

Figure 4 shows the proposed select logic that selects three instructions to a 24-bank register file.

Issue Port Arbiters

The upper half of the figure represents conventional select logic composed of *cascaded* three arbiters, each of them selects at most one from at most W requests, where W is the instruction window size [5]. These arbiters work *in series* by withdrawing the requests to the next arbiter when granted. They produce gp[i][p] for

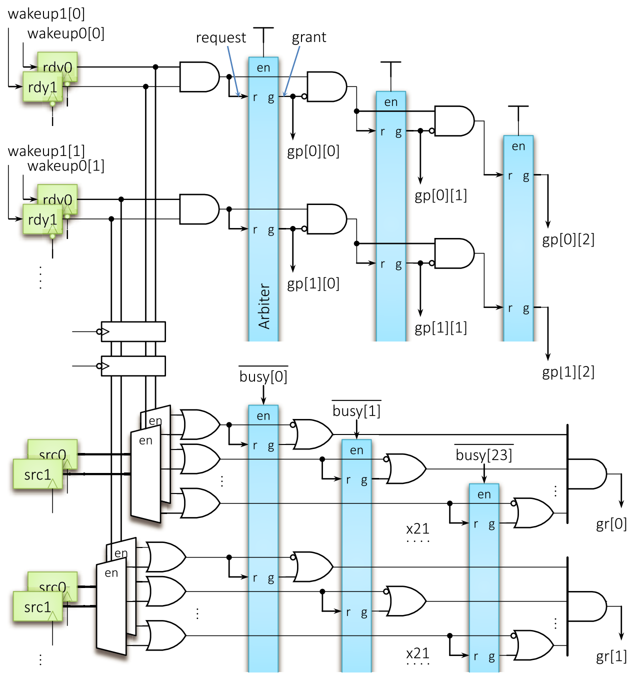


Fig. 4 Proposed select logic (3 insts to 24 banks).

the i-th instruction to be issued from the p-th issuing port.

Read Arbiters

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. We 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 [5]. It is unrealistic to add read ports to the CAM for reading the bank numbers of all of the ready instructions for arbitration. Thus, the point of the proposed logic is to change these fields into discrete registers so that the bank numbers can be provided to the arbiters without adding read ports to the CAM.

Because the scheduler has the bank numbers as its fields, there are several design options to reuse these fields also for the arbiters. In the evaluation in Section 5, we added dedicated registers for the arbiters to evaluate our proposal independently with the scheduler design.

Write Arbiters

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.

More precisely, an instruction requests not the register file banks but the *timeslots* of the banks, i.e., the banks in the cycles when the instruction reads and writes the target banks. Thus, the logic has the separate read and write arbiters in order to request the banks in different cycles between read and write.

As shown in the figure, the read arbiters are disabled by the busy signals when the bank is used for the write of an instruction issued several cycles before.

Three Types of Arbiters

Finally, the i-th instruction is selected to be issued from the p-th port when

$$\mathrm{gp}[i][p]\ \&\&\ (\mathrm{gr}[i]\ \&\&\ \mathrm{gw}[i]) == 1. \qquad (1)$$

3.2 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 Fig. 4. 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 correspond to the first && operator in Eq. (1).

3.3 Unified Scheduler

As mentioned above, an instruction requests the timeslots of the banks. Thus, a unified scheduler, which holds several types of instructions with different latencies, requires several sets of write arbiters for the different cycles when the instructions write to the banks.

Here it should be noted that, in general, instructions issued from the same issue port are designed to write to the register file in the same cycles after they have been issued, in order to realize pipelined behavior without resource conflict, where the execution resources include the register file read ports, the execution units, and the register file write ports. For example, if two instructions that write to the register file in the l-th and $(l-1)$-th cycles after they have been issued are issued in consecutive cycles, they will cause a conflict on the register file write port. Insertion of a bubble to avoid this conflict not only requires extra logic but also degrades the IPC. This is also true for scheduling with level-1 cache hit/miss prediction [17].

Table 1 summarizes the latencies of the instruction types in this sense for the unified scheduler evaluated in Section 4. This scheduler requires, (in addition to two sets of read arbiters for the integer and floating-point register files) two sets of write arbiters for the integer register file (1-cycle latency integer, and 4-cycle latency load instructions), and one set of write arbiters for the floating-point register file (4-cycle latency floating-point and floating-point load instructions).

Table 1 Latencies of instructions in Sections 4 and 5.

inst type / destination	int	int load	fp load	fp
int RF	1	4	▨	▨
fp RF	▨	▨	4	

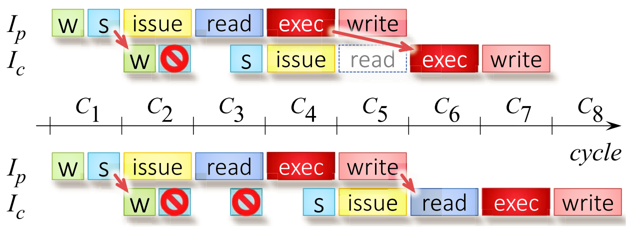


Fig. 5 Scheduling w/(upper) and w/o (lower) delayed request, where the stages are W: wakeup, S: select, exec: execution, and read/write: register read/write.

3.4 Bypass-aware Scheduling

Figure 5 shows the pipelined behavior 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 half of the figure, the issue of I_c is delayed 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 half of the figure.

To solve this problem for bank-aware instruction scheduler, the following logical trick is introduced.

In Fig. 4, 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 delays the requests for the read arbiters for two cycles after wakeup.

In the case of Fig. 5, I_c does not request the bank in the cycles C_2 and C_3. This is the same situation as I_c does not have the source operands. As a result, I_c is selected in C_3 as shown in the upper half of the figure.

3.5 Inter- and Intra-Instruction Conflict

Bank conflicts can be categorized into inter- and intra-instruction ones. An **inter-instruction conflict** occurs between two (or more) source operands of two (or more) different (or more) instructions, while an **intra-instruction conflict** occurs between the two source operands of a single instruction. The bank-aware scheduling can solve inter- but not intra-instruction conflicts. On an intra-instruction conflict, the backend pipeline must be stalled to make a cycle to read the second operand.

We should note that it is difficult to solve intra-instruction conflicts with register renaming. Physical registers have already been allocated to the destination operands of the dependent instructions, and this mapping cannot be changed for the convenience of the source operands of the instruction to be scheduled.

Table 2 Types of conflicts and solutions.

	Type		Solution
	Instruction	Operands	
🟦	Inter-	—	Scheduling
🟨	Intra-	Different	Stalling
🟧		Identical	Stalling (or Duplication)

3.6 Intra-Instruction Conflict for Identical Operand

In addition, intra-instruction conflicts are categorized into two cases. When the two source operands of a single instruction are different, an intra-instruction conflict occurs with a probability. On the contrary, when the two source operands are identical, an intra-instruction conflict occurs with probability 1.

A single instruction with identical source operands is sometimes used for an optimization technique known as *strength reduction*. For example, we found that gromacs in SPEC 2006 have a number of floating-point instructions to calculate $x + x = 2x$ and $x \times x = x^2$.

When the source operands are identical, the bank conflict can be avoided not by the scheduling or stalling but by **duplication**, i.e., the bank is read only once and the read value is duplicated in the read switch.

Table 2 summarizes the types of conflicts and the solutions for them.

However, the evaluation results show that the opportunity for strength reduction is rare and this duplication improves the averaged relative IPC of 29 programs in SPEC 2006 only by 0.35%.

Thus, we did not adopt this duplication in the evaluation in the next section. In this case, the pipeline is stalled if a bank conflict occurs in a single instruction regardless of whether the source operands are identical or different.

4. Evaluation of IPC

This section shows evaluation results on IPC with a processor simulator.

4.1 Evaluation Environment

We used the whole set of the SPEC CPU 2006 benchmark, including 29 programs, with the *ref* data sets [18]. The programs were compiled with gcc 4.2.2 –O3. We evaluated the 1G instructions after the first 10G instructions.

We used the Onikiri 2 [19] simulator, which was also used to evaluate NORCS [12]. This simulator is completely 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.

4.2 Evaluated Models

Table 3 shows the evaluated models of their default configurations. We chose as the default the minimum configurations with which Proposal and NORCS show average relative IPC of more than 0.96.

Table 3 Evaluated models and their default configurations.

Name	Technique	Ports	RF read stages		
(baseline)	Replicated Pair	5/4r+5w (int/fp)	3		RF:3
🟩 NORCS	NORCS	3r+3w (MRF)	2	RC:1	MRF:1
🟦 Proposal	Proposal	1rw (bank)	1		RF:1
🟧 Plain	Plain Multibank		2	a/r:1	

RF/RC: Register File/Cache, MRF: Main RF, a/r: arbitration/register number routing

Table 4 Configuration of baseline model.

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

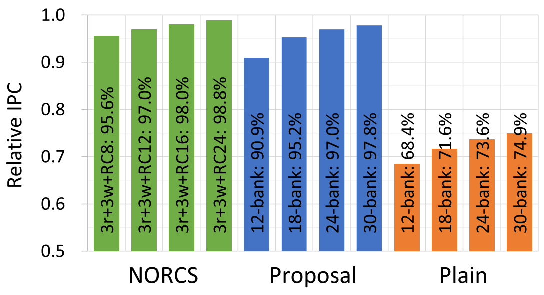


Fig. 6 Averaged relative IPC of models.

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 Section 1 [8].

Table 4 gives its configuration, which follows modern 8-issue cores such as the IBM POWER 8, and Intel Haswell and Skylake processors [2], [4].

As described in Section 2.2, we assumed that the arbitration and register number routing of Plain model take one cycle (denoted as "a/r: 1" in Table 3).

Unfortunately, the register file latency is not documented for recent cores [17]. We assumed that the latency of the baseline model is 3 cycles and those of the other models are reduced to 2 or 1 as shown in Table 3. It should be noted that, this difference of one cycle has less significant effect on the IPC of recent cores with highly accurate predictors than bank conflicts. In this evaluation, the average IPC decreased by 1.4% because of one-cycle increase in latency.

4.3 Relative IPC

Figure 6 shows 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, and for NORCS with a 8 to 24-entry register cache and a 3-read + 3-write main register file. Regarding NORCS, we 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.

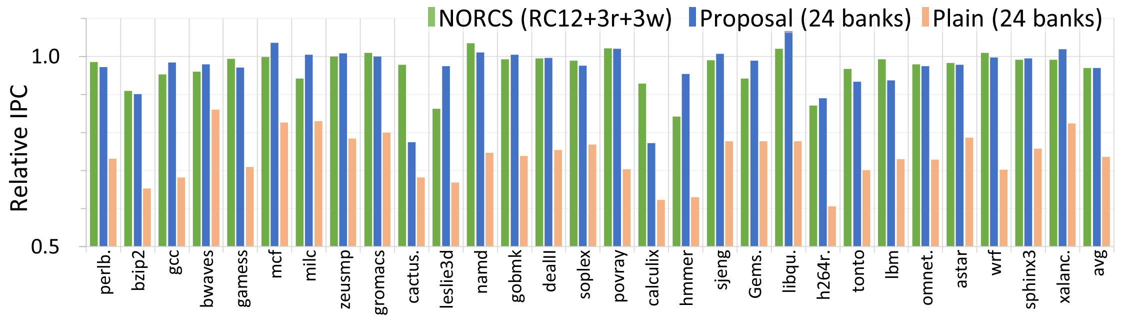


Fig. 7 Relative IPC of models with default configurations (Table 3) for SPEC CPU 2006.

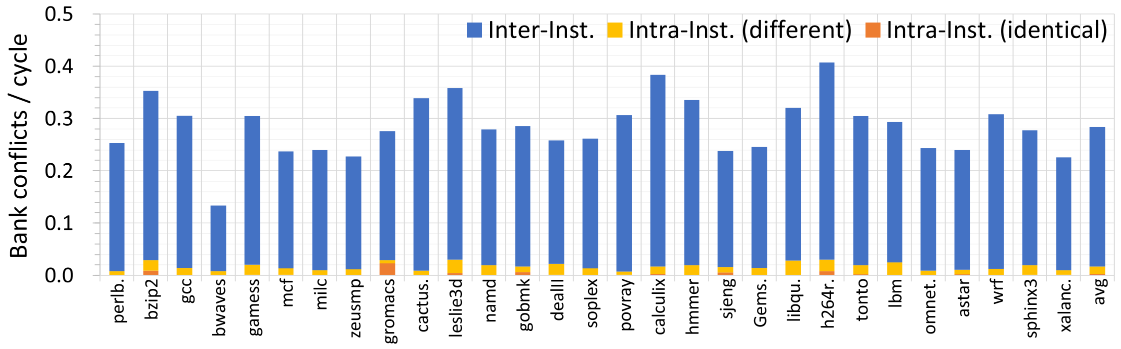


Fig. 8 Bank conflicts per cycle for 24 banks for SPEC CPU 2006.

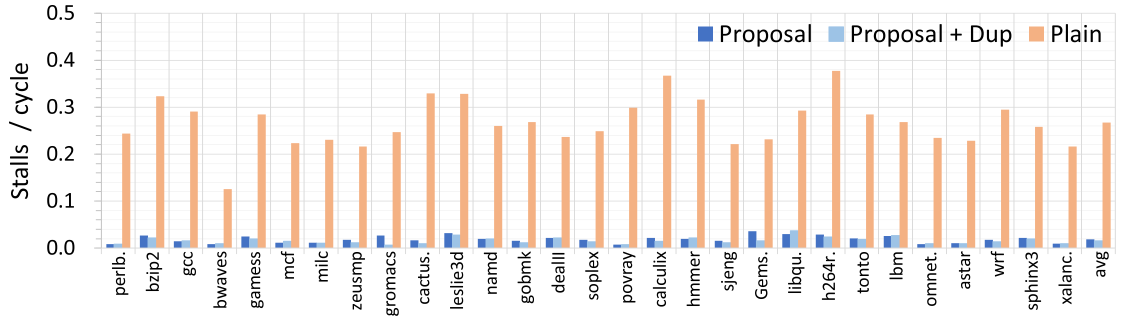


Fig. 9 Bank-conflict-induced stalls per cycle for 24 banks for SPEC CPU 2006.

We evaluated the number of banks in multiples of 6 based on the layout constraint derived in Section 5.2. While Plain cannot achieve sufficient IPC even with 30 banks, Proposal achieves a relative IPC of as high as 97.0% with 24 banks.

Figure 7 shows the relative IPC of the models with the default configurations shown in Table 3 for all the 29 programs in SPEC CPU 2006. We chose the default configurations so that Proposal and NORCS show average relative IPC of more than 0.96. However, most of them show the relative IPC of as low as 0.9.

4.4 Bank Conflicts

Figure 8 shows the number of *potential* bank conflicts per cycle for 24 banks for all the 29 programs in SPEC CPU 2006. This data was retrieved with the baseline model with full-port register files. Though the model is actually free from bank conflicts, we counted *potential* bank conflicts observing the fields corresponding to the bank number in the accessed register numbers.

The bank conflicts are categorized into inter- and intra-instruction ones as shown in Table 2. For the average of 29 programs, the number of inter-, intra- (different operands) and intra- (identical operands) instruction bank conflicts per cycle are 0.267, 0.015 and 0.002, respectively.

Among the three types of bank conflicts, the inter-instruction conflicts (0.267) and the intra-instruction conflicts with the identical operands (0.002) can be eliminated by the bank-aware scheduling and the operand duplication, respectively; however, the latter is considerably rare.

4.5 Pipeline Stalls Caused by Bank Conflicts

In contrast to Fig. 8, **Fig. 9** shows the number of *actual* pipeline stalls caused by bank conflicts per cycle for 24 banks for all the 29 programs in SPEC CPU 2006. Unlike Fig. 8, this data was retrieved with the Plain and Proposal models.

The average number of stalls per cycle is reduced from 0.267 (Plain) to 0.019 (Proposal). The difference between them shows strong correlation with the average number of inter-instruction bank conflicts per cycle (0.267) shown in Fig. 8, which is eliminated by bank-aware scheduling.

Figure 9 also shows the effect of the identical operand duplication. The difference in the average number of stalls per cycle between without (Proposal) and with the duplication (Proposal+Dup) is only 0.003. Thus, we do not recommend the duplication of identical operands as described in Section 3.6.

5. Evaluation of Area and Energy

This section shows evaluation results on the area and energy based using a process design kit.

5.1 Evaluation Methodology

We used FreePDK15, a predictive process design kit for 15 nm FinFET technology [20], and NanGate Open Cell Library [21].

Because this library does not include RAMs or switches, we used CACTI [6], [7], [22] with minor modifications to evaluate them. 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. We adjusted the scale of the RAMs and switches using the formula of CACTI, and the standard cells of FreePDK15, by their minimum pitch of wires.

Because the areas of small cells strongly depend on the designers' efforts, we investigated recent researches on small-port memory cells [23], [24], and verified that the values are quite consistent.

We described the entire systems of Plain and Proposal in System Verilog. Then, we 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. The RAMs and switches are treated as large cells which have parameters estimated with CACTI.

5.2 Layout

Figure 10 shows the place-and-route results of the Plain and Proposal integer register files and the bank arbiters of Proposal. This figure also shows the shapes of the datapaths of the baseline and NORCS integer register files for reference.

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

In this 8-bit-slice, 6 register file banks are arranged. This is the reason why the number of banks of the multibanked models is the multiple of 6. We 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.

The heights (the horizontal direction in these figures) 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 4 banks, and a read and a write switch.

The most part of the control circuit is pipeline latches and switches in Proposal. In contrast, the control circuit of Plain also contains the arbiters of the banks. Proposal has these arbiters in its instruction scheduler instead. The arbiters of Proposal is larger than that of Plain, because the number of requests of the bank arbiters is 64 and 15 for Proposal and Plain, respectively. The number of requests of Proposal equals the size of the instruction window, and that of Plain equals the 10-read+5-write of register file.

5.3 Area and Energy Consumption

Figure 11 shows the relative area and energy consumption of the integer and floating-point register files. The Plain and Proposal areas include dead spaces produced by layout constraint. The energy is calculated using the access count produced by the simulation in Section 4.

Area

The areas of the multibanked models are considerably smaller than those of the other models with the default configurations. As

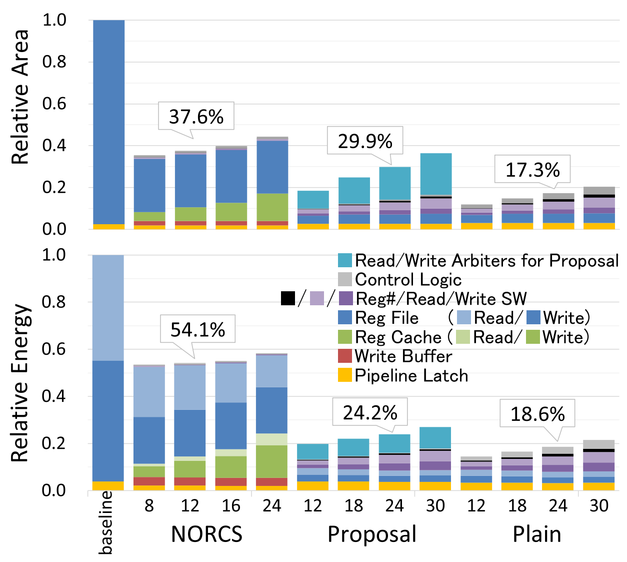


Fig. 11 Relative area (upper) and energy consumption (lower).

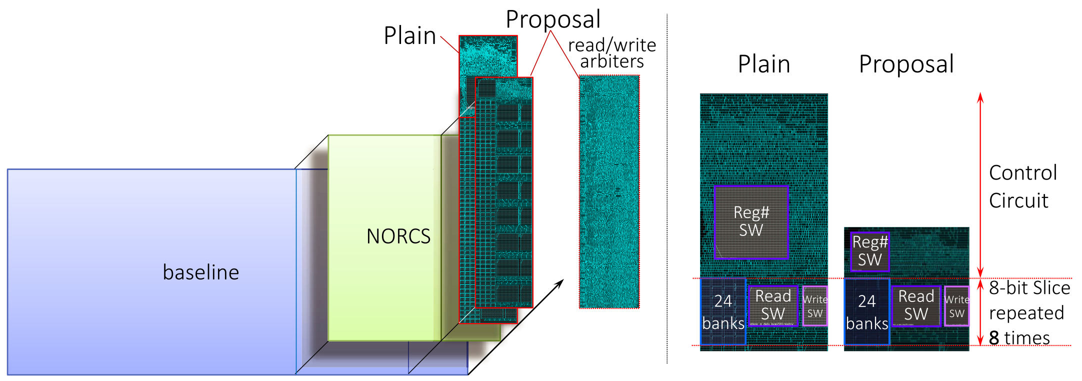


Fig. 10 Layout of integer register files of each model.

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.

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

Energy Consumption

As shown in Fig. 11, the result of energy consumption is basically proportional to 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.

Read/Write Arbiters

As shown in Fig. 10, Proposal has the read/write arbiters in its instruction scheduler. In Proposal with 24 banks, 52.5% and 32.9% of the area and energy consumption come from the read/write arbiters, respectively.

In this area and energy consumption, only 0.543% and 1.31% come from the additional bank number registers described in Section 3.1. The latter percentage is larger than the former mainly because the switching rate of the registers is higher than that of the logic.

5.4 Scheduler Latency

We evaluated the critical path of the instruction scheduler. We applied 250 ps as a constraint of logic synthesis for 2 GHz operation. Because wakeup and select should be performed in a single cycle, half cycle is assigned for select logic. As a result of the logic synthesis, the latencies of the conventional scheduler and the bank arbiter of our proposal were 216 ps and 202 ps, respectively. Thus, the bank arbiter of our proposal is not the critical path of the scheduler.

5.5 Area and Energy Efficiency

The graphs in **Fig. 12** show the relative IPC with respect to the relative area and energy consumption. The graphs are simply derived from the graphs in Figs. 6 and 11 to show the trade-off between IPC and area, and between IPC and energy consumption. For techniques 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.

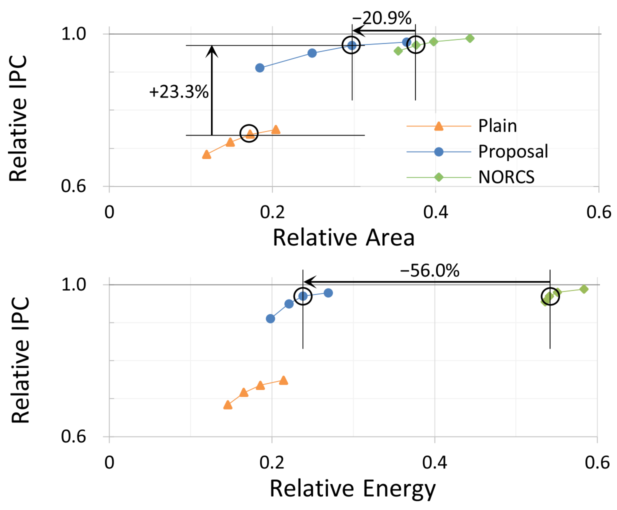


Fig. 12 Relative IPC vs. relative area (upper) and energy consumption (lower).

In each of the graphs, the points for Proposal and NORCS with their default configurations (denoted by circles) are located within the region where the average relative IPC is more than 0.96, from left to right in this order, which proves that Proposal reduces more area and energy than NORCS while keeping the same level of IPC. Compared with NORCS, Proposal achieves a 20.9% and 56.0% reduction in area and energy consumption, respectively.

6. Conclusion

The region including the register file is a hot spot of a processor core that limits the clock frequency and the scale of the core. Although a multibanked register file drastically reduces its area and energy consumption to mitigate the hot spot problem, conventional implementations suffer from low IPC because of bank conflicts. Bank-aware scheduler schedules instructions not to cause bank conflicts.

Although the prior studies considered that the bank-aware scheduling is unrealistic because of increased latency, our design showed that it is not true. The evaluation results show that, from NORCS [12], which is the latest architecture to reduce the area and energy consumption of a register file, the proposed system achieves a 20.9% and 56.0% reduction in area and energy consumption, respectively.

Acknowledgments This work is supported by JSPS KAKENHI Grant Number 16H05855, 16H02797 and 26280012. This work is supported by VLSI Design and Education Center (VDEC), the University of Tokyo in collaboration with Cadence Design Systems, Inc.

References

[1] Sinharoy, B., Kalla, R., Starke, W.J., Le, H.Q., Cargnoni, R., Norstrand, J.A.V., Ronchetti, B.J., Stuecheli, J., Leenstra, J., Guthrie, G.L., Nguyen, D.Q., Blaner, B., Marino, C.F., Retter, E. and Williams, P.: IBM POWER7 multicore server processor, *IBM Journal of Research and Development*, Vol.55, No.3, pp.1:1–1:29 (online), DOI: 10.1147/ JRD.2011.2127330 (2011).

[2] Sinharoy, B., Norstrand, J.A.V., Eickemeyer, R.J., Le, H.Q., Leenstra, J., Nguyen, D.Q., Konigsburg, B., Ward, K., Brown, M.D., Moreira, J.E., Levitan, D., Tung, S., Hrusecky, D., Bishop, J.W., Gschwind, M., Boersma, M., Kroener, M., Kaltenbach, M., Karkhanis, T. and Fernsler, K.M.: IBM POWER8 processor core microarchitecture, *IBM Journal of Research and Development*, Vol.59, No.1, pp.2:1–2:21 (online), DOI: 10.1147/JRD.2014.2376112 (2015).

[3] Krewell, K.: Intel's Haswell Cuts Core Power, *Microprocessor Report* (2012).

[4] Fayneh, E., Yuffe, M., Knoll, E., Zelikson, M., Abozaed, M., Talker, Y., Shmuely, Z. and Rahme, S.A.: 14nm 6th-generation Core processor SoC with low power consumption and improved performance, *IEEE International Solid-State Circuits Conference on (ISSCC 2016), Digest of Technical Papers*, pp.72–73 (online), DOI: 10.1109/ISSCC.2016.7417912 (2016).

[5] Shen, J.P. and Lipasti, M.H.: *Modern Processor Design: Fundamentals of Superscalar Processors*, Waveland Press, Inc. (2013).

[6] Thoziyoor, S., Muralimanohar, N., Ahn, J.H. and Jouppi, N.P.: CACTI 5.1., Technical Report HPL-2008-20, HP Laboratories (2008).

[7] Muralimanohar, N., Balasubramonian, R. and Jouppi, N.P.: CACTI 6.0: A tool to model large caches, Technical Report HPL-2009-85, HP Laboratories (2009).

[8] Fischer, T., Arekapudi, S., Busta, E., Dietz, C., Golden, M., Hilker, S., Horiuchi, A., Hurd, K.A., Johnson, D., McIntyre, H., Naffziger, S., Vinh, J., White, J. and Wilcox, K.: Design solutions for the Bulldozer 32nm SOI 2-core processor module in an 8-core CPU, *IEEE International Solid-State Circuits Conference on (ISSCC 2011), Digest of Technical Papers*, pp.78–80 (online), DOI: 10.1109/ISSCC.2011.5746227 (2011).

[9] Golden, M., Arekapudi, S. and Vinh, J.: 40-Entry unified out-of-order scheduler and integer execution unit for the AMD Bulldozer x86-64 core, *IEEE International Solid-State Circuits Conference on (ISSCC 2011), Digest of Technical Papers*, pp.80–82 (online), DOI: 10.1109/ISSCC.2011.5746228 (2011).

[10] McIntyre, H., Arekapudi, S., Busta, E., Fischer, T., Golden, M., Horiuchi, A., Meneghini, T., Naffziger, S. and Vinh, J.: 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 (online), DOI: 10.1109/JSSC.2011.2167823 (2012).

[11] Butts, J.A. and Sohi, G.S.: Use-Based Register Caching with Decoupled Indexing, *Proc. 31st Annual International Symposium on Computer Architecture* (*ISCA*), pp.302–313 (online), DOI: 10.1109/ISCA.2004.1310783 (2004).

[12] Shioya, R., Horio, K., Goshima, M. and Sakai, S.: Register Cache System Not for Latency Reduction Purpose, *Proc. 43rd Annual International Symposium on Microarchitecture* (*MICRO*), pp.301–312 (online), DOI: 10.1109/MICRO.2010.43 (2010).

[13] Gebhart, M., Johnson, D.R., Tarjan, D., Keckler, S.W., Dally, W.J., Lindholm, E. and Skadron, K.: Energy-efficient Mechanisms for Managing Thread Context in Throughput Processors, *Proc. 38th Annual International Symposium on Computer Architecture* (*ISCA*), pp.235–246 (online), DOI: 10.1145/2000064.2000093 (2011).

[14] Gebhart, M., Johnson, D.R., Tarjan, D., Keckler, S.W., Dally, W.J., Lindholm, E. and Skadron, K.: A Hierarchical Thread Scheduler and Register File for Energy-Efficient Throughput Processors, *ACM Trans. Computer Systems*, Vol.30, No.2, pp.8:1–8:38 (online), DOI: 10.1145/2166879.2166882 (2012).

[15] Balasubramonian, R., Dwarkadas, S. and Albonesi, D.H.: Reducing the complexity of the register file in dynamic superscalar processors, *Proc. 34th Annual International Symposium on Microarchitecture* (*MICRO*), pp.237–248 (online), DOI: 10.1109/MICRO.2001.991122 (2001).

[16] Tseng, J.H. and Asanović, K.: Banked multiported register files for high-frequency superscalar microprocessors, *Proc. 30th Annual International Symposium on Computer Architecture* (*ISCA*), pp.62–71 (online), DOI: 10.1109/ISCA.2003.1206989 (2003).

[17] Perais, A., Seznec, A., Michaud, P., Sembranty, A. and Hagersten, E.: Cost-Effective Speculative Scheduling in High Performance Processors, *Proc. 42nd Annual International Symposium on Computer Architecture* (*ISCA*), pp.247–259 (online), DOI: 10.1145/2749469.2749470 (2015).

[18] SPEC CPU 2006 (online), available from ⟨http://www.spec.org/cpu2006/⟩.

[19] Onikiri 2 (online), available from ⟨https://github.com/onikiri/onikiri2/⟩.

[20] Bhanushali, K. and Davis, W.R.: FreePDK15: An Open-Source Predictive Process Design Kit for 15nm FinFET Technology, *Proc. International Symposium on Physical Design* (*ISPD*), pp.165–170 (online), DOI: 10.1145/2717764.2717782 (2015).

[21] Martins, M., Matos, J.M., Ribas, R.P., Reis, A., Schlinker, G., Rech, L. and Michelsen, J.: Open Cell Library in 15nm FreePDK Technology, *Proc. International Symposium on Physical Design* (*ISPD*), pp.171–178 (online), DOI: 10.1145/2717764.2717783 (2015).

[22] Li, S., Ahn, J.H., Strong, R.D., Brockman, J.B., Tullsen, D.M. and Jouppi, N.P.: The McPAT Framework for Multicore and Manycore Architectures: Simultaneously Modeling Power, Area, and Timing, *ACM Trans. Architecture and Code Optimization*, Vol.10, No.1, pp.5:1–5:29 (online), DOI: 10.1145/2445572.2445577 (2013).

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

[24] Yabuuchi, M., Fujiwara, H., Tsukamoto, Y., Tanaka, M., Tanaka, S. and Nii, K.: A 28nm high density 1R/1W 8T-SRAM macro with screening circuitry against read disturb failure, *Proc. IEEE Custom Integrated Circuits Conference* (*CICC*), pp.1–4 (online), DOI: 10.1109/CICC.2013.6658451 (2013).

Junji Yamada was born in 1980. He received his B.E from Shinshu University in 2010. He received his M.E. and Ph.D. degrees in Information and Communication Engineering from the University of Tokyo in 2012 and 2017, respectively. From 2004 to 2015, he was an engineer at Micron Memory Japan, Inc. (formerly Elpida Memory, Inc.). He is currently an engineer at Toshiba Memory Corporation. He is a member of IPSJ and IEICE.

Ushio Jimbo was born in 1990. He received his M.E. degree in Information and Communication Engineering from the University of Tokyo in 2015. He is currently a doctoral student in the Department of Informatics, School of Multidisciplinary Sciences, SOKENDAI (Graduate University for Advanced Studies). He received the xSIG Poster Award in 2017. He is a member of IPSJ.

Ryota Shioya was born in 1981. He received his M.E. and Ph.D. degrees in Information and Communication Engineering from the University of Tokyo in 2008 and 2011, respectively. He was a research fellow of the Japan Society for the Promotion of Science from 2009. In 2011, he became an assistant professor at Nagoya University, where he became an associate professor in 2016. Since 2018, he has been an associate professor at the University of Tokyo. He received IEEE Computer Society Japan Chapter Young Author Award in 2011. He is a member of IPSJ, IEICE and IEEE.

Masahiro Goshima was born in 1968. He received his M.E. in engineering and Ph.D. in informatics from Kyoto University in 1994 and 2004, respectively. He was a research fellow of the Japan Society for the Promotion of Science from 1994. From 1996, he was an assistant professor in Kyoto University. From 2005, he was an associate professor in the University of Tokyo. Since 2014, he has been a professor in the National Institute of Informatics. He has been engaged in the research area of computing systems. He received IPSJ Yamashita SIG Research Award and IPSJ Best Paper Award in 2001 and 2002, respectively. He wrote a book titled "Digital Circuits." He is a member of IPSJ (senior member since 2015) and IEEE.

Shuichi Sakai was born in 1958. He received his B.S., M.S. and D.E degrees from the University of Tokyo in 1981, 1983 and 1986, respectively. He worked at the Electrotechnical Laboratory (1986–1998), Massachusetts Institute of Technology (MIT, 1991–1992), Real World Computing (RWC, 1993–1996), University of Tsukuba (1996–1998). In 1998, he became an Associate Professor of the University of Tokyo where he has continuously been a Full Professor since 2001. His major concerns are computer systems and their applications, especially computer architecture, interconnect networks, optimizing compilers, low power architecture and dependable/security systems. He received several awards, including IPSJ Best Paper Award (1991), IBM Science Award (1991), Ichimura Academic Award (1995), IEEE Outstanding Paper Award (1995), Sun Distinguished Speaker Award (1997). He is a member of IPSJ (fellow since 2010), IEICE (fellow since 2011, President of ISS since 2016), JSAI, ACM, and IEEE.