SWQUE: A Mode Switching Issue Queue with Priority-Correcting Circular Queue

Hideki Ando
Nagoya University
Nagoya, Chikusa, Aichi, Japan
ando@nuee.nagoya-u.ac.jp

ABSTRACT

The improvement of single-thread performance is much needed. Among the many structures that comprise a processor, the issue queue (IQ) is one of the most important structures that influences high single-thread performance. Correctly assigning the issue priority and providing high capacity efficiency are key features, but no conventional IQ organizations do not sufficiently have these.

In this paper, we propose an IQ called the *switching issue queue* (SWQUE), which dynamically configures the IQ as a modified circular queue (CIRC-PC) or random queue with an age matrix (AGE) by responding to the degree of capacity demand. CIRC-PC corrects the issue priority when wrap-around occurs by exploiting the finding that instructions that are wrapped around are latency-tolerant. CIRC-PC is used for phases in which capacity efficiency is less important and the correct priority is more important; and AGE is used for phases in which capacity efficiency is more important. Our evaluation results using SPEC2017 benchmark programs show that SWQUE achieved higher performance by averages of 9.7% and 2.9% (up to 24.4% or 10.6%) for integer and floating-point programs, respectively, compared with AGE, which is widely used in current processors.

CCS CONCEPTS

• **Computer systems organization** → **Superscalar architectures**.

KEYWORDS

issue queue, instruction scheduling

ACM Reference Format:
Hideki Ando. 2019. SWQUE: A Mode Switching Issue Queue with Priority-Correcting Circular Queue. In *The 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-52), October 12–16, 2019, Columbus, OH, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3352460.3358293

1 INTRODUCTION

Dennard scaling, which was the largest source of single-thread performance improvement, ended around 2005 [28] for several reasons, including the cooling problem. The clock frequency of a processor has hardly increased since then. Simultaneously, the interests of many computer architects have moved from core microarchitecture toward multicore and uncore architectures. Since then, few studies have been conducted on core microarchitecture. This academic trend has also made single-thread performance sluggish. However, improving single-thread performance is still strongly required for many applications.

Correct priority and capacity efficiency in the issue queue: For high single-thread performance, the issue queue (IQ) plays an important role. The IQ schedules instructions to be executed by selecting and issuing instructions from the ready-to-execute instructions in each cycle. For higher performance, two features are required: 1) correct priority and 2) large capacity.

It is essential that the IQ assigns the *correct priority* when scheduling instructions. We say that the priority of the instructions is correct if the execution time is minimized when these instructions are scheduled based on their priority. Theoretically, the correct priority is assigned based on the height of the node of the instruction in the dataflow graph. Intuitively, the higher the node of an instruction, the longer the total execution time until the leaf instruction is executed, and thus, this instruction should be executed earlier. Consider two instructions: instruction A, which is off the critical path of the dataflow graph, and instruction B, which is the highest node on the critical path. If the priority assigned to A is higher than that assigned to B, then this can hinder the issuing of instruction B. If the issuing of instruction B is delayed, then this will increase the execution time. Unfortunately, it is difficult for hardware to assign the correct priorities to instructions. However, it is widely known that age-based priority, where a higher priority is assigned to older instructions, is sub-optimal. The rationale behind this heuristic is that a critical path is composed of a long dependence chain; thus, the instructions on a critical path remain in the IQ for a long time. Older instructions are therefore likely to be those on a critical path. Hereafter, we say that a priority is correct if it is assigned based on age.

Capacity is also an important feature for high performance. The larger the IQ, the higher the probability that ready instructions will be found. Apparently, programs with a large amount of instruction-level parallelism (ILP) require this feature. High capacity is also important for memory-intensive programs, in which memory-level parallelism (MLP) is the main source for high performance. If the IQ is larger, more loads will be allowed to be executed for a short time. If multiple loads cause cache misses, then their memory accesses overlap. MLP can hide the memory access times of several loads using the memory access time of the counterpart. This reduces the negative impact of cache misses on the execution time, thereby

MICRO-52, October 12–16, 2019, Columbus, OH, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6938-1/19/10…$15.00
https://doi.org/10.1145/3352460.3358293

improving performance. Given the amount of hardware allocated to the IQ, *capacity efficiency* is important for an effectively large capacity. Capacity efficiency is defined as the ratio of the number of instructions held in the IQ to the total number of IQ entries.

IQ organizations: There are three types of IQ: shifting queue (SHIFT) [9], circular queue (CIRC), and random queue (RAND). SHIFT achieves the highest IPC among these types because it assigns perfectly correct priorities and has good capacity efficiency. However, the circuit is complex, and thus its delay is longest among the types of IQ and it consumes a large amount of power. Therefore, SHIFT was only used in old processors with a small IQ and is not currently in use.

By contrast, the delays of CIRC and RAND are short because of the circuits' simplicity. However, the IPC of CIRC is low because of incorrect priorities caused by wrap-around and low capacity efficiency. CIRC is also not used in current processors.

RAND offers good capacity efficiency; however, the priority is incorrect because of the random ordering of instructions. Therefore, RAND is not used alone. A circuit called the *age matrix* [22, 24], which selects the single oldest ready instruction, is used together with RAND in current processors [11, 22, 26]. We call this organization AGE. AGE considers the ages of instructions; however, it only assigns the single oldest instruction to the highest priority, and does not assign the correct priority to all instructions.

Proposal: In this paper, we propose a novel IQ organization called the *switching issue queue* (SWQUE), which achieves correct prioritization and high capacity efficiency when either property is more needed using a simple circuit. SWQUE dynamically configures the IQ as a modified CIRC or AGE based on the degree of capacity demand, which is monitored during execution.

For less capacity-demanding phases (i.e., priority-sensitive phases), SWQUE operates as a modified CIRC. Here, the modified CIRC, which we call the *priority-correcting CIRC* (CIRC-PC), essentially uses the CIRC organization, but it correctly prioritizes instructions even when wrap-around occurs. The scheme corrects the priority by exploiting the finding that the instructions that are wrapped around are *latency-tolerant*. Capacity efficiency remains low, similar to the original CIRC; however, CIRC-PC is used for phases in which the correct priority is more important than capacity efficiency. Always assigning the correct priority to instructions achieves high performance.

By contrast, for capacity-demanding phases, SWQUE operates as an AGE with high capacity efficiency. The correct priority is not assigned to all instructions; however, high capacity efficiency still delivers high performance.

Section organization: The remainder of this paper is organized as follows: In Section 2, we explain the organization of conventional IQs to aid the understanding of the organization of SWQUE. We propose SWQUE in Section 3 and present the evaluation results in Section 4. In Section 5, we describe related work. Finally, we present our conclusions in Section 6.

2 ORGANIZATION AND CIRCUIT OF CONVENTIONAL IQS

In this section, we explain the organization and circuit of conventional IQs to aid the understanding of SWQUE and CIRC-PC.

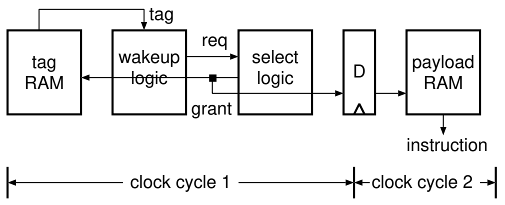


Figure 1: Organization of the IQ.

2.1 Basic Organization

There are two types of IQ organizations, depending on the type of wakeup logic circuit: content-addressable memory (CAM) and RAM [13]. These are both used in commercial processors. For example, the CAM type is used in the AMD Bulldozer [11], whereas the RAM type is used in the IBM POWER8 [26]. We assume the CAM type in this paper. Applying the SWQUE scheme to the RAM-type IQ is our future work.

The IQ mainly comprises the wakeup logic, select logic, tag RAM, and payload RAM [13, 20, 30], as illustrated in Figure 1. The wakeup logic is a one-dimensional array in which each wakeup logic entry holds the tags for two source operands and ready flags indicating the data dependence state (resolved or not) for the corresponding instruction. If both data dependences are resolved, an issue request is sent to the select logic, which grants some requests by considering resource constraints. The grant signals are sent to the payload RAM that holds instructions. Instructions are read (issued) from the payload RAM and sent to the function units. The grant signals are also sent to the tag RAM, the wordlines of the tag RAM are directly connected to the grant signals (i.e., there is no address decoder), and the destination tags are read by responding to the grant signals. These tags are broadcast to the wakeup logic to update the ready flags.

The issue operation described above is pipelined. The wakeup, select, and tag RAM read are performed in the first cycle whereas the payload RAM is read in the second cycle. The first cycle operations are not generally pipelined; if pipelined, dependent instructions cannot be issued back-to-back;[1], thus the IPC is degraded. The path composed of these three operations is the critical path of the IQ, and is a critical path of the processor [21].

2.2 Circuits

In this section, we briefly describe the circuits of the conventional IQ. The following three sections explain the wakeup logic, select logic, and tag RAM.

2.2.1 Wakeup Logic. The wakeup logic is composed of CAM. Each entry in CAM holds the tags for two source operands of an instruction and two ready flags that indicate whether the corresponding source operand is ready. The *IW* destination tags of issued instructions are broadcast to all *IQS* entries in CAM using the search lines, which we call *tag lines*. Here, *IW* and *IQS* denote the issue width and IQ size, respectively. The broadcast destination tags are compared with two source operand tags in each entry. If there is a match, then the ready flag is set. If both ready flags that correspond

[1]Pipelining these operations was studied in [7, 27] and is discussed in Section 5.

to two operands are set, then an issue request is output to the select logic.

2.2.2 Select Logic. The select logic receives an issue request signal for each entry and outputs IW grant signals ($grant_0, \ldots, grant_{IW-1}$) for each entry, where each grant signal corresponds to a specific issue port [12, 30].

The most widely known circuit used as select logic is a *tree arbiter* [21]. This circuit uses a tree structure to connect multiple small arbiter cells. For example, if the small arbiter cell arbitrates four requests, then the overall tree arbiter is organized using a quad-tree. For an IW-issue IQ, IW tree arbiters are stacked and the grants are determined from the highest priority in order from the bottom of the stacked arbiters [12, 21].

Note that the priority in arbitration is not flexible but is fixed with respect to the IQ position to make the delay short. Recall that the IQ critical path is a critical path in a processor; thus, the shorter delay is strongly required for high performance. Generally, the lower the position, the higher the priority.

Also note that $grant_0, \ldots, grant_{IW-1}$ correspond to the instructions in descending order in terms of the priority [12, 21]; that is, $grant_0$ is the grant signal of the instruction with the highest priority, $grant_1$ is that of the instruction with the second-highest priority, and so on. This ordering is used because the grants are determined from the highest priority in order, as previously described.

Another select logic circuit exists that uses prefix-sum logic [12, 30]. Although this logic is faster than the tree arbiter, it is not different in terms of the fixed position-based priority and grant signal ordering.

2.2.3 Tag RAM. The tag RAM consists of an SRAM that stores destination tags without an address decoder. It has IW ports, with IW-bit grant lines per entry from the select logic directly connected to the IW wordlines per entry. Up to IW destination tags held in the SRAM cells connected to the asserted wordlines are read.

Note that, for this study, the tag RAM outputs bogus data from a read port when the corresponding wordline is not asserted (i.e., no instruction is issued from the corresponding issue port). The bogus data are zeros for all bits in current SRAM designs in nano-LSI technology, where eight-transistor (8T) cells are used rather than conventional six-transistor (6T) cells to ensure read margins under low supply voltages [29]. In fact, Intel switched from 6T to 8T cells for its 45nm line of Core processors [18]. When using 8T cells, each bitline is single and an inverter is used to read the value of the bitline. Therefore, if a wordline is not asserted, the precharged bitline is simply read, and the output is thus zero. The bogus data from the tag RAM do not represent a valid tag, but are functionally used as a *bogus tag* that is not used as a tag for any in-flight instructions, and thus never matches any of the source operand tags in the wakeup logic.

2.3 Taxonomy

There are three types of IQs in terms of instruction ordering. The first type is the *shifting queue* (SHIFT). SHIFT was used in old processors (e.g., DEC Alpha 21264 [9] two decades ago), where the IQ size was small. In the SHIFT, instructions stay physically ordered by age from the head to the tail of the queue. A *compaction circuit*, which computes shift amount and shifts instructions to fill the "holes" created by the instructions that have been issued, is responsible for this. Because the select logic is position based, the priority assigned to instructions is always perfect. Additionally, compaction achieves good capacity efficiency. However, the compaction circuit is highly complex, and is inserted into the critical path of the IQ [9]. Thus, the delay of the IQ is significantly increased. Additionally, compaction needs to perform numerous shifts of instructions, and thus consumes a large amount of power. SHIFT is no longer used in current processors.

The second type of IQ is the *circular queue* (CIRC), which is composed of a circular buffer. In this queue, instructions remain physically ordered by age, as per SHIFT, but it does not have a compaction circuit. Although this queue is simple, unlike SHIFT, remaining "holes" causes serious capacity inefficiency. This significantly degrades performance in capacity-sensitive programs. Additionally, wrap-around occurs in instruction order, and this reverses the issue priority, further degrading performance. CIRC was often assumed in previous studies on IQs (e.g., [6, 15]), but is not used in current processors.

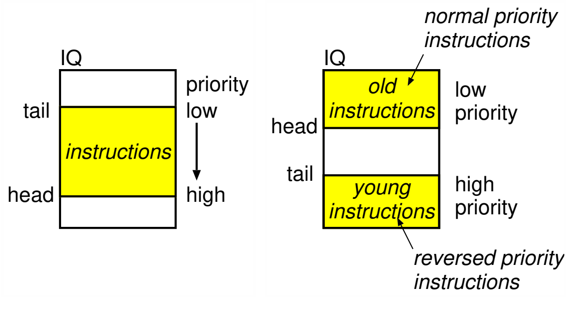
The last type of IQ is the *random queue* (RAND), where instructions are simply dispatched into the "holes." Although the capacity is fully used, the order of instructions in the queue becomes random because "holes" arise randomly in the long term. Thus, the issue priorities of the instructions are assigned randomly because of their random ordering. Therefore, the IPC is low in priority-sensitive programs.

To mitigate IPC degradation in the RAND, an additional circuit called the *age matrix* [22, 24] is used. We refer to this type of IQ as AGE. The age matrix is used in parallel with the select logic [11], and selects only the single oldest ready instruction. The circuit of the age matrix is a logic matrix, where each row and column of the matrix is associated with an instruction in the IQ. Each cell of the matrix holds a single bit that represents age ordering information. In each row, the circuit determines whether the input issue request is the oldest by bitwise ANDing the row vector stored in the matrix with the transposed issue request vector. Although the age matrix considers age, the priority correctness is imperfect because only the single oldest instruction is selected. Therefore, effectiveness is limited.

We have described all the published IQ organizations used in commercial processors to the best of our knowledge in this section. Although all processor vendors do not publish their IQ organizations, AGE is generally used in modern processors [11, 22, 26].

3 SWQUE: SWITCHING ISSUE QUEUE

In this section, we propose the *switching issue queue* (SWQUE). SWQUE configures the IQ as the priority-correcting CIRC (CIRC-PC) or AGE depending on the degree of capacity demand. We first explain CIRC-PC in Section 3.1. Then we explain the switching scheme in Section 3.2.



(a) No wrap-around (b) Wrap-around

Figure 2: Priority depending on whether wrap-around occurs in CIRC.

3.1 CIRC-PC: Priority-Correcting Circular Queue

In this section, we provide details on CIRC-PC. In Section 3.1.1, we explain the problem of the incorrect priority caused by wrap-around. Then we describe a possible straightforward circuit to solve this problem, and demonstrate its high complexity in Section 3.1.2. In Section 3.1.3, we describe our approach to this problem. Finally, we provide details on the circuits of CIRC-PC.

3.1.1 Reversed Priority Problem. In CIRC, the issue priority becomes incorrect when wrap-around occurs. Figure 2 illustrates how the priority becomes dependent on wrap-around, where the select logic circuit is configured such that instructions in lower entries are assigned a higher priority.

As shown in Figure 2(a), the priority is correct when wrap-around has not occurred. By contrast, as shown in Figure 2(b), the incorrect priorities are assigned to instructions when wrap-around occurs. Despite the instructions in the area from the head entry to the top of the queue being older than those in the area from the tail entry to the bottom of the queue, the former instructions are assigned a lower priority, whereas the latter instructions are assigned a higher priority. Hereafter, we call the instructions in the area close to the top the *normal priority instructions* (NR instructions), whereas those in the area close to the bottom are the *reversed priority instructions* (RV instructions). The priority of the RV instructions must be corrected so that it is lower than that of the NR instructions.

3.1.2 Possible Straightforward Circuit. One possible but straightforward circuit that solves the incorrect priority problem caused by wrap-around with only minimal modification of the IQ involves exchanging the requests from the NR instructions with those from the RV instructions in terms of the position by inserting an exchange network. The exchange network is essentially composed of many IQS-to-1 MUXes for each input of the select logic. These MUXes are controlled using the values of the head and tail pointers. Although this circuit can be implemented theoretically, the huge fan-out of request signals with vertically traversing long wires and huge fan-in of MUXes significantly increase the delay and energy consumption.

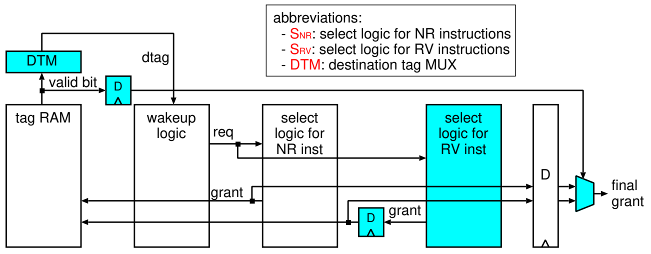


Figure 3: CIRC-PC organization.

3.1.3 Approach to Correct Reversed Priority. To enable CIRC-PC to correct the priority when wrap-around occurs, we prepare an additional select logic denoted by S_{RV} for the RV instructions only; the original select logic, which is denoted by S_{NR}, is used for the NR instructions only. Issue requests from the wakeup logic are first sent to either S_{NR} or S_{RV}, depending on whether they are from the NR or RV instructions, respectively. Some requests are then granted independently in either select logic. At this moment, unlike a normal IQ, instructions to be issued are not yet determined. The grant signals from each select logic then read destination tags from the tag RAM. Finally, up to IW tags are selected from these read destination tags to determine the instructions to be issued, with higher priority assigned to the tags that originated from S_{NR}. In this scheme, the final tag selection process is responsible for the correction of the issue priority because higher priority is assigned to the tags of the NR instructions.

Although the scheme described above successfully corrects the priority, it has a serious implementation problem, because it doubles the number of ports of the tag RAM, which significantly increases the delay of the IQ. Therefore, we implement time-sliced tag RAM access; that is, the tag RAM is accessed at the end of each clock cycle as per normal for the NR instructions, whereas it is accessed at the beginning of the *next* clock cycle for the RV instructions. Because the tag RAM is a small circuit (see Figure 13), doubling the number of accesses is possible without increasing the IQ delay. However, the downside is that the wakeup–select for the RV instructions requires two cycles. This virtually increases the latency of the RV instructions by one cycle, and thus the IPC is reduced. However, our evaluation in Section 4.4 indicates that this adverse effect is very small. This means that ready instructions with low priority (i.e., ready RV instructions, which are placed near the tail of the queue and are thus young) are off the critical path and thus latency-tolerant.

3.1.4 Organization Details . Figure 3 shows the organization of the CIRC-PC. The blue boxes represent added logic. The additional select logic, S_{RV}, is placed on the right-hand side of S_{NR}, and receives issue requests for the RV instructions from the wakeup logic. D-FFs are placed between S_{RV} and S_{NR} to allow the grant signals from S_{RV} to be sent to the tag RAM in the next cycle. The final tag selection, which we call *tag merge*, is performed in the logic called the *destination tag multiplexer* (DTM), which is placed above the tag RAM.

Figure 4 shows the pipeline timing chart, which explains the detailed timing of the wakeup–select operation. Instructions `i0`,

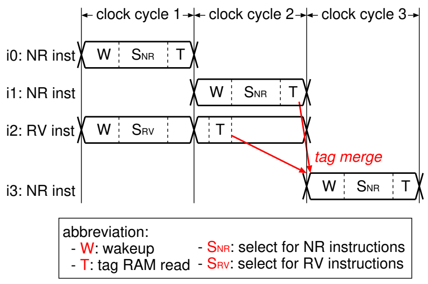


Figure 4: Timing of operations in CIRC-PC. The length of each phase in the chart is proportional to the delay evaluated by our HSPICE simulation.

i1, and i3 are NR instructions, whereas instruction i2 is an RV instruction. In the figure, the phases of W and T represent the wakeup and tag RAM read, respectively, whereas the phases of S_{NR} and S_{RV} represent the selection for the NR and RV instructions, respectively. The length of each phase in the chart is proportional to the delay evaluated by our HSPICE simulation of the IQ for our default processor model described in Section 4.1.

For NR instructions i0 and i1, W, S_{NR}, and T are performed in a single cycle as per normal. By contrast, for RV instruction i2, only W and S_{RV} are performed in the first cycle and T is then performed at the beginning of the next cycle[2]. The destination tags of the RV instructions read from the tag RAM are temporarily stored in the DTM (see Figure 3). To achieve this, the DTM contains latches called *pending tag latches* (PTLs). (The DTM circuit is explained in detail in Section 3.1.5.) After the NR instructions are read from the tag RAM, the DTM merges the tags of the NR instructions i1 with the tags stored in the PTLs, with higher priority assigned to the tags of the NR instructions in the merging operation (see the red line in Figure 4). The merged tags are output to the tag lines of the wakeup logic. Therefore, the correction of the priority is performed during the merging operation in the DTM.

There are several concerns about this implementation, which are described as follows, but they are not substantial problems:

- The issue operation of the RV instructions takes two cycles, which reduces the IPC. However, the RV instructions are young, and are thus unlikely to be on the critical path of the dataflow if they are ready. This implies that the ready RV instructions are *latency-tolerant*, and the increase in the number of cycles of the issue operation thus hardly affects the IPC. Our evaluation in Section 4.4 indicates that the adverse impact of this increase on performance is only 1.1% on average.
- The DTM increases the IQ delay. However, it only represents 1.3% of the entire IQ delay, according to our LSI design and HSPICE simulation, as described in Section 4.7.
- The tag RAM is accessed twice within a single cycle. To achieve this, double the delay of a single tag RAM read including the precharge time must be accommodated within a single cycle. According to our LSI design and HSPICE simulation results (for the methodology, see Section 4.1), the total delay of the double tag RAM accesses including the precharge time is 66% of the entire IQ delay, and thus there is a large margin.
- The additional select logic increases the processor cost. However, the impact on the core or entire chip in a current commercial processor (Intel Skylake) is only 0.034% or 0.010%, respectively.

[2]The blank at the beginning of the cycle in the timing chart represents the precharging of the tag RAM for the immediate next read. For the wakeup and select logic, one precharges whereas the other operates, and thus precharging is hidden.

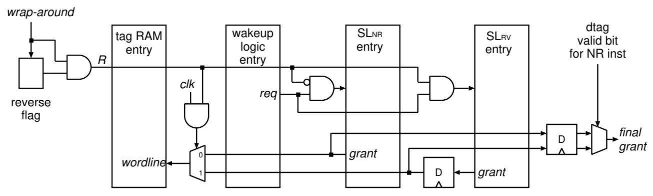


Figure 5: Entry slice circuit of CIRC-PC.

3.1.5 *Circuit Details.*

In this section, we provide details on the circuits that must be added to those of the conventional IQ.

Entry Slice: Figure 5 illustrates the circuits for the entry slice of the CIRC-PC, where the entry slice represents a single row of the IQ. The reverse flag (which is not illustrated in Figure 3) indicates that the corresponding instruction is an RV instruction at dispatch time. This flag is set when an instruction is dispatched if wrap-around occurs; otherwise, it is cleared. Signal R, which is the reverse flag ANDed with the signal that indicates that the instructions in the IQ are currently wrapped around, controls whether the issue request signal from the wakeup logic is input to S_{NR} or S_{RV}; that is, if R is true, the request signal is input to S_{RV} and is masked for S_{NR}; otherwise, the opposite operation is performed. Similarly, grant signals are selected by the clock signal masked by R (the clock is high during the first half of the clock cycle and low for the remaining time). If the control signal to the MUX is true, then the grant signals from S_{RV} are selected; otherwise, the grant signals from S_{NR} are selected.

The delay of this circuit is only increased by the single AND gate and MUX. Thus, it is negligibly small.

DTM: The DTM is used for *tag merge*, where the tags of the NR instructions and those of the RV instructions stored in the PTLs are merged, with higher priority assigned to the tags of the NR instructions. Figure 6 shows the circuit ($IW = 4$ as an example). The MUXes select the tags output to the wakeup logic from the tags of NR instructions ($dtag^{NR}_{0..3}$) and those in the PTLs ($dtag^{RV}_{0..3}$). The MUXes are controlled using valid bits ($V_{0..3}$) to indicate that the tags of the NR instructions are valid. We modified the tag RAM to output the validity of each output data, where the validity is created by the wordlines. This means that if the output tag is valid, the corresponding valid bit is true; otherwise (i.e., bogus tag), it is false. The tags of the NR instructions are prioritized during selection as follows: if V_i is true, then $dtag^{NR}_i$ is selected; otherwise, the tag of the PTL is selected.

Suppose that the priority is higher when the suffix of $dtag$ is smaller for both the NR and RV instructions; that is, the priorities of $dtag^{NR}_0$ to $dtag^{NR}_3$ and those of $dtag^{RV}_0$ to $dtag^{RV}_3$ are both

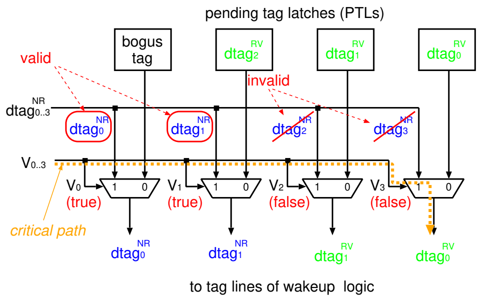


Figure 6: Destination tag MUX circuit (DTM) with $IW = 4$ for tag merge.

Table 1: Tag merge examples.

(a) example 1

inst	destination tag			
NR	$dtags_0^{NR}$	$dtags_1^{NR}$	invalid	invalid
RV	bogus	$dtags_2^{RV}$	$dtags_1^{RV}$	$dtags_0^{RV}$
merged	$dtags_0^{NR}$	$dtags_1^{NR}$	$dtags_1^{RV}$	$dtags_0^{RV}$

(b) example 2

inst	destination tag			
NR	$dtags_0^{NR}$	invalid	invalid	invalid
RV	bogus	bogus	$dtags_1^{RV}$	$dtags_0^{RV}$
merged	$dtags_0^{NR}$	bogus	$dtags_1^{RV}$	$dtags_0^{RV}$

descending in terms of the priority. Note that tags are aligned on one side in terms of priority because of the select logic circuit, as described in Section 2.2.2. As shown in Figure 6, we arrange the tags, $dtag_i^{NR}$ and $dtag_i^{RV}$, in opposing order as inputs to the MUXes. This means that $dtag_0^{NR}$ to $dtag_3^{NR}$ are aligned from left to right, whereas $dtag_0^{RV}$ to $dtag_3^{RV}$ are aligned from right to left. This opposing alignment successfully merges tags in terms of priority.

In the example shown in Figure 6, two $dtags_i^{NR} (i = 0, 1)$ are valid, whereas three $dtags_i^{RV}$ $(i = 0, 1, 2)$ in the PTLs are valid (the remaining PTL holds a bogus tag). The same example is presented in Table 1(a) for clarification. In this case, the two MUXes on the left-hand side output $dtag_0^{NR}$ and $dtag_1^{NR}$, whereas the two MUXes on the right-hand side output $dtag_0^{RV}$ and $dtag_1^{RV}$. Note that $dtag_2^{RV}$ remains, but it is simply discarded during this cycle (i.e., the PTL is overwritten in the next cycle).

Another example is shown in Table 1(b). In this example, only a single valid tag appears for the NR instructions, whereas two valid tags are held in the PTLs for the RV instructions. As shown in the merge results, the MUX second from the left outputs a bogus tag.

As described previously, the DTM increases the IQ delay. We indicate the critical path of the DTM with the broken orange line in Figure 6. This path includes the valid bit lines because the PTLs are updated at the beginning of each clock cycle, whereas selection using the valid bits is performed immediately before the tag broadcast to the wakeup logic. We evaluate the increased delay in Section 4.7.

Finally, we explain how the final grant signal $grant\_final_i$ output to the payload RAM is generated from the two sets of grant signals: $grant_i^{NR}$ (the grant from S^{NR}) and $grant_i^{RV}$ (the grant from S^{RV}) $(i = 0, \ldots, IW - 1)$. As performed in the DTM, the final grant signal is selected using the valid bits for the NR instructions from the tag RAM (V_i). Given that the tags are merged in opposing order in terms of priority in the DTM, the final grant signals are obtained as follows:

$$grant\_final_i = V_i \cdot grant_i^{NR} \vee V_i' \cdot grant_{IW-i-1}^{RV}.$$

Note that this operation is performed in the pipeline stage in which the payload RAM is read (see Figures 3 and 5). Because the access time for the small payload RAM is short (only 43% of the IQ critical path delay according to our HSPICE simulation), this additional operation does not increase the clock cycle time.

3.2 Switching Scheme between CIRC-PC and AGE

The IQ configuration is switched between CIRC-PC and AGE, depending on the degree of capacity demand. If the capacity demand is high, then the IQ is configured as AGE; otherwise, it is configured as CIRC-PC.

3.2.1 Metrics to Estimate the Degree of Capacity Demand. Two metrics are used to estimate the degree of capacity demand that corresponds to two performance sources, ILP and MLP. For MLP, a large capacity is beneficial for memory-intensive phases because if more loads are issued within a short time, then many last-level cache (LLC) misses can be overlapped with higher probabilities. Therefore, we monitor the frequency of LLC misses, specifically in terms of LLC misses per kilo-instructions (MPKI). If the MPKI is higher than a predetermined threshold, then the current phase is considered to be capacity-demanding.

For ILP, we monitor the frequency of instruction issues from the predetermined lowest priority region of the IQ. We call this metric the frequency of low-priority instruction issues (*FLPI*). If the FLPI is high, this means that ready instructions reside throughout the IQ; that is, many ready instructions reside in the lowest priority region, not only in the higher priority region. We thus determine that a phase is capacity-demanding if the FLPI is higher than a predetermined threshold.

3.2.2 Reconfiguration of the IQ. The reconfiguration of the IQ is determined periodically based on two metrics. If the MPKI and FLPI are both high or both low in the current interval, then the IQ will be configured as AGE or CIRC-PC in the next interval, respectively. If they disagree in the current interval, then the IQ is configured as AGE in the next interval. This AGE-favoring policy achieves better performance than the CIRC-favoring policy, according to our evaluation.

The IQ hardware reconfiguration is simple. The head and tail pointers and wrap-around signal (see Section 3.1.5) operate in CIRC-PC mode, whereas a free entry list for the IQ is maintained in AGE mode. In the AGE configuration, the wrap-around signal is always set to false. This disables the use of SL_{RV} and any requests go to SL_{NR}. When switching between AGE and CIRC-PC modes, the scheme flushes the pipeline and ensures that the necessary

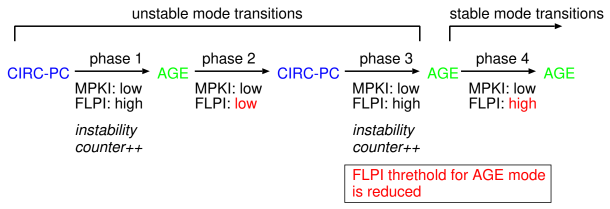


Figure 7: Stabilizing the mode transition using the instability counter.

structures work. Because of the requirement for a pipeline flush, a penalty is imposed on switching. This penalty is similar to that for branch misprediction.

3.2.3 Stabilizing Mode Transitions. Two FLPI thresholds are required that correspond to AGE and CIRC-PC modes. This causes an instability problem in the mode transitions during low MPKI phases. It is difficult to determine the thresholds for each mode, which causes the state to remain in an optimal mode with higher stability, according to our preliminary evaluation. In the worst case scenario, each mode repeatedly dictates that the other mode is beneficial, and the mode goes back and forth as a result. To stabilize the mode transition, we emphasize the decision in CIRC-PC mode, and dynamically adjust the FLPI threshold in AGE mode to make AGE mode more likely to remain unchanged.

For this purpose, we prepare a single small saturated resetting counter, called the *instability counter*, which evaluates the mode transition instability. The counter is incremented if the FLPI decision in CIRC-PC mode determines that AGE mode is beneficial; otherwise, it is reset to zero. If the instability counter reaches a predetermined value, then we determine that the mode transition is unstable. In this case, the FLPI threshold AGE mode is reduced by a predetermined value. This makes AGE mode more likely to remain unchanged, and the mode transition thus becomes more stable. Additionally, we periodically reset both the instability counter and FLPI threshold in AGE mode to restart learning and adapt phase changes.

Figure 7 illustrates an example of this behavior. In this example, the instability counter is initially zero, and its threshold is 2. Additionally, the MPKI is low through all phases. The mode starts from CIRC-PC mode. At the end of phase 1, suppose that the FLPI is found to be high. The mode is thus transferred to AGE mode. This mode transition increments the instability counter. Next, suppose that the FLPI is low in phase 2. The mode is then transferred back to CIRC-PC mode. Next, suppose that the FLPI is high again in phase 3. The mode is then transferred back to AGE mode again. The instability counter is incremented at this time, and reaches the threshold, thereby determining that the mode transition is unstable. Thus, the FLPI threshold for AGE mode is reduced so that remaining in AGE mode is more likely. With this low threshold, the FLPI becomes high in phase 4, unlike in phase 2. Accordingly, the mode transition becomes more stable, remaining in AGE mode.

Table 2: Base processor configuration.

Pipeline width	6-instruction wide for each of fetch, decode, issue, and commit
Reorder buffer	256 entries
IQ	128 entries
Load/store queue	128 entries
Physical registers	256(int) + 256(fp)
Branch prediction	12-bit history 4K-entry PHT gshare, 2K-set 4-way BTB, 10-cycle misprediction penalty
Function unit	3 iALU, 1 iMULT/DIV, 2 Ld/St, 2 FPU
L1 I-cache	32KB, 8-way, 64B line
L1 D-cache	32KB, 8-way, 64B line, 2 ports, 2-cycle hit latency, non-blocking
L2 cache	2MB, 16-way, 64B line, 12-cycle hit latency
Main memory	300-cycle min. latency, 8B/cycle bandwidth
Data prefetch	stream-based: 32-stream tracked, 16-line distance, 2-line degree, prefetch to L2 cache

4 EVALUATION RESULTS

We first describe the evaluation methodology in Section 4.1. We then compare the performance of SWQUE with that of the conventional IQ organizations in Section 4.2. In Section 4.3, we evaluate how effective SWQUE will be in a future large-sized processor. In Section 4.4, we evaluate the effectiveness of CIRC-PC by comparing it with the performance of the idealized CIRC. Next, we evaluate the energy consumption and area overhead of SWQUE in Sections 4.5 and 4.6, respectively. Then, we evaluate the circuit delay specific to SWQUE in Section 4.7. In Section 4.8, we evaluate the performance impact of the mode switch penalty. Finally, we evaluate the performance of SWQUE and AGE when we enhance the AGE scheme using multiple age matrices in Section 4.9.

4.1 Methodology

We built a simulator based on the SimpleScalar Tool Set version 3.0a [1] to evaluate the IPC. The instruction set used was Alpha ISA. We used all the programs from the SPEC2017 benchmark suite except *gcc* and *wrf*; these programs were excluded because they do not run correctly on our simulator at present. The programs were compiled using gcc version 4.5.3 with option -O3.

The configuration of the base processor used for the evaluation is summarized in Table 2.

We simulated 100M instructions after the first 16B instructions were skipped using the *refspeed* inputs. The parameters specific to SWQUE are summarized in Table 3.

In addition to the IPC evaluation, we evaluated the area and delay of the IQ circuits. We designed the IQ at the transistor level, assuming MOSIS design rules [2]. We then performed a circuit simulation using HSPICE to evaluate the delay, assuming the 16nm predictive transistor model [3] developed by Arizona State University for HSPICE, and the resistance and capacitance per unit length of the wire predicted in the International Technology Roadmap for Semiconductors [16]. Drivers and repeaters were optimally inserted on

Table 3: Parameters for SWQUE.

Switch interval	10k instructions
Switch penalty	10 cycles
Switch MPKI threshold	1.0
FLPI threshold	0.04
Instability counter threshold	2
Reduction of FLPI threshold at instability	0.01
Instability counter reset interval	1M instructions

long wires to reduce the delay, in accordance with experimentation results.

4.2 Performance Comparison

Before presenting a detailed performance comparison of SWQUE with AGE, we present the average performance comparison results for various conventional IQs. Figure 8 shows the performance (IPC) degradation of the various IQs relative to that of SHIFT for integer (INT) and floating-point (FP) programs on the geometric mean ("GM int" and "GM fp," respectively). The longer the bars, the worse the performance; thus, shorter bars are better. Note that SHIFT is the best in terms of the IPC in the various IQs evaluated because of its perfectly correct priority and highest capacity efficiency.

As shown in Figure 8, the performance of CIRC and RAND significantly degraded in both the INT and FP programs by more than 10% compared with SHIFT. The performance degradation of RAND is mitigated by adding the age matrix (i.e., AGE), but there is still considerable room for improvement. By contrast, SWQUE achieves nearly the same performance as SHIFT in the INT programs (only 0.8% worse), and also achieves good performance in the FP programs (only 2.4% worse).

Figure 9 shows the speedup of SWQUE when compared with AGE for each of the programs. Two bars for each program represent the speedups in medium- and large-sized processors. The medium-sized processor is the default processor; the processor configuration is listed in Table 2, whereas the large-sized processor scales the seven parameters of the configuration as listed in Table 4; the other parameters remain unchanged from their default values. In this section, we discuss the results for the medium-sized processor; the results for the large-sized processor are discussed in Section 4.3. The notation above the graph for each program represents programs with a moderate amount of ILP (blue box), rich ILP (red box), or MLP (green box) programs. The IPC threshold between the moderate and rich ILP programs is 3.3, and that for MPKI, whether the program is MLP or not, is 3.0.

As shown in the figure, SWQUE achieves a significant speedup over AGE in five out of seven moderate ILP programs in the INT programs. In particular, in *deepsjeng, exchange2, leela* and *mcf*, the speedups are considerably higher by more than 10% (up to 24.4%). In the moderate ILP programs, the IQ is configured as CIRC-PC for most of the execution time, as shown in Figure 10. The results show that the correct prioritization for all instructions is performed very well by the CIRC-PC scheme, in contrast to AGE, where the prioritization is correct for only a single instruction. However, SWQUE achieves almost no speedup in the MLP programs because SWQUE

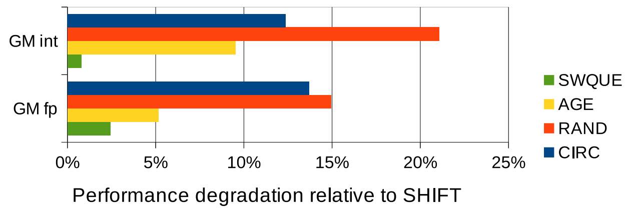


Figure 8: Performance degradation relative to that of SHIFT for various IQs.

is essentially configured as an AGE for these programs. On average, SWQUE achieves a 9.7% speedup in the INT programs.

In the FP programs, SWQUE again achieves a significant speedup in the moderate ILP programs, but no speedup in the MLP programs. The FP programs include rich ILP programs, unlike the INT programs. Rich ILP programs are capacity-demanding, and thus SWQUE is essentially configured as AGE, thereby achieving no speedup. Because moderate ILP programs constitute only half the total number of FP programs, the speedup is limited (2.9% on average).

4.3 Performance in a Large-Sized Processor

In this section, we discuss the results obtained when the processor size is scaled up to verify the effectiveness of SWQUE in the future. The results are shown as red bars in Figure 9. The larger the window size (i.e., the size of IQ, load/store queue, reorder buffer, and register file), the more issue conflicts occur, and the importance of the issue priority thus increased. Additionally, the IQ capacity shortage in the CIRC-PC mode of SWQUE is mitigated. Hence, SWQUE becomes more advantageous. By contrast, fewer the issue conflicts occur as the issue width and the number of function units increase, and the effectiveness of SWQUE thus decreases.

As shown in Figure 9, the speedup of SWQUE when compared with AGE is increased in most of the moderate ILP programs, compared with that in the medium-sized processor. In the capacity-demanding programs (rich ILP and MLP programs), there is almost no speedup as per the medium-sized processor. On average, the speedups in the INT and FP programs are 13.4% and 4.0%, respectively.

4.4 Evaluation of CIRC-PC

In this section, we discuss the effectiveness of priority correction in CIRC. Figure 11 shows the evaluation of the average performance of various CIRC configurations for INT and FP programs. The Y-axis represents the performance degradation relative to SHIFT. The longer the bars, the worse the performance; thus, shorter bars are better. Three types of CIRC are compared. CIRC-CONV is the conventional circular IQ, which is simply called CIRC in the other sections. CIRC-PPRI is the circular queue with perfect priority, which assigns the correct priority even when wrap-around occurs. Finally, CIRC-PC is the circular queue that we proposed in Section 3.1, which is used in SWQUE.

As shown in the figure (also evaluated in Section 4.2), CIRC-CONV degrades performance in both INT and FP programs because

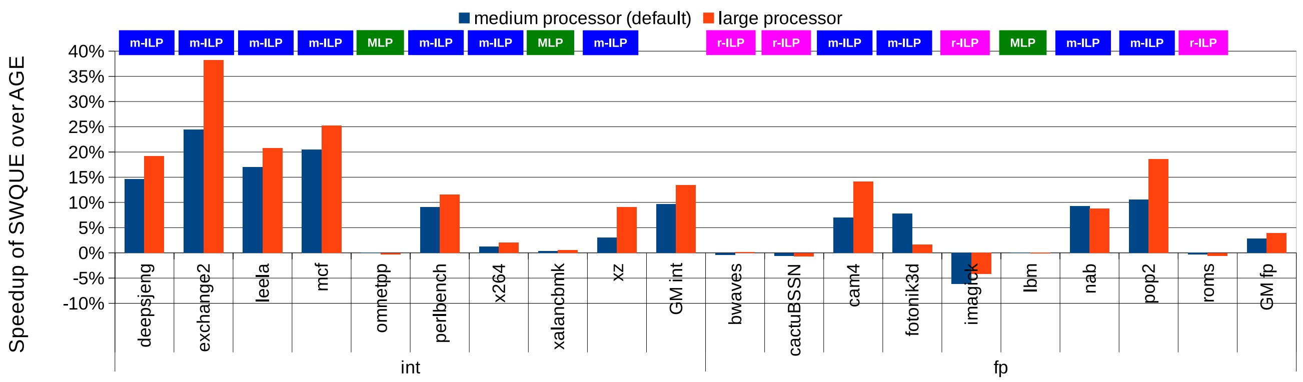


Figure 9: Speedup over AGE for medium- (default) and large-sized processors. "m-ILP," "r-ILP," and "MLP" represent moderate ILP, rich ILP, and MLP programs, respectively.

Table 4: Parameters of medium/large-sized processor models.

Parameter	Processor model	
	Medium	Large
Fetch/decode/Issue/commit width	6	8
IQ size	128	256
Load/store queue size	128	256
Reorder buffer size	256	512
Physical regs (int+fp)	256+256	512+512
Number of iALUs	3	4
Number of FPUs	2	3

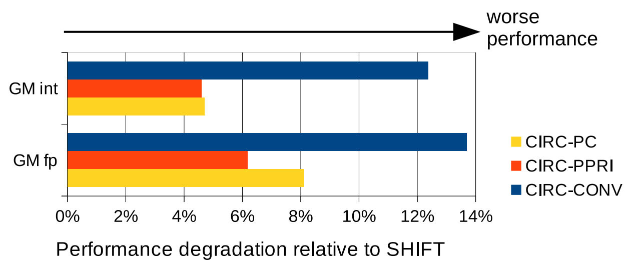


Figure 11: Performance degradation relative to SHIFT for various CIRC configurations.

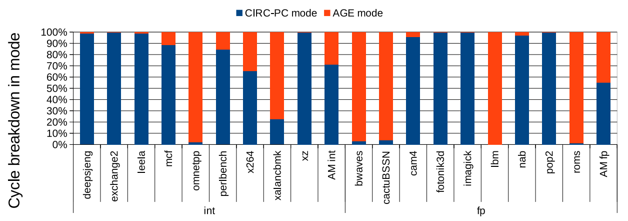


Figure 10: Breakdown of the execution cycles in terms of modes.

of the incorrect priority that it assigns when wrap-around occurs and capacity inefficiency. By contrast, CIRC-PPRI improves the performance because the correct priority is perfectly assigned.

For CIRC-PC, although the RV instructions require two cycles to be issued, the performance is almost identical to that of CIRC-PPRI in the INT programs, and it is only slightly worse in the FP programs. This means that the RV instructions are latency-tolerant, and the issue delay thus hardly affects performance adversely.

4.5 Energy Consumption

In this section, we evaluate the energy consumption of a core using McPAT [19], assuming 22nm LSI technology (22nm technology is the finest that McPAT supports), and a temperature of 320K. We compare SWQUE with the idealized shifting queue (I-SHIFT) where the increases of energy consumption and delay caused by compaction are not considered, which means that it is the simplest circuit and the execution time is shortest (a longer execution time equates to more static energy being consumed) among the IQs we evaluated in this paper. We could not compare AGE because McPAT does not support the age matrix. For the same reason, the energy consumption evaluation for SWQUE does not include the energy consumed by the age matrix.

Figure 12 shows the energy consumption relative to that of I-SHIFT. The bar is divided into four parts: static and dynamic energies for the basic operation of the IQ and SWQUE-specific operation (the extra select logic operation and twice as many tag RAM accesses). As the figure shows, the energy consumed by the SWQUE-specific operation is very low (the static energy is too low to be visible in the figure). Consequently, the energy consumption of SWQUE is almost the same (only 0.5% increase) compared with that of the I-SHIFT.

As described previously, the energy consumption of the age matrix is not included in this evaluation. However, it is smaller than that of the select logic, and thus, if included, it would be added as a small constant value to both I-SHIFT and SWQUE, and so the graph would hardly be changed.

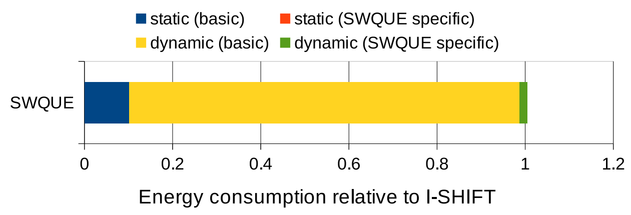


Figure 12: Energy consumption relative to I-SHIFT.

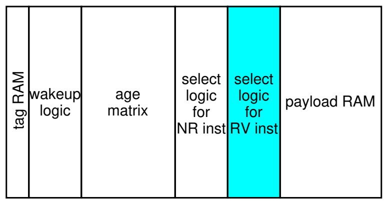


Figure 13: Relative size of each circuit in SWQUE. The size of each circuit is scaled in proportion to the actual size.

Table 5: Transistor density comparison.

Design	Circuit	Tr. density ($\times 10^{-3}/\lambda^2$)
Author	tag RAM	1.399
	wakeup logic	1.586
	select logic	0.740
	age matrix	1.708
Sun Micro[25]	512KB L2 cache	3.957
Fujitsu[14]	54-bit multiplier for FP	0.726
Intel	processor (Skylake)	0.701

Table 6: Additional costs and cost-neutral performance comparison.

additional cost	value	0.0029mm^2
	vs. Skylake core	0.034%
	vs. Skylake chip	0.010%
perf improvement over baseline AGE	SWQUE (128 entries)	9.8%(INT), 3.7%(FP)
	AGE (150 entries)	-0.6%(INT), -0.1%(FP)

4.6 Area Overhead

The SWQUE area overhead is the additional select logic. In this section, we estimate the impact of this area overhead.

McPAT calculates the area of the various structures in a core of a processor, including the IQ, but it neglects the area of the select logic[3]. Additionally, it does not assume that the IQ has an age matrix, as described in Section 4.5. Therefore, we manually drew the LSI layout of the IQ, assuming MOSIS design rules [2], and compared the area with that of the Intel Skylake fabricated using 14nm LSI technology.

Before providing the results, we compare the transistor density of our design with that of several examples in Table 5 to illustrate that our layout design is reasonable. As the table shows, the transistor density of any circuit composed of an IQ is sparser than the single-port 512KB L2 cache (one of the densest structures in processors), but denser than the 54-bit multiplier (dense logic array). Additionally, when compared with the entire Intel Skylake processor chip, the transistor density of the circuits in our design is comparable or denser. These results indicate that our layout design is reasonable.

Our evaluation indicates that the area overhead is 17% compared with the baseline IQ area. Figure 13 illustrates the relative sizes of each circuit in SWQUE.

Table 6 lists the cost impact on a processor along with the cost-neutral performance comparison results. The first three rows of the table list the additional area (assuming 14nm LSI technology) and its ratio relative to that of the Intel Skylake core and entire chip. As listed in the table, the additional costs and their impact are very small. The two remaining rows in Table 6 list the cost-neutral performance comparison results. We evaluated the performance of AGE, which has a 17% increase in entries (150 entries) when using the additional area for SWQUE. As listed, simply increasing the small number of entries does not enhance the performance (it even slightly reduces the performance by increasing issue conflicts).

[3]McPAT calculates the energy consumption of the select logic.

4.7 Delay Issues

The SWQUE delay is increased by the DTM. As shown in Figure 6, the broken orange line represents the critical path of the DTM. However, the delay of this critical path is not completely exposed to the IQ delay because valid signals (V) in the DTM travel parallel to tags ($dtag$) from the left-hand side of the tag RAM to the wakeup logic along the top of the tag RAM. These tags must even travel in the IQ with the basic organization, without the DTM. Therefore, the increased net delay produced by the DTM is caused by the increase in the load capacitance provided by the gate capacitance of the transistors for MUX control. We compared the IQ delay with and without the DTM via HSPICE simulation, and found that the additional delay caused by the DTM is only 1.3% of the entire IQ delay.

Another concern is whether the double accesses to the tag RAM are accommodated within a single cycle. We evaluated this using HSPICE simulations and found that the total delay of the double tag RAM accesses, including precharge time is 66% of the entire IQ delay, and thus there is a large margin. This short access time is related to the small size of the tag RAM as shown in Figure 13.

4.8 Switch Penalty Sensitivity

The default penalty at the transition between AGE and CIRC-PC is 10 cycles, as listed in Table 3. We evaluated the sensitivity of this penalty by changing it to 40 cycles. SWQUE performance degradation relative to the case in which the default penalty was used was only 0.02%. This insensitivity arises because the transition occurs infrequently at 8 times per 1M clock cycles, on average.

4.9 Using Multiple Age Matrices

One approach to enhance the AGE scheme is to prepare multiple age matrices, and thus divide and conquer the priority problem using these matrices. This approach *logically* prepares multiple *instruction buckets*, where each bucket uses a single age matrix. Instructions are steered to one of these buckets at dispatch (i.e., write) to the IQ, and the single oldest ready instruction is selected from the instructions in each bucket. It is reasonable to prepare these buckets based on function units. Note that the buckets do not *physically* exist, and thus the IQ is thus not split. Because the IQ remains monolithic, the capacity efficiency also remains high. This approach can increase the IPC by having multiple age matrices select multiple high-priority instructions, although they are selected from each bucket (not from the entire IQ). The downside of this approach is simply the need for multiple age matrices, which increases the size of the IQ. Because the age matrix is a large circuit (see Figure 13) compared with the other circuits in the IQ, the increase in the IQ area is significant. This increases the IQ delay by requiring global wires (for request and grant signals) to traverse the IQ, and can have a significant adverse impact on the clock cycle time. Therefore, the IQ size must be reduced to compensate for the delay increase in practice so that the IQ delay does not increase. In this section, we evaluate the IPC as a performance; we do not consider the increased delay and do not reduce the IQ size.

In the evaluation, the enhanced AGE scheme uses seven and nine age matrices for the medium- and large-sized processors, respectively. The buckets are prepared based on function units; that is, the IQ of the medium-sized processor uses three, two, and two buckets for INT, memory, and FP instructions, respectively. The buckets are prepared in a similar manner for the large-sized processor. A load-balancing algorithm is used to steer instructions to the buckets. As with previous evaluations in this paper, the priorities of all ready instructions are essentially assigned based on the IQ position, but instructions selected by the age matrices are assigned the highest priority independently of their IQ position.

Figure 14 shows the evaluation results. The Y-axis represents the average speedup over the baseline AGE with a single age matrix for the medium- and large-sized processors. The blue (left), red (middle), and yellow (right) bars represent the speedups of SWQUE with a single age matrix (SWQUE-1AM) (the default evaluated in Section 4.2), AGE with multiple age matrices (AGE-multiAM), and SWQUE with multiple age matrices (SWQUE-multiAM), respectively.

Although, as shown in the figure, AGE-multiAM increases the performance by 1.4%, the performance gap versus that of SWQUE is still very large in the INT programs. This is because SWQUE strictly selects issue instructions in priority order from all the ready instructions in the IQ, whereas AGE-multiAM is constrained in the selection of high priority instructions because of the buckets; that is, instructions with the second or lower priority in a bucket are still placed in the random selection, even if they should have been selected when all the ready instructions were considered.

Note that the source of the speedup in SWQUE is the CIRC-PC scheme in the INT programs, as described in Section 4.2, and thus having multiple age matrices only affects the performance slightly. By contrast, the SWQUE speedup is at a level similar to that for

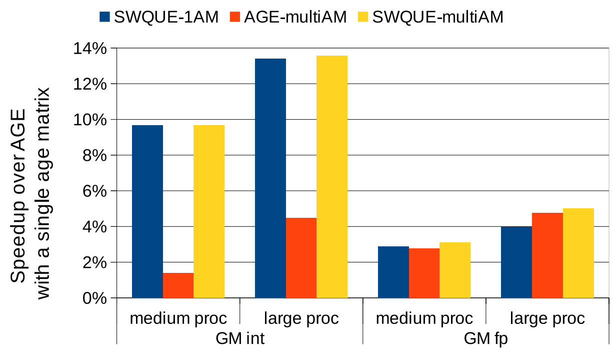


Figure 14: Performance comparison when using multiple age matrices.

AGE-multiAM in the FP programs because SWQUE is configured as an AGE in many programs in the FP programs, as described in Section 4.2.

5 RELATED WORK

The IQ was extensively studied around 2000, and a comprehensive survey was performed by Abella et al [4].

One processor that implemented SHIFT was the DEC Alpha 21264 [9]. Although this scheme outperforms the other schemes we evaluated in this paper in terms of IPC, the shifting and compaction of the instructions needed to keep the order result in more complex circuits, increasing the delay and consuming large amounts of power, as described in Section 2.3. SHIFT is no longer used in modern processors.

Butler et al. investigated the effect of several select policies of the IQ, including random selection and selection based on the number of dependent instructions [8]. Their evaluation results showed that the select policies they investigated delivered almost the same performance for the INT programs (SPEC89 benchmark) but the performance differed by up to 20% for the FP programs. They noted that similar performance arose from the number of ready instructions in a clock cycle being heavily skewed to zero. By contrast, our results (SHIFT vs. RAND) differ from their results, particularly for the INT programs. These differences have arisen because Butler et al. assumed full function units capable of integer execution, which caused no issue conflict on function units. We also confirmed that the number of ready instructions is skewed to zero, but there is still a significant number of clock cycles where the number of ready instructions is greater than zero. These statistics strongly depend on the program.

Although the age-aware select policy implements the correct priority, it is only a heuristic policy. Ideally, for high performance, the instructions on a critical path of the dataflow should be selected with high priority. Fields et al. proposed a scheme to identify the criticality of an instruction [10]. However, the scheme is difficult to implement because of its high complexity and large area.

Stark et al. proposed breaking the wakeup–select loop into different pipeline stages [27]. Brown et al. proposed issuing instructions without selection [7], where the select operation is eliminated from the critical path. These schemes are similar to our CIRC-PC in terms of pipelining the issue operation. However, these schemes pipeline the issue operation for all instructions to reduce the clock cycle time, whereas our scheme only pipelines the issue operation for latency-tolerant instructions. Additionally, these schemes are speculative; thus, misspeculation occurs frequently. This significant increases power consumption.

Brekelbaum et al. proposed a *hierarchical scheduling window* that divides the IQ into a large slow queue and a small fast queue [6]. The larger queue requires multiple cycles for scheduling whereas the small queue requires only a single cycle. The large queue is CIRC and this scheme searches for several non-ready instructions from its head, which are moved to the small queue. The critical instructions are issued from the fast small queue, whereas latency-tolerant instructions are issued from the slow large queue. This scheme is similar to ours because the instruction operations are segregated in terms of latency tolerance; however, moving the oldest non-ready instructions complicates the IQ, increases the delay, and consumes large amounts of power.

Henry et al. addressed the problem caused by wrap-around (i.e., incorrect issue priority) in CIRC at the circuit level [15]. However, in their proposed circuits, the wires traverse vertically within the IQ to transfer information from the top to the bottom, and thus they increase the delay.

The delay of the conventional age matrix increases the IQ delay. Because the wire delay is dominant in the delay of this circuit, the reduction of the number of cells that the wires must traverse is effective to reduce the delay. It is also helpful to reduce wire delay traversing on this circuit. Sassone et al. proposed a scheme that dynamically allocates transposed issue request lines for a group of instructions to reduce the age matrix width [24]. The downside of this scheme is that it still requires an arbiter to arbitrate the requests among the instructions in a group. Because the instructions are distributed within the IQ even for a single group (because the queue is a random queue), the wires for the arbiter traverse the IQ vertically. The arbiter delay is thus not trivial, and the effectiveness is consequently reduced.

Kora et al. proposed configuring the IQ to exploit ILP and MLP [17]. To exploit ILP, the IQ is reduced, whereas to exploit MLP, the IQ is increased and pipelined. This scheme is similar to ours in terms of configuring the IQ depending on which parallelism the IQ attempts to exploit. However, that study did not address the problem of incorrect priority.

Sakai et al. proposed an IQ scheme that assigns high priority to multiple oldest instructions [23] (not the single oldest instruction as in the age matrix). The scheme divides the IQ into a large *main queue* and small *old queue*, and the select logic is shared among both queues and assigns a higher priority to the instructions in the old queue than those in the main queue. The scheme moves multiple oldest instructions in the main queue to the old queue each cycle, and consequently succeeds in assigning higher priority to multiple oldest instructions.

Ando proposed an IQ scheme that reduces the branch misprediction penalty by issuing instructions in an *unconfident branch slice* as early as possible [5]. Unconfident branch slices are defined as instructions that a branch directly or indirectly depends on and its branch prediction is unconfident. The scheme reserves several priority entries in the IQ for the instructions in the unconfident branch slices. During execution, the scheme constructs a pointer table that links instructions to the associated branch if there is a dependence. The confidence of branch prediction is estimated, and then whether instructions belong to an unconfident branch slice is determined by looking up the constructed pointer table. If an instruction is determined to belong to an unconfident branch slice, then it is dispatched into one of the reserved priority entries, and consequently issued with the highest priority.

6 CONCLUSIONS

The stagnation of single-thread performance improvement is currently very serious. Following the end of Dennard scaling, architectural improvements are now more necessary than ever. In particular, improvements in the IQ are essential, and correct issue priority and high capacity efficiency are both vital for high performance.

In this paper, we proposed a new IQ called SWQUE that dynamically configures the IQ as a modified CIRC (CIRC-PC) or AGE based on the degree of capacity demand. The IQ is configured as CIRC-PC when the correct priority is more important than the capacity efficiency; otherwise, it is configured as AGE. CIRC-PC is effective for phases with moderate amounts of ILP, whereas AGE is effective for phases with large amounts of ILP or memory-intensive phases.

CIRC-PC, which we proposed in this paper, corrects the incorrect priority caused by wrap-around with a simple circuit by exploiting the finding that the instructions that are wrapped around are latency-tolerant.

Using CIRC-PC, SWQUE achieves higher performance than the IQ with the age matrix (AGE) that is currently used by 9.7% and 2.9% (up to 24.4% and 10.6%) in INT and FP programs, respectively.

ACKNOWLEDGMENTS

The author thanks the anonymous reviewers for their useful comments. This work was supported by the VLSI Design and Education Center (VDEC), the University of Tokyo in collaboration with Synopsys Inc.

REFERENCES

[1] http://www.simplescalar.com/.
[2] http://www.mosis.com/.
[3] http://ptm.asu.edu/.
[4] J. Abella, R. Canal, and A. Gonzalez. 2003. Power- and Complexity-Aware Issue Queue Designs. *IEEE Micro* 23, Issue 5, 5 (September-October 2003).
[5] H. Ando. 2018. Performance Improvement by Prioritizing the Issue of the Instructions in Unconfident Branch Slices. In *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture.* 82–94.
[6] E. Brekelbaum, J. Rupley, C. Wilkerson, and B. Black. 2002. Hierarchical Scheduling Windows. In *Proceedings of the 35th Annual IEEE/ACM International Symposium on Microarchitecture.* 27–36.
[7] M. D. Brown, J. Stark, and Y. N. Patt. 2001. Select-Free Instruction Scheduling Logic. In *Proceedings of the 34th Annual IEEE/ACM International Symposium on Microarchitecture.* 204–213.
[8] M. Butler and Y. Patt. 1992. An Investigation of the Performance of Various Dynamic Scheduling Techniques. In *Proceedings of the 25th Annual IEEE/ACM International Symposium on Microarchitecture.* 1–9.
[9] J. A. Farrell and T. C. Fischer. 1998. Issue Logic for a 600-MHz Out-of-Order Execution Microprocessor. *Journal of Solid-State Circuits* 33, 5 (May 1998), 707–712.
[10] B. Fields, S. Rubin, and R. Bodík. 2001. Focusing Processor Policies via Critical-Path Prediction. In *Proceedings of the 28th Annual International Symposium on Computer Architecture.* 74–85.
[11] M. Golden, S. Arekapudi, and J. Vinh. 2011. 40-Entry Unified Out-of-Order Scheduler and Integer Execution Unit for the AMD Bulldozer x86-64 Core. In

2011 IEEE International Solid-State Circuits Conference, Digest of Technical Papers. 80–82.

[12] M. Goshima. 2004. *Research on High-Speed Instruction Scheduling Logic for Out-of-Order ILP Processor.* Ph.D. Dissertation. Kyoto University.

[13] M. Goshima, K. Nishino, T. Kitamura, Y. Nakashima, S. Tomita, and S. Mori. 2001. A High-Speed Dynamic Instruction Scheduling Scheme for Superscalar Processors. In *Proceedings of the 34th Annual IEEE/ACM International Symposium on Microarchitecture.* 225–236.

[14] G. Goto, A. Inoue, R. Ohe, S. Kashiwakura, S. Mitarai, T. Tsuru, and T. Izawa. 1997. A 4.1-ns Compact 54 × 54-b Multiplier Utilizing Sign-Select Booth Encoders. *IEEE Journal of Solid-State Circuits* 32, 11 (December 1997), 1676–1682.

[15] D. S. Henry, B. C. Kuszmaul, G. H. Loh, and R. Sami. 2000. Circuits for Wide-Window Superscalar Processors. In *Proceedings of the 27th Annual International Symposium on Computer Architecture.* 236–247.

[16] International Technology Roadmap for Semiconductors (http://www.itrs2.net/).

[17] Y. Kora, K. Yamaguchi, and H. Ando. 2013. MLP-Aware Dynamic Instruction Window Resizing for Adaptively Exploiting Both ILP and MLP. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture.* 37–48.

[18] R. Kumar and G. Hinton. 2009. A Family of 45nm IA Processors. In *2009 IEEE International Solid-State Circuits Conference, Digest of Technical Papers.* 58–59.

[19] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. 2009. McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture.* 469–480.

[20] S. Palacharla, N. P. Jouppi, and J. E. Smith. 1996. *Quantifying the Complexity of Superscalar Processors.* Technical Report CS-TR-1996-1328. University of Wisconsin-Madison.

[21] S. Palacharla, N. P. Jouppi, and J. E. Smith. 1997. Complexity-Effective Superscalar Processors. In *Proceedings of the 24th Annual International Symposium on Computer Architecture.* 206–218.

[22] R. P. Preston, R. W. Badeau, D. W. Bailey, S. L. Bell, L. L. Biro, W. J. Bowhill, D. E. Dever, S. Felix, R. Gammack, V. Germini, M. K. Gowan, P. Gronowski, D. B. Jackson, S. Mehta, S. V. Morton, J. D. Pickholtz, M. H. Reilly, and M. J. Smith. 2002. Design of an 8-wide Superscalar RISC Microprocessor with Simultaneous Multithreading. In *2002 IEEE International Solid-State Circuits Conference, Digest of Technical Papers.* 334–472.

[23] S. Sakai, T. Suenaga, R. Shioya, and H. Ando. 2018. Rearranging Random Issue Queue with High IPC and Short Delay. In *Proceedings of the 36th IEEE International Conference on Computer Design.* 123–131.

[24] P. G. Sassone, J. Rupley II, E. Brekelbaum, G. H. Loh, and B. Black. 2007. Matrix Scheduler Reloaded. In *Proceedings of the 34th Annual International Symposium on Computer Architecture.* 335–346.

[25] J. L. Shin, B. Petrick, M. Singh, and A. S. Leon. 2005. Design and Implementation of an Embedded 512-KB Level-2 Cache Subsystem. *IEEE Journal of Solid-State Circuits* 40, 9 (September 2005), 1815–1820.

[26] B. Sinharoy, J. A. Van Norstrand, R. J. Eickemeyer, H. Q. Le, J. Leenstra, D. Q. Nguyen, B. Konigsburg, K. Ward, M. D. Brown, J. E. Moreira, D. Levitan, S. Tung, D. Hrusecky, J. W. Bishop, M. Gschwind, M. Boersma, M. Kroener, M. Kaltenbacha, T. Karkhanis, and K. M. Fernsler. 2015. IBM POWER8 Processor Core Microarchitecture. *IBM Journal of Research and Development* 59, issue 1 (January - February 2015), 2:1 – 2:21.

[27] J. Stark, M. D. Brown, and Y. N. Patt. 2000. On Pipelining Dynamic Instruction Scheduling Logic. In *Proceedings of the 33rd Annual IEEE/ACM International Symposium on Microarchitecture.* 57–66.

[28] H. Sutter. 2005. The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software. *Dr. Dobb's Journal* 30, 3 (2005), 202–210.

[29] N. H. E. Weste and D. M. Harris. 2011. *CMOS VLSI Design: A Circuits and Systems Perspective, fourth edition.* Addition Wesley.

[30] K. Yamaguchi, Y. Kora, and H. Ando. 2011. Evaluation of Issue Queue Delay: Banking Tag RAM and Identifying Correct Critical Path. In *Proceedings of the 29th International Conference on Computer Design.* 313–319.