A Tightly Coupled Heterogeneous Core with Highly Efficient Low-Power Mode

Yasumasa Chidai[1](✉), Kojiro Izuoka[1], Ryota Shioya[1], Masahiro Goshima[2], and Hideki Ando[1]

[1] Nagoya University, Nagoya, Aichi, Japan
{chidai,izuoka}@ando.nuee.nagoya-u.ac.jp,
{shioya,ando}@nuee.nagoya-u.ac.jp
[2] National Institute of Informatics, Tokyo, Japan
goshima@nii.ac.jp

Abstract. A tightly coupled heterogeneous core (TCHC) has heterogeneous execution units with different characteristics inside the core. The composite core (CC) and the front-end execution architecture (FXA) are examples of state-of-the-art TCHCs. These TCHCs have in-order and out-of-order execution units in the core. They selectively execute instructions in-order and it improves the energy efficiency without significant performance degradation compared to out-of-order execution. However, these TCHCs cannot improve the energy efficiency sufficiently. CC has a large switching penalty of the execution units, and thus, CC cannot sufficiently execute instructions in-order. FXA cannot suspend energy consuming out-of-order execution units when it executes instructions in-order. We propose a dual-mode frontend execution architecture (DM-FXA), which is based on the FXA. DM-FXA has our proposed low-power execution mode, which completely suspends the out-of-order execution unit on in-order execution, and thus, DM-FXA consumes less energy than does the FXA. In addition, DM-FXA has a smaller switching penalty than CC. In our evaluation, the proposed methods reduce energy consumption by 34.7% compared with a conventional out-of-order processor, and performance degradation is within 3.2%.

1 Introduction

A *heterogeneous multicore (HMC)* is an effective method for improving the energy efficiency of processors [1–5]. HMCs consist of multiple cores with different performance and energy-efficiency characteristics. HMCs execute each program phase using the most energy-efficient core by switching the active core. ARM big.LITTLE [5] is a commercialized example of an HMC.

However, because each core has dedicated caches and predictors, core switching causes significant penalty cycles. As a result, core switching granularity is restricted to be long intervals (e.g., 100M instructions) [1].

To reduce the switching penalty, a *tightly coupled heterogeneous core (TCHC)* was proposed [6–12]. A typical TCHC has two execution units, in-order (InO)

M. Berekovic et al. (Eds.): ARCS 2018, LNCS 10793, pp. 211–224, 2018.
https://doi.org/10.1007/978-3-319-77610-1_16

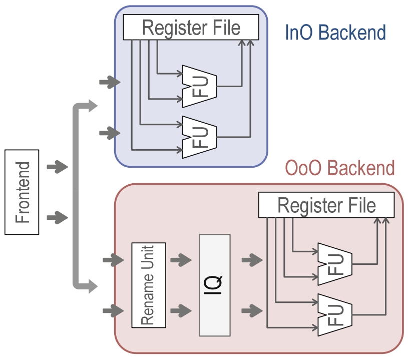

Fig. 1. Block diagram of CC.

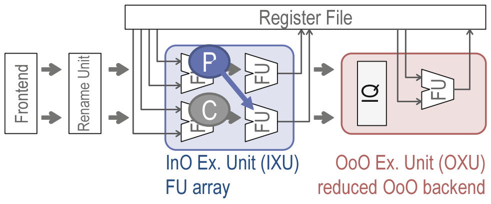

Fig. 2. Block diagram of FXA.

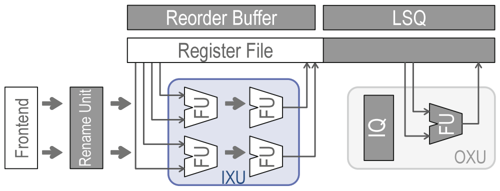

Fig. 3. LP mode in DM-FXA.

and out-of-order (OoO), in a single core. Because the caches and predictors are shared by the units in the core, the TCHC enables the fine-grained switching of execution units, and thus, it improves energy efficiency.

The *composite core (CC)* [6,7] is an example of state-of-the-art TCHCs. CC [6,7] has an InO and an OoO backend, as shown in Fig. 1. CC has two execution modes: a *low-power (LP)* mode, using the InO backend and a *high-performance (HP)* mode, using the OoO backend. CC shares the frontend and L1D/I cache between the backends, thereby reducing the penalty for mode switching.

However, because CC has the independent backends, there is always a penalty when switching modes. Although this switching penalty is much shorter than those of the HMCs, it has a non-negligible effect on fine-grained mode switching in CC (e.g., 500 instructions interval). As a result, the switching penalty significantly reduces opportunities for LP mode execution in CC.

The *front-end execution architecture (FXA)* [8] is a TCHC contrasting to CC. FXA has two execution units: an in-order execution unit (IXU) and an OoO execution unit (OXU), as shown in Fig. 2 [8]. The difference between FXA and CC is that the IXU and OXU are connected serially, and the IXU serves as a filter for the OXU. The IXU is placed in the processor frontend and executes instructions that can fetch all their source operands in the frontend. The instructions executed in the IXU are not dispatched to the OXU, which reduces the energy consumption of the OXU. As a result, FXA does not have the mode switching penalty and can execute instructions in-order at instruction granularity.

In addition, the IXU has a higher capability than an InO processor. An InO processor usually stalls the pipeline when dependent instructions are decoded at the same cycle. In contrast, the IXU can execute such instructions without pipeline stall. As a result, the IXU can execute many instructions (approximately 50% [8]).

However, FXA cannot sufficiently reduce the energy consumption compared to CC. This is because it is necessary for FXA to keep to operate some OoO components such as a rename unit and a re-order buffer (ROB) even if instructions are executed in the IXU. As a result, the in-order execution in FXA is less energy efficient than the LP mode in CC, because the LP mode in CC completely stops the OoO components.

As described above, CC and FXA have the following problems: (1) the mode-switching penalty of CC is still large, and (2) the in-order execution in FXA is not energy efficient. In order to resolve these problems, we propose a *dual-mode frontend execution architecture (DM-FXA)*, which is based on FXA and has LP and HP modes. The LP mode executes all instructions in-order using the IXU only and suspends the OXU, whereas the HP mode executes instructions in the same way as in FXA. Similarly to CC, DM-FXA executes instructions by switching between these modes and improves the energy efficiency.

The contributions of DM-FXA are as follows:

1. DM-FXA can switch from LP to HP mode without incurring a penalty. This is because the switching from in-order (LP) to OoO (HP) execution can be realized only by resuming dispatch to the OXU since the IXU and OXU are connected serially. Consequently, it mitigates the large switching penalty of CC, and thus, it improves an LP mode use rate.
2. The LP mode in DM-FXA completely omits the processes required by OoO execution, and consequently, it solves the inefficient energy reduction of the in-order execution in FXA.
3. Since the LP mode of DM-FXA leverages the IXU, the LP mode of DM-FXA has higher performance than LP mode of CC using a normal InO processor. As a result, DM-FXA can get more opportunities to perform LP mode execution than CC without performance degradation.
4. Our evaluation shows that the performance-energy ratio (the inverse of the energy-delay product) of DM-FXA is 24.1% and 12.1% higher than those of CC and FXA, respectively.

The rest of the paper is organized as follows. Section 2 describes CC and FXA. In Sect. 3, we propose DM-FXA, and in Sect. 4, we evaluate our proposed method. Then, Sect. 5 summarizes related work.

2 Existing TCHC Architecture

2.1 Composite Core

CC [6,7] has *InO* and *OoO backends* (Fig. 1), which are used in LP and HP mode, respectively. The two backends share some units, such as the branch predictor and L1D/I caches, to avoid a cold start on switching. This allows CC to perform fast execution-mode switching. Other latency-critical components, such as a register file (RF), are independent for each backend.

Execution Mode Selection. CC basically selects the execution mode as follows. In execution mode selection, CC controls the (estimated) increase of execution cycles from the HP mode within a user-configured range (e.g., 5%). That is, supposing that a part of a program is executed by both HP and LP modes, and if the increase in cycles of LP mode execution is estimated to be smaller than that of the HP mode execution, then CC switches its mode to LP mode. We call such a part an *LP-friendly interval* when LP mode effectively improves energy efficiency.

The authors of CC found that the LP-friendliness fluctuates wildly in fine granularity (e.g., 500 instructions) [6,7], and CC improves energy efficiency by exploiting such fine-grained LP-friendly intervals. In their evaluation, the use rate of LP mode in CC is approximately 30% while the performance degradation is within 5% [7].

Execution Mode Switching. CC switches its execution mode as follows. (1) CC stops the instruction fetch and waits until all instructions are retired from the active backend. (2) In parallel with the fetch stop, CC speculatively starts to migrate register values between the dedicated RFs. (3) After the retirement of instructions from the active backend, the values for which speculative migration failed are remigrated. (4) When the remigration is completed, the switching destination backend starts instruction execution. The cycles required for this retirement and migration is the switching penalty.

Although the penalty cycles are significantly shorter (about 37 cycles in our evaluation) than those in conventional heterogeneous multicore processors (approximately 20 μs [13]), the penalty has non-negligible negative effects because CC switches modes with a considerably fine granularity. As a result, CC's switching granularity is restricted to more than hundreds of instructions [7].

2.2 Front-End Execution Architecture

Structure and Behavior. FXA also has two execution units: OXU and IXU (Fig. 2). While the OXU is a reduced backend of a conventional OoO processor, the IXU consists of an array of functional units (FUs). The IXU is placed after the rename stage in the frontend and executes instructions in order. FXA has additional register ports for the IXU; however, the total number of register ports is not significantly increased because the OXU size is reduced as shown in Figs. 1 and 2.

FXA handles instructions as follows (assuming integer instructions with one-cycle latency). (1) FXA attempts to obtain source operands at the register read stage in the frontend by (1-a) reading from RF or (1-b) bypassing from the FUs in the IXU (not from the OXU). Operands are bypassed between FUs on different stages, e.g., a result calculated on the first stage is received by the second stage. FXA then checks whether all the operands are obtained, that is, whether instructions are ready to be executed. (2) A ready instruction is executed in the IXU and is not dispatched to the IQ. (3) A non-ready instruction goes through the IXU as a NOP and is then dispatched to the IQ in OXU and executed. Note that an InO processor stalls its pipeline until its readiness is resolved; however non-ready instructions in FXA go through the IXU pipeline as a NOP without stalling the pipeline.

IXU Capability. The IXU has a higher capability than an InO processor. An InO processor usually stalls the pipeline when dependent instructions are decoded at the same cycle. In contrast, the FU array in the IXU can execute such instructions without pipeline stall. For example, in Fig. 2, when the producer instruction $\textcircled{P}$ is executed in the first stage of the IXU, its consumer $\textcircled{C}$ decoded at the same cycle goes through the first stage as a NOP and then is executed in the second stage. As a result, the IXU can execute many instructions (approximately 50% [8]).

IXU executes not only integer instructions but also load/store and branch instructions. Other instructions such as floating-point (FP) instructions are not executed in IXU because the resource overhead for additional FPUs is large.

Merits and Demerits. FXA has the following merits:

- The energy consumption is reduced. IXU has no instruction-scheduling hardware and thus it can execute instructions with high energy efficiency. Moreover, IXU filters many instructions (e.g. 50%); thus, the size of OXU can be reduced without performance degradation.
- The performance is improved because the number of FUs is increased as shown in Figs. 1 and 2. CC can execute up to two instructions, while FXA can execute up to five instructions per cycle.

However, the IXU execution in FXA cannot sufficiently reduce energy consumption compared with the LP mode in CC, which can completely omit OoO execution. In FXA, OXU must execute instructions not filtered by IXU, and thus FXA must keep operating OoO execution components such as a rename unit, ROB, and load/store queue (LSQ).

3 Dual-Mode Front-End Execution Architecture

As described before, CC and FXA have the following problems:

1. CC: The switching penalty is large (Sect. 2.1).
2. FXA: The reduction in energy consumption is not sufficient (Sect. 2.2).

In order to solve these problems, we propose a **dual-mode front-end execution architecture (DM-FXA)**, based on FXA, with LP and HP modes.

3.1 Implementation of LP Mode

The LP mode in DM-FXA executes all instructions using the IXU only, whereas the HP mode executes instructions in the same way as in FXA. The execution mode selection is performed in the same way of CC (Sect. 2.1).

Figure 3 shows DM-FXA in the LP mode. DM-FXA has a similar physical architecture to FXA. In this figure, the shadowed units, such as IQ, are stopped in the LP mode. Note that power gating is not applied to the deactivated units as in the prior work of CC [6,7] because recovery from power gating requires significant time.

The LP mode stalls the pipeline when its source operands cannot be obtained and waits for resolving dependencies as in an InO processor. When decoding complex instructions not supported by the IXU (Sect. 2.2) in the LP mode, DM-FXA immediately switches to the HP mode.

The LP mode completely omits OoO execution as follows:

1. It deactivates the OXU including IQ, LSQ, and ROB.
2. It stops register renaming, and RF is accessed using logical register numbers.
3. Only a partial region in RF is accessed when accessing RF with the logical numbers; thus, the other region of RF can be deactivated. We call active partial region *head region*. If the number of logical registers is 32, the head region is from register 0 to 31. In modern processors, RFs generally consist of hierarchical SRAMs [14,15]; thus, the LP mode deactivates not accessed SRAMs.

Thus, the LP mode resolves the problem of inefficient energy reduction in FXA (Sect. 2.2).

3.2 Switching from HP to LP Mode

The switching from the HP to LP mode occurs as follows.

1. The fetch instruction is stopped, and DM-FXA waits for the retirement of all instructions in a similar way to the as CC.
2. DM-FXA rearranges the values in the RF in the order of the logical register numbers so that the RF is accessed using logical numbers. In this rearrangement, live values in the head region are temporally migrated to the other region in order to clean out the head region, and then all values are migrated to the head region.

Additional cycles for this temporal migration are small because the number of live registers is equal to that of logical registers after the retirement of all instructions; thus, there is a low probability of live values in the head region (e.g., 32 logical regs/160 physical regs = 0.2). In addition, the required cycles for this rearrangement are shorter (maximally 20 cycles in our evaluation) than those required for the retirement of instructions, and thus, does not cause serious problems.

3.3 Switching from LP to HP Mode

The switching from the LP to HP mode is performed by (1) initializing a register alias table (RAT), and (2) restarting OoO execution. Unlike CC, DM-FXA does not incur a penalty when switching from the LP to HP mode.

The RAT must be initialized before returning to the HP mode because the register values are rearranged when switching modes. In this case, the RAT is initialized because each logical register number points to the same number of a physical register entry. This initialization can be performed in parallel with execution in the LP mode because the LP mode does not use the RAT.

3.4 Execution Correctness

When switching from the LP to HP mode, unlike CC, it is not necessary for DM-FXA to wait for the instructions to retire from the pipeline. In this behavior, the execution correctness in DM-FXA is still maintained. For describing the reason, we refer to instructions fetched in LP and HP mode as LP and HP instructions, respectively.

In the LP mode of DM-FXA, ROB and LSQ entries are not allocated to LP instructions; consequently, if HP instructions are executed before the execution of all LP instructions, the correctness of execution is not maintained.

However, HP instructions do not overtake LP instructions in execution. Figure 4 shows the switching of an execution mode from the LP to HP mode, and both LP and HP instructions simultaneously exist in the pipeline. The ovals labeled as LP and HP show LP and HP instructions respectively, and the instructions are arranged from right to left in the program order.

The left-sided HP instructions cannot be dispatched to IQ in OXU before all the right-sided LP instructions are executed because all the instructions must proceed in order in the frontend pipeline including IXU. As a result, when HP instructions are dispatched to IQ in OXU, all the LP instructions have been completed to maintain execution correctness.

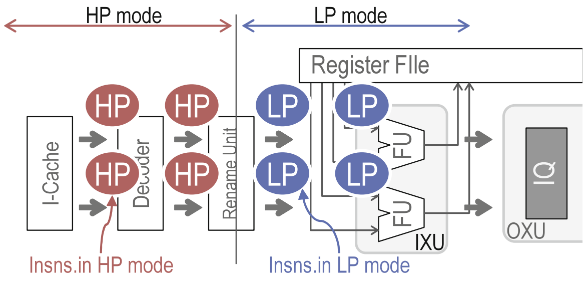

Fig. 4. Switching from LP to HP mode.

3.5 LP Mode Utilization

DM-FXA can get more opportunities to perform LP mode execution than CC without performance degradation for the following reasons:

1. As described above, DM-FXA can switch from the LP to HP mode without incurring the switching penalty. As a result, compared with CC, execution cycles in which the core is stopped due to the switching penalty are reduced, and more instructions can be executed by LP mode.
2. As described in Sect. 2.2, the IXU has the higher capability than an InO processor. Since the LP mode of DM-FXA leverages the IXU, the LP mode of DM-FXA has higher performance than LP mode of CC using a normal InO processor.

The evaluation results in Sect. 4.2 show that DM-FXA can execute about 3 times more instructions in LP mode than CC with the same performance.

3.6 Hardware Cost

The additional hardware cost of DM-FXA compared with FXA is mainly composed of switching control hardware that is almost the same as that for CC. The circuit area and energy consumption of this additional hardware are negligible compared with those of the whole processor [6,7].

4 Evaluation

4.1 Evaluation Environment

We evaluate the IPCs using an in-house cycle-accurate processor simulator. Similar to gem5 [16], this simulator is execution driven, but it more accurately simulates dynamic scheduling in OoO processors, such as a replay mechanism on cache misses. We evaluate the energy consumption using the McPAT simulator [17], with the parameters shown in Table 1.

We use the programs from the SPEC CPU 2006 INT benchmark suite [18]. The programs were compiled using GCC 4.5.3 with "-O3" option. We skipped the first 2G instructions, and evaluate the next 100M instructions using *ref* data sets. These benchmarks and evaluated instructions are basically the same as those used in prior CC-related studies [6,7].

Table 1. Device configurations

Technology	22 nm, Fin-FET
Temperature, VDD	320 K, 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)

Table 2. Processor configurations.

	BASE/CC(OoO)	FXA/DM-FXA	CC(InO)
Fetch width	3	←	←
Issue width	4	2	3
Retire width	3	←	N/A
Function unit	ALU:2, FPU:2, MEM:2	←	ALU:2, FPU:1, MEM:1
IQ	64 entries	32 entries	N/A
Ld./St. queue	32/32 entries	←	N/A
ROB	128 entries	←	N/A
I/D TLB	64/64 entries	←	←
u-op cache	4 KB, one cycle	←	←
L1 I-cache	48 KB, two cycles	←	←
L1 D-cache	32 KB, two cycles	←	←
L2 cache	512 KB, 12 cycles	←	←
L2 prefetcher	Stream prefetcher	←	←
Main memory	200 cycles	←	←
IXU	N/A	5 FUs, 3 stages [8]	N/A
ISA	Alpha	←	←

We evaluate the following models:

BASE: A baseline model for an OoO superscalar processor.

CC5: A CC model with an execution mode selection algorithm proposed in [7]. As in previous studies, we use a trace-based phase predictor with a 9-bit index and the corresponding 512-entry PHT. The switching length [7] is set to 500 instructions. In this model, the allowable increase rate for the execution cycles is set to 1.05 (5% slowdown) from BASE, as in previous studies [6,7].

CC10: This model is CC5 with an allowable increase rate of 1.1 (10% slowdown).

FXA: An FXA model with an IQ with issue width and half the capacity of those in BASE because the IXU filters instructions to the IQ without performance degradation, as described in Sect. 2.2.

DMFXA5: A DM-FXA model with the same mode selection algorithm of the CC models. The length of the switching interval is set to 500 instructions. In this model, the allowable increase rate is set to 1.05 (5% slowdown), not from BASE, but from FXA. This model has a comparable IPC to that of BASE because the IPC for FXA is 6.8% higher than that for BASE, as described in Sect. 2.2. The IQ in this model is also half that in BASE.

DMFXA10: This model is DMFXA5 with an allowable increase rate of 1.1 (10% slowdown). For the same reason as in DMFXA5, this model has almost same IPC as CC5.

Table 2 lists the configurations for these models. The parameters are based on those in ARM big.LITTLE architecture, which consists of ARM Cortex-A57 [19] and A53 [20]. The InO backend used in CC5 is a three-issue in-order superscalar processor. These configurations are similar to those used in prior FXA studies [8].

4.2 Evaluation Results

Control Accuracy and Performance. First, we evaluate the performance control accuracy for each model. The average error rates for CC5, CC10, DMFXA5, and DMFXA10 are 0.35%, 0.15%, 0.28%, and 0.55%, respectively. Thus, the performance follows each target performance with high accuracy. These results show that the mode selection algorithm for CC can be also applicable to DM-FXA with high control accuracy.

Figure 5 shows the performance of all models, normalized by that of BASE. As mentioned above, the performance of FXA is 6.8% higher than BASE, on average. Therefore, DMFXA5 has almost the same performance as BASE, and DMFXA10 has almost the same performance as CC5. Compared to BASE, the performance levels of CC5 and DMFXA10 are 95.2% and 96.8%, respectively.

LP Mode Utilization. Figure 6 shows the rate of instructions executed in LP mode. The use rate in CC5 is 21.7%, on average, whereas that of DMFXA5, which has the same 5% slowdown rate as CC5, is 38.0%, on average. Moreover, the use rate for DMFXA10, the performance of which is nearly equal to that of CC5,

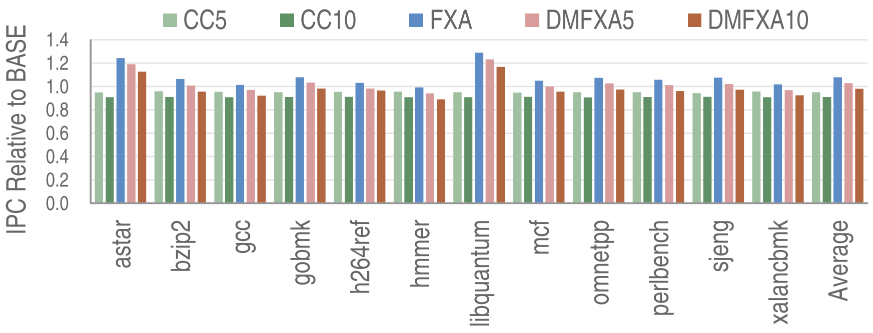

Fig. 5. IPC relative to BASE.

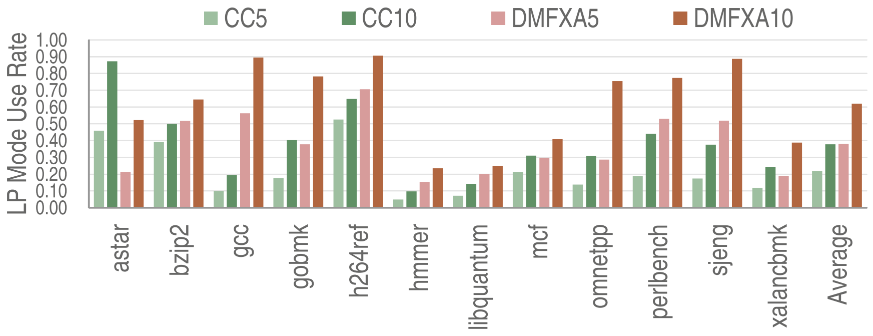

Fig. 6. LP mode use rate.

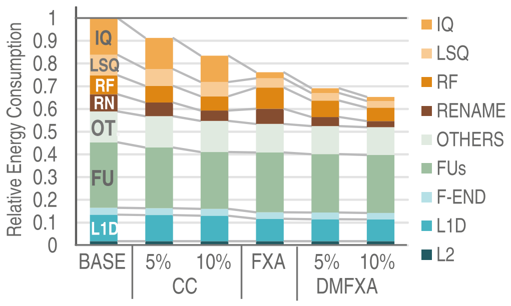

Fig. 7. Energy consumption relative to BASE.

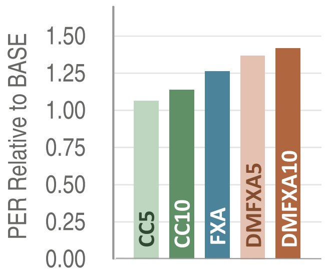

Fig. 8. PER relative to BASE.

is 60.0%. Such improvements are achieved by the following reasons: (1) DM-FXA can reduce the switching penalty compared to CC and (2) since IXU has better performance than normal in-order execution, it gives more opportunities to execute LP mode than CC (Sect. 3).

Energy Consumption. Figure 7 shows the energy consumption of each model relative to that of BASE, on average. These results include static and dynamic energy consumption.

DMFXA10 reduces the energy consumption by 34.7% compared with BASE because the energy consumption of the units for OoO execution, such as IQ, LSQ, RF and a rename logic, is significantly reduced. DMFXA10 reduces energy consumption by 28.5% compared with CC5, which has almost the same performance as DMFXA10. This is because the use rate of LP mode increases significantly, as described before. DMFXA10 reduces energy consumption by 14.2% compared with FXA, because the energy consumption of the components that the FXA must continue activating, such as the rename logic (Sect. 3.1), is reduced in the LP mode of DM-FXA.

Note that the breakdown of BASE is very similar to that in AMD Steamroller [21] and ARM Cortex A15 [22]. The energy consumption of the L2 cache is very small in all the models. This is because we use Fin-FET technology [23] and low-standby-power transistors for the L2 caches (Table 1), and consequently, the static energy consumption of the L2 caches is very small.

Performance Energy Ratio. This section shows the performance-energy ratio (PER) of each model, which is equal to the inverse of the energy-delay product (EDP)[1]. The PER shows how each model reduces energy consumption with respect to its performance degradation. Figure 8 shows the PER of each model relative to that of BASE. In the figure, DMFXA10 improves the PER compared to BASE, CC10, and FXA by 41.5%, 24.1%, and 12.1%, respectively. This high

[1] We use a PER instead of an EDP because it is easy to understand. That is, a larger PER shows better energy efficiency.

PER is achieved because DMFXA10 significantly reduces the energy consumption compared to its performance degradation. Noted that FXA has the better PER than its reduced energy consumption, because FXA does not degrade but improves performance [8].

Switching Penalty. In the DM-FXA, there is no penalty for switching from LP to HP mode; thus, the number of penalty cycles per switch is smaller than that of the CC. The number of average penalty cycles per switch is 56 cycles for the CC and 32 cycles for the DM-FXA, respectively. As described in Sect. 3.5, DM-FXA takes advantage of this short penalty and increases the number of instructions executed in the LP mode

5 Related Work

Typical HMCs include heterogeneity based on the difference in the microarchitecture between the cores, such as InO and OoO cores [1–5]. Other methods have also been proposed with heterogeneity based on the DVFS control [3,24].

Unlike the naive HMCs, TCHCs have tightly coupled heterogeneous execution units within a single core. Therefore, TCHCs perform energy-efficient execution at a finer granularity. We summarize this research in the following paragraphs.

The heterogeneous block architecture (HBA) [9] improves energy efficiency by caching dynamically scheduled instructions and executing the cached instructions InO. DynaMOS [10] also caches scheduled instructions and executes them InO. DynaMOS is based on the CC architecture and improves the InO backend execution ratio by executing the *scheduled* instructions in the InO backend. Compared with our proposed method, these methods require special mechanisms, such as a special trace cache for storing scheduled instructions and additional renaming logic.

Early OoO late execution (EOLE) [12] is similar to the FXA in that EOLE executes instructions InO using FUs added to the frontend. EOLE must also continue activating components for OoO execution for the same reason as the FXA, and consequently, the reduction of consumed energy is restricted compared with that of the DM-FXA.

6 Conclusion

The CC and FXA introduce heterogeneity within a single core and were previously proposed for to improve energy efficiency. However, these TCHCs have problems with their switching latency hardware, and the amount of energy consumption reduction is insufficient. In order to resolve these problems, we proposed the DM-FXA, which is based on the FXA and has an LP mode. In our evaluation, our proposed method achieved a reduction in energy consumption by 34.7%, with a 3.2% performance overhead in comparison with a conventional superscalar processor.

Acknowledgment. This work was supported by JSPS KAKENHI Grant Number 16H05855.

References

1. Kumar, R., Farkas, K.I., Jouppi, N.P., Ranganathan, P., Tullsen, D.M.: Single-ISA heterogeneous multi-core architectures: the potential for processor power reduction. In: Proceedings of the 36th Annual International Symposium on Microarchitecture (MICRO), pp. 81–92, December 2003
2. Becchi, M., Crowley, P.: Dynamic thread assignment on heterogeneous multiprocessor architectures. In: Proceedings of the 3rd Conference on Computing Frontiers, pp. 29–40, May 2006
3. Rangan, K.K., Wei, G.Y., Brooks, D.: Thread motion: fine-grained power management for multi-core systems. In: Proceedings of the 36th Annual International Symposium on Computer Architecture, pp. 302–313, June 2009
4. Joao, J.A., Suleman, M.A., Mutlu, O., Patt, Y.N.: Bottleneck identification and scheduling in multithreaded applications. In: Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pp. 223–234, April 2012
5. Greenhalgh, P.: Big.LITTLE Processing with ARM Cortex-A15 and Cortex-A7. Whitepaper, September 2011
6. Lukefahr, A., Padmanabha, S., Das, R., Sleiman, F.M., Dreslinski, R., Wenisch, T.F., Mahlke, S.: Composite cores: pushing heterogeneity into a core. In: Proceedings of the 45th Annual International Symposium on Microarchitecture, pp. 317–328, December 2012
7. Padmanabha, S., Lukefahr, A., Das, R., Mahlke, S.: Trace based phase prediction for tightly-coupled heterogeneous cores. In: Proceedings of the 46th Annual International Symposium on Microarchitecture, pp. 445–456, December 2009
8. Shioya, R., Goshima, M., Ando, H.: A front-end execution architecture for high energy efficiency. In: Proceedings of the 47th Annual International Symposium on Microarchitecture, pp. 419–431, December 2014
9. Fallin, C., Wilkerson, C., Mutlu, O.: The heterogeneous block architecture. In: Proceedings of the International Conference on Computer Design (ICCD), pp. 386–393, October 2014
10. Padmanabha, S., Lukefahr, A., Das, R., Mahlke, S.: DynaMOS: dynamic schedule migration for heterogeneous cores. In: Proceedings of the 48th International Symposium on Microarchitecture, December 2015
11. Khubaib, Suleman, M.A., Hashemi, M., Wilkerson, C., Patt, Y.N.: MorphCore: an energy-efficient microarchitecture for high performance ILP and high throughput TLP. In: Proceedings of the 45th Annual International Symposium on Microarchitecture, pp. 305–316, December 2012
12. Perais, A., Seznec, A.: EOLE: paving the way for an effective implementation of value prediction. In: Proceeding of the 41st Annual International Symposium on Computer Architecture, pp. 481–492, June 2014
13. ARM: ARM Unveils its Most Energy Efficient Application Processor Ever; Redefines Traditional Power And Performance Relationship With big.LITTLE Processing (2011)
14. Weste, N.H.E., Harris, D.M.: CMOS VLSI Design: A Circuits and Systems Perspective, 4th edn. Pearson/Addison-Wesley, Boston (2011)

15. Golden, M., Arekapudi, S., Vinh, J.: 40-Entry unified out-of-order scheduler and integer execution unit for the AMD Bulldozer x86-64 core. In: Proceedings of the International Solid-State Circuits Conference (ISSCC), pp. 80–82, February 2011
16. Binkert, N.: The gem5 simulator. SIGARCH Comput. Archit. News **39**(2), 1–7 (2011)
17. Li, S., Ahn, J.H., Strong, R.D., Brockman, J.B., Tullsen, D.M., Jouppi, N.P.: McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In: Proceedings of the 42nd Annual International Symposium on Microarchitecture, pp. 469–480, December 2009
18. The Standard Performance Evaluation Corporation: SPEC CPU 2006 Suite. http://www.spec.org/cpu2006/
19. Bolaria, J.: Cortex-A57 Extends ARM's Reach. Microprocessor Report 11/5/12-1, November 2012
20. Krewell, K.: Cortex-A53 Is ARM's Next Little Thing. Microprocessor Report 11/5/12-2, November 2012
21. Gillespie, K., et al.: Steamroller: an x86-64 core implemented in 28nm bulk CMOS. In: International Solid-State Circuits Conference (ISSCC). Presentation Slides (2014)
22. NVIDIA: NVIDIA Tegra 4 Family CPU Architecture. Whitepaper (2013)
23. Auth, C., et al.: A 22 nm 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), pp. 131–132 (2012)
24. Lukefahr, A., Padmanabha, S., Das, R., Dreslinski Jr., R., Wenisch, T.F., Mahlke, S.: Heterogeneous microarchitectures trump voltage scaling for low-power cores. In: Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT), pp. 237–250, July 2014