2014 47th Annual IEEE/ACM International Symposium on Microarchitecture

A Front-end Execution Architecture for High Energy Efficiency

Ryota Shioya*, Masahiro Goshima† and Hideki Ando*

* *Department of Electrical Engineering and Computer Science, Nagoya University, Aichi, Japan*
† *Information Systems Architecture Research Division, National Institute of Informatics, Tokyo, Japan*
Email: {shioya, ando}@nuee.nagoya-u.ac.jp, goshima@nii.ac.jp

***Abstract*—Smart phones and tablets have recently become widespread and dominant in the computer market. Users require that these mobile devices provide a high-quality experience and an even higher performance. Hence, major developers adopt out-of-order superscalar processors as application processors. However, these processors consume much more energy than in-order superscalar processors, because a large amount of energy is consumed by the hardware for dynamic instruction scheduling. We propose a Front-end Execution Architecture (FXA). 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 comprises functional units and a bypass network only. The IXU is placed at the processor front end and executes instructions without scheduling. Fetched instructions are first fed to the IXU, and the instructions that are already ready or become ready to execute by the resolution of their dependencies through operand bypassing in the IXU are executed in-order. Not-ready instructions go through the IXU as a NOP; thereby, its pipeline is not stalled, and instructions keep flowing. The not-ready instructions are then dispatched to the OXU, and are executed out-of-order. The IXU does not include dynamic scheduling logic, and its energy consumption is consequently small. Evaluation results show that FXA can execute over 50% of instructions using IXU, thereby making it possible to shrink the energy-consuming OXU without incurring performance degradation. As a result, FXA achieves both a high performance and low energy consumption. We evaluated FXA compared with conventional out-of-order/in-order superscalar processors after ARM big.LITTLE architecture. The results show that FXA achieves performance improvements of 67% at the maximum and 7.4% on geometric mean in SPECCPU INT 2006 benchmark suite relative to a conventional superscalar processor (big), while reducing the energy consumption by 86% at the issue queue and 17% in the whole 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).**

***Keywords*-Core Microarchitecture, Hybrid In-Order/Out-of-Order Core, Energy Efficiency**

I. Introduction

The single thread performance of a core remains important, even in the era of multi-core processors, and is considered to be important even for mobile devices, such as smart phones and tablets, which have become widespread and dominant in the computer market. To provide high single thread performance, major developers have adopted out-of-order superscalar processors in these mobile devices. For example, iPhone, iPad, and major Android devices are equipped with out-of-order superscalar processors, such as ARM Cortex A9 and its successors [1], [9], [3]. The applications in these devices have become increasingly complex in order to provide a high-quality user experience and they require an ever higher processor performance. These applications are usually implemented on HTML5 with JavaScript or application virtual machines. They are generally slower than native binary applications, and it is essentially difficult to parallelize them. Consequently, these mobile devices are equipped with out-of-order superscalar processors with a high performance.

Although the performance level of out-of-order superscalar processors is high, they consume much more energy than do in-order superscalar processors, because a large amount of energy is consumed by hardware for dynamic instruction scheduling [6], [7], such as an issue queue (IQ) and a load/store queue (LSQ), which comprises mainly 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 [23]. Moreover, the number of accesses is also increased with the issue width. Consequently, its energy consumption is very large.

We propose a *Front-end Execution Architecture (FXA)*. FXA has two execution units, an *out-of-order execution unit (OXU)* and an *in-order execution unit (IXU)*, using which it achieves a higher performance and lower energy consumption than do conventional out-of-order superscalar processors. The OXU is the execution core of a common out-of-order superscalar processor, which includes several components, such as an IQ and functional units (FUs). In contrast, the IXU comprises FUs and a bypass network only and is placed in the processor front end. 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 also executes instructions whose dependency is dissolved in it, in addition to instructions that are ready to execute at reading source operands. The IXU can thereby execute many instructions. Moreover, in the IXU, FUs are placed over multiple stages and this allows the IXU to

1072-4451/14 $31.00 © 2014 IEEE
DOI 10.1109/MICRO.2014.35

execute more instructions (Section II). The evaluation results presented in Section VI show that over 50% of instructions are executed in the IXU.

This execution in the IXU reduces the energy consumption of the OXU. In particular, the energy consumption of the IQ is greatly reduced. Since the IXU can execute many instructions, the capacity and issue width of the IQ is reduced without performance degradation being incurred. Moreover, the number of accesses to the IQ is greatly reduced, because instructions executed in the IXU are not dispatched to the OXU. The evaluation results presented in Section VI show that the energy consumption of the IQ is reduced by 86%. The energy consumption of the LSQ is also reduced, because the detection of the order violation of load/store instructions and the writing of load instructions to the LSQ are partially omitted (details are described in Section V).

In addition to reducing energy consumption, the instruction execution 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.

As a result, FXA achieves both high performance and low energy consumption. We evaluated FXA compared with conventional out-of-order/in-order superscalar cores after ARM big.LITTLE architecture [13], [3], [8]. The evaluation results show that FXA achieves IPC improvements of 5.7% (and 7.4% with integer benchmarks only) on geometric mean in SPECCPU 2006 benchmark suite to the big core, while reducing the energy consumption 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.

However, our goal is not to replace both of the big and little cores by FXA cores but rather to replace only the big core by an FXA core as described in Section VI-I. In this way, enjoying the energy optimization of big.LITTLE, application programs that require high-performance of big cores can be executed with lower energy consumption.

The rest of the paper is organized as follows. In Section II, the basic characteristics of FXA are described, and in Section III, its details and optimization. In Section IV and Section V, its performance and energy consumption are given. In Section VI, evaluation results are presented. In Section VII, related work is summarized.

II. 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.

A. Structure

We describe the structure of FXA by comparing it to that of a conventional out-of-order superscalar processor.

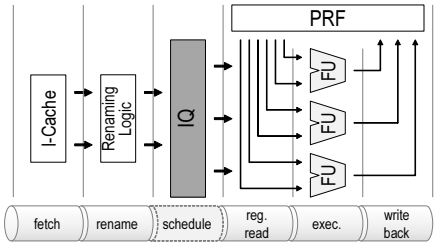


Figure 1: Conventional superscalar architecture.

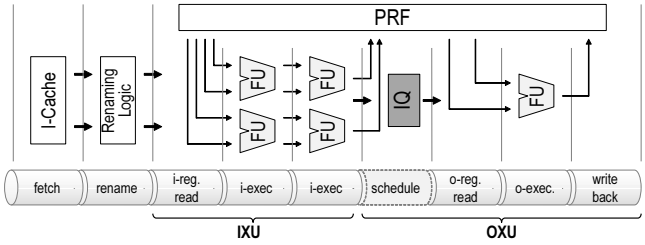


Figure 2: Front-end execution architecture.

Figure 1 shows the pipeline and block diagram of a conventional out-of-order superscalar processor. This figure shows a physical register-based architecture [25], [11], [20], [14]. Hereafter, the term "conventional superscalar processor" refers to this architecture. In contrast, Figure 2 shows the pipeline and the block diagram of FXA. FXA has two execution units:

1) **Out-of-order execution unit (OXU)**: This unit is the execution core of a conventional superscalar processor. In Figure 2, the element placed after the "schedule" stage is an OXU.
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 Figure 2, the IXU is placed between the rename stage and the dispatch 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 using these, instructions are executed *in-order*.

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 [25] [1]. Hereafter, the term "scoreboard" refers to the PRF scoreboard.

B. 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 Section II-D below.

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 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[2].
   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 [11], [16]. This is because serial placement greatly reduces the complexity of bypassing, wake-up, and steering as described in Section III and VII-A. 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.

[1]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 [25].

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

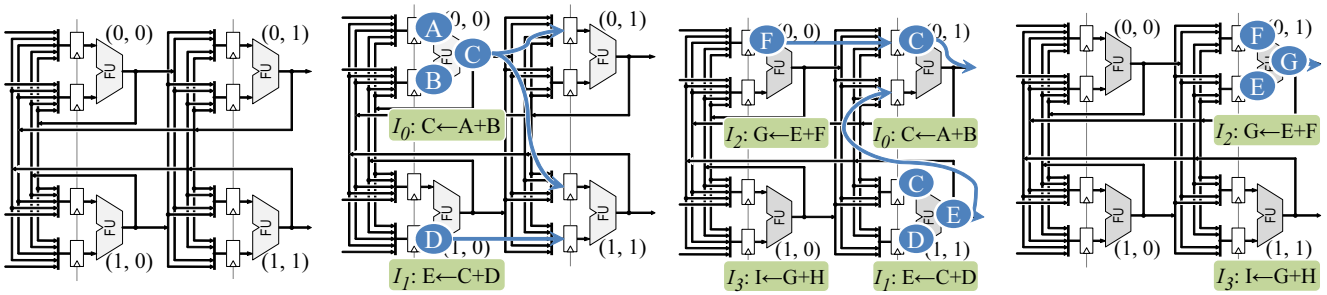
C. Detailed Behavior of IXU

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

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 3a shows a datapath example of an IXU with the FUs of 2-instruction width $\times$ 2 stages. In this figure, $FU(y, x)$ denotes an FU at the x-th position from the left and y-th from the top. The FUs are connected through the bypass network that allows each FU to use each other's execution results. In Figure 3a, 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 using an example where the code shown in Figure 4 is executed on the IXU shown in Figure 3a. The code includes the serially dependent instructions from I_0 to I_3. All the source operands, except C, E, and G, shown in Figure 4, have already been read from the PRF. The behavior of each cycle is as follows.

- **Cycle 1:** Figure 3b 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 3c shows the cycle that follows the cycle shown in Figure 3b. In Figure 3c, 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 3d 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.



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

Figure 3: In-order execution unit.

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

Figure 4: Code executed in IXU.

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 shown in Figure 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 Figure 3d. Generally, dependent instructions are rarely placed in a long and consecutive chain[3], and consequently, an IXU can execute many instructions.

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.

D. 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.

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

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) 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.

3) Load/Store: FXA assumes a scheme that issues loads/store instructions speculatively using a dependency predictor [4], [11], such as the store-set predictor [4]. 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 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 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[4]. 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. Thereby, the pipeline is not stalled in this case, and the performance degradation is small.

[4]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.

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:

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 omission reduces the energy consumption of the LSQ, as described in Section V-D.

III. Detailed Design and Optimization

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

A. Operand Bypassing

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

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 $\rightarrow$ 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 Figure 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 $\rightarrow$ 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 are separated, and its complexity and energy consumption are similar to those of the bypass network in conventional superscalar processors, as described below.

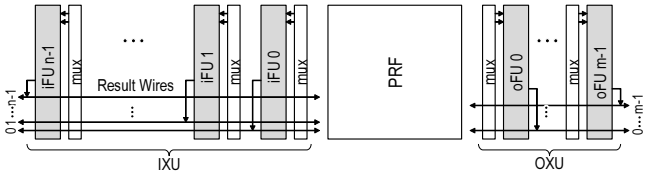


Figure 5: Bit-slice of bypass network.

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 Section VI show that the performance of a configuration with $3ways \times 1stage + 1way \times 2stages = 5$ FUs is almost same as that of a configuration with $3ways \times 3stages = 9$ FUs.

Additionally, we partially omit operand-bypassing in the IXU. As described in Section II-C, 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 Section VI show that performance degradation is negligible when operand-bypassing between FUs that are more distant than two stages is omitted.

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 [16]. Figure 5 shows the layout of the bypass network [16], [24] that we assume in this paper. This figure shows a bit-slice of the datapath. There are n FUs in the IXU, which are from iFU_0 to iFU_{n-1}, and m FUs in the OXU, which are 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 Section III-A1, 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 Figure 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 Section VI. 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.

B. Physical Register File

In the layout presented in Figure 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 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, and 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 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[23]. 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 (Section II-A) 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 Section VI.

Note that, in this paper, 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 the ports is not significantly increased. This is because the number of the 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[5], [19], can mitigate the complexity of the increased ports.

C. Scoreboard

FXA reads the scoreboard twice per instruction to provide correct execution. The first reading is carried out before instructions go into the IXU (Section II-B), and the second reading is carried out in a dispatch stage (Section II-A). These two readings cannot be combined into one time. This is because there is a possibility that a not ready instruction that goes through the IXU as NOPs becomes ready when its 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.

IV. Performance of FXA

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

A. Instructions Executed in IXU

As described in Section II, instructions executed in the IXU are categorized as:

- **(a)** Instructions that are already ready when they are entered to 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 Section VI, 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 Section VI show that the IXU with one-stage FUs can execute 35% of instructions. Moreover, FXA can execute more instructions by serially placing FUs over multiple stages in the IXU, as described in Section II-C. This makes it possible to increase the number of (b) significantly. The evaluation results presented in Section VI show that an IXU with three-stage FUs can execute 54% of instructions.

B. 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.

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 Section VI, 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 Section VI. This is because FP applications still include many integer and load/store instructions [5].

2) Reducing Branch Misprediction Penalty: FXA can execute branch instructions and detect branch misprediction in the IXU (Section II-D). 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 [20], [3], [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 the stages of the IXU, which is usually four or five, because its pipeline length is prolonged. However, more than 50% of instructions are executed in the IXU (Section VI), and thus, the total penalty is reduced.

V. 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.

A. 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.

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.

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

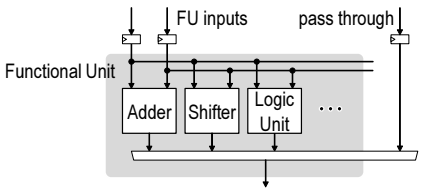


Figure 6: Functional Unit in IXU

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 [24]. When source operands or execution results are sent to the next stage in the IXU, the path for passing through in 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 larger amount of static energy. However, the number of the transistors in the integer FUs for the IXU is much smaller than that of the 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, comprises 4k − 6k transistors [10], [22]. In contrast, an FP multiplier and an FP adder, which require fast transistors as integer FUs, comprise over 200 k transistors[22], which is several tens times of that in integer adders. Consequently, the static energy consumed by the few integer FUs for the IXU is negligible.

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 Section III-A, the bypass networks in the IXU and the OXU comprise mainly result wires and multiplexers[6]. 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 the FUs in the layout shown in Figure 5. Consequently, the energy consumption

[6]The result wires of the IXU and the OXU are separated (Section III-A).

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 Section VI are used, in FXA, the IXU has $n = 5$ FUs and the OXU has $m = 4$ FUs. In this configuration, over 50% of 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 ($m = 4$)[7].

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 Section III-A2, 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.

B. 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 the 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 Figure 1 and FXA in Figure 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 Section II-A. As a result, the number of the 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.

C. 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 the ports being proportional to the issue width. Their energy consumption (and area) is proportional to the number of their ports and their capacities [23]. Consequently, the reduction in the capacity and issue width significantly reduces the energy consumption of the IQ per access. Moreover, the number of the accesses is also significantly reduced, because instructions executed in the IXU are not dispatched to the OXU. The evaluation results presented in Section VI show that the energy consumption of the IQ is reduced to 14% of that of the IQ of a conventional superscalar processor.

D. Load/Store Queue

The LSQ also comprises mainly CAMs and RAMs, and it consumes a large amount of energy. As described in Section II-D3, 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.

VI. Evaluation

We evaluated FXA and other processor architectures.

A. Evaluation Environment

We evaluated IPCs using an in-house cycle-accurate processor simulator. We used all the programs from the SPEC

Table I: 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	←	←

Table II: Device Configurations

technology	22 nm, Fin-FET[18]
temperature	320 K
VDD	0.8 V
device type (core)	high performance (I_off: 127 nA/um)
device type (L2)	low standby power (I_off: 0.0968 nA/um)

[7] All these FUs are integer FUs.

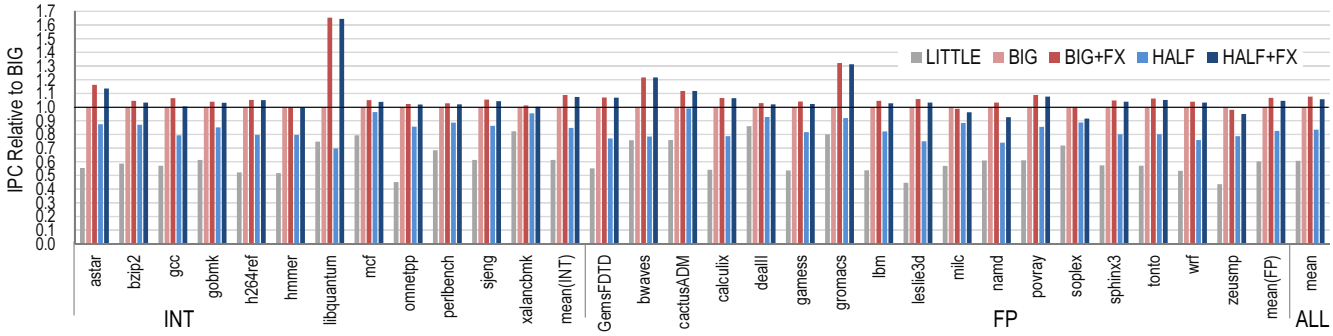


Figure 7: IPC relative to BIG.

CPU 2006 benchmark suites [21] with *ref* data sets. The programs were compiled using gcc 4.5.3 with the "-O3" option. We skipped the first 4G instructions and evaluated the next 100 M instructions. We evaluated energy consumption and chip areas using the McPAT simulator [15] with the parameters shown in Table II.

B. 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 [3] and Cortex-A53 [13]. Table I shows the detailed parameters of the evaluated models.

- **BIG:** BIG is the baseline model of this evaluation. It is 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 those in BIG. Its other parameters are the same as 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 the 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 Section III-B. 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 Section III-A2, 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. It is determined on the basis of the discussion about complexity in Section III-A.
- **BIG+FX:** This is a model of FXA. It has an IQ whose issue width and capacity are the same as 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.

C. 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.

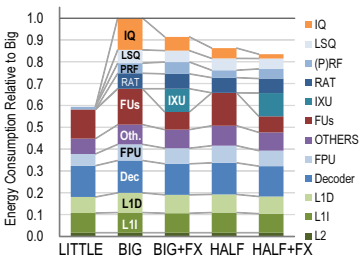
In libquantum and gromacs, HALF+FX significantly improves the IPC compared with BIG. This is because HALF+FX can execute more INT operations compared with BIG in a single cycle. In this case, the term "INT operations" include logical, add/sub, shift, and branch instructions and not include load/store instructions. In BIG, the maximum number of INT operations executed in a single cycle is two operations because the number of INT FUs is two. In contrast, HALF+FX can execute upto seven INT operations in a single cycle[8]. libquantum and gromacs include significantly more INT operations (over 80%) than the other applications include (50% on average). Consequently, HALF+FX with high INT-operation-throughput significantly improves performance in the programs[9].

The IPC degradation of HALF as compared to BIG is significant: 16% on geometric mean. This is because the width and size of IQ in HALF are half of those in BIG.

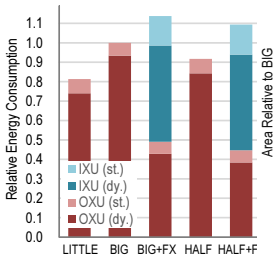
HALF+FX improves the IPC as compared to HALF by 27% on geometric mean, which is significant, although HALF+FX has the same IQ as HALF. HALF+FX can be considered to be the combination of HALF and an additional

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

[9]This high INT-operation-throughput is achieved by the IXU without widening the width of the IQ (Section IV-B1).

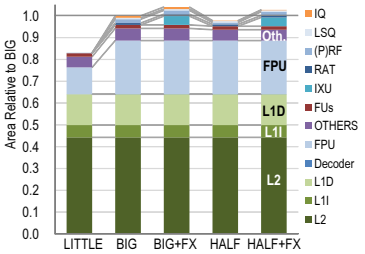


(a) Whole

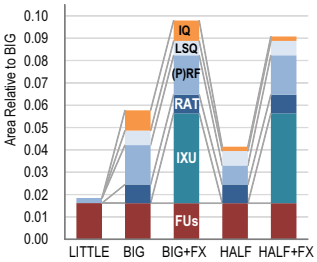


(b) FUs and bypass networks

Figure 8: Energy consumption relative to BIG.



(a) Whole



(b) FUs to IQ

Figure 9: Circuit area relative to BIG.

IXU. This shows that the addition of the IXU significantly improves the IPC, as described in Section IV-B.

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 sufficient 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.

D. Energy Consumption

Figure 8a 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" is the energy consumption of the FUs and the bypass network in the OXU. Similarly, "IXU" is the energy consumption of the FUs and the bypass network in the IXU. "OTHERS" is 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 Section V. 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 those 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 Section II-D3. 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 Section VI-E. 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 those 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 II. Fin-FET technology significantly reduces leakage current[2]. Furthermore, 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 II. 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.

E. Energy Consumption of FUs and Bypass Networks

Figure 8b 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 those 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 with that in BIG, but this does not cause a serious overall problem as described above.

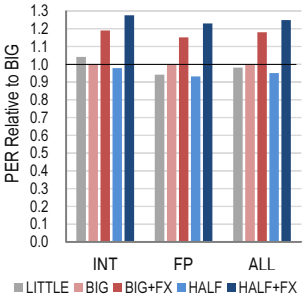


Figure 10: Performance/energy ratio.

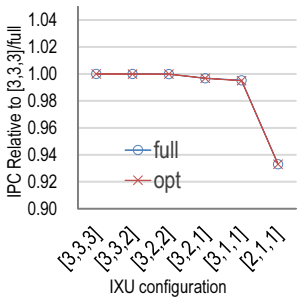


Figure 11: IPC versus IXU configurations.

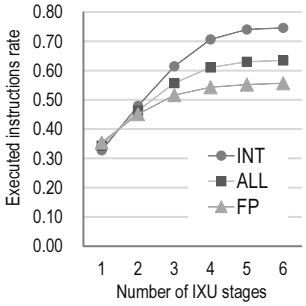


Figure 12: Executed instructions rate in IXU.

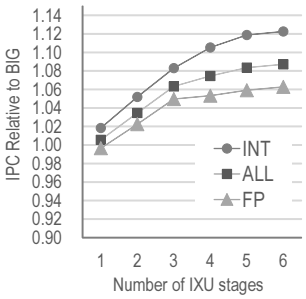


Figure 13: IPC versus IXU stages.

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 Sections V-A1 and VI-F.

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.

F. Circuit Area

We evaluated the circuit areas of the models. Figure 9a shows the areas of a whole processor for the other models relative to that of BIG. The labels in this figure are the same as in Figure 8a. The areas of the several units shown in the upper part of Figure 9a are not clear to see, and thus, Figure 9b shows the areas of those units[10]. The area of HALF+FX is increased by the addition of the IXU, as shown in Figure 9b. However, the area of the IXU is significantly smaller than that of the whole processor, as shown in Figure 9a, 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 the transistors in the IXU is small compared to that of transistors in the whole processor (Section V-A1), 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[11], and thus, their areas are the same.

[10]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 (Section V-C).

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

G. 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.

H. Sensitivity

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

1) Optimization of IXU: We evaluated the effect of the optimization of the IXU described in Section III-A2. 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 Section III-A2 is negligible.

2) Function Units Stages in IXU: We evaluated HALF+FX while varying the FUs depth in the IXU from 1 to 6. It should be noted that the optimization of the IXU described in Section III-A2 is not applied to HALF+FX in this section. Figure 12 shows the rate of instructions executed in the IXU relative to all executed instructions (hereafter, in this section, "executed rate" refers to this rate). Each line in Figure 12 shows the executed rate of the geometric mean of the INT benchmark group, FP benchmark group, and all benchmark programs, respectively. Figure 12 shows that the executed rate increases with the increase in the

depth. The figure shows that HALF+FX can execute many instructions, and that with the one-stage IXU it executes 35% of instructions on geometric mean. HALF+FX with the three-stage IXU executes more than half of instructions; the executed rate is 54%. The executed rate in the INT benchmark group is significantly higher than that in the FP benchmark group. The executed rates for the INT and FP benchmark groups in HALF+FX with the three-stage IXU are 61% and 51%, respectively. This difference is due to the IXU not having FP units, as described in Section II-D.

Similarly, Figure 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 the increase in the depth. When the depth constitutes more than three stages, the IPC increase per one depth is less than 1%. This is because, if the issue width of the OXU is sufficient, the effect of an IXU, similarly to widening the issue width (Section IV-B1), does not improve performance.

I. Evaluation Summary and Discussion

The above evaluation results shows that FXA achieves performance improvements of 5.7% relative to the conventional out-of-order superscalar processor (big core), while reducing the energy consumption 17%. The PER of FXA is 25% higher than that of the big core and 27% higher than that of the conventional in-order superscalar processor (little core).

FXA has better PER compared with both of the big and little cores, but 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 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, and consequently, the energy consumption of the FXA and big cores is always bigger than that of the little core. Thus, the little core is useful when the smallness of energy consumption is important and high-performance is not important.

Our goal is not to replace both of the big and little cores by FXA cores but rather to replace only the big core by FXA core. In this way, enjoying the energy optimization of big.LITTLE, application programs that require high-performance of big cores can be executed with lower energy consumption.

VII. Related Work

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

A. Clustered Architecture

Both FXA and clustered architecture (CA), such as Alpha 21264 [11], 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 as follows:

- Operand Bypassing and Wakeup: It is necessary to bypass operands and wakeup instructions between the clusters in CA [11], 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 order relation, as described in Section III-A1.
- Instruction Steering: For mitigating additional latencies for communication between the clusters, the method used for steering instructions to the clusters for CA [16]. If there is a CA with in-order/out-of-order clusters and instructions that remain not executed for long time are steered to the in-order cluster, and 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 consequently, instruction steering is not necessary.

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

B. Reducing Issue Queue Complexity

For directly reducing the complexity of an IQ, Forwardflow [7] 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 [12] was proposed. Half Price Architecture focuses on the fact that many instructions have fewer than two source operands, and reduces the number of the 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 major 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.

C. Processing Instructions in Front End

RENO (RENaming Optimizer) [17] 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.

VIII. Conclusion

Smart phones 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 OXU. This makes it possible for FXA to achieve both a 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 27% higher performance/energy ratio than a conventional in-order superscalar processor.

Acknowledgment

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

References

[1] ARM, "The ARM Cortex-A9 Processors," *ARM White Paper*, 2007.

[2] C. Auth *et al.*, "A 22nm High Performance and Low-power CMOS Technology Featuring Fully-depleted Tri-gate Transistors, Self-aligned Contacts and High density MIM Capacitors," in *Symposium on VLSI Technology (VLSIT)*, 2012, pp. 131–132.

[3] J. Bolaria, "Cortex-A57 Extends ARM's Reach," *Microprocessor Report 11/5/12-1*, pp. 1–5, 2012.

[4] G. Chrysos and J. Emer, "Memory Dependence Prediction Using Store Sets," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 1998, pp. 142 –153.

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

[6] D. Folegnani and A. González, "Energy-Effective Issue Logic," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2001, pp. 230–239.

[7] D. Gibson and D. A. Wood, "Forwardflow: A Scalable Core for Power-Constrained CMPs," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2010, pp. 14–25.

[8] P. Greenhalgh, "Big.LITTLE Processing with ARM Cortex-A15 and Cortex-A7," *ARM White Paper*, 2011.

[9] L. Gwennap, "How Cortex-A15 Measures Up," *Microprocessor Report 5/27/13-1*, pp. 1–7, 2013.

[10] S. Kao, R. Zlatanovici, and B. Nikolic, "A 240ps 64b Carry-Lookahead Adder in 90nm CMOS," in *Proceedings of the International Solid-State Circuits Conference (ISSCC)*, 2006, pp. 1735–1744.

[11] R. Kessler, "The Alpha 21264 microprocessor," *IEEE Micro*, vol. 19, no. 2, pp. 24–36, 1999.

[12] I. Kim and M. Lipasti, "Half-Price Architecture," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2003, pp. 28–38.

[13] K. Krewell, "Cortex-A53 Is ARM's Next Little Thing," *Microprocessor Report 11/5/12-2*, pp. 1–4, 2012.

[14] K. Krewell, "Intel's Haswell Cuts Core Power," *Microprocessor Report 9/24/12*, pp. 1–5, September 2012.

[15] 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," HP Laboratories, Technical Report HPL-2009-206, 2009.

[16] S. Palacharla, N. P. Jouppi, and J. E. Smith, "Quantifying the Complexity of Superscalar Processors," University of Wisconsin-Madison, Tech. Rep., Nov 1996.

[17] V. Petric, T. Sha, and A. Roth, "Reno: A Rename-Based Instruction Optimizer," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2005, pp. 98–109.

[18] *Model for Assessment of CMOS Technologies and Roadmaps (MASTAR) http://www.itrs.net/models.html*, Semiconductor Industries Association, 2007.

[19] R. Shioya, K. Horio, M. Goshima, and S. Sakai, "Register Cache System not for Latency Reduction Purpose," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2010, pp. 301–312.

[20] 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.

[21] *SPEC CPU2006 Suite*, The Standard Performance Evaluation Corporation. [Online]. Available: http://www.spec.org/cpu2006/

[22] S. Vangal, Y. Hoskote, N. 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.

[23] N. H. E. Weste and D. M. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective 4th Edition*. Pearson/Addison-Wesley, 2011.

[24] S. Wijeratne, N. Siddaiah, S. Mathew, M. Anders, R. Krishnamurthy, J. Anderson, M. Ernest, and M. Nardin, "A 9-GHz 65-nm Intel reg; Pentium 4 Processor Integer Execution Unit," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 1, pp. 26–37, 2007.

[25] K. Yeager, "The MIPS R10000 Superscalar Microprocessor," *IEEE Micro*, vol. 16, no. 2, pp. 28–41, 1996.