IEICE TRANS. INF. & SYST., VOL.E99–D, NO.4 APRIL 2016

PAPER

FXA: Executing Instructions in Front-End for Energy Efficiency*

Ryota SHIOYA[†a)], ***Member***, **Ryo TAKAMI**[††], **Masahiro GOSHIMA**[†††], ***Nonmembers***, ***and*** **Hideki ANDO**[†], ***Member***

SUMMARY Out-of-order superscalar processors have high performance but consume a large amount of energy for dynamic instruction scheduling. We propose a front-end execution architecture (FXA) for improving the energy efficiency of out-of-order superscalar processors. FXA has two execution units: an out-of-order execution unit (OXU) and an in-order execution unit (IXU). The OXU is the execution core of a common out-of-order superscalar processor. In contrast, the IXU consists only of functional units and a bypass network only. The IXU is placed at the processor front end and executes instructions in order. The IXU functions as a filter for the OXU. Fetched instructions are first fed to the IXU, and the instructions are executed in order if they are ready to execute. The instructions executed in the IXU are removed from the instruction pipeline and are not executed in the OXU. The IXU does not include dynamic scheduling logic, and thus its energy consumption is low. Evaluation results show that FXA can execute more than 50% of the instructions by using IXU, thereby making it possible to shrink the energy-consuming OXU without incurring performance degradation. As a result, FXA achieves both high performance and low energy consumption. We evaluated FXA and compared it with conventional out-of-order/in-order superscalar processors after ARM big.LITTLE architecture. The results show that FXA achieves performance improvements of 7.4% on geometric mean in SPECCPU INT 2006 benchmark suite relative to a conventional superscalar processor (big), while reducing the energy consumption by 17% in the entire processor. The performance/energy ratio (the inverse of the energy-delay product) of FXA is 25% higher than that of a conventional superscalar processor (big) and 27% higher than that of a conventional in-order superscalar processor (LITTLE).

key words: *superscalar processor, hybrid in-order/out-of-order core, energy efficiency*

1. Introduction

Out-of-order superscalar processors have been widely adopted in PCs and server systems for achieving high performance [2], [3]. These days, high performance is considered to be important even for recent mobile devices, such as smartphones and tablets, and major developers have recently adopted out-of-order superscalar processors in these mobile devices. For example, iPhone, iPad, and most of the popular Android devices are equipped with out-of-order superscalar processors, such as ARM Cortex A9 and its successors [4]–[6].

Although out-of-order superscalar processors provide high performance, they consume a large amount of energy. This is because a large amount of energy is consumed by hardware for dynamic instruction scheduling [7], [8], such as an issue queue (IQ) and a load/store queue (LSQ), which mainly include heavily multi-ported memories. The energy consumption per access of a multi-ported memory is proportional to its capacity and the number of its ports [9]. Moreover, the number of accesses increases with an increase in the issue width, and corresponding energy consumption is significantly large. As a result, out-of-order superscalar processors consume a large amount of energy compared with in-order superscalar processors, which do not have such hardware for dynamic instruction scheduling.

We propose a *front-end execution architecture (FXA)* for improving the energy efficiency of out-of-order superscalar processors. FXA has two execution units: an out-of-order execution unit (OXU) and an in-order execution unit (IXU). The OXU is the execution core of a common out-of-order superscalar processor, which includes several components, such as an IQ, LSQ and functional units (FUs). In contrast, the IXU comprises FUs and a bypass network only. The IXU does not include dynamic scheduling logic, and thus, its energy consumption is low. The IXU is placed at the processor front-end and executes instructions in order. The IXU functions as a filter for the OXU. In the front-end, source operands are read, and then instructions that are ready to execute at this time are executed in the IXU and not dispatched to the OXU.

The IXU can execute many instructions. This is because the IXU also executes instructions whose dependency is dissolved in it, in addition to instructions that are ready to execute while reading source operands. Moreover, we propose a multiple-staging mechanism for the IXU, which places FUs over multiple stages; this allows the IXU to execute more instructions. The evaluation results presented in Section 6 show that more than 50% of the instructions are executed in the IXU.

FXA achieves both low energy consumption and high performance. Since the IXU can execute many instructions, it is possible to reduce the number of accesses to the OXU and shrink the energy-consuming OXU without incurring performance degradation. In addition, the instruction exe-

Manuscript received August 12, 2015.
Manuscript revised November 21, 2015.
Manuscript publicized January 6, 2016.

[†]The authors are with Graduate School of Engineering, Nagoya University, Nagoya-shi, 464–8603 Japan.
[††]The author is with MegaChips Corporation, Osaka-shi, 532–0003 Japan.
[†††]The author is with Information Systems Architecture Research Division, National Institute of Informatics, Tokyo, 101–8430 Japan.

*This paper is an extended version of [1], containing a more detailed description and analysis.

a) E-mail: shioya@nuee.nagoya-u.ac.jp

DOI: 10.1587/transinf.2015EDP7316

cution in the IXU improves performance. The IXU does not have multi-ported structures, such as an IQ. Hence, IXU can have many FUs without a large energy overhead being incurred, which makes it possible to improve performance.

We evaluated FXA and compared it with conventional out-of-order/in-order superscalar cores after ARM big.LITTLE architecture [6], [10], [11]. The evaluation results show that FXA achieves IPC improvements of 5.7% (and 7.4% with integer benchmarks only) in SPECCPU 2006 benchmark suite with respect to the big core, while reducing the energy consumption by 17%. The performance/energy ratio (the inverse of the energy-delay product) of FXA is 25% higher than that of the big core, and even 27% higher than that of the little core. We also evaluated FXA with performance-centric superscalar cores after Intel Haswell [3] or IBM POWER8 [2], and FXA achieves IPC improvements of 5.0% while reducing the energy consumption by 23%.

The rest of this paper is organized as follows: In Section 2, the basic characteristics of FXA are described, and in Section 3, its details and optimization. In Sects. 4 and 5, its performance and energy consumption are given. In Sect. 6, evaluation results are presented. In Sect. 7, the related work is summarized.

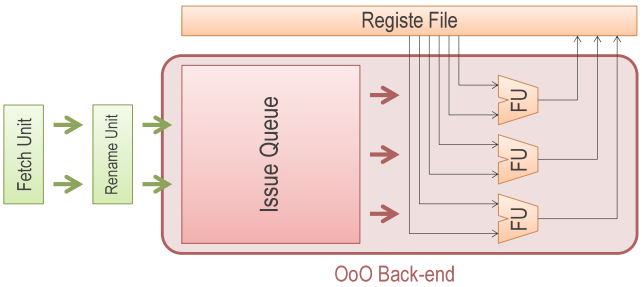


Fig. 1 Conventional superscalar architecture.

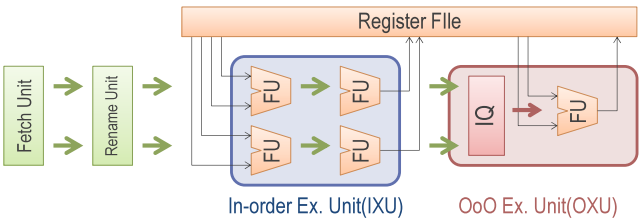


Fig. 2 Front-end execution architecture. The area of the IQ is significantly smaller than that in Fig. 1 because both the width and capacity are decreased as described in Sect. 5.3. Although the PRF ports are added for the IXU, the total number of ports is not different from that in Fig. 1, because the number of ports for the OXU is reduced.

2. Front-End Execution Architecture

We propose a *front-end execution architecture (FXA)*. In this section, we first describe the structure of FXA, and then, its behavior and details.

2.1 Structure

We describe the structure of FXA by comparing it to that of a conventional out-of-order superscalar processor. Figure 1 shows the block diagram of a conventional out-of-order superscalar processor. This figure shows a physical register-based architecture [3], [12]–[14]. Hereafter, the term "conventional superscalar processor" refers to this architecture. In contrast, Fig. 2 shows the the block diagram of FXA. FXA has the following two execution units:

1. **Out-of-order execution unit (OXU)**: This unit is the execution core of a conventional superscalar processor, which is shrunk compared with that of the conventional SSP.
2. **In-order execution unit (IXU)**: This unit is an execution structure that is unique to FXA. The main components of an IXU are FUs and a bypass network. As shown in Fig. 2, the IXU is placed after the rename stage in the front-end. FXA has a register read stage after the rename stage in addition to that in the OXU. Source operand values read from the register read stage in the front-end are fed to the IXU, and instructions are executed *in order* by using these.

FXA has a datapath that accesses the PRF in the front-end for supplying source operands to the IXU. The IXU and OXU partially share the ports of the PRF. The IXU accesses the shared ports only when the OXU does not access them. Additionally, the scoreboard of the PRF is accessed in the front-end for checking whether the values read from the PRF are available. Each entry of the scoreboard is a 1-bit flag that indicates whether a value in a corresponding entry of the PRF is available. It should be noted that this scoreboard is a module that a conventional superscalar processor originally provides [12]†. Hereafter, the term "scoreboard" refers to the PRF scoreboard.

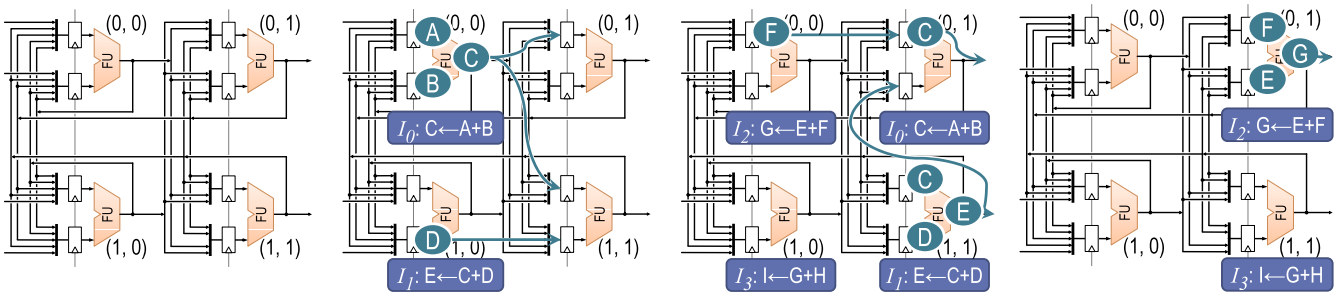
2.2 Basic Behavior of FXA

The IXU functions as a filter for the OXU. That is, instructions executed in the IXU are removed from the instruction pipeline and are not executed in the OXU. This section describes this behavior by comparing conventional superscalar processors and FXA. It should be noted that, in this section, instructions are assumed to be integer instructions with 1-cycle latency. The execution of other types of instructions is described in Sect. 2.4.

The behavior of FXA above a rename stage is the same as that of conventional superscalar processors. FXA processes instructions below the rename stage as follows:

1. Read from the PRF and the scoreboard at the register read stage in the front-end.
2. Check whether instructions are ready. Hereafter, we

†Operands that are already written to the PRF are not woken up, and consequently, they must be dispatched to the IQ as initially ready. For detecting such initially ready operands, this scoreboard is used [12].



(a) Datapath (b) Cycle 1 Behavior (c) Cycle 2 Behavior (d) Cycle 3 Behavior

Fig. 3 In-order execution unit.

use the term *ready instruction* to refer to an instruction that is ready to execute. Source operands can be obtained through the following two paths, and if all source operands are thus obtained, the instruction is ready.

a. Read from the PRF.
b. Bypassed from the FUs in the IXU.

Whether values read from the PRF are available is checked by reading the scoreboard.

3. Depending on whether an instruction is ready, the instruction is processed as follows:

a. A ready instruction is executed in the IXU and is not dispatched to the IQ. Its execution result is written to the PRF after it exits the IXU. The instruction is committed later as in conventional superscalar processors[†].
b. A not-ready instruction goes through the IXU as a NOP. The instruction is dispatched to the IQ and is executed in much the same way as it is executed in conventional superscalar processors.

It should be noted that the behavior of the IXU for not-ready instructions is different from that of in-order superscalar processors. When a not-ready instruction is decoded, an in-order superscalar processor stalls its pipeline until its readiness is resolved. In contrast, in FXA, not-ready instructions go through its pipeline as a NOP; thereby, its pipeline is not stalled, and instructions keep flowing.

The IXU and the OXU are placed in series and not in parallel as in a clustered architecture [13], [15]. This is because serial placement considerably reduces the complexity of bypassing, wake-up, and steering, as described in Sects. 3 and 7.1. In contrast, parallel placement cannot reduce this complexity and its merit is negligible; its branch misprediction penalty is slightly reduced by the shortened pipeline length.

2.3 Detailed Behavior of IXU

In this section, first, the structure and behavior of the IXU are described, and then, its control.

[†]The entries of a reorder buffer are allocated for all instructions for implementing a precise exception.

$$
\begin{aligned}
I_0 &: \ \mathbf{C} \leftarrow \mathrm{A} + \mathrm{B} \\
I_1 &: \ \mathbf{E} \leftarrow \mathbf{C} + \mathrm{D} \\
I_2 &: \ \mathbf{G} \leftarrow \mathbf{E} + \mathrm{F} \\
I_3 &: \ \mathrm{I} \leftarrow \mathbf{G} + \mathrm{H}
\end{aligned}
$$

Fig. 4 Code executed in IXU.

2.3.1 Structure and Behavior of IXU

An IXU has FUs serially placed over 2-3 stages to increase the number of instructions executed in the IXU. Figure 3 (a) shows a datapath example of an IXU with the FUs of 2-instruction width × 2 stages. In this figure, $FU(y, x)$ denotes an FU at the x-th position from the left and y-th position from the top. The FUs are connected through the bypass network that allows each FU to use each other's execution results. In Fig. 3 (a), it can be seen that source operands read from the PRF are fed from the left side, and the execution results are fed to the right-hand side and written to the PRF.

We describe the behavior of an IXU by using an example where the code shown in Fig. 4 is executed on the IXU shown in Fig. 3 (a). The code includes the serially dependent instructions from I_0 to I_3. All the source operands, except C, E, and G, shown in Fig. 4, have already been read from the PRF. The behavior of each cycle is as follows:

- **Cycle 1:** Figure 3 (b) shows the state of the first cycle. In this cycle, I_0 and I_1 are at the first stage of the IXU. I_0 on $FU(0, 0)$ is executed because all its source operands are ready, and the execution result C is outputted. An execution result is fed through the bypass network and received by FUs, which use the execution result in the next cycle. In this case, the execution result C is received by $FU(1, 1)$, which uses it in the next cycle. In contrast, I_1 on $FU(1, 0)$ is not executed, because its source operand C has not yet been executed, and the other source operand D is fed to the next stage.
- **Cycle 2:** Figure 3 (c) shows the cycle that follows the cycle shown in Fig. 3 (b). In Fig. 3 (c), I_0 and I_1 are moved to the second stage, and I_2 and I_3 are fed to the first stage. I_0 on $FU(0, 1)$ does nothing in this cycle

because it has already been executed in the previous cycle. I_1 on $FU(1,1)$ is executed in this cycle because the source latch has the source operand C executed in the previous cycle. The execution result E is received by $FU(0,1)$, which uses it and executes I_2 in the next cycle. I_2 on $FU(0,0)$ and I_3 on $FU(1,0)$ are not executed, because their source operands have not yet been executed.

- **Cycle 3:** Figure 3 (d) shows the next cycle. I_2 and I_3 are moved to the second stage. I_2 on $FU(0,1)$ is executed in this cycle and outputs the execution result G. I_3 on $FU(1,1)$ is not executed in the IXU, because its source operand G has not yet been executed in this cycle.

The IXU leverages FUs serially placed over multiple stages and can execute dependent instructions, such as the instructions from I_0 to I_2. In contrast, the IXU cannot execute instructions after a *long and consecutive* chain of dependent instructions, such as I_3. However, an IXU can execute instructions in a dependent chain when the length of the chain is long but the chain is not consecutive. For example, if there is an independent instruction between I_2 and I_3, as shown in Fig. 4, the IXU can execute I_3, because I_3 can use the execution result of I_2 in the next cycle of the cycle shown in Fig. 3 (d). In general, dependent instructions are rarely placed in a long and consecutive chain[†], and consequently, an IXU can execute many instructions.

Note that, when the number of stages in an IXU is larger than the width of the IXU, the IXU still cannot execute all instructions. Stages that execute instructions in a consecutive chain are moved downstream, and finally, dependent instructions cannot be executed when there are no remaining downstream stages. For example, I_0 is executed in the first stage, and then, I_1 and I_2 are executed in the second stage. Finally, I_3 cannot be executed, because there is no third stage.

2.3.2 Control of IXU

The control of an IXU, that is, the decision as to which FU executes an instruction and the control of operand-bypassing, is determined by comparing register numbers in the same way as operand-bypassing is conducted in conventional superscalar processors. The control signals are generated in parallel with renaming and register read, because this process uses logical register numbers, which can be used after decoding. The generated control signals and instructions are fed to the IXU, and the fed signals control the FUs and the bypass network. Consequently, the critical path is not prolonged by the generation of the control signals.

2.4 Behavior When Executing Other Instruction Types

We described the behavior of FXA that executes integer instructions. In this section, we describe its behavior when executing other types of instructions.

2.4.1 Branch

In the same way as it executes integer instructions, the IXU executes branch instructions. The FU in the IXU compares a branch prediction result and a branch executed result, and then, detects a branch misprediction. If a branch misprediction is detected, the pipeline is flushed and recovered at this time.

2.4.2 Floating Point

The IXU does not have floating point (FP) units for avoiding area overhead increase and performance degradation due to a prolonged pipeline length. The latency of FP operations generally constitutes multiple cycles, and hence, the pipeline length is significantly prolonged if multiple FP units are serially placed.

2.4.3 Load/Store

FXA assumes a scheme that issues loads/store instructions speculatively by using a dependency predictor [13], [16], such as the store-set predictor [16]. In this scheme, load/store instructions are issued not from the LSQ but from the IQ. The following part briefly describes how this scheme executes load/store instructions. 1) After a load instruction is issued, this scheme searches the LSQ by using the address of the load instruction. If a *predecessor* store instruction with the same address is detected and has already been executed, its data is forwarded to the load instruction. At the same time, its data address is written to the LSQ. 2) After a store instruction is issued, this scheme searches the LSQ by using the address of the store instruction. If a *successor* load instruction with the same address is detected and has already been executed, an order violation is detected. At the same time, its data address and data are written to the LSQ.

The IXU executes load/store instructions according to results of the arbitration of resources between the IXU and the OXU[††]. These resources are the LSQ and the L1 data cache. In this arbitration, instructions in the OXU have higher priority than those in the IXU. If the arbiter determines that instructions cannot be executed in the IXU, they are simply dispatched to the IQ. Therefore, the pipeline is not stalled in this case, and the performance degradation is small.

In FXA, the LSQ itself is not different from that of conventional superscalar processors. The differences are that the LSQ is accessed by both the IXU and the OXU and the processes of the LSQ are partially omitted as follows:

[†]Therefore, in-order superscalar processors have a higher performance than do scalar processors.

[††]When the IXU executes store instructions, stored data are written to the LSQ only, in much the same way as in conventional superscalar processors, and then, stored data are written to the data cache in the commit stages.

1. Omitting Order Violation Detection: When a store instruction is executed in the IXU, there is no successor load instruction that has already been executed. Consequently, in this case, an order violation never occurs, and the search in the LSQ can be omitted.
2. Omitting Load Instruction Writing: When a load instruction is executed in the IXU and all its predecessor store instructions have already been executed, an order violation caused by the load instruction never occurs. This is because the predecessor store instructions and the load instruction are executed in order. Consequently, it is not necessary to detect an order violation caused by the load instruction, and writing the load instruction to the LSQ can be omitted.

These omissions reduce the energy consumption of the LSQ, as described in Sect. 5.4.

3. Detailed Design and Optimization

In this section, we describe the design of FXA in detail and its optimization for mitigating complexity.

3.1 Operand Bypassing

In this section, we describe the design of the bypass network in detail and discuss its complexity.

3.1.1 Bypassing between IXU and OXU

Between the IXU and the OXU, data are communicated only through the PRF, and there is no other datapath between them. The following description describes the reasons for the absence of operand-bypassing between the IXU and the OXU for each direction.

- IXU → OXU: It is not necessary to bypass data from the IXU to the OXU. The IXU and the OXU have an order relation, as instructions not executed in the IXU go into the OXU, as shown in Fig. 2. Consequently, when instructions are executed in the IXU, their consumers are not in the OXU, and it is not necessary to pass the execution results to the OXU.
- OXU → IXU: We omit the operand-bypassing from the OXU to the IXU because the performance degradation caused by this omission is negligible. Between the IXU and the OXU, there are the IQ and several pipeline stages. Thus, instructions executed in the IXU are distant from instructions executed in the OXU in a program order, and the probability that they have dependencies is low. Consequently, if execution results cannot be passed directly from the OXU to the IXU, the number of affected instructions is small, and thus, performance degradation is not significant.

The operand-bypassing of the IXU and the OXU is separated, and its complexity and energy consumption are similar to those of the bypass network in conventional superscalar processors, as described below.

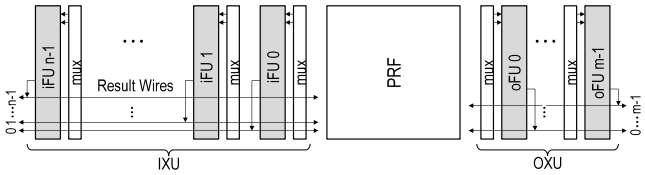


Fig. 5 Bit-slice of the bypass network.

3.1.2 Optimization of IXU

When FUs are placed over multiple stages in the IXU, the latency of its bypass network is increased, and operand-bypassing and the operation of an FU may not be completed in one cycle.

To mitigate its complexity, we decrease the number of FUs in the backward stages in the IXU. The number of instructions executed in the backward stages in the IXU is relatively small, and thus, decreasing the number of the FUs in the backward stages does not significantly degrade performance. The evaluation results presented in Sect. 6 show that the performance of a configuration with $3ways \times 1stage + 1way \times 2stages = 5$ FUs is almost the same as that of a configuration with $3ways \times 3stages = 9$ FUs.

Additionally, we partially omit operand-bypassing in the IXU. As described in Sect. 2.3, instructions that are mutually distant are executed on FUs that are distant. Consequently, operand-bypassing for FUs that are distant seldom occurs. The evaluation results presented in Sect. 6 show that performance degradation is negligible when operand-bypassing between FUs that are more distant than two stages is omitted.

3.1.3 Complexity of IXU

In this section, we discuss the complexity of the bypass networks in FXA. The bypass network in the IXU has a structure similar to that of the bypass network of conventional superscalar processors. The main components of both bypass networks are result wires for broadcasting execution results and multiplexers for selecting operands [15]. Figure 5 shows the layout of the bypass network [15], [17] that we assume in this study. This figure shows a bit-slice of the datapath. There are n FUs in the IXU, ranging from iFU_0 to iFU_{n-1}, and m FUs in the OXU, ranging from oFU_0 to oFU_{m-1}. These FUs are placed on both sides across the PRF. Each FU broadcasts its execution result over a result wire, and the result is selected by multiplexers and received by FUs that execute consumers. As described in Sect. 3.1.1, operands are not bypassed between the IXU and the OXU, and thus, the result wires of the IXU and the OXU are mutually independent.

The latency of a bypass network is determined mainly by the length of its result wires and the number of its multiplexer inputs. The length of result wires is proportional to the number of FUs, when the layout is as illustrated in Fig. 5. The maximum number of multiplexer inputs is the number

of result wires, and thus, it is proportional to the number of FUs. The number of FUs is $n = 5$ in the configuration used in the evaluations presented in Sect. 6. Consequently, the latency of the bypass network in the IXU is not significantly different from that of the bypass network in 4-issue conventional superscalar processors.

3.2 Physical Register File

In the layout presented in Fig. 5, the IXU and the OXU are directly placed on both sides of the PRF, and thus, the IXU and the OXU can access the PRF with a short latency. The bitlines of the PRF are placed horizontally in this figure. For the PRF ports shared by the IXU and the OXU, sense-amps are placed on both ends of the bitlines; they supply values to the IXU and the OXU. Consequently, the IXU and the OXU can access the PRF with a latency that is similar to that of the PRF of conventional superscalar processors, and the shared ports do not increase its latency.

The additional sense-amp does not significantly increase the latency of the PRF. A bitline is generally connected to two transistors that a sense-amp consists of [9]. The parasitic capacitance of the two transistors is significantly smaller than that of the bitline and the access transistors of RAM cells. Consequently, the additional parasitic capacitance does not significantly increase the latency of the PRF.

When the PRF is read in the front-end, invalid entries that have not yet been written may be read, and these reads increase its energy consumption. To mitigate this problem, the scoreboard (Sect. 2.1) is read before the PRF is read, and then, the PRF is read only when values in the PRF are available. This sequential access makes it possible to reduce the energy consumed by invalid reading. Although the latency of the scoreboard is very small, because its capacity is much smaller than that of the PRF, this sequential access may prolong the critical path. For this reason, an additional stage is added to the front-end in the evaluations presented in Sect. 6.

Note that, in this study, we assume that the ports of the PRF are partially shared by the IXU and the OXU, but even though the ports are not shared for avoiding arbitration, the number of ports is not significantly increased. This is because the number of ports of the PRF required for the IXU is increased, but that required for the OXU is decreased. Moreover, methods that reduce the complexity of the PRF, such as a hierarchical PRF [18], [19], can mitigate the complexity of the additional ports.

3.3 Scoreboard

FXA reads the scoreboard twice per instruction to provide correct execution. The first reading is carried out before instructions go into the IXU (Sect. 2.2), and the second reading is carried out in the dispatch stage (Sect. 2.1). These two readings cannot be combined into one. This is because there is a possibility that a not ready instruction that goes through the IXU as NOPs becomes ready when the corresponding producer is executed in the OXU at the same time. If such instructions are dispatched to the IQ as not ready, their operands are never woken up, and the processor cannot continue execution correctly. The second reading from the scoreboard makes it possible to dispatch such instructions as ready correctly.

4. Performance of FXA

In this section, we describe the instructions that can be executed in IXU, and then, the performance of FXA.

4.1 Instructions Executed in IXU

As described in Sect. 2, instructions executed in the IXU are categorized as follows:

- **(a)** Instructions that are already ready when they are entered into the IXU. All their source operands have already been obtained from the PRF.
- **(b)** Instructions that become newly ready in the IXU. They receive execution results executed in the IXU, and all their source operands are complete in the IXU.

The number of (a), which comprise mainly instructions dependent only on registers that have not been updated for a long time, is small. Using the configuration used in the evaluation presented in Sect. 6, the ratio of the number of (a) to the number of all executed instructions is 5.5% on average. In contrast, the number of (b) is large, and instructions executed in the IXU comprise mainly (b). The evaluation results presented in Sect. 6 show that the IXU with one-stage FUs can execute 35% of the instructions. Moreover, FXA can execute more instructions by serially placing FUs over multiple stages in the IXU, as described in Sect. 2.3. This makes it possible to increase the number of (b) significantly. The evaluation results presented in Sect. 6 show that an IXU with three-stage FUs can execute 54% of the instructions.

4.2 Performance Improvement

FXA improves the performance as compared to conventional superscalar processors. This improvement is achieved as a result of the FUs added in the IXU and the reduced branch misprediction penalty.

4.2.1 Effects of FUs in IXU

In FXA, the number of FUs is increased as compared to that in conventional superscalar processors, because the IXU is added. If the IXU executes many instructions, FXA can improve performance in a manner similar to that used to widen its issue width. For example, with the configurations used in the evaluation presented in Sect. 6, the conventional superscalar processor can execute up to four instructions per cycle. In contrast, FXA can execute up to seven instructions per cycle with an OXU of two-instruction issue width and an IXU with five FUs.

Moreover, the instructions executed in the IXU are not dispatched to the OXU, and thus, other instructions can use IQ entries and the issue ports that are supposed to be used by the instructions executed in the IXU. The OXU is shrunken to the degree at which performance is not significantly decreased in all applications, and consequently, FXA improves performance in applications where the IXU can execute many instructions.

In FP applications, integer instructions that are executed in the IXU are not dispatched to the OXU, and thus, FXA also improves performance, as shown in the evaluation results presented in Sect. 6. This is because FP applications still include many integer and load/store instructions[†].

4.2.2 Reducing Branch Misprediction Penalty

FXA can execute branch instructions and detect branch misprediction in the IXU (Sect. 2.4). If a misprediction is detected in the IXU, the misprediction penalty is reduced, because the IXU is placed at the front-end. Recent superscalar processors have pipelines with more than 10 stages [3], [6], [14], and thus, their branch misprediction penalty is also more than 10 cycles. In FXA, if a misprediction is detected in the IXU, its misprediction penalty is reduced by approximately half.

In contrast, if a misprediction is detected in the OXU, the misprediction penalty is increased by the number of stages of the IXU, which is usually four or five, because its pipeline length is prolonged. However, more than 50% of the instructions are executed in the IXU (Sect. 6), and thus, the total penalty is reduced.

5. Reducing Energy Consumption

In this section, we describe how FXA reduces energy consumption. The energy consumption reduction of FXA is based on the following: 1) the IXU does not significantly increase energy consumption; and 2) the energy consumption of the IQ and the LSQ is reduced.

5.1 IXU

The energy consumption of the IXU comprises mainly that of the FUs and the bypass network. This section describes the energy consumption of these components.

5.1.1 Functional Units

The dynamic energy consumption of the FUs is determined by 1) the dynamic energy consumption of each FU per access, and 2) the number of its accesses. 1) The dynamic energy consumption of each FU per access in FXA and conventional superscalar processors is the same, because their FUs are exactly the same. 2) The numbers of accesses to the FUs in FXA and conventional superscalar processors are not significantly different, because all instructions using FUs are executed once on any FU. The total dynamic energy consumption is calculated from the product of 1) and 2), and consequently, the total dynamic energy consumption of the FUs in FXA is not significantly different from that of the FUs in conventional superscalar processors.

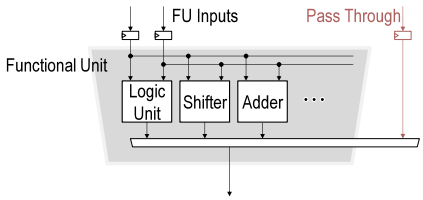


Fig. 6 Functional unit in IXU.

It should be noted that FUs do not consume dynamic energy when instructions go through on the FUs in the IXU as NOPs. Figure 6 shows a block diagram of the FUs in the IXU. The FU in this figure has a structure where the outputs of several units, such as the adder and the shifter, are selected by the multiplexer [17]. When source operands or execution results are sent to the next stage in the IXU, the path for passing through on the right-hand side of the figure is used and data are selected by the multiplexer. In this case, the source latches of the FU are controlled such that they are not updated, and thus, switching does not occur in the FU, and the FU does not consume dynamic energy.

The static energy consumption is increased by the additional FUs in the IXU, but this increase is relatively small and does not cause a serious problem. The static energy consumption is proportional to the number of transistors in a circuit, and transistors in fast circuits such as FUs consume a relatively large amount of static energy. However, the number of the transistors in the integer FUs for the IXU is much smaller than that of transistors of other circuits, and thus, its static energy is relatively small. For example, integer adder, which occupies most of the circuit area of an integer FU, consists of 4k–6k transistors [20], [21]. In contrast, an FP multiplier and an FP adder, which require fast transistors as integer FUs, consists of more than 200k transistors [21], which is several tens of times the number of transistors in integer adders. Consequently, the static energy consumed by the few integer FUs for the IXU is negligible.

5.1.2 Bypass Network

The energy consumption of the bypass network in FXA is not significantly different from that in conventional superscalar processors. This is because the energy consumption is increased by the bypass network in the IXU, but the energy consumption of the bypass network in the OXU is reduced.

As described in Sect. 3.1, the bypass networks in the IXU and the OXU comprise mainly result wires and mul-

[†]Using the configuration used in the evaluation presented in Sect. 6, in SPECCPU FP 2006 applications, the average ratio of FP instructions in all executed instructions is 30.8% and the maximum is 52.0%.

tiplexers[††]. In the bypass networks, energy is consumed mainly for driving the result wires. Each FU executes an instruction and drives its own result wire. At this time, energy is consumed proportionally to the parasitic capacitance of the result wire; the parasitic capacitance is proportional to the length of the result wire. The length of the result wire is proportional to the number of FUs in the layout shown in Fig. 5. Consequently, the energy consumption is proportional to the number of FUs.

On the basis of this assumption, we compare the energy consumption of the bypass networks. When the configurations used in the evaluation presented in Sect. 6 are used, in FXA, the IXU has $n = 5$ FUs and the OXU has $m = 4$ FUs. In this configuration, more than 50% of the instructions are executed in the IXU, and thus, the energy consumption of the bypass network in the IXU is proportional to the average of $n = 5$ and $n = 4$, which is $n = 4.5$. Consequently, the energy consumption per bypassing in the IXU does not increase significantly as compared to that of conventional superscalar processors ($m = 4$)[†].

It should be noted that some result wires in the IXU are short, because it is not necessary for each instruction to send its execution result to its predecessor according to the program order. Moreover, as described in Sect. 3.1.2, the operand-bypassing in the IXU is partially omitted, and this makes it possible to reduce the length of the result wires. Consequently, the actual energy consumption of the bypass network is smaller than that assumed in the above discussion.

5.2 Physical Register File

The energy consumption of the PRF is determined by 1) the energy consumption of each PRF per access, and 2) the number of its accesses.

1) The energy consumption of each PRF per access in FXA and conventional superscalar processors is not significantly different, because the areas of their PRFs are almost the same. This is because the number of ports of the PRF required for the IXU is increased, but that required for the OXU is decreased. For example, both the PRFs of the conventional superscalar processor in Fig. 1 and FXA in Fig. 2 have nine ports. Moreover, the ports of the PRF are partially shared by the IXU and the OXU, and the IXU accesses the shared ports only when the OXU does not access them, as described in Sect. 2.1. As a result, the number of ports of the PRF in FXA is not different from that in conventional superscalar processors.

2) The numbers of accesses to the PRF in FXA and conventional superscalar processors are not significantly different, because all instructions access the PRFs once.

The total energy consumption is calculated from the product of 1) and 2), and consequently, the total energy consumption of the PRFs of FXA and that of conventional superscalar processors are not significantly different.

The capacity of the scoreboard is significantly smaller (1/64) than that of the PRF, and consequently, the energy consumption of the scoreboard is negligible.

5.3 Issue Queue

In FXA, the energy consumption of the IQ is significantly reduced as compared to that of a conventional superscalar processor. FXA reduces the capacity and the issue width of the IQ without performance degradation being incurred, because the IXU can execute many instructions.

The IQ comprises mainly CAMs and RAMs, with the number of ports being proportional to the issue width. Their energy consumption (and area) is proportional to the number of their ports and their capacities [9]. Consequently, the reduction in the capacity and issue width significantly reduces the energy consumption of the IQ per access. Moreover, the number of accesses is also significantly reduced, because instructions executed in the IXU are not dispatched to the OXU. The evaluation results presented in Sect. 6 show that the energy consumption of the IQ is reduced to 14% of that of the IQ of a conventional superscalar processor.

5.4 Load/Store Queue

The LSQ also comprises mainly CAMs and RAMs, and it consumes a large amount of energy. As described in Sect. 2.4.3, FXA partially omits processing in its LSQ. The LSQ in FXA is not different from that in conventional superscalar processors, but the number of accesses is reduced by omitting the processing. Consequently, the energy consumption of the LSQ is reduced.

6. Evaluation

We evaluated FXA and other processor architectures.

6.1 Evaluation Environment

We evaluated IPCs by using an in-house cycle-accurate processor simulator. We used all the programs from the SPEC CPU 2006 benchmark suites [22] with *ref* datasets. The programs were compiled using gcc 4.5.3 with the "-O3" option. We skipped the first 4G instructions and evaluated the next 100M instructions. We evaluated energy consumption and chip areas by using the McPAT simulator [23] with the parameters shown in Table 2.

6.2 Evaluation Models

We evaluated the following models whose configurations are based on those used in most ARM big.LITTLE architecture, which consists of ARM Cortex-A57 [6] and Cortex-A53 [10]. Table 1 shows the detailed parameters of the evaluated models.

- **BIG:** BIG is the baseline model of this evaluation. It is

[††]The result wires of the IXU and the OXU are separated (Sect. 3.1).

[†]All these FUs are integer FUs.

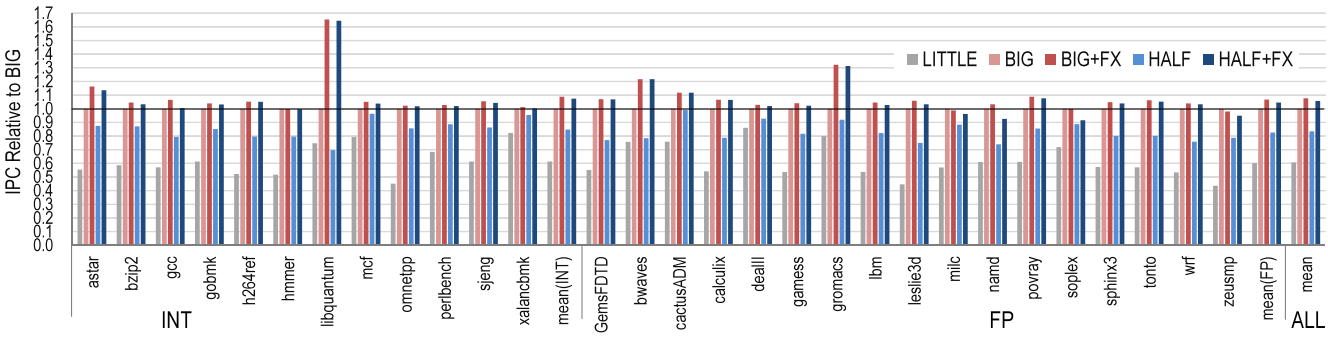


Fig. 7 IPC relative to BIG.

Table 1 Processor configurations.

	BIG	HALF	LITTLE
type	out-of-order	←	in-order
fetch width	3 inst.	←	2
issue width	4 inst.	2 inst.	2
issue queue	64 entries	32 entries	N/A
FU (int, mem, fp)	2, 2, 2	←	2, 1, 1
ROB	128 entries	←	N/A
int/fp PRF	128/96 entries	←	N/A
ld/st queue	32/32 entries	←	N/A
branch pred.	g-share, 4K PHT, 512 entries BTB	←	←
br. mispred. penalty	11 cycles	←	8 cycles
L1C (I)	48 KB, 12 way, 64 B/line, 2 cycles	←	←
L1C (D)	32 KB, 8 way, 64 B/line, 2 cycles	←	←
L2C	512 KB, 8 way, 64 B/line, 12 cycles	←	←
main mem.	200 cycles	←	←
ISA	Alpha	←	←

an out-of-order superscalar processor. Its major micro-architectural parameters are the same as those of ARM Cortex-A57, which include parameters such as fetch width, issue width, the size of an instruction window, the number of FUs, cache sizes, and branch predictors.

- **HALF:** This model has an IQ whose issue width and capacity are half of those in BIG. Its other parameters are the same as those of BIG.
- **LITTLE:** This is a model of an in-order processor. Like BIG, its major micro-architectural parameters are the same as those of ARM Cortex-A53.
- **HALF+FX:** This is a model of FXA. The other elements of an IXU are basically the same as those of HALF, but the number of ports of the PRF is the same as that of BIG. The ports of the PRF shared by the IXU and the OXU are accessed as described in Sect. 3.2. The IXU has three stages and five FUs (the first stage has three FUs, and the second and third stage each have one FU). As described in Sect. 3.1.2, operand bypassing from the third stage to the first stage in the IXU is omitted. This configuration of the IXU has the highest performance among configurations whose bypass network complexity is similar to that of the bypass network in BIG. The bypass network complexity is considered on the basis of the discussion on complexity in Sect. 3.1.
- **BIG+FX:** This is a model of FXA. It has an IQ whose issue width and capacity are the same as those of BIG. Its other parameters are the same as those of HALF+FX.

Through the evaluation of these models, we show that HALF+FX achieves both a higher performance and lower energy consumption than BIG.

6.3 IPC

Figure 7 shows the IPCs for each model relative to BIG. HALF+FX improves the IPC of BIG by 5.7% on geometric mean. The IPC improvement of HALF+FX in the INT benchmark programs is significant; the maximum improvement is 67% for libquantum and the geometric mean is 7.4%. The IPC of HALF+FX in FP benchmark programs is also improved: it is 4.5% on geometric mean. These IPC improvements are achieved because many instructions are executed in the IXU. The rates of instructions executed in the IXU are 61%, 51%, and 54% in the INT benchmark group, FP benchmark group, and all benchmark programs, respectively.

In libquantum and gromacs, HALF+FX significantly improves the IPC compared with BIG. This is because HALF+FX can execute more INT operations compared with BIG in a single cycle. In this case, the term "INT operations" includes logical, add/sub, shift, and branch instructions and does not include load/store instructions. In BIG, the maximum number of INT operations executed in a single cycle is two because the number of INT FUs is two. In contrast, HALF+FX can execute up to seven INT operations in a single cycle[†]. libquantum and gromacs include significantly more INT operations (more than 80%) than the other applications include (50% on average). Consequently, HALF+FX with high INT-operation-throughput significantly improves performance in the programs[††].

The IPC degradation of HALF as compared to BIG is significant: 16% on geometric mean. This is because the

[†]It can execute five operations steadily in a single cycle.

[††]This high INT-operation-throughput is achieved by the IXU without increasing the width of the IQ (Sect. 4.2.1).

width and size of IQ in HALF are half of those in BIG.

HALF+FX improves the IPC as compared to HALF by 27% on geometric mean, which is significant, although HALF+FX has the same IQ as HALF. HALF+FX can be considered to be a combination of HALF and an additional IXU. This shows that the addition of the IXU significantly improves the IPC, as described in Sect. 4.2.

The improvement of the IPC by BIG+FX as compared to HALF+FX is slight: 1.8% on geometric mean. This is because the IXU executes a sufficient number of instructions and the number of instructions fed to the OXU is small. As a result, the increase in the width/size of the IQ does not significantly improve the IPC.

LITTLE significantly degrades the IPC as compared to the other models, because LITTLE is an in-order superscalar processor. The IPC degradation of LITTLE as compared to BIG is 40% on geometric mean.

6.4 Energy Consumption

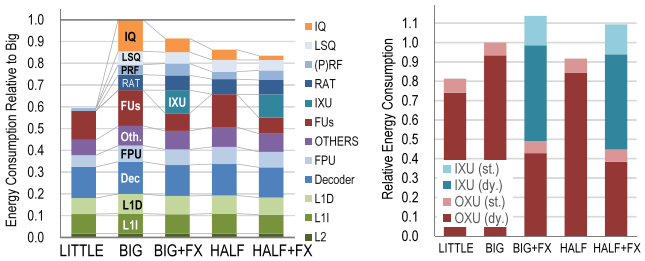
Figure 8 (a) shows the energy consumption for each model relative to BIG. This energy consumption is the sum of static and dynamic energy consumption. In the graph, "FUs" denotes the energy consumption of the FUs and the bypass network in the OXU. Similarly, "IXU" denotes the energy consumption of the FUs and the bypass network in the IXU. "OTHERS" represents the energy consumption of the other units, such as TLBs, fetch queues, and branch predictors.

HALF+FX reduces the energy consumption as compared to BIG and HALF by 17% and 3.3%, respectively. This is mainly because the energy consumption of the IQ and the LSQ is reduced, as described in Sect. 5. In particular, the energy consumption of the IQ is significantly reduced. The energy consumption of the IQ in HALF+FX is reduced to 14% and 42% of that in BIG and HALF, respectively. The energy consumption of the IQ in HALF+FX is also reduced as compared to that in HALF, although HALF+FX and HALF have the same IQ, whose width/size is half of that in BIG. This is because the number of instructions dispatched to the IQ is reduced by the execution of instructions in the IXU. The energy consumption of the LSQ in HALF+FX is reduced to 77% of that in BIG, and its effect is small compared with the case of the IQ. This is because all processes of the LSQ are not omitted when load/store instructions are executed in the IXU, as described in Sect. 2.4.3. The energy consumption of the FUs and the bypass network in HALF+FX is increased by 9.3% as compared with that in BIG, but this increase in energy consumption is smaller than the energy consumption decrease effected by the IQ and the other parts. This detailed energy consumption of the FUs and the bypass network is discussed in Sect. 6.5. The energy consumption of the other parts in HALF+FX is slightly smaller than those in BIG, because HALF+FX improves its performance, and thus, static energy consumption is reduced.

BIG+FX reduces the energy consumption as compared to BIG by 8.7%. Although BIG+FX has the same OXU as BIG, the energy consumption is reduced. This is because the number of instructions fed to the OXU is reduced as compared to BIG.

The energy consumption of LITTLE is much smaller than that of the other models: 60% and 71% of that of BIG and HALF+FX, respectively.

It should be noted that the energy consumption of the L2 cache is very small in all the models because we use Fin-FET technology and low-standby-power transistors for the L2 caches, as shown in Table 2. Fin-FET technology significantly reduces leakage current [25]. Furthermore, the leakage current of low-standby-power transistors used in L2 caches is considerably small compared to that of high-performance transistors used in the cores, as shown in Table 2. Thus, the static energy consumption of the L2 caches is very small. The dynamic energy consumption of the L2 caches is also small, because the hit rates of L1 data caches are more than 95% on average in all the models, and thus, the number of accesses to the L2 caches is small.



(a) Whole (b) FUs and bypass networks

Fig. 8 Energy consumption relative to BIG.

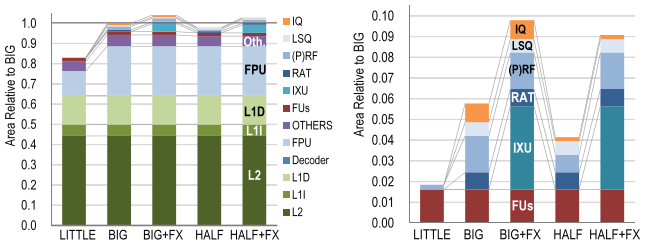
Table 2 Device configurations.

technology	22 nm, Fin-FET [24]
temperature	320 K
VDD	0.8 V
device type (core)	high performance (I_off: 127 nA/μm)
device type (L2)	low standby power (I_off: 0.0968 nA/μm)

6.5 Energy Consumption of FUs and Bypass Networks

Figure 8 (b) shows the energy consumption of the FUs and the bypass network for each model relative to BIG. In the graph, "dy." and "st." are the dynamic and static energy consumption of each module, respectively. In HALF+FX, the energy consumption of the FUs and the bypass network in the OXU is reduced as compared to that in BIG and HALF, but the energy consumption of the IXU is increased. This increase is due mainly to the static energy consumption of the FUs in the IXU. As a result, the energy consumption of the FUs and the bypass network in HALF+FX is increased by 9.4% as compared to that in BIG, but this does not cause a serious overall problem as described above.

The static energy consumption of the IXU is small, because the area of the IXU is small and the number of the transistors that the IXU comprises is small, as described in



(a) Whole (b) FUs to IQ

Fig. 9 Circuit area relative to BIG.

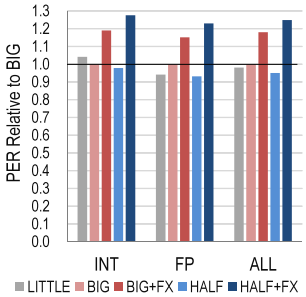


Fig. 10 Performance/energy ratio.

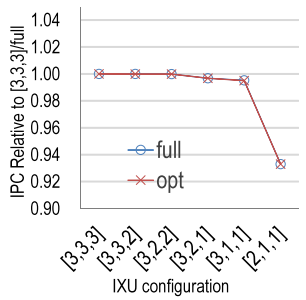


Fig. 11 IPC versus IXU configurations.

Sects. 5.1.1 and 6.6.

In LITTLE, the energy consumption of the FUs and the bypass network is smaller than that of the other models. This is because LITTLE does not perform out-of-order execution, and thus, the number of instructions uselessly executed and flushed on branch misprediction is significantly smaller than that of the other models. The energy consumption of the FUs and the bypass network in HALF is reduced to 92% of that in BIG, for the same reason as in LITTLE. The number of instructions speculatively executed in HALF is smaller than that in BIG because the IQ is shrunk.

6.6 Circuit Area

We evaluated the circuit areas of the models. Figure 9 (a) shows the areas of the entire processor for the other models relative to that of BIG. The labels in this figure are the same as in Fig. 8 (a). The areas of the several units shown in the upper part of Fig. 9 (a) are not clear to see, and thus, Fig. 9 (b) shows the areas of these units[†]. The area of HALF+FX is increased by the addition of the IXU, as shown in Fig. 9 (b). However, the area of the IXU is significantly smaller than that of the entire processor, as shown in Fig. 9 (a), and consequently, the area of HALF+FX is slightly bigger than that of BIG; the area growth is 2.7%. These results support that the number of transistors in the IXU is small compared to that of transistors in the entire processor (Sect. 5.1.1), and thus, the static energy consumption of the IXU is small, as described above. In all the models, an L2 cache and FP units occupy a large area. In HALF+FX, the areas of the L2 cache and FP units are 44% and 24% of the entire area, respectively. These units are basically the same in all the evaluated models except LITTLE[††], and thus, their areas are the same.

6.7 Performance/Energy Ratio

In this section, we show the performance/energy ratio (PER) of each model, which is equal to the inverse of the energy-delay product (EDP). Figure 10 shows the PER of each model relative to that of BIG. In the figure, it can be seen that HALF+FX improves the PER as compared to BIG and LITTLE by 25% and 27%, respectively. This high PER is achieved because HALF+FX improves both the IPC and energy consumption.

6.8 Sensitivity

In this section, we describe the sensitivity of HALF+FX to variations in the IXU.

6.8.1 Optimization of IXU

We evaluated the effect of the optimization of the IXU described in Sect. 3.1.2. Figure 11 shows the IPCs of HALF+FX compared to that of the HALF+FX with nine FUs over three stages and the full bypass network. The "full" line shows the results of HALF+FX with the full bypass network, and the "opt" line shows those with the bypass network that omits operand-bypassing between FUs that are more distant than two stages. The horizontal axis shows the number of the FUs in each stage of the IXU. For example, "[3,1,1]" shows that the first stage has three FUs, and the second and third stage each has one FU. The performance degradation of the model with [3,1,1] and opt, which is used in the other evaluations, is only 0.5% as compared to the model with nine FUs and the full bypass network. These results show that the performance degradation caused by the optimization described in Sect. 3.1.2 is negligible.

6.8.2 Function Units Stages in IXU

We evaluated HALF+FX while varying the depth of FUs

[†]The area of the IQ in HALF and HALF+FX is significantly smaller than that in BIG because both the width and capacity are decreased (Sect. 5.3).

[††]LITTLE has fewer FP units than the other models.

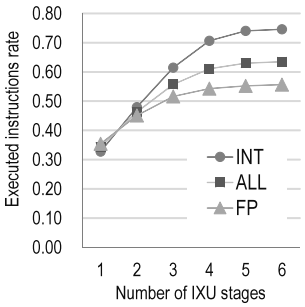


Fig. 12 Executed instructions rate in IXU.

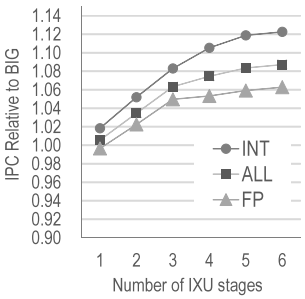


Fig. 13 IPC versus IXU stages.

in the IXU from 1 to 6. It should be noted that the optimization of the IXU described in Sect. 3.1.2 is not applied to HALF+FX in this section. Figure 12 shows the rate of instructions executed in the IXU relative to all the executed instructions (hereafter, in this section, "execution rate" refers to this rate). Each line in Fig. 12 shows the execution rate of the geometric mean of the INT benchmark group, FP benchmark group, and all benchmark programs. Figure 12 shows that the execution rate increases with an increase in the depth. The figure shows that HALF+FX can execute many instructions, and HALF+FX with the one-stage IXU executes 35% of the instructions on geometric mean. HALF+FX with the three-stage IXU executes more than half of instructions; the execution rate is 54%. The execution rate in the INT benchmark group is significantly higher than that in the FP benchmark group. The execution rates for the INT and FP benchmark groups in HALF+FX with the three-stage IXU are 61% and 51%, respectively. This difference is attributed to the lack of FP units in IXU, as described in Sect. 2.4.

Similarly, Fig. 13 shows the IPC of HALF+FX relative to that of BIG when the FUs' depth is varied. This figure shows that the IPC increases with an increase in the depth. When the depth constitutes more than three stages, the IPC increase per depth is less than 1%. This is because of the fact that if the issue width of the OXU is sufficient, the effect of an IXU, similar to the effect of widening the issue width (Sect. 4.2.1), does not improve performance.

6.9 Evaluation on Performance-Centric Cores

Thus far we have evaluated models based on ARM big.LITTLE, and in this section, we show the evaluation results of FXA on *performance-centric cores*. The big cores in ARM big.LITTLE are designed for mobile devices, and thus, the maximum width of instruction execution is limited to four instructions for reducing energy consumption. In contrast, PCs and server systems are usually equipped with performance-centric cores such as Intel Haswell [3] and IBM POWER8 [2], which can execute eight or more instructions in a cycle and are much bigger than those in ARM big.LITTLE. In this section, we show how the performance and energy consumption of FXA are changed in such performance-centric cores. For simplicity, we refer to the big cores in ARM big.LITTLE as *efficiency-centric cores* in this section.

Table 3 Processor configurations (Performance-centric).

	BIG	HALF
fetch width	4 inst.	←
issue width	8 inst.	3 inst.
issue queue	80 entries	40 entries
FU (int, mem, fp)	4, 2, 2	←
ROB	180 entries	←
int/fp PRF	160/160 entries	←
ld/st queue	40/40 entries	←
L2C	2MB, 8 way, 64 B/line, 12 cycles	←

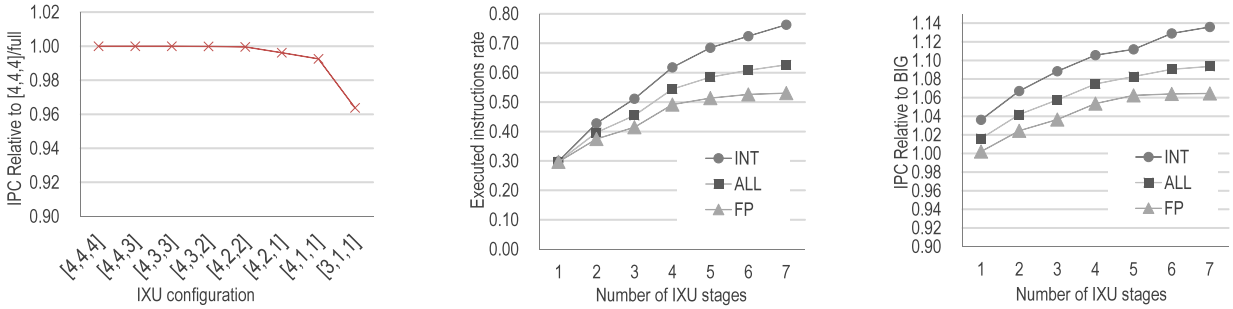
We evaluated the models presented in Sect. 6.2 except LITTLE, because the performance improvement is generally negligible when the size of an in-order superscalar processor is increased. Table 3 shows the detailed parameters of the evaluated models whose configurations are based on performance-centric cores such as Intel Haswell [3] and IBM POWER8 [2]. Other parameters such as an L1D cache configuration are not different from those listed in Table 1.

The FXA models have an IXU with a [4,1,1] configuration. This configuration differs from that in the efficiency-centric cores ([3,1,1]), because the fetch width is widened, and thus, optimal configuration changes. In the performance-centric cores, the fetch width is widened from three to four instructions compared with the efficiency-centric cores, and consequently, the available maximum width of the IXU is also widened to four. The [4,1,1] IXU has the highest performance among configurations whose bypass network complexity is similar to that of the bypass network of BIG in performance-centric cores[†]. The bypass network complexity is considered on the basis of the discussion on complexity in Sect. 3.1 in the same way as the efficiency-centric cores. The detailed sensitivity of the IXU configurations is described in Sect. 6.9.2.

6.9.1 IPC and Energy Consumption

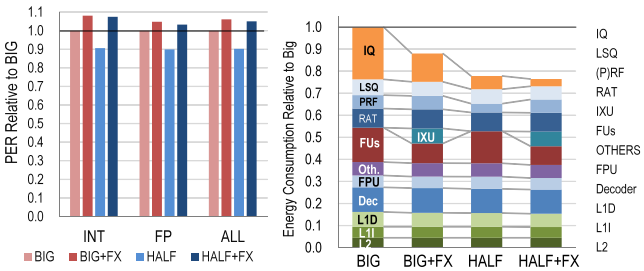
FXA based on performance-centric cores shows the same tendency of the results of the efficiency-centric cores in

[†]The bypass network of BIG in performance-centric cores is more complex than that of BIG in the efficiency-centric cores, because the number of functional units connected through the bypass network is increased.



(a) IPC versus IXU configurations. (b) Executed instructions rate in IXU. (c) IPC versus IXU stages.

Fig. 15 Sensitivity of IXU configurations on performance centric cores.



(a) IPC (b) Energy Consumption

Fig. 14 IPC/energy consumption relative to BIG on performance centric cores.

terms of performance and energy consumption. Figure 14 (a) shows the IPCs for each model relative to BIG. The IPC improvement of HALF+FX is 5.0% while the improvement is 5.7% in the efficiency-centric cores, and this shows that FXA can also function well with performance-centric cores. Figure 14 (b) shows the energy consumption for each model relative to BIG. The labels in this figure are the same as in Fig. 8 (a). HALF+FX reduces the energy consumption as compared to BIG by 24%, while the reduction is 17% in the efficiency-centric cores. This is because the configurations of the L1 caches are not changed, but hardware for dynamic instruction scheduling is relatively enlarged. FXA can reduce the energy consumption of such hardware; consequently FXA functions better in the performance-centric cores.

6.9.2 Sensitivity of IXU Configurations

In this section, we describe the sensitivity of the IXU configurations on the performance-centric cores. Figure 15 (a) shows the IPCs of HALF+FX compared to that of the HALF+FX with a [4,4,4] IXU configuration. The labels and lines in this figure are the same as in Fig. 11. This figure shows the IPCs of HALF+FX with the bypass network that omits operand-bypassing between FUs that are more distant than two stages, as described in Sect. 3.1.2. The IPC degradation of [4,1,1] as compared to [4,4,4] is slight (0.7%), but the IPC degradation of [3,1,1] as compared to [4,1,1] is not negligible (2.9%). Consequently, we use the [4,1,1] configuration in the evaluations of the performance-centric cores.

In addition, we evaluated HALF+FX on the performance-centric cores while varying the depth of FUs in the IXU from 1 to 7. It should be noted that the optimization of the IXU described in Sect. 3.1.2 is not applied to HALF+FX in this evaluation. Figure 15 (b) shows the rate of instructions executed in the IXU relative to all the executed instructions. Figure 15 (c) shows the IPC of HALF+FX relative to that of BIG when the FUs' depth is varied. The labels and lines in these figure are the same as in Figs. 12 and 13. Although these results show the same tendency of the results on the efficiency-centric cores, the executed rates and IPC improvements are smaller than those on the efficiency-centric cores. This is because the fetch width is widened compared to the efficiency-centric cores. When the fetch width is widened, it increases the probability that dependent instructions are in the same fetch group. These dependent instructions cannot be executed in the same stage of the IXU as described in Sect. 2.3, and consequently, the number of the executed instructions in the IXU is decreased. However, this degradation is not significant, and the IXU with three stages, which is used for the evaluations on the performance-centric cores, still can execute 45% instructions, while the executed rate is 54% in the efficiency-centric cores.

7. Related Work

We describe works related to the proposed method in this section.

7.1 Clustered Architecture

Both FXA and clustered architecture (CA), such as Alpha 21264 [13], have multiple execution units. The major difference between them is that the clusters in CA do not have an order relation, but the IXU and the OXU in FXA have an order relation as instructions not executed in the IXU go into the OXU. Consequently, FXA is simpler than CA in terms of the following:

- Operand Bypassing and Wakeup: It is necessary to bypass operands and wakeup instructions between the clusters in CA [13], and they require additional datapath and wakeup ports. These operand-bypassing and wakeup operations are performed across the clusters,

and thus, additional latencies are required. In contrast, it is not necessary to bypass operands and wakeup instructions between the IXU and the OXU, because they have an order relation, as described in Sect. 3.1.1.

- Instruction Steering: For mitigating additional latencies for communication between the clusters, it is important for CA that instructions are appropriately steered to the clusters [15]. If there is a CA with in-order/out-of-order clusters and instructions that remain not executed for a long time are steered to the in-order cluster, then its performance is significantly decreased. Consequently, more careful instruction steering and a complex logic are required. In contrast, the IXU and the OXU in FXA have an order relation, and thus, instruction steering is not necessary.

As described above, FXA is simpler than CA; moreover, FXA has more FUs than CA. As a result, FXA has a higher performance and lower energy consumption than does CA.

7.2 Heterogeneous Multicore

A heterogeneous multicore architecture improves energy efficiency with a hybrid system of out-of-order/in-order cores. One example of a heterogeneous multicore architecture is ARM big.LITTLE [6], [10], [11]. ARM big.LITTLE consists of big and little cores. Big cores have a higher performance than little cores, but big cores consume a larger amount of energy than little cores do. ARM big.LITTLE reduces the energy consumption by executing instructions with little cores when processing speed is not important (e.g., receiving e-mails in the background). It makes it possible to improve energy efficiency without compromising user experience.

Our goal is not to replace both the big and little cores by FXA cores but rather to replace only the big core by an FXA core. FXA has better PER than both the big and little cores, but the FXA core cannot replace both of the big and little cores. This is because the energy consumption of the little core for processing a single instruction is always smaller than that of the FXA and the big cores. For processing a single instruction, the little core consumes energy for steps such as fetch, decode, register access, and execute. In contrast, the FXA and big cores consume energy for additional steps such as rename and scheduling in addition to the energy consumed in the little core; consequently, the energy consumption of the FXA and big cores is always bigger than that of the little core. Thus, the little core is still useful when the smallness of energy consumption is important and high performance is not important. Therefore our goal is to replace only the big core by an FXA core. In this way, enjoying the energy optimization of big.LITTLE, application programs that require the high performance of big cores can be executed with lower energy consumption.

7.3 Processing Instructions in Front-End

RENO (which stands for RENaming Optimizer) [26] reduces the complexity of an execution core by removing irrelevant instructions on register renaming. It dynamically performs optimization, such as move elimination and common subexpression elimination. Both RENO and FXA reduce the number of instructions dispatched to the execution core by processing instructions in the front-end. The major difference between RENO and FXA is that RENO reduces the number of executed instructions itself by optimization in the front-end, and FXA actually executes instructions in the front-end. Optimization in RENO is implemented by modifying the renaming logic, and thus, this optimization can be implemented in FXA, and improved results can be achieved by combining them.

7.4 Array Processor

Researchers proposed *array processors*, which improve performance and energy efficiency by leveraging FUs placed as an array [27]–[30]. The array processors have a structure similar to that of the IXU in that functional units are serially placed over multiple stages.

The major difference between array processors and the IXU is whether instructions are statically assigned to FUs or dynamically flow over FUs.

Array processors generally target loop iterations, and instructions in iterations are statically assigned to FUs by compilers; that is, each FU always executes the same static institution. Consequently, when the length of an FU array is longer than that of a loop, the loop cannot be executed in the array processor. Moreover, when a loop includes a complex control flow graph with branches, a compiler cannot effectively map instructions to each FU.

In contrast, in the IXU, instructions flow over the FUs, as described in Sect. 2, and it does not statically map instructions to each FU. Thus, the IXU does not have the limitation of array processors assuming loop iterations, and it can flexibly execute a wide range of instructions in programs including a complex control flow without any compiler support.

7.5 Reducing Issue Queue Complexity

For directly reducing the complexity of an IQ, Forwardflow [8] was proposed. In Forwardflow, instructions are directly woken up through pointers, and thus, it omits CAMs or dependency matrices, and its energy consumption is therefore reduced.

As an approach that focuses on the number of source operands, Half Price Architecture [31] was proposed. Half Price Architecture focuses on the fact that many instructions have fewer than two source operands, and reduces the number of ports of the wakeup logic and the register file.

Both the approaches proposed in these related studies and FXA reduce the complexity of the IQs. The ma-

jor difference between them is that FXA reduces its energy consumption by executing instructions in the IXU and reducing the number of instructions dispatched to the issue queue. Moreover, these approaches can be applied to the IQ in FXA, and energy consumption is reduced further if they are combined.

8. Conclusion

Smartphones and tablets have recently become widespread and dominant in the computer market, and major developers have adopted out-of-order superscalar processors for these mobile devices. However, out-of-order superscalar processors consume much more energy than in-order superscalar processors. In this paper, we proposed FXA, which has two execution units, the IXU and OXU. The simple IXU operates as a filter for the complex OXU by executing instructions in the front-end. The IXU executes many instructions and reduces the number of instructions dispatched to the OXU. This makes it possible for FXA to achieve both high performance and low energy consumption. In a comparison with the models based on ARM big.LITTLE architecture, the evaluation results show that FXA achieves a 5.7% higher performance, 17% lower energy consumption, and 25% higher performance/energy ratio (the inverse of energy-delay product) than does a conventional superscalar processor, and a 27% higher performance/energy ratio than a conventional in-order superscalar processor. We also evaluated FXA with performance-centric superscalar cores after Intel Haswell [3] or IBM POWER8 [2], and FXA achieved IPC improvements of 5.0% while reducing the energy consumption by 24%.

Acknowledgments

This work was supported by JSPS KAKENHI Grant Number 24680005. We would like to thank Haruka Hirai, Kazuo Horio, and Yasuhiro Watari for support.

References

[1] R. Shioya, M. Goshima, and H. Ando, "A Front-end Execution Architecture for High Energy Efficiency," Proceedings of the International Symposium on Microarchitecture (MICRO), pp.419–431, 2014.
[2] B. Sinharoy, J.A. Van Norstrand, R.J. Eickemeyer, H.Q. Le, J. Leenstra, D.Q. Nguyen, B. Konigsburg, K. Ward, M.D. Brown, J.E. Moreira, D. Levitan, S. Tung, D. Hrusecky, J.W. Bishop, M. Gschwind, M. Boersma, M. Kroener, M. Kaltenbach, T. Karkhanis, and K.M. Fernsler, "IBM POWER8 Processor Core Microarchitecture," IBM J. Res. & Dev., vol.59, no.1, pp.2:1–2:21, 2015.
[3] K. Krewell, "Intel's Haswell Cuts Core Power," Microprocessor Report 9/24/12, pp.1–5, Sept. 2012.
[4] ARM, "The ARM Cortex-A9 Processors," ARM White Paper, 2007.
[5] L. Gwennap, "How Cortex-A15 Measures Up," Microprocessor Report 5/27/13-1, pp.1–7, 2013.
[6] J. Bolaria, "Cortex-A57 Extends ARM's Reach," Microprocessor Report 11/5/12-1, pp.1–5, 2012.
[7] D. Folegnani and A. González, "Energy-Effective Issue Logic," Proceedings of the International Symposium on Computer Architecture (ISCA), pp.230–239, June 2001.
[8] D. Gibson and D.A. Wood, "Forwardflow: A Scalable Core for Power-Constrained CMPs," Proceedings of the International Symposium on Computer Architecture (ISCA), pp.14–25, 2010.
[9] N.H.E. Weste and D.M. Harris, CMOS VLSI Design: A Circuits and Systems Perspective 4th Edition, Pearson/Addison-Wesley, 2011.
[10] K. Krewell, "Cortex-A53 Is ARM's Next Little Thing," Microprocessor Report 11/5/12-2, pp.1–4, 2012.
[11] P. Greenhalgh, "Big.LITTLE Processing with ARM Cortex-A15 and Cortex-A7," ARM White Paper, 2011.
[12] K.C. Yeager, "The MIPS R10000 Superscalar Microprocessor," IEEE Micro, vol.16, no.2, pp.28–41, 1996.
[13] R.E. Kessler, "The Alpha 21264 Microprocessor," IEEE Micro, vol.19, no.2, pp.24–36, 1999.
[14] B. Sinharoy, R. Kalla, W.J. Starke, H.Q. Le, R. Cargnoni, J.A. Van Norstrand, B.J. Ronchetti, J. Stuecheli, J. Leenstra, G.L. Guthrie, D.Q. Nguyen, B. Blaner, C.F. Marino, E. Retter, and P. Williams, "IBM POWER7 Multicore Server Processor," IBM J. Res. Dev., vol.55, no.3, pp.191–219, 2011.
[15] S. Palacharla, N.P. Jouppi, and J.E. Smith, "Quantifying the Complexity of Superscalar Processors," Technical Report, University of Wisconsin-Madison, 1996.
[16] G.Z. Chrysos and J.S. Emer, "Memory Dependence Prediction Using Store Sets," Proceedings of the International Symposium on Computer Architecture (ISCA), vol.26, no.3, pp.142 –153, 1998.
[17] S.B. Wijeratne, N. Siddaiah, S.K. Mathew, M.A. Anders, R.K. Krishnamurthy, J. Anderson, M. Ernest, and M. Nardin, "A 9-GHz 65-nm Intel Pentium 4 Processor Integer Execution Unit," IEEE J. Solid-State Circuits, vol.42, no.1, pp.26–37, 2007.
[18] J.-L. Cruz, A. González, M. Valero, and N.P. Topham, "Multiple-Banked Register File Architecture," Proceedings of the International Symposium on Computer Architecture (ISCA), pp.316–325, 2000.
[19] R. Shioya, K. Horio, M. Goshima, and S. Sakai, "Register Cache System Not for Latency Reduction Purpose," Proceedings of the International Symposium on Microarchitecture (MICRO), pp.301–312, 2010.
[20] S. Kao, R. Zlatanovici, and B. Nikolic, "A 240ps 64b Carry-Lookahead Adder in 90nm CMOS," Proceedings of the International Solid-State Circuits Conference (ISSCC), pp.1735–1744, 2006.
[21] S.R. Vangal, Y.V. Hoskote, N.Y. Borkar, and A. Alvandpour, "A 6.2-GFlops Floating-Point Multiply-Accumulator with Conditional Normalization," Journal of Solid-State Circuits, vol.41, no.10, pp.2314–2323, 2006.
[22] The Standard Performance Evaluation Corporation, SPEC CPU2006 Suite.
[23] S. Li, J.H. Ahn, J.B. Brockman, and N.P. Jouppi, "McPAT 1.0: An Integrated Power, Area, and Timing Modeling Framework for Multicore Architecture," technical report hpl-2009-206, HP Laboratories, 2009.
[24] Semiconductor Industries Association, Model for Assessment of CMOS Technologies and Roadmaps (MASTAR) http://www.itrs.net/models.html, 2007.
[25] C. Auth, C. Allen, A. Blattner, D. Bergstrom, M. Brazier, M. Bost, M. Buehler, V. Chikarmane, T. Ghani, T. Glassman, R. Grover, W. Han, D. Hanken, M. Hattendorf, P. Hentges, R. Heussner, J. Hicks, D. Ingerly, P. Jain, S. Jaloviar, R. James, D. Jones, J. Jopling, S. Joshi, C. Kenyon, H. Liu, R. McFadden, B. McIntyre, J. Neirynck, C. Parker, L. Pipes, I. Post, S. Pradhan, M. Prince, S. Ramey, T. Reynolds, J. Roesler, J. Sandford, J. Seiple, P. Smith, C. Thomas, D. Towner, T. Troeger, C. Weber, P. Yashar, K. Zawadzki, and K. Mistry, "A 22nm High Performance and Low-power CMOS Technology Featuring Fully-depleted Tri-gate Transistors, Self-aligned Contacts and High density MIM Capacitors," Symposium on VLSI Technology (VLSIT), pp.131–132, 2012.
[26] V. Petric, T. Sha, and A. Roth, "Reno: A Rename-Based Instruction Optimizer," Proceedings of the International Symposium on Com-

puter Architecture (ISCA), pp.98–109, 2005.
[27] B. Mei, S. Vernalde, D. Verkest, and R. Lauwereins, "Design Methodology for a Tightly Coupled VLIW/Reconfigurable Matrix Architecture: A Case Study," Proceedings of the Design, Automation and Test in Europe Conference and Exhibition, pp.1224–1229 vol.2, 2004.
[28] F. Bouwens, M. Berekovic, B. De Sutter, and G. Gaydadjiev, "Architecture Enhancements for the ADRES Coarse-Grained Reconfigurable Array," High Performance Embedded Architectures and Compilers, pp.66–81, 2008.
[29] N. Devisetti, T. Iwakami, K. Yoshimura, T. Nakada, J. Yao, and Y. Nakashima, "LAPP: A Low Power Array Accelerator with Binary Compatibility," Proceedings of the International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), pp.854–862, 2011.
[30] J. Yao, Y. Nakashima, N. Devisetti, K. Yoshimura, and T. Nakada, "A Tightly Coupled General Purpose Reconfigurable Accelerator LAPP and Its Power States for HotSpot-Based Energy Reduction," vol.E97-D, no.12, pp.3092–3100, 2014.
[31] I. Kim and M.H. Lipasti, "Half-Price Architecture," Proceedings of the International Symposium on Computer Architecture (ISCA), pp.28–38, 2003.

Ryota Shioya was born in 1981. He received his M.E. and Ph.D. 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 to 2011. Since 2011, he is an assistant professor at the Graduate School of Engineering, Nagoya University. He is a member of IEICE, IPSJ, and IEEE.

Ryo Takami received his B.E. degree from Nagoya Institute of Technology, Nagoya, Japan, in 2012. He received his M.E. degree from Nagoya University, Nagoya, Japan, in 2014. Since then, he has been with MegaChips Corporation.

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 National Institute of Informatics. He has been engaging in the research area of computer architecture. 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 and IEEE.

Hideki Ando received his B.S. and M.S. degrees in Electronic Engineering from Osaka University, Suita, Japan, in 1981 and 1983, respectively. He received his Ph.D. degree in Information Science from Kyoto University, Kyoto, Japan, in 1996. From 1983 to 1997, he was with Mitsubishi Electric Corporation, Itami, Japan. From 1991 to 1992, he was a visiting scholar at Stanford University. In 1997 he joined the faculty of Nagoya University, Nagoya, Japan, where he is currently a professor in the Department of Electrical Engineering and Computer Science. He received IPSJ best paper awards in 1998 and 2002, and a best paper award at the Symposium on Advanced Computing Systems and Infrastructures in 2013. His research interests include computer architecture and compilers.