Geneva: A Dynamic Confluence of Speculative Execution and In-Order Commitment Windows

Yanghee Lee
Yonsei University
Seoul, Korea
yanghee.lee@yonsei.ac.kr

Jiwon Lee
Yonsei University
Seoul, Korea
jiwon.lee@yonsei.ac.kr

Jaewon Kwon
Yonsei University
Seoul, Korea
jaewon.kwon@yonsei.ac.kr

Yongju Lee
Yonsei University
Seoul, Korea
yongju.lee@yonsei.ac.kr

Won Woo Ro
Yonsei University
Seoul, Korea
wro@yonsei.ac.kr

ABSTRACT

Modern out-of-order microprocessors are increasingly expanding resources such as reorder buffer (ROB) and instruction queue (IQ) for memory-level parallelism (MLP). While this expansion effectively addresses the memory wall challenge, it also incurs notable cost and energy trade-offs. To tackle this, we propose *Geneva*, a microarchitecture that improves performance and saves energy. Geneva reallocates a portion of an ROB to serve as a dynamic queue (DQ), used as an ROB, IQ, or both depending on operational needs. Geneva saves energy by 15.6% and improves performance by 2.6% compared to the conventional out-of-order core.

CCS CONCEPTS

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

KEYWORDS

Dynamic Scheduling, Speculative Execution, In-Order Commitment

ACM Reference Format:
Yanghee Lee, Jiwon Lee, Jaewon Kwon, Yongju Lee, and Won Woo Ro. 2024. Geneva: A Dynamic Confluence of Speculative Execution and In-Order Commitment Windows. In *61st ACM/IEEE Design Automation Conference (DAC '24), June 23–27, 2024, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3649329.3655924

1 INTRODUCTION

Contemporary processors exploit out-of-order execution techniques to push the boundaries of instruction-level parallelism (ILP). As the complexity and diversity of modern applications grow, the memory wall challenge caused by the performance gap between the CPU and memory exacerbates. Such dynamics saturate the ILP extraction of out-of-order machines. A simple method to address this challenge is to exploit memory-level parallelism (MLP) through resource expansion, specifically in the context of reorder buffers (ROBs) and instruction queues (IQs). However, expanding these resources often leads to higher costs and energy consumption. In particular, extending an out-of-order IQ requires content-addressable memory (CAM) logic that searches the entire IQ entries to generate optimized instruction scheduling. Such an approach can enhance performance while significantly increasing energy consumption. Therefore, it is paramount to design out-of-order cores that achieve both performance improvement and energy savings without expanding hardware resources.

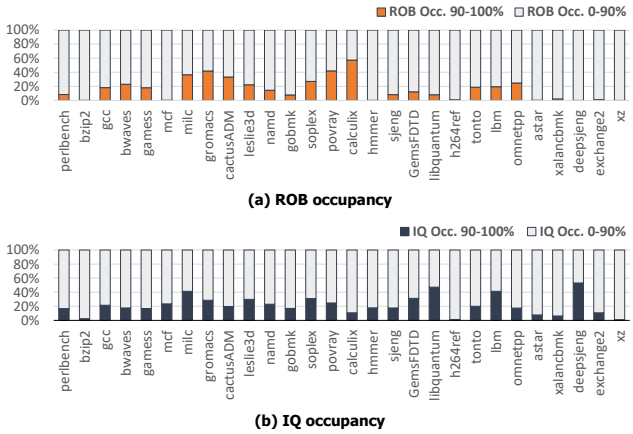

Figure 1: Distribution of (a) ROB occupancy when IQ occupancy exceeds 90% and (b) IQ occupancy when ROB occupancy exceeds 90%.

Unfortunately, prior approaches necessitate alternative components to optimize resource utilization, thereby improving performance or energy efficiency in their core designs. Previous works on out-of-order cores [1, 12] suggest saving energy with acceptable performance degradation by employing simple in-order queues as second-tiers for their out-of-order IQs. These methods classify instructions by criticality and readiness. They then steer classified instructions, especially non-critical ones, to second-tier in-order queues, alleviating pressure on the out-of-order IQ. However, they require at least a 128-entry second-tier in-order queue to realize the benefits of reducing the size of the out-of-order IQs.

In this work, we aim to improve both the performance and the energy efficiency of out-of-order cores without additional resources.

DAC '24, June 23–27, 2024, San Francisco, CA, USA

ACM ISBN 979-8-4007-0601-1/24/06
https://doi.org/10.1145/3649329.3655924

To this end, we propose *Geneva*[1], a microarchitecture that dynamically balances resource allocation between an ROB and an IQ instead of adding an in-order queue. Geneva is devised based on two key observations. First, we observe that out-of-order cores suffer from the imbalance between ROBs and IQs. Figure 1 plots the portion of cycles when ROB or IQ occupancy is higher than 90% and the utilization ratio of its counterpart also exceeds 90% at the same time. The results show that the ROB and IQ are highly utilized only for 3.9% (Figure 1a) and 14.8% (Figure 1b) of cycles when the other component is busy. This implies that underutilized resources can be repurposed to extend the size of other hardware components, and the ROB is a better candidate as it exhibits a lower utilization rate than the IQ. Second, we find that simply reducing the size of a specific component cannot mitigate the resource imbalance problem. Figure 2a compares the resource utilization rates when the baseline out-of-order cores have their ROB or IQ entries reduced by 32. The baseline employs a 224-entry ROB and 96-entry IQ. The results show that such an approach does not decrease the gap between the resource utilization of the ROB and IQ, and rather the imbalance problem is aggravated. Therefore, it is necessary to design an out-of-order core that dynamically balances its hardware resources.

To address the resource imbalance problem of out-of-order cores, Geneva dynamically balances the utilization rates of an ROB and an IQ. Geneva repurposes a portion of an ROB as a dynamic queue (DQ) and dynamically allocates DQ entries according to the occupancy of its ROB and IQ. We design the DQ as a set of multiple dynamic in-order queues (D-IQs). Each D-IQ consists of four entries because more than 85% of chains of dependent instructions have a length equal to or less than four (Figure 2b). Initially, D-IQs are idle, and Geneva monitors the occupancy rate of the ROB and IQ. When Geneva detects that the occupancy of the ROB, IQ, or both exceeds certain thresholds, it allocates available D-IQs to the resources that exceed the thresholds. The enhanced hardware utilization in Geneva allows for improved performance and energy efficiency, despite the reduction in the size of an out-of-order IQ.

The contributions of this work are as follows:

- We show that conventional out-of-order cores suffer from the imbalance between an ROB and an IQ. In addition, we find that simply reducing the size of either component intensifies the problem.
- We propose a Geneva microarchitecture that balances the utilization rate of an ROB and IQ using a DQ. A DQ enables the *dynamic confluence* of an ROB and IQ and thereby enhances performance and reduces energy consumption.
- We quantitatively analyze the proposed Geneva microarchitecture. Our analysis shows that Geneva saves 15.6% of energy and increases instruction per cycle (IPC) by 2.6% over the baseline out-of-order cores, even with a reduction of 32 entries in the out-of-order IQ.

[1]Geneva is renowned for being the confluence of the Rhône and Arve rivers. In the same context, we use Geneva as the name of our proposal, which is a dynamic confluence of speculative execution and in-order commitment windows.

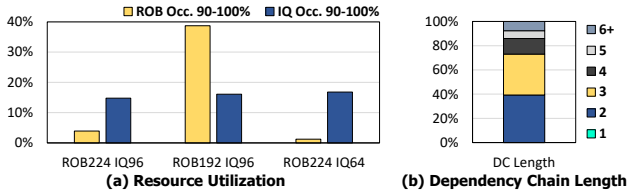

Figure 2: (a) Comparison of resource utilization between the baseline and the baseline with 32 fewer ROB or IQ entries. (b) Percentage distribution of dependency chain length.

2 BACKGROUND AND MOTIVATION

2.1 Speculative Execution

Modern out-of-order processor designs improve performance by extracting ILP and MLP through *speculative execution*. To realize speculative execution, processors employ both an out-of-order IQ to optimize instruction scheduling, and an ROB that maintains program semantics by committing instructions in order.

Dynamic Scheduler (Out-of-Order IQ): An out-of-order IQ operates as a dynamic scheduler, allowing instructions to be executed speculatively based on operand readiness rather than their program order. In conventional out-of-order cores, each IQ entry comprises a busy bit, opcode, ready bits, and physical register (P-Reg) IDs or values for both destination and source operands.

In-Order Commitment Window (ROB): All instructions must be committed in program order within an instruction pipeline. Also, in scenarios where speculative execution leads to incorrect speculation or exceptions arise (*e.g.*, division by zero, segmentation faults), the unexpected event must be correctly recovered and handled. To satisfy these requirements, an ROB functions as an in-order commitment window that manages instructions in program order and maintains all data for instructions, including recovery data. Typically, an ROB entry contains a busy bit, status flags, P-Regs, and recovery data, which includes the previous physical registers (PP-Regs).

2.2 Instruction Dependency Chain

A dependency chain (DC) of instructions denotes the correlation arising from dependencies wherein a destination register of a certain instruction is used as a source register for other ones. The former and latter are the producer and consumer instructions. The consumer instruction can also be a producer instruction for other ones. In other words, a DC can exhibit successive producer-consumer relationships. In this paper, we refer to the first instruction of each DC as the *dependency chain head instruction (DC-Head)*.

2.3 Exploiting Second-Tier In-Order Queues

Second-Tier In-Order Queues: Prior approaches employ simple FIFO queues as second-tier in-order queues to improve either performance or energy efficiency of core designs. Studies targeting in-order cores adopt second-tier in-order queues to construct multiple in-order IQs, and such IQs operate as a restricted form of an out-of-order IQ [3, 5, 7, 8, 11]. On the other hand, in the realm of out-of-order cores, instructions classified as less critical are retained and processed in second-tier in-order queues. This alleviates the pressure on the out-of-order IQ and consequently reduces energy consumption [1, 12].

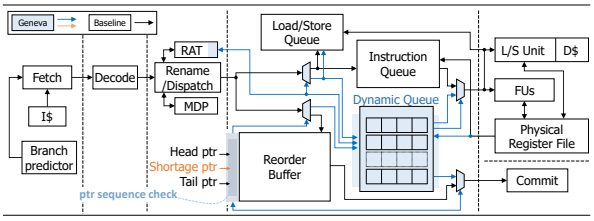

Figure 3: Geneva microarchitecture overview

Instruction Classification: To effectively exploit second-tier in-order queues, which are fundamentally simple FIFO queues, the instructions stored and processed in these queues should consist only of those that do not adversely impact the desired performance or energy consumption of cores. For this purpose, previous studies implement instruction classification schemes to maximize the utilization of second-tier in-order queues [1, 3, 5, 7, 8, 11, 12]. Methods for instruction categorization include identifying instruction slices related to missed load instructions either backward [3] or forward [8] and classifying based on the producer-consumer relationship among instructions [5, 11].

Exploiting Second-Tier In-Order Queues: Prior schemes that adopt an instruction slice technique in in-order or out-of-order cores categorize instructions into critical, non-critical, ready, and non-ready groups. Methods for in-order cores aim to improve performance [3, 7, 8], while those associated with out-of-order cores strive to reduce energy consumption with acceptable performance degradation [1, 12]. However, since the instruction slice approaches mainly target instructions related to missed load instructions to extract MLP, they can degrade the performance of MLP-insensitive applications [12]. Moreover, these approaches require employing one or more in-order queues to achieve desired performance or energy savings. For instance, Long Term Parking (LTP) [12] requires the addition of a 128-entry in-order queue, and Delay and Bypass (DNB) [1] necessitates 128-entry and 32-entry FIFO queues.

On the contrary, in-order core-based designs, such as Complexity-Effective Superscalar (CES) [11] and Ballerino [5], employ different strategies. They classify instructions by tracking DCs of instructions and utilize parallel in-order queues (P-IQs) to independently schedule distinct instruction DCs. Such an approach allows their IQs to behave more closely to an out-of-order IQ than those in the previous studies [3, 7, 8]. Although they also have the drawback of requiring resource expansion, the uniform role of the second-tier in-order queues provides scalability. Inspired by this, we design the D-IQs in Geneva based on P-IQs. Note that we construct the D-IQs by repurposing part of an ROB, thereby avoiding the need for employing additional resources, unlike CES and Ballerino.

3 GENEVA MICROARCHITECTURE

In this section, we introduce *Geneva*, a novel out-of-order core that harnesses the *dynamic confluence* of both speculative execution and in-order commitment windows. Compared to conventional out-of-order cores, the proposed Geneva microarchitecture achieves higher IPC performance and improves energy efficiency. More importantly, Geneva effectively addresses the resource expansion demands, which are the common drawbacks in previous studies. The primary features of the Geneva microarchitecture are as follows:

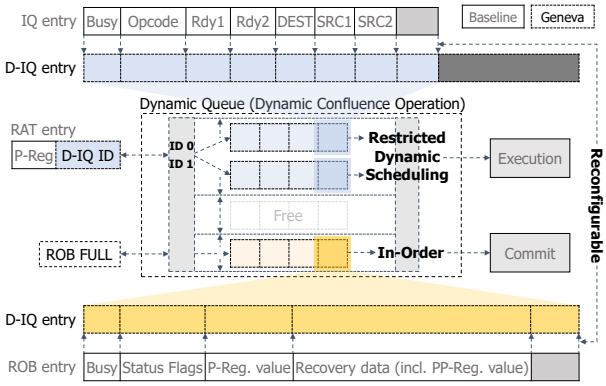

Figure 4: Dynamic confluence operation

- **Dynamic Queue Allocation:** Geneva reduces the size of an ROB and reallocates the saved portion to a *set of general-purpose in-order queues (i.e., DQ).*
- **Dynamic Confluence:** The DQ functions as an in-order commitment window (*i.e., dynamic queue as reorder buffer (DQ-ROB)*) or as a restricted dynamic scheduler (*i.e., dynamic queue as instruction queue (DQ-IQ)*). We refer to the structure in which the DQ operates, either the DQ-ROB, the DQ-IQ, or both, as dynamic confluence.

3.1 Overview

Figure 3 illustrates the overview of the proposed Geneva microarchitecture. In the figure, the components colored in gray and black lines represent those in conventional out-of-order cores, while the parts marked in blue and orange indicate the hardware components extended in Geneva. The Geneva microarchitecture exhibits three primary distinctions from conventional out-of-order core designs, summarized as follows:

(1) A *shortage pointer* is added as one of the ROB pointers to ensure that operation conditions for the DQ-ROB, such as the ROB being full, are met and the in-order commitment rule is maintained.
(2) In register aliasing table (RAT) entries, a *D-IQ ID* is recorded for three purposes: to verify the DC of dispatched instructions, to serve as a prerequisite for a specific instruction to be inserted into a DQ-IQ, and to index DQ-IQ entries.
(3) For the dynamic confluence, Geneva employs a DQ and *steering logic*. The DQ consists of three partitions (*i.e.,* DQ-ROB, DQ-IQ, and free), and the steering logic ensures instructions are directed appropriately to their respective DQ partitions.

3.2 Dynamic Confluence Partition Policy

Geneva realizes the dynamic confluence by partitioning a DQ. Specifically, the DQ is split and assigned into a DQ-ROB, a DQ-IQ, and a free partition. When the occupancy of an ROB or an IQ exceeds a certain threshold, the dynamic confluence partition policy of Geneva allocates D-IQs, which is a subset of a DQ, to the DQ-ROB partition or the DQ-IQ partition, respectively. When a D-IQ in the DQ-ROB or DQ-IQ partition is not in use and the ROB or IQ occupancy is below the threshold, the queue is reassigned to the free partition to conserve energy.

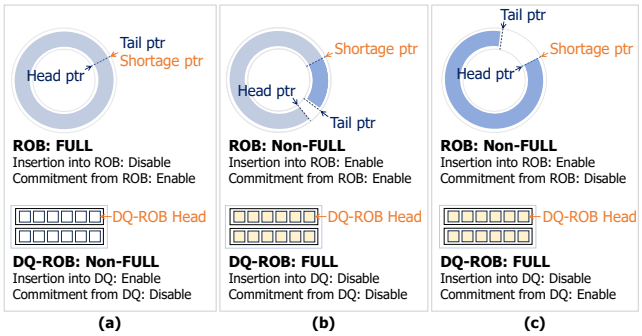

Figure 5: Example of DQ-ROB operation

The size of each partition varies adaptively based on the occupancy rate of the ROB or the IQ, following the policies described below.

(1) Initially, all DQ entries belong to the free partition.
(2) When the occupancy of an ROB exceeds 90%, one D-IQ from the free partition is allocated to the DQ-ROB partition and held in reserve. This *proactive reservation* prevents unnecessary pipeline stalls that occur when an ROB is full.
(3) If IQ occupancy is above 50%, one D-IQ from the free partition is *assigned to the DQ-IQ partition*. For efficient utilization of D-IQs, there is a need to ensure sufficient DC length. Thus the allocation of DQ-IQs happens more sensitively than that of the DQ-ROB.
(4) If the occupancy conditions are not met, further allocation of D-IQs to the DQ-ROB or DQ-IQ is prevented, and already allocated D-IQs *are returned to the free partition* if they become vacant.

3.3 Dynamic In-Order Queue (D-IQ)

Inspired by a P-IQ [11], we design D-IQs to realize the dynamic confluence for balanced resource utilization in Geneva. Geneva configures the number of entries in each D-IQ to four since more than 85% of instruction DCs have a length of four or less as presented in Figure 2b. For the operation of the dynamic confluence, each D-IQ entry must be able to serve as an entry for both the ROB and IQ. Figure 4 illustrates a reconfigurable scheme of D-IQs as an entry for both ROB and IQ. A D-IQ has an entry size identical to an ROB, which is relatively larger than an IQ due to the recovery data. Since a DQ is repurposed from an ROB, the DQ can function as the dynamic confluence without changing the entry size of a D-IQ.

3.4 Dynamic Confluence Operation

ROB Operation: A DQ can seamlessly function as the second-tier in-order queue of an ROB. To ensure a DQ-ROB upholds the in-order commitment between instructions, Geneva adopts a shortage pointer. Figure 5 depicts the cooperation between an ROB and a DQ-ROB, under various operation scenarios.

(a) The operation of a DQ-ROB begins when the shortage pointer indicates the same ROB entry as the head and tail pointers of an ROB, which happens when the ROB is full.
(b) When the DQ-ROB is full and there are empty entries in the ROB, subsequently dispatched instructions are steered back to the ROB. The in-order commitment sequence between the ROB and the DQ-ROB is strictly maintained by checking whether the shortage pointer continues to direct to the same entry.
(c) If the ROB head pointer meets again with the shortage pointer, it means that all of the instructions in the DQ-ROB are older than those in the ROB. Thus, the DQ-ROB is prioritized for instruction commitment.

IQ Operation: All DCs for instructions are managed via a RAT. A register renaming stage verifies whether each instruction is included in a specific DC, is a new DC-Head, or neither. Once a new DC-Head is determined and there is an empty D-IQ in the DQ-IQ partition, the steering logic of Geneva allocates the DC-Head to the D-IQ. When this happens, the index of the D-IQ is stored in the D-IQ ID, one of the bit vectors that constitutes a RAT entry for the destination register of the DC-Head. In the register renaming stage, subsequent instructions that are dependent on the DC-Head inherit the same D-IQ ID, and they are directed to the same D-IQ. Note that if the D-IQ becomes full, the steering logic signals a busy status to the RAT to prevent steering further instructions to that D-IQ until the busy status is resolved.

3.5 Instruction Steering Mechanism

The instruction steering mechanism of Geneva exhibits unique characteristics through the operation of the dynamic confluence as described in Section 3.4. To explain the details of the instruction steering mechanism, we design an out-of-order core that adopts P-IQs from CES [11] as second-tier queues, which we refer to as CES-enhanced out-of-order core (CES-Enhanced OoO), and then compare it with Geneva. Figure 6 demonstrates the instruction steering mechanism of Geneva with that of the CES-Enhanced OoO. In this example, we assume a situation where the dispatch of instructions from i59 to i65 is stalled due to an ROB reaching its full capacity. The commit sequence and dependency chains among instructions are shown in Figure 6a. Figure 6b illustrates the operation of CES-Enhanced OoO when its ROB fills up after enqueuing i58. Thus, CES-Enhanced OoO suffers from dispatch stalls for instructions from i59 to i65, thereby resulting in energy wastage due to unoccupied, yet allocated P-IQs. In contrast, the dynamic confluence partition policy of Geneva enables the DQ-ROB partition to be available before the ROB reaches its full capacity as depicted in Figure 6c. This allows dispatching instructions from i59 to i65 even after an ROB is full. Each DC-Head is inserted into an empty D-IQ of the DQ-IQ, and instructions dependent on the DC-Head are inserted into the same D-IQ. Unused D-IQs are managed in the free partition to reduce energy consumption.

4 EVALUATION

4.1 Experimental Setup

We use a cycle-level x86 simulator [15] that is integrated with a DDR4 DRAM model [6]. Applications are selected from both SPEC2006 [14] and 2017 [2] benchmark suites. We run the most representative region of 300 million instructions using the SimPoint methodology [13] after warming up for 300 million instructions. Runtime dynamic energy results are derived from a modified version of McPAT [9] at the ITRS 22nm technology.

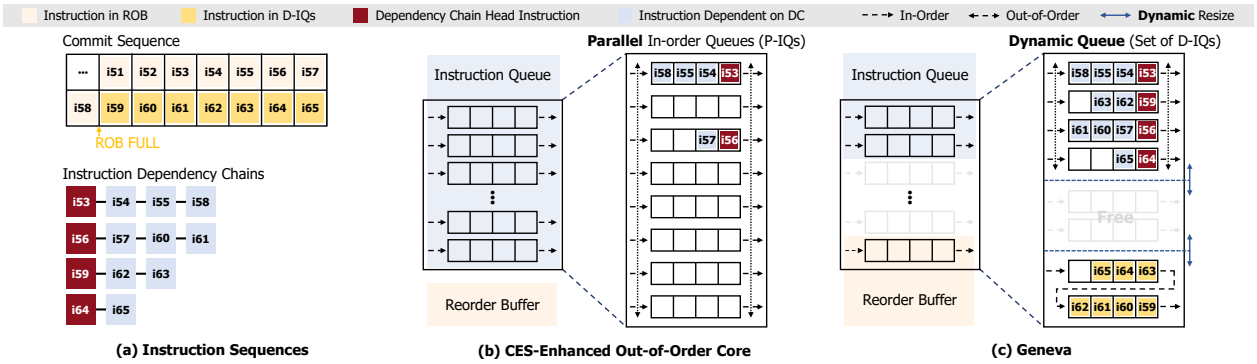

Figure 6: Comparison of instruction steering mechanism between CES-Enhanced OoO and Geneva microarchitecture

Table 1: Core and Memory Configurations

Resource	Baseline	DNB	CES-Enhanced OoO	Geneva
Core	8-wide superscalar @3.4 GHz			
Load queue	72			
Store queue	56			
Exe. resources	4 Int ALUs, 1 Int DIV, 1 Int MUL, 2 Fp ADD, 1 Fp DIV 2 Fp MULs, 4 AGUs, 2 Branches, 180 Int RF, 168 Fp RF			
Branch predictor	TAGE: 17-bit GHR with 1 bimodal and 4 tagged predictors, 512 sets 4-way BTB			
L1 I/D cache	32 KiB, 8-way, 4-cycle latency, 8 MSHRs, stride-based prefetcher			
L2 cache	256 KiB, 8-way, 12-cycle latency, 32 MSHRs			
L3 cache	1 MiB, 4-way, 42-cycle latency, 64 MSHRs			
Memory	4 GiB, DDR4 DRAM, 2400 MT/s 1 Ch., 1rank			

Table 2: ROB and IQ Configurations

Design	ROB	Out-of-Order IQ	In-Order IQ
Baseline	224	96(/64)	
DNB	224	64	CRQ: 32, DLQ: 128
CES-Enhanced OoO	192	64	P-IQs: 8x4
Geneva	192	64	D-IQs: 8x4

We model DNB, CES-Enhanced OoO, and Geneva based on a Skylake-like out-of-order processor [4], and their detailed configurations are listed in Table 1. DNB, CES-Enhanced OoO, and Geneva adopt different-size ROBs and IQs compared to the baseline, as presented in Table 2. Geneva and CES-Enhanced OoO reduce the ROB and IQ by 32 entries each compared to the baseline. Instead, they employ eight D-IQs or P-IQs with each queue consisting of four instruction entries (*i.e.,* a total of 8x4 entries). DNB and baseline-IQ32 feature an out-of-order IQ with 32 fewer entries compared to the baseline, and DNB uses two additional separate second-tier in-order queues: one with 128 entries (*i.e.,* Delay Queue (DLQ)) and another with 32 entries (*i.e.,* Critical-Ready Queue (CRQ)).

4.2 Performance Analysis

Figure 7 plots the performance of tested schemes, normalized to the baseline. The results show that while baseline-IQ32, DNB, and CES-Enhanced OoO degrade performance by 5.3%, 4.1%, and 3.1% respectively, Geneva improves IPC by 2.6%. Due to the reduction in the number of IQ entries, baseline-IQ32 performs worse than the baseline. DNB uses a DLQ and a CRQ, in place of 32 out-of-order IQ entries. However, in our experiment environment, we observe that DNB is unable to produce a sufficient number of critical and ready instructions to effectively mitigate the performance degradation caused by the reduction of out-of-order IQ, compared to the baseline. By reallocating a portion of an ROB, CES-Enhanced OoO employs P-IQs that operate as a restricted dynamic instruction scheduler, but the smaller ROB leads to an increase in instructions dispatch stall cycles, which in turn makes it difficult for the P-IQs to operate effectively. In contrast, Geneva outperforms the other tested designs across almost all of the 28 SPEC benchmarks, resulting in an average performance increase of 2.6% over the baseline. Such an improvement is attributed to the operation of dynamic confluence, which effectively mitigates the resource constraints in an ROB and IQ. In other words, Geneva can well utilize its dynamic instruction scheduler by minimizing instruction stall cycles caused by the shortage of ROB entries since the DQ in Geneva can operate as a DQ-IQ, a DQ-ROB, or both.

4.3 Resource Utilization Balance Analysis

Geneva enhances performance and reduces energy consumption by achieving a utilization balance between an ROB and an IQ. Figure 9 compares the resource utilization balance of the baseline, baseline-IQ32, CES-Enhanced OoO, and Geneva (*i.e.,* cases when IQ occupancy exceeds 90% while ROB occupancy is also above 90%, and vice versa). The results show that the balance in baseline-IQ32 becomes skewed since it uses 32 fewer IQ entries. CES-Enhanced OoO utilizes P-IQs as a restricted dynamic scheduler, thereby expecting a similar effect to an increase in the total IQ size. However, the resource utilization balance of CES-Enhanced OoO shows the opposite behavior compared to baseline-IQ32 due to smaller ROB hardware. In contrast, Geneva alleviates the resource utilization imbalance between an ROB and an IQ. Specifically, the proposed dynamic confluence allows Geneva to dynamically utilize a DQ as a DQ-ROB, a DQ-IQ, or both. Consequently, Geneva exhibits an improved resource utilization balance compared to other architectures. Note that the resource utilization balance of DNB cannot be fairly compared with that of other designs it employs two IQs (*i.e.,* CRQ and DLQ) as second-tier queues. Therefore, the experiment results of DNB are not presented in Figure 9.

4.4 Energy Analysis

Figure 8 plots the energy consumption among different core designs, normalized to the baseline. DNB (+14.3%), CES-Enhanced

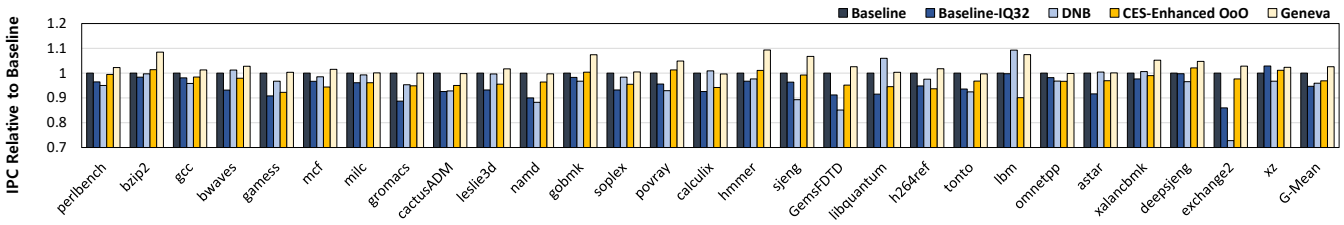

Figure 7: IPC performance of different core designs normalized to the baseline

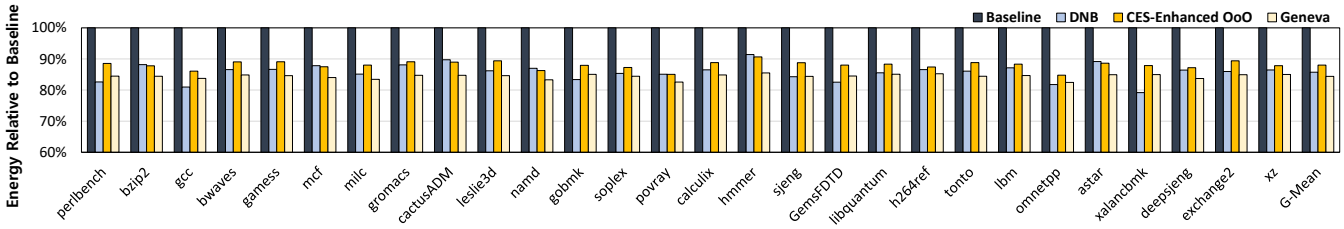

Figure 8: Energy consumption of different core designs normalized to the baseline

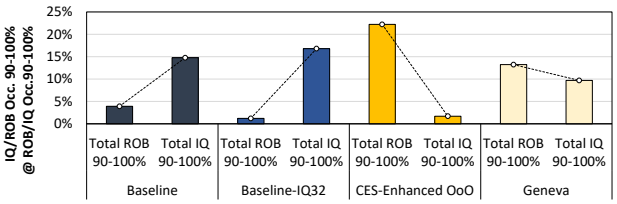

Figure 9: Comparison of resource utilization balance

OoO (+12.0%), and Geneva (+15.6%) reduce energy consumption over the baseline core. DNB saves energy by replacing 32 out-of-order IQ entries with 128-entry and 32-entry simple FIFO queues (*i.e.,* DLQ and CRQ). Additionally, DNB reduces energy consumption through a virtual register renaming technique [10] that defers register renaming for instructions that reside in a DLQ or a CRQ until the instructions are issued. Similarly, CES-Enhanced OoO conserves energy by reducing the size of out-of-order IQ by 32 entries. However, P-IQs, the restricted dynamic scheduling logic of CES-Enhanced OoO, remain perpetually active, even when the queues are not in use. Thus, CES-Enhanced OoO consumes more energy than Geneva. Geneva achieves the highest energy savings among the tested schemes for two reasons. First, Geneva uses 32 fewer CAM-based out-of-order IQ entries. Second, Geneva minimizes energy consumption by setting D-IQs within the free partition as inactive since they are not used in both the DQ-ROB and DQ-IQ.

5 CONCLUSION

In out-of-order processors, it is infrequent that an ROB and an IQ are both fully occupied. By leveraging this, we propose a Geneva microarchitecture that improves performance and saves energy by addressing this utilization gap. Experiments show that Geneva improves performance by 2.6% and achieves energy savings of 15.6% even with 32 fewer IQ entries over the conventional out-of-order core.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their insightful feedback. Won Woo Ro is the corresponding author.

REFERENCES

[1] M. Alipour, S. Kaxiras, D. Black-Schaffer, and R. Kumar. 2020. Delay and Bypass: Ready and Criticality Aware Instruction Scheduling in Out-of-Order Processors. In *Proceedings of IEEE International Symposium on High Performance Computer Architecture*. IEEE, 424–434.

[2] J. Bucek, K.-D. Lange, and J. v. Kistowski. 2018. SPEC CPU2017: Next-Generation Compute Benchmark. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*. 41–42.

[3] T. E. Carlson, W. Heirman, O. Allam, S. Kaxiras, and L. Eeckhout. 2015. The Load Slice Core Microarchitecture. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*. 272–284.

[4] J. Doweck, W.-F. Kao, A. K.-y. Lu, J. Mandelblat, A. Rahatekar, L. Rappoport, E. Rotem, A. Yasin, and A. Yoaz. 2017. Inside 6th-Generation Intel Core: New Microarchitecture Code-Named Skylake. *IEEE Micro* 37, 2 (2017), 52–62.

[5] I. Jeong, J. Lee, M. K. Yoon, and W. W. Ro. 2022. Reconstructing Out-of-Order Issue Queue. In *Proceedings of 55th IEEE/ACM International Symposium on Microarchitecture*. IEEE, 144–161.

[6] Y. Kim, W. Yang, and O. Mutlu. 2015. Ramulator: A Fast and Extensible DRAM Simulator. *IEEE Computer Architecture Letters* 15, 1 (2015), 45–49.

[7] R. Kumar, M. Alipour, and D. Black-Schaffer. 2019. Freeway: Maximizing MLP for Slice-Out-of-Order Execution. In *Proceedings of IEEE International Symposium on High Performance Computer Architecture*. IEEE, 558–569.

[8] K. Lakshminarasimhan, A. Naithani, J. Feliu, and L. Eeckhout. 2020. The Forward Slice Core Microarchitecture. In *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*. 361–372.

[9] 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.

[10] T. Monreal, A. González, M. Valero, J. González, and V. Viñals. 1999. Delaying Physical Register Allocation Through Virtual-Physical Registers. In *Proceedings of the 32nd Annual ACM/IEEE International Symposium on Microarchitecture*. IEEE, 186–192.

[11] 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.

[12] A. Sembrant, T. Carlson, E. Hagersten, D. Black-Shaffer, A. Perais, A. Seznec, and P. Michaud. 2015. Long Term Parking (LTP): Criticality-aware Resource Allocation in OOO Processors. In *Proceedings of the 48th International Symposium on Microarchitecture*. 334–346.

[13] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. 2002. Automatically Characterizing Large Scale Program Behavior. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 45–57.

[14] C. D. Spradling. 2007. SPEC CPU2006 Benchmark Tools. *ACM SIGARCH Computer Architecture News* 35, 1 (2007), 130–134.

[15] R. Ubal, B. Jang, P. Mistry, D. Schaa, and D. Kaeli. 2012. Multi2Sim: A Simulation Framework for CPU-GPU Computing. In *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*. 335–344.