Energy Efficiency Improvement of Renamed Trace Cache through the Reduction of Dependent Path Length

Ryota Shioya* and Hideki Ando*
* Department of Electrical Engineering and Computer Science,
Nagoya University, Nagoya, Aichi, Japan
Email: {shioya, ando}@nuee.nagoya-u.ac.jp

***Abstract*—A renaming logic is a high-cost module in a superscalar processor, and it consumes significant energy. For mitigating this, renamed trace cache (RTC), which caches renamed operands, was proposed. However, conventional RTCs have several problems such as low capacity-efficiency, large hardware overhead and insufficient caching of renamed operands. We propose a semi-global renamed trace cache (SGRTC) that caches only renamed operands whose distances from producers outside traces are short, and it solves the problems of conventional RTCs. Evaluation results show that SGRTC achieves 64% lower energy consumption for renaming with a 0.2% performance overhead compared to a conventional processor.**

I. Introduction

In out-of-order superscalar processors, logical registers are *renamed* for removing false dependencies between instructions. Register numbers are renamed by accessing a table called *register map table (RMT)*. The circuit area of an RMT has recently increased owing to the widespread use of SMT and other reasons described in Section II. For example, Alpha 21464 has a rename logic that is larger than its 64KB L1-data cache[14]. Consequently, it makes its complexity, energy consumption, and heat generation, a considerably serious issue. As a result, the RMT in the Intel P6 architecture consumes 4% of the energy consumed by the processor, which is comparable to that of its reservation station[9], and its power-density is the fourth highest on-chip[11].

To overcome these problems, Vajapeyam et al. proposed a *renamed trace cache (RTC)* [22] that extends a trace cache (TC)[15]. A conventional TC caches instructions ordered by a dynamic execution sequence beyond branch instructions. An RTC is a TC that caches instructions with renamed operands. When instructions with renamed operands are obtained from an RTC, register renaming can be omitted, thus it is possible to reduce the number of ports of an RMT. An RMT generally comprises a RAM, and the area of a RAM grows proportionally with the square of the number of ports. Consequently, the reduction in the number of ports makes the area of the RMT very small. However, the RTC method has a limitation on caching renamed operands, i.e., the RTC method can cache renamed operands only when operands refer producers, which are dependent instructions of the operands, in the same trace. As a result, its advantage is limited, and the RTC method can cache renamed operands only for approximately 30% operands (Section VI).

On the other hand, Ichibayashi et al. independently proposed another RTC method that can cache renamed operands that refer producers outside a trace[4]. For simplicity, we refer to the RTC of Vajapeyam et al. as *local renamed trace cache(LRTC)* and that of Ichibayashi et al. as *global renamed trace cache(GRTC)*.

GRTC does not have the limitation of LRTC for caching operands, but it considerably reduces capacity-efficiency, which is the number of cached static instructions per a certain amount of cache capacity. A similar problem is exhibited by a conventional TC, but the problem of GRTC is more severe. In a conventional TC, its capacity-efficiency is reduced because there are multiple traces starting from the same basic block. Additionally, in GRTC, each trace depends on its successor path outside a trace, and it is necessary to cache different traces starting from the same basic block for each successor path. The increase in the length of a dependent path, exponentially increases the number of generated traces, and the capacity-efficiency is considerably reduced. Moreover, in GRTC, it is necessary to store the target addresses of indirect jumps to its tag array for checking a path, and it considerably increases a circuit area. As a result, its energy-overhead mostly cancels reduced energy consumption by reducing the RMT, as described in Section VI.

We propose *semi-global renamed trace cache (SGRTC)*, which is based on GRTC. Our proposal caches only renamed operands whose distances from producers outside traces are short. The number of generated traces is exponential with distance from producers outside traces, thus limiting off-trace distances makes it possible to reduce the number of generated traces, and it improves its capacity-efficiency. Additionally, limiting dependent distances, probabilities that there are indirect jumps in a dependent path decrease. Consequently, storing the target addresses of indirect jumps can be omitted without significant performance degradation, and it makes it possible to reduce energy consumption.

Although the researches of conventional LRTC and GRTC suggested that the complexity of a rename logic can be mitigated, but their energy consumption has not been quantitatively evaluated yet. One of the contributions of this paper is that we quantitatively evaluated the energy consumption and the performance of these conventional methods.

Evaluation results presented in Section VI show that energy consumption for register renaming is reduced by 64% from a processor with a conventional instruction cache, and it suffers only 0.2% performance degradation. Further, the evaluation results show that the proposed method achieves 8.5% and 4.5% higher performance, 23% and 60% lower energy consumption, and 41% and 151% higher performance-energy ratio (the inverse of energy-delay product) than conventional LRTC and GRTC, respectively.

978-1-4799-6492-5/14/$31.00 ©2014 IEEE

II. Register Map Table

In this section, we first describe the problems of an RMT, and then, we summarize related work.

A. Problems of Register Map Table

Register numbers are renamed by reading and writing an RMT. An RMT is a table with relationship information between logical and physical register numbers, and it generally comprises multi-ported RAM[24], [13][1].

An RMT occupies a considerably large circuit area. For general ISAs with a three-operand format, an RMT requires 4 ports per instruction[17], [24]. For example, in a processor whose decode width is 8, such as IBM Power 8[7], [2], a straightforward implementation of an RMT requires 32 ports[2]. The area of a RAM grows proportionally with the square of the number of ports[23], and therefore, the area of the RMT is large despite of its small number of entries.

Moreover, spreading SMT processors makes the RMT considerably larger. SMT processors require RMTs with capacities that are proportional to the number of threads for retaining thread contexts. In fact, Alpha 21464 with 4-threads SMT has a rename logic that is larger than its L1-data cache[14]. The recent IBM Power 8[7], [2] supports 8-threads SMT and this problem gets more severe.

As a result, the energy consumption and generated heat of the RMT is considerably important issue. The energy consumption of a RAM is proportional to its area and access frequency[23]. Almost all instructions access the RMT more than once, thus the number of accesses to the RMT is considerably high. Consequently, the RMT consumes significantly larger energy than a data cache in the same area. As a result, the RMT in the Intel P6 architecture consumes 4% of the energy consumed by the processor, which is comparable to that of its reservation station[9]. In recent processors such as IBM Power 8, its rename width is significantly wider than that of the Intel P6 and they support SMT, thus, their RMT is significantly larger than that of the Intel P6 and consumes more energy.

B. Related Work

For mitigating these problems, several techniques were proposed for RMTs. Some researchers focused on the fact that all ports of an RMT are generally not entirely used, and they proposed methods that reduce the number of the ports of an RMT for mitigating its complexity[10], [18]. These methods can reduce the number of ports proportionally to the effective use rate of the ports, but this reduction is limited.

Moshovos et al. proposed a method that focuses on checkpointing an RMT for recovering from branch misprediction[11]. This method takes checkpoints only for branch instructions with low confidence, and it makes it possible to reduce the resources for the RMT. However this method requires an additional confidence estimator for branch instructions, and it consumes additional energy.

Liu et al. proposed a method that uses a register map cache (RMC)[8]. The RMC is smaller than the main RMT, and it can reduce its latency and energy consumption on hit; however, its miss penalty may degrade performance.

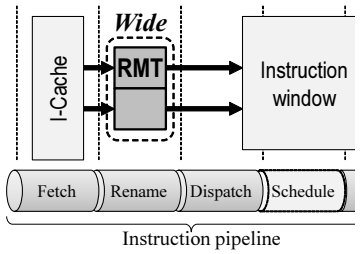

Fig. 1. General Front-end

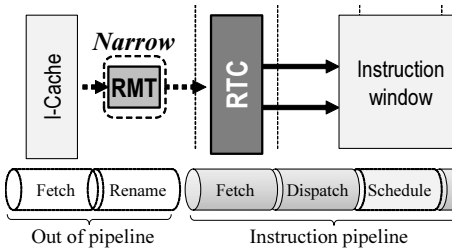

Fig. 2. GRTC's Front-end

There are only two conventional methods that cache renamed operands, to our knowledge, and they are LRTC[22] and GRTC[4]. We think that this is because caching renamed operands is essentially difficult. The following sections describe why caching renamed operands is difficult, and describe how to cache renamed operands in the conventional methods.

C. Caching Renamed Operands

For mitigating the problems, caching renamed operands is an attractive idea; however, it is generally difficult for the following reasons: 1) Physical registers are assigned from a free list, and consequently, assigned register numbers are not reproducible and reusable. 2) The producers of source operands are changed depending on the executed paths, and consequently, the physical register numbers of source operands are not reusable.

LRTC[22] described in Section I avoids the above problems and partially reuses renamed operands by the following methods. LRTC uses a physical register file (RF) that is local for each trace in addition to a conventional physical RF. Each entry of this local physical RF is sequentially assigned to each instruction in a trace, and is referred with an offset from the start of a trace[3]. In a trace, the positional relation between instructions and their execution paths are guaranteed to be fixed for the behavior of TC. Consequently, LRTC can resolve the above two reasons and reuse renamed operands in a trace.

However, this means that LRTC can cache renamed operands only when source operands refer destination operands generated in the same trace. As a result, LRTC cannot work well and it can only cache approximately 30% of renamed operands (Section VI).

III. Global Renamed Trace Cache

Unlike LRTC, GRTC is proposed for caching renamed operands that refer producers outside a trace[4]. Figures 1 and 2 show the front-end of a conventional superscalar processor

[1]There is a method with a CAM-based RMT[5], and our proposal also can be implemented on a CAM-based method.

[2]In processors with wide decode width, techniques that restrict the number of renames allowed per processor clock cycle can be used to reduce the number of rename ports and hence the RMT area and energy consumption.

[3]There is a more optimized implementation, but we describe this implementation for simplicity.

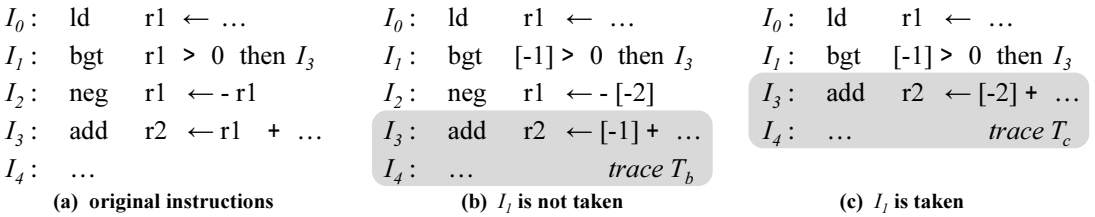

Fig. 3. Example of Conversion to Dualflow Ops

and that with GRTC, respectively. A conventional superscalar processor renames operands for each fetched instruction. On the other hand, in GRTC, fetched instructions are directly dispatched to the instruction window, because their operands are already renamed. Operands are renamed only when traces are generated on an RTC miss, and consequently, the number of ports of an RMT can be reduced without performance degradation. This reduction of ports considerably reduces its area and energy consumption.

GRTC modifies the allocation and reference of physical RFs, and it makes it possible to reuse renamed operands without the restriction of caching in LRTC. This section describes GRTC.

A. Register File Method

GRTC uses two RFs, physical register file (PRF) and logical register file (LRF). The entries in these RFs are allocated in the same manner as the allocation of a reorder buffer and LRF in a processor with a reservation station. However, only the allocation method is same. The issue queue in GRTC does not have values and the RFs are accessed only after issuing from the issue queue. In particular, the PRF has a ring structure and its entries are sequentially allocated. The LRF has the execution results of retired instructions. Instructions write their execution results to the LRF in sequential order.

B. Instruction Method

In GRTC, source operands are referred with the *displacement* between instructions as "the execution result before n instructions." This displacement is equal to displacement between entries allocated to instructions on the PRF, because the entries of the PRF are sequentially allocated to instructions. Each instruction obtains the physical register number of source operands by adding its displacement to the physical register number of its own destination operand. When a producer is distant more than WS, which is the size of an instruction window[4], the producer is guaranteed to be retired and its execution result is obtained from the LRF.

An instruction with this displacement is called *dualflow op*. GRTC dynamically converts instructions to dualflow ops for generating traces on RTC miss. This conversion is carried out with an RMT that is similar to an RMT in an usual rename-logic [4].

C. Caching Dualflow Ops

GRTC generates traces including dualflow ops for each execution path, and separately caches them. The term "trace" means instructions ordered by a dynamic execution sequence. This caching uses a structure that is similar to a conventional TC. The conventional TC generates and caches traces for each execution path *in a trace*. On the other hand, GRTC generates and caches traces for each backward execution path *outside a trace* in addition to those in a trace.

[4] WS is the size of a reorder buffer.

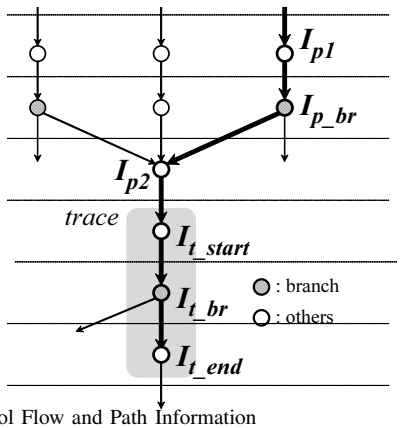

Fig. 4. Control Flow and Path Information

We explain this scheme by using an example presented in Figure 3. In this figure, instructions in (a) are converted to dualflow ops in (b) when the branch I_1 is not taken, and those in (c) are vice versa. I_3 in (b) and (c) have different displacements, -1 and -2, respectively. Their displacements are different depending on the backward execution path of I_3. GRTC generates traces T_b and T_c for each pattern and caches them for each backward execution path.

GRTC distinguishes traces by using *path information* that is similar to that used in a conventional TC. The term "path information" is specifically the directions of branches and the target addresses of indirect jumps. GRTC stores this path information and the start address of path to a tag array. When GRTC decides hit or miss of the RTC, it generates path information from PC and branch history information and compares it with the path information stored in a tag array.

D. Reusing Renamed Operands

GRTC resolves the two reasons for which renamed operands cannot be reused (Section II-C). First, the non-reproducibility of physical register numbers is solved by the sequential assignment of registers and operand access with displacement. The displacement, which is a static distance between instructions, is always constant. Consequently, the displacement once converted can be reused. Second, producer changing for each path is solved by caching traces for each backward execution path. As a result, GRTC can cache and reuse renamed operands.

IV. Problem of Global Renamed Trace Cache

This section describes two problems of GRTC.

A. Low Capacity-Efficiency

The first problem of GRTC is its low capacity-efficiency. Compared to TC and LRTC, each trace depends on its

backward path outside a trace, and it is necessary to cache different traces starting from the same basic block for each backward path. The increase in the length of a dependent path exponentially increases the number of generated traces, and the capacity-efficiency is considerably reduced.

We explain this problem by using Figure 4. This figure shows the control flow graph, and each node in the graph represents an instruction. In the following, we explain the fetch of the trace that starts from I_{t_start} to I_{t_end} by comparing TC and GRTC. Note that we do not explain a case of LRTC, because the fetch methods of LRTC and TC are identical.

(1) TC : A trace hits if the path from I_{t_start} to I_{t_end} in the trace matches path information in the tag array. In this case, the length of path information is three instructions, which is the length of a trace.

(2) GRTC : In addition to the path in a trace, it requires matching the backward execution path of a trace and path information in the tag array. The length of the backward path of a trace is determined by the largest displacement in the source operands in a trace (we refer to it as *dependent path length*). For example, when the source operands of I_{t_start}, I_{t_br} and I_{t_end} refer the destination operand of I_{p1}, the length of path information is six instructions, which is three instructions in the trace and three instructions in the backward path from I_{p1} to I_{t_start}. This is longer than that of TC, which is three instructions.

The maximum dependent path length in GRTCS is the size of instruction window, WS. This is because producers that are distant more than WS must be retired, and their results are guaranteed to be in the LRF, as described in Section III-A.

Because of the difference of dependent path lengths, the numbers of generated traces in TC and GRTC are considerably different. For example, in TC whose length of a trace is three instructions, the number of generated traces that started from the same address is $2^3 = 8$ in the worst case when all instructions in a trace are branches. On the other hand, in GRTC, each trace depends on its backward path outside a trace in addition to the inside of a trace. The maximum dependent path length is WS, which is more than 200 instructions in recent processors[7], [2], and consequently, the number of generated traces is more than 2^{200} in the worst case. As described above, the number of generated traces is considerably higher and the capacity-efficiency of GRTC is consequently decreased.

B. Overhead of Tag

The second problem is the hardware overhead of the stored path information in a tag:

(1) Increased Information for Branch Directions: The path information of GRTC is similar to the information of branch directions in a trace in TC. As described before, the dependent path length of GRTC is considerably longer than that of TC, and consequently, more hardware is necessary to store it.

(2) Stored Target Address of Indirect Jumps : In TC, the target address of an indirect jump is not stored to its tag array to avoid the hardware overhead. Consequently, a trace is finished if an indirect jump exists in a case of TC. This does not decrease performance significantly because the appearance frequency of indirect jumps is sufficiently low.

On the other hand, in GRTC, if the target address of an indirect jump is not stored to a tag array, its performance degrades sufficiently[4]. This is because hit/miss decision in GRTC depends on its backward path outside a trace. When the target addresses are not stored to a tag array, it is unable to cache a trace with indirect jumps that are placed backward within the dependent path length of the trace, because matching its path is not guaranteed. For example, in Figure 4, if I_{p_br} is an indirect jump, instructions above I_{p1} can not be cached. As a result, when an indirect jump appears, it is unable to cache a considerable number of instructions in front of the indirect jump.

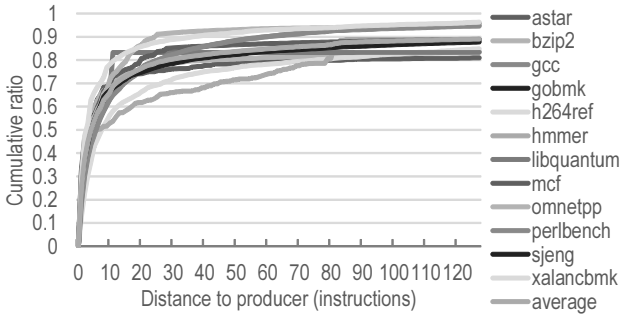

Fig. 5. Distribution of Distances to Producers

Thus, GRTC stores some target addresses of indirect jumps to a tag array. If there are indirect jumps more than the maximum number, traces are not cached and are renamed with an RMT. Ichibayashi et al. described that it requires two-target addresses in a tag array for avoiding performance degradation; however, they did not evaluate its overhead such as energy consumption[4].

However, the overhead of its additional hardware is considerably large. For example, when an instruction length is four bytes and a trace length is 4 instructions, the required capacity for the data part of a trace is 16 bytes. On the other hand, when the length of a target address is 64 bits, the tag part with two target addresses for the path information consumes capacity more than its data part. Moreover, it requires WS directions of branches, and consequently, the capacity of the tag array is considerably larger than that of a conventional cache and TC.

V. Semi-Global Renamed Trace Cache

We propose *semi-global renamed trace cache (SGRTC)*, which is based on GRTC. Our proposal caches only renamed operands whose distances from producers outside traces are short.

A. Motivation

Many source operands of consumers generally refer producers near to the consumers. Figure 5 shows the distribution of the distances from each source operands to its producer in SPECCPU INT 2006 [20][5]. The horizontal axis shows the distances from each source operand to its producer, and the vertical axis shows the cumulative ratio to all source operands. For example, the point plotted at 10 on the horizontal axis and 0.7 on the vertical axis means that: source operands whose distances to their produces are less than 10 instructions occupy 70% of all source operands. This preliminary evaluation result shows that about 70% of source operands refer producers within ten instructions in SPECCPU INT 2006.

On the other hand, the dependent path length of a trace (Section IV-A) is considerably longer than the distance from

[5]Evaluation environment is the same as that used in Section VI

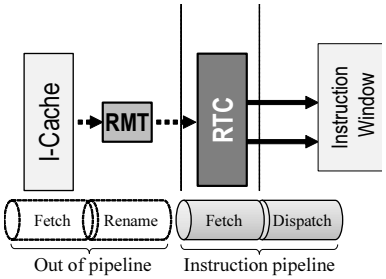

Fig. 6. GRTC's Front-end

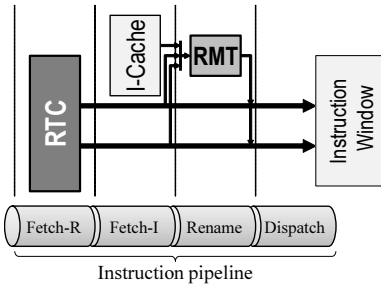

Fig. 7. SGRTC's Front-end

each source operand to its producer. This is because the dependent path length of a trace is determined by the largest distance from a source operand to a producer in the trace. As a result, even though the distance from most source operands to those producers is within ten instructions, the average dependent path length is over 80 instructions in GRTC.

B. Limiting Caching by Dependent Path Length

We focus on this fact that many source operands generally refer producers near to the consumers, and propose a method that caches only renamed operands whose distances from producers outside traces are short. The other operands are stored to an RTC as logical register numbers, and they are renamed with an RMT after instruction decoding.

We describe our proposal, SGRTC, by comparing the conventional GRTC and SGRTC. Figures 6 and 7 show the front-end pipeline of GRTC and SGRTC, respectively. In GRTC, on RTC hit, fetched traces are immediately dispatched to the instruction window. On RTC miss, instructions are fetched from an instruction cache, and then they are renamed. The stages accessing an instruction cache and an RMT are placed outside an instruction pipeline, and those stages work only on RTC miss. On the other hand, SGRTC has the stages that can access an instruction cache and an RMT regardless of RTC hit/miss, and these stages work as follows:

(1) RTC Hit: In Figure 7, the successive Fetch-I stage does *not* access the instruction cache. The successive Rename stage only renames operands stored in traces as logical register numbers.

(2) RTC Miss: The Fetch-I stage accesses the instruction cache, and the Rename stage renames operands.

The traces in RTC are replaced and filled in the same way as a conventional TC[15]. As described above, on RTC miss, instructions are fetched from the instruction cache and are renamed. Then the renamed instructions are dispatched to the instruction window, and at the same time, they are fed to a fill unit with path information. The fill unit generates trace data from the fed instructions and path information, and it is filled

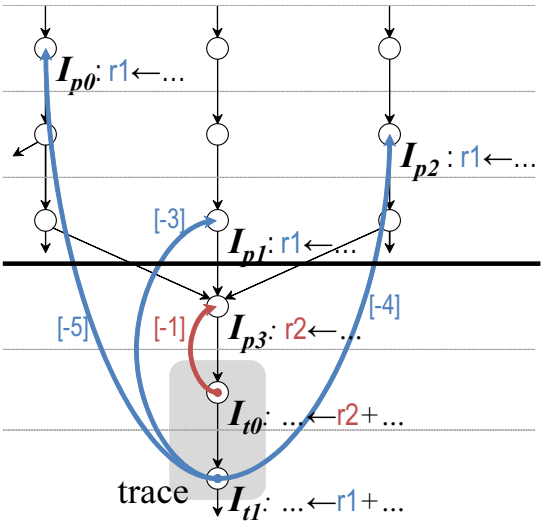

Fig. 8. Reference Distance for Each Instruction

to RTC. In this time, a replacement target is determined with LRU policy in the same way as conventional caches.

SGRTC can reduce the number of ports of its RMT as in the conventional GRTC. When the number of ports is short, the front-end is stalled, and the operands in a fetch group are renamed in multi-cycles.

C. Effects of Proposal

Although our proposal is very simple, it can effectively solve the two problems of the conventional GRTC described in Section IV, and it can cache renamed operands more than LRTC does.

1) Improving Capacity-Efficiency: Our proposal limits the dependent path lengths of traces to short distances, and it improves the capacity-efficiency of RTC. We explain this improvement by using the example shown in Figure 8, which is the control flow graph similar to Figure 4. In this figure, each instruction has only one source operand. We focus on the trace with I_{t0} and I_{t1}, and there are three backward paths that arrive in this trace.

In general, when the distance from a consumer to a producer increases, the number of paths to its producer also increases. The producer of I_{t0} is near to I_{t0}, and is always I_{p3} regardless of which backward path is executed. On the other hand, the producers of I_{t1} are different for each backward path, and they are I_{p0}, I_{p1} and I_{p2}.

In conventional GRTC, the dependent path length is determined by the *largest* displacement in the source operands in a trace. Consequently, different traces are generated for every three paths arriving at I_{t0}.

On the other hand, we assume a case of SGRTC that limits dependent path length to 1 instruction. The producer of I_{t0} is within 1 instruction from I_{t0}, and consequently, its renamed operand is cached. On the other hand, all the producers of I_{t1} that are distant from I_{t1} more than by 1 instruction, their renamed operands are not cached. That is, unrenamed r1 is cached to RTC as it is regardless of which backward path is executed, and then it is renamed even if RTC hits.

In this case, there is only one path that arrives at I_{t0} from 1 instruction before, and consequently, the number of traces that may be generated is 1. The number of generated traces is

reduced, and it improves its capacity-efficiency.

As described in Section IV, increase in the dependent path lengths of traces exponentially increases the number of generated traces, and the capacity-efficiency of RTC is considerably reduced. Our proposal can shorten the dependent path lengths of traces from over 200 instructions to about 12 instructions (Section VI), and consequently, the number of generated traces is considerably reduced, which significantly improves its capacity-efficiency.

Renaming operands stored as logical register numbers does not cause significant overheads. As described before, most source operands refer producers within ten instructions, and therefore, the number of source operands stored as logical register numbers is small. Consequently, these source operands can be renamed by a small RMT that is originally used for renaming when traces are generated on RTC misses in GRTC, and its energy consumption is small.

2) Reducing Hardware Overhead of Tag: Our proposal can omit storing the target addresses of indirect jumps to the tag array without significant performance degradation. As described in Section IV-B, the conventional GRTC requires some target addresses in a tag array for avoiding significant performance degradation. On the other hand, our proposal can shorten the dependent path lengths of traces to about ten instructions. Consequently, if the target addresses are not stored to the tag array, and it cannot cache traces with indirect jumps that are placed backward within the dependent path lengths of the traces, the number of non-cachable instructions is not more than ten instructions.

VI. Evaluation

We evaluated SGRTC and other methods, and this section presents the evaluation results.

A. Evaluation Environment

We evaluated IPCs using an in-house cycle-accurate processor simulator. We used all the programs from the SPEC CPU 2006 benchmark suites[20] with *ref* data sets. The programs were compiled using gcc 4.2.2 with the "-O3" option. We skipped the first 10G instructions and evaluated the next 100M instructions.

TABLE I. Baseline Configuration

fetch width	8 inst.
rename width	8 inst.
issue width	8 inst.
issue queue	64 entries, unified
execution unit	int:4, fp:4, mem:4
ROB	224 entries
branch predictor	8 KB g-share, 2K entries BTB
SMT	8 threads
L1DC	64 KB, 8 way, 64 B/line, 2 cycles
L1IC	32 KB (34.3 KB[1]), 8 way, 64 B/line, 2 cycles
L2C	512 KB, 8 way, 64 B/line, 8 cycles
L3C	8 MB, 8 way, 64 B/line, 24 cycles
main memory	200 cycles
ISA	Alpha

TABLE II. Configurations of TC and RTC

TC	1 K traces (37 KB[1]), 8 way, trace:8 inst., 3 cycles
LOCAL	1 K traces (39 KB[1]), 8 way, trace:8 inst., 3 cycles
GLOBAL	1 K traces (82.9 KB[1]), 8 way, trace:8 inst., 3 cycles
S_GLOBAL	1 K traces (39.4 KB[1]), 8 way, trace:8 inst., 3 cycles

[1]These capacities include the tag parts.

We evaluated the energy consumption for register renaming by evaluating the arrays and their peripheral circuits related to our proposal, which are the RMT, TC, RTC and instruction cache. We do not evaluate the energy consumption of the comparators for dependency checking on register renaming[17], [24], but each comparator consists of few transistors, and thus, we think that their energy consumption is comparatively small. Moreover, our proposal omits this dependency checking on RTC hit, and consequently, this results in a conservative energy comparison for our technique.

The energy consumption for register renaming and the area of each array are evaluated by using CACTI 6.5 [12], which simulates cell arrays and peripheral circuits. We assume 32nm technology (ITRS-HP/nominal), 4GHz operating frequency, 0.9V VDD, and 320K junction temperature. We assume that each cell consists of a regular 4T storage and two access transistors per port. The arrays are not banked[6]. The static and dynamic access energy of the arrays is calculated by CACTI at the stated PVT assumptions. Temperature is assumed to be fixed and leakage is thus modeled as a constant. The number of accesses to each array structure and elapsed time is provided by our cycle-accurate processor simulator. The dynamic array access energy from CACTI is multiplied by the array access frequency from the simulator to form the total dynamic energy.

B. Evaluated Models

Table I lists the configuration of a baseline processor. Its major micro-architectural parameters are based on those of IBM Power 8[7], [2], which include parameters such as fetch width, the size of instruction window, the number of FUs, and cache hierarchies. Note that we use the configuration based on the IBM Power 8 with wide front-end, because it is well known that the throughput of a processor front-end has a significant impact to its performance[15], [16], [1], and the front-end width of recent high-performance commercial processors is increased[19], [7], [2], [6]. Moreover, recent high-performance processors generally equip SMT[19], [7], [2], [6], [3], and thus their RMTs have a considerably large area and energy consumption(Section II). Our proposal makes it possible to solve the problems of such high-performance processors.

We evaluated the following models based on baseline configuration:

(1) BASE: A model with an I-cache only.

(2) TC: A model with TC.

(3) LOCAL: A model with LRTC.

(4) GLOBAL: A model with GRTC.

(5) S_GLOBAL: A model with SGRTC, which is our proposal.

Table II summarizes the configurations of TC and RTC used in these models. The capacity of RTC in S_GLOBAL is set to 1K traces that has a similar area to that of the conventional I-cache. In each model, each trace includes two branches. The TC and RTC in the other models have the same capacity. Hit/miss of TC and RTC is determined in the first cycle, and then the I-cache is accessed on TC/RTC misses. In GLOBAL, the number of indirect jumps stored to a tag array is two(Section IV-B).

The RMT in BASE and TC has 32 ports in total. An RMT generally requires 1-write and 3-read ports per a single

[6]Details are described in [12], [21]

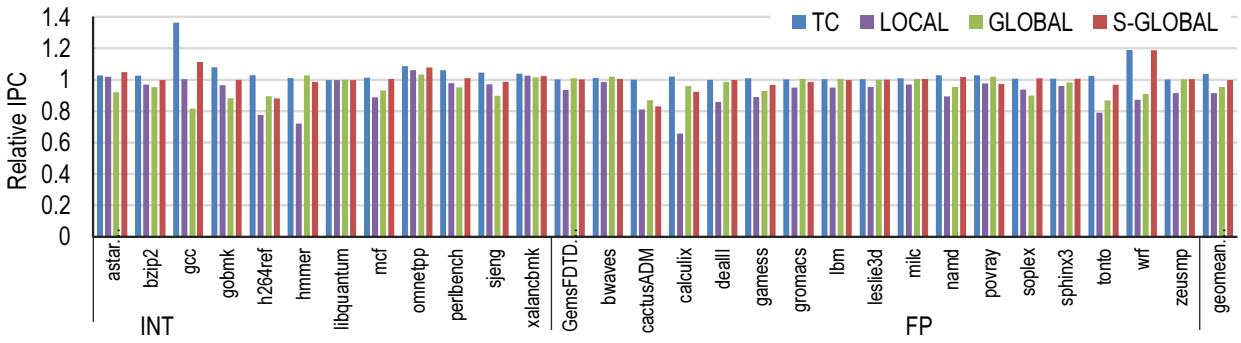

Fig. 9. IPC Relative to BASE

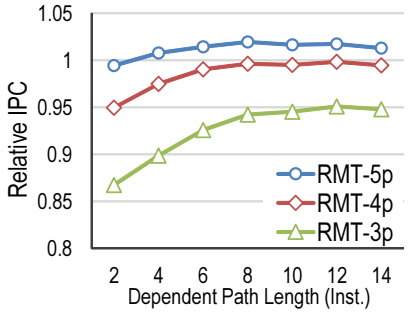

Fig. 10. IPC Versus RMT Read Ports and Dependent Path Length

2-source operand instruction[17], [24]: 1) 1-write port for updating new destination mapping, 2) 2-read port for reading source operand mapping, and 3) 1-read port for reading old destination mapping. As a result, each instruction with 2-source operand requires 4 ports. We assume that the rename-width of the processor is 8 as presented in Table I, and thus, the RMT requires $4 \times 8 = 32$ ports in total. In the other models, the number of the read ports of the RMT is reduced. The specific parameters of the read ports are determined by the evaluation in the next section.

C. Investigating Configurations

First, for determining the configuration of S_GLOBAL, we evaluated its performance while varying its dependent path length and the number of read ports of its RMT. Figure 10 shows the IPCs of S_GLOBAL relative to BASE. These IPCs are on the geometric mean of all benchmark programs. Each line labeled as "RMT-Np" shows the IPCs of a configuration with an RMT that has N read ports. Figure 10 shows that IPCs degrade if the number of read ports of the RMT is less than 4, and in the configuration with the 4-read ports RMT, the configuration whose dependent path length is 12 instructions has the highest performance. Thus, hereafter, we evaluate S_GLOBAL with a 4-read ports RMT and 12-instructions dependent path length.

D. IPC

Figure 9 shows the IPCs for each model relative to BASE. S_GLOBAL degrades the IPC of BASE by only 0.2% on a geometric mean. As described in Section V-C1, our proposal improves the capacity-efficiency, and consequently, its performance penalty described in Section IV-A is considerably reduced, and it cancels each other out by the improved fetch throughput in the same manner as TC. On the other hand, S_GLOBAL degrades the IPC of TC by 3.7% on the geometric mean, because TC also improves the fetch throughput.

TABLE III. RATIO OF CACHED RENAMED OPERANDS

LOCAL	GLOBAL	S_GLOBAL
31.7%	69.3%	57.3%

LOCAL degrades the IPC of BASE by 8.5% on the geometric mean, and its IPC degradation is considerably larger than that of S_GLOBAL. This is because LOCAL can cache only 31.7% renamed operands on average as shown in Table III, and this ratio is about half of that in S_GLOBAL. As a result, the shortage of the RMT ports stalls the front-end and it degrades its performance.

GLOBAL also degrades the IPC of BASE by 4.5% on the geometric mean. This is because of its low capacity-efficiency described in Section IV-A.

E. Energy Consumption

Figure 11 shows the energy consumption[7] of the RMT in each model normalized by BASE. This energy consumption is on the average of all benchmark programs. "overhead" in each model shows the overhead energy by the addition of the TC or RTC compared to BASE. This overhead energy includes the energy consumption for reading the instruction cache and filling generated traces on RTC miss. In S_GLOBAL, the RMT is used for renaming source operands stored as logical register numbers into traces on RTC hit, as described in Section V-C1. The part labeled as RMT in Figure 11 includes the energy consumption for this renaming.

S_GLOBAL considerably reduces the energy consumption of the RMT including its overhead. The energy consumption of the RMT in S_GLOBAL is reduced by 64% compared with BASE, because both the area and the access frequency of the RMT are reduced. The energy consumption is reduced by 65%, 23%, and 60% compared with TC, LOCAL, and GLOBAL, respectively. LOCAL does not reduce its energy consumption compared to S_GLOBAL, because LOCAL cannot cache renamed operands sufficiently, as described above. GLOBAL has a large overhead caused by the tag array in RTC as described in Section IV, and consequently, this overhead cancels out the reduction of the energy consumption in the RMT. The energy consumption of the RMT without the overhead in GLOBAL is smaller than that of the RMT in S_GLOBAL, and those are 16% and 22% of that of the RMT in BASE, respectively. This is because renaming is completely omitted

[7]This includes both dynamic and static energy consumption.

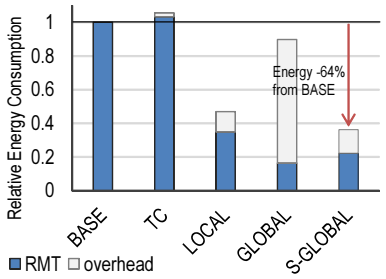

Fig. 11. Energy Consumption Relative to BASE

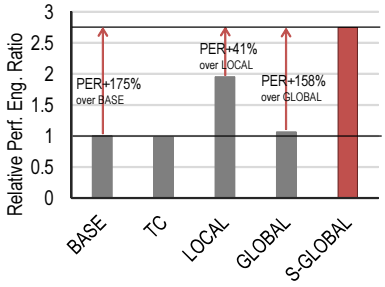

Fig. 12. Performance Energy Ratio (Inverse of EDP) Relative to BASE

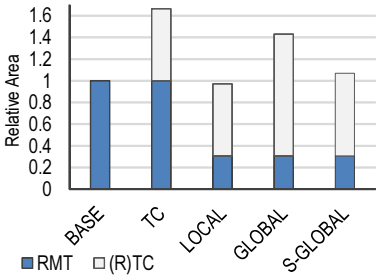

Fig. 13. Area Relative to BASE

in GLOBAL on RTC hit, but few operands are still renamed in S_GLOBAL as described in Section V-C1. However, this energy consumption for renaming on RTC hit in S_GLOBAL is negligible, and the total energy consumption in S_GLOBAL is significantly reduced compared with that in GLOBAL and BASE as described before.

F. Performance Energy Ratio

In this section, we show a performance/energy ratio (PER) of each model, which is equal to the inverse of energy-delay product (EDP), and a higher PER shows better efficiency. Figure 12 shows the PER relative to that of BASE. S_GLOBAL considerably improves its PER compared to the other models because the performance is improved/maintained and the energy consumption is reduced at the same time. Figure 12 shows that the PER of S_GLOBAL is improved by 175%, 180%, 41%, and 158% compared to BASE, TC, LOCAL, and GLOBAL, respectively.

G. Area

Figure 13 shows the circuit area of the RMT and RTC in each model. These areas are normalized by that of BASE. S_GLOBAL considerably reduces the area of the RMT compared to BASE, but the addition of the RTC slightly increases its total area by 7.0%.

VII. Conclusion

We propose a SGRTC that caches only renamed operands whose distances from producers outside traces are short. Evaluation results show that SGRTC achieves 64% lower energy consumption for register renaming with only 0.2% performance overhead compared to a conventional processor. Further, the evaluation results show that the proposed method achieves 8.5% and 4.5% higher performance, 23% and 60% lower energy consumption, and 41% and 151% higher performance-energy ratio (the inverse of energy-delay product) than conventional LRTC and GRTC, respectively.

Acknowledgment

This work was supported by JSPS KAKENHI Grant Number 24680005.

References

[1] B. Black, B. Rychlik, and J. P. Shen, "The block-based trace cache," in *ACM SIGARCH Computer Architecture News*, vol. 27, no. 2, 1999, pp. 196–207.

[2] E. Fluhr, J. Friedrich, D. Dreps, V. Zyuban, G. Still, C. Gonzalez, A. Hall, D. Hogenmiller, F. Malgioglio, R. Nett, J. Paredes, J. Pille, D. Plass, R. Puri, P. Restle, D. Shan, K. Stawiasz, Z. Deniz, D. Wendel, and M. Ziegler, "Power8: A 12-core server-class processor in 22nm soi with 7.6tb/s off-chip bandwidth," in *Proc. of the Int. Solid-State Circuits Conference*, 2014, pp. 96–97.

[3] T. R. Halfhill, "Oracle says sparc is tops," *Microprocessor Report 4/15/13*, pp. 1–6, 4 2013.

[4] H. Ichibayashi, R. Shioya, H. Irie, M. Goshima, and S. Sakai, "Anti-dualflow architecture," *IPSJ Trans. on Advanced Computing Systems*, vol. 1, no. 2, pp. 22–33, 2008.

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

[6] K. Krewell, "Intel's haswell cuts core power," *Microprocessor Report 9/24/12*, pp. 1–5, September 2012.

[7] K. Krewell, "Power8 muscles up for servers," *Microprocessor Report 9/2/13*, pp. 1–6, September 2013.

[8] T. Liu and S.-L. Lu, "Performance improvement with circuit-level speculation," in *Proc. of the Int. Symp. on Microarchitecture*, 2000, pp. 348–355.

[9] S. Manne, A. Klauser, and D. Grunwald, "Pipeline gating: Speculation control for energy reduction," in *Proc. of the Int. Symp. on Computer Architecture*, 1998, pp. 132–141.

[10] A. Moshovos, "Power-aware register renaming," *Computer Engineering Group Technical Report, University of Toronto*, pp. 01–08, 2002.

[11] A. Moshovos, "Checkpointing alternatives for high-performance power-aware processors," 2003, pp. 318–321.

[12] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Cacti 6.0: A tool to model large caches," HP Laboratories, Tech. Rep., 2009.

[13] S. Palacharla, N. P. Jouppi, and J. E. Smith, "Quantifying the complexity of superscalar processors," University of Wisconsin-Madison, Tech. Rep., Nov 1996.

[14] R. Preston, R. Badeau, D. Bailey, S. Bell, L. Biro, W. Bowhill, D. Dever, S. Felix, R. Gammack, V. Germini, M. Gowan, P. Gronowski, D. Jackson, S. Mehta, S. Morton, J. Pickholtz, M. Reilly, and M. Smith, "Design of an 8-wide superscalar risc microprocessor with simultaneous multithreading," in *Proc. of the Int. Solid-State Circuits Conference*, vol. 1, 2002, pp. 334–472.

[15] E. Rotenberg, S. Bennett, and J. E. Smith, "Trace cache: A low latency approach to high bandwidth instruction fetching," in *Proc. of the Int. Symp. on Microarchitecture*, 1996, pp. 24–35.

[16] E. Rotenberg, S. Bennett, and J. E. Smith, "A trace cache microarchitecture and evaluation," *IEEE Transactions on Computers*, vol. 48, no. 2, pp. 111–120, 1999.

[17] E. Safi, A. Moshovos, and A. Veneris, "On the latency and energy of checkpointed superscalar register alias tables," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 3, pp. 365–377, Mar. 2010.

[18] R. Sangireddy, "Reducing rename logic complexity for high-speed and low-power front-end architectures," *IEEE Transactions on Computers*, vol. 55, no. 6, pp. 672–685, 2006.

[19] 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, May 2011.

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

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

[22] S. Vajapeyam and T. Mitra, "Improving superscalar instruction dispatch and issue by exploiting dynamic code sequences," in *Proc. of the Int. Symp. on Computer Architecture*, 1997, pp. 1–12.

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

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