Design Tradeoffs in Backend Organization in Out-Of-Order RISC-V Processors.

Esther Alonso
Computer Engineering Group
University of Cantabria
Santander, Spain
alonsogae@unican.es

Pablo Abad
Computer Engineering Group
University of Cantabria
Santander, Spain
abadp@unican.es

Pablo Prieto
Computer Engineering Group
University of Cantabria
Santander, Spain
prietop@unican.es

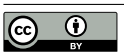
Valentin Puente
Computer Engineering Group
University of Cantabria
Santander, Spain
vpuente@unican.es

Abstract

The primary role of the processor backend is to schedule instructions exposed by the frontend onto available execution resources while preserving sequential semantics. Optimizing this process is critical in modern superscalar out-of-order architectures, as it involves navigating complex tradeoffs in issue logic design and functional unit (FU) organization.

Recent high-performance RISC-V cores with aggressive out-of-order pipelines tend to adopt highly distributed backends, featuring multiple execution ports, each fed by a private instruction queue and scheduler. This contrasts with conventional designs relying on unified or partially shared scheduling structures spanning multiple ports. Motivated by this divergence, this work analyzes the performance implications of scheduling scope and execution resource organization. We evaluate three backend organizations: (i) a unified issue queue and global scheduler serving as an idealized performance upper bound, (ii) a fully distributed per-port scheduling organization, and (iii) a replicated sliced backend composed of homogeneous clusters, each one integrating an independent issue queue and a complete set of functional units. Using detailed simulation across representative benchmark suites, we explore the design space under both ideal and latency-aware assumptions.

Our results show that while unified scheduling can provide up to ~20% performance benefit over distributed queues under equal latency assumptions, this advantage is highly sensitive to scheduling delay and largely vanishes with the addition of a single cycle to the wakeup–select loop. In contrast, replicated sliced backends demonstrate robust competitiveness, matching distributed designs with fewer execution ports and remaining within 5–10% performance degradation even under conservative latency penalties. These findings suggest that backend replication could provide an alternative scaling tradeoff for execution resources and could be considered alongside conventional distributed organizations in next-generation high-performance RISC-V microarchitectures.



SPAA '26, London, United Kingdom

ACM ISBN 979-8-4007-2761-0/2026/07
https://doi.org/10.1145/3816782.3819204

CCS Concepts

• **Computer systems organization** → **Superscalar architectures**; **Reduced instruction set computing**; • **General and reference** → Performance.

Keywords

Instruction Queue, RISC-V, Processor Backend, Scheduling

ACM Reference Format:
Esther Alonso, Pablo Abad, Pablo Prieto, and Valentin Puente. 2026. Design Tradeoffs in Backend Organization in Out-Of-Order RISC-V Processors.. In *38th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '26), July 06–10, 2026, London, United Kingdom.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3816782.3819204

1 Introduction

Although processor microarchitecture was considered less critical after frequency scaling ended two decades ago and the focus shifted to multicore designs, it was never truly set aside. In practice, many of the promised gains from multicore architectures were not realized due to the ever-growing complexity of software. Consequently, instruction-level parallelism (ILP) remains highly relevant in modern chip design and continues to be a key market driver. The advent of open-source hardware has reinvigorated this interest. RISC-V [36], the most prominent example of such, is currently playing a transformative role comparable to that of open-source software four decades ago. Several key stakeholders argue that a similar paradigm shift may once again take place [23]. Since its relatively recent introduction, RISC-V has made substantial progress, and current trends indicate that the performance gap with established commercial competitors is steadily narrowing. Its open nature not only enables global collaboration on complex design challenges, but also supports agile development practices inspired by those widely adopted in software engineering [23, 38].

A clear example of this convergence can be found in the most recent superscalar RISC-V processors introduced by companies such as SiFive [2], Alibaba's Xuantie [13], as well as open-source initiatives like XiangShan [38]. In all cases, these microarchitectures exhibit a level of complexity comparable to that of the latest high-performance cores from traditional vendors such as Intel and AMD.

They feature wide out-of-order pipelines, sophisticated branch prediction mechanisms (e.g., TAGE-SC-L [31]), large decode, issue, and retire widths (6–8), a high number of execution ports (12 or more), and support for both scalar and vector instruction set extensions. In microarchitectures of such complexity, backend design relies on a small set of key decisions that shape the overall organization and directly affect the complexity of supporting hardware structures. Among these, the issue logic and the functional unit (FU) organization play a central role.

The issue logic is fundamental to exploiting instruction-level parallelism (ILP) in an out-of-order processor. It typically consists of two main components: the Issue Queue (IQ), where instructions wait for operand and FU availability, and the scheduling logic, which selects ready instructions for dispatch to free functional units. The hardware structures involved in this stage frequently lie on the processor's critical path [21, 26, 27, 37], making the complexity–performance trade-off particularly important for maximizing ILP while sustaining high clock frequencies. Given the highly heterogeneous nature of general-purpose workloads, the execute stage comprises multiple execution paths with different latencies and levels of complexity. Functional units are conventionally grouped into execution ports, and the number of ports determines the maximum number of instructions that can be dispatched from the issue logic per cycle. This organization serves to constrain the complexity of several critical components, including the scheduling logic, the bypass/forwarding networks, and the register file read and write ports [27].

The design of both the issue logic and the FU organization is therefore pivotal in backend microarchitecture. An examination of recent high-performance RISC-V cores reveals a recurring design pattern. For instance, SiFive's latest P870 microarchitecture [2] implements 13 execution ports in the backend, each fed by an independent issue queue (typically comprising ten to twenty entries) and equipped with a per-port scheduler whose scheduling scope is restricted to the entries of its associated queue. Despite implementation-specific differences, other processors such as Xuantie's C930 [13] and XiangShan's Kunminghu [38] adopt a similar backend organization. Given the complexity of wakeup and select logic in modern schedulers [21, 26], and its frequent impact on the processor's critical path [27], this distributed organization offers clear advantages in terms of scalability and bounded design complexity.

In contrast to the RISC-V architectures described above, more established vendors such as Intel and AMD have historically followed different backend design philosophies in their high-performance processors. Early AMD Zen generations [35] relied on a more distributed organization with multiple issue queues, while Intel architectures traditionally employed fewer and more centralized scheduling structures. Over time, however, both vendors appear to have evolved toward partially centralized organizations with a small number of shared issue queues grouped by operation domain. For example, AMD's Zen family progressively transitioned from the distributed organization of early Zen generations to a backend featuring three major issue queues in Zen 5 [33], corresponding to integer arithmetic, memory operations, and floating-point/vector execution. Each queue is managed by a scheduler capable of dispatching multiple instructions per cycle to several execution ports. Intel architectures such as Meteor Lake [18] and Lunar Lake [8] although still architecturally distinct, exhibit a backend organization with a broadly similar philosophy, relying on a limited number of shared scheduling domains rather than fully distributed per-port queues.

More recently, Andes Technology has introduced its high-performance RISC-V processor Cuzco [17], which can be viewed as an alternative point in the design space. While it preserves the RISC-V trend of assigning a dedicated issue queue and scheduler to each execution port, it significantly increases the functionality available per port. In fact, each port replicates the full set of execution capabilities required by the ISA, allowing the backend to operate with a smaller number of ports.

The divergence between these backend organizations, together with the emergence of modular and clustered alternatives, motivates the present work. Our goal is to analyze, through a systematic exploration of the backend design space using a representative set of applications, how scheduling organization affects performance. As previously mentioned, despite implementing the same instruction set architecture (ISA), prominent processors from AMD and Intel have markedly different backend architectures. This raises the question of whether these differences stem from legacy constraints, or deliberate architectural and technological trade-offs. Our goal is to address this issue in the context of RISC-V by systematically exploring the design space and analyzing the impact of this key architectural decision. First, we quantify the gap to an idealized backend featuring a single unified issue queue with full scheduling scope. Then, we estimate the impact of implementation cost by modeling increasing scheduling delays, reflecting the growing complexity of selecting from a larger instruction window. Finally, we evaluate where novel architectures such as Cuzco lie with respect to both extremes of the design space (fully distributed versus fully unified scheduling), considering both idealized and more realistic implementations.

The main contributions of this work are as follows:

(1) A thorough design-space exploration of scheduling scope in modern superscalar backends, quantifying the performance gap between unified and fully distributed issue organizations under both ideal and latency-aware assumptions.
(2) An evaluation of clustered, fully replicated backend slices as an alternative to conventional distributed port organizations, identifying the conditions under which replication can match or exceed the performance of wider distributed designs.
(3) A latency-sensitivity analysis of critical backend stages, demonstrating that the performance advantage of unified scheduling is highly fragile to additional wakeup–select delay, while replicated organizations exhibit greater robustness to moderate writeback latency variations.
(4) Architectural insights into execution-resource scaling trade-offs in emerging high-performance RISC-V cores, highlighting replication as a viable alternative scaling path alongside traditional distributed backends.

2 Background: Superscalar Backend

Within processor backend, a small number of components play a non-linear role in determining both performance and implementation complexity. This section focuses on two such components: the issue logic, responsible for dynamically scheduling instructions to available execution resources, and the writeback and forwarding structures, which govern result propagation and dependency resolution. We review their organization, scalability challenges, and the key design tradeoffs that motivate the backend organizations evaluated in the remainder of this work.

2.1 Issue

The issue logic is responsible for dispatching instructions from the Issue Queue (IQ) to the available functional units (FUs). In out-of-order processors, this process is fully dynamic: instructions are issued as soon as their source operands become available, enabling true out-of-order execution. This stage is governed by two tightly coupled mechanisms, wakeup and select. The wakeup logic tracks the completion of producer instructions and determines which IQ entries have all their operands ready, while the select logic chooses, among the ready instructions, those that will be issued in the current cycle based on resource availability and scheduling policies.

The literature describes two dominant implementation schemes for issue logic. The first and most widely adopted approach, sketched in Figure 1, combines RAM-based structures to store instruction state with content-addressable memory (CAM) logic for tag matching. In this design, when producer instructions complete, their destination tags are broadcast to all IQ entries, which perform parallel comparisons to determine operand readiness. An alternative approach implements the issue logic using a bit-matrix representation whose dimensions are proportional to the number of in-flight instructions and the number of physical registers. In this case, instruction completion events clear all bits in the corresponding matrix row except those associated with non-ready source operands. In both approaches, issue queue size and issue width increase logic complexity, CAM-based designs suffer from an increasing number of tag comparisons and broadcasts while matrix-based designs scale poorly due to the growing size and wiring complexity of the matrix. As a result, the wakeup–select loop frequently lies on the processor's critical path, making scheduling one of the most timing- and energy-sensitive stages [27].

A substantial body of work has addressed these challenges. Abella et al. [3] provide a comprehensive survey that consolidates most classical scheduling optimizations proposed during the early 2000s, when increasing superscalar widths motivated extensive research on scalable issue logic. These techniques include reducing unnecessary tag comparisons by filtering empty or ready entries [16], selectively disabling portions of the issue queue [15, 16], combining CAM-based structures with simpler in-order queues [25], and optimizing matrix-based implementations through partitioning or structural reuse [9, 20]. More recent studies revisit these ideas with a renewed focus on limiting timing and energy costs in modern implementations, for example through partitioned issue queues [30], fragmented matrix-based scheduling structures [5], or optimized wakeup logic [6, 26].

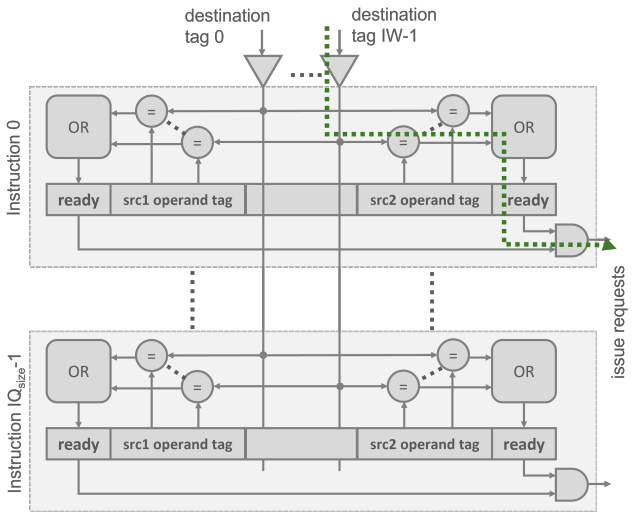


Figure 1: IQ and wakeup logic in the Issue stage and its critical path.

While these techniques significantly improve the scalability and efficiency of issue logic, they are largely orthogonal to the architectural design choices explored in this work. Most prior proposals apply to both globally scheduled issue queues and distributed, per-port scheduling organizations.

2.2 Execute (+ Writeback)

The number of instructions that flow from the issue stage to the execute stage in each cycle has a direct impact on the sizing and cost of several backend components. A clear example is the register file: as the number of issued instructions increases, more operands must be read per cycle, significantly increasing both area and power consumption. A similar effect arises in the final stages of the backend, where writeback bandwidth requirements and the complexity of bypass networks grow with the number of concurrently active functional units. These bypass networks forward results directly between functional units to minimize pipeline stalls caused by data dependencies.

Given that the amount of instruction-level parallelism (ILP) present in most applications is inherently limited, there exists a point beyond which increasing the number of functional units available for simultaneous execution no longer improves performance and instead only amplifies the cost of backend structures. In contemporary out-of-order processors, this performance-cost trade-off is addressed by grouping functional units into execution ports, such that at most one instruction can be issued per port in a given cycle. Functional units are typically distributed across ports with the goal of maximizing execution bandwidth, tailoring the grouping to the instruction mix observed in representative workloads [28]. Despite this grouping, in modern wide-issue designs the bypass network remains a critical component, as it directly impacts area, power consumption, and the critical path of the execute stage. As a result, many clustered backend architectures restrict bypassing to a subset

of possible paths, typically those within the same cluster, and do not provide full inter-cluster bypassing.

As long as the register file is centralized and shared across the entire backend (or replicated in functionally equivalent copies), limitations in the bypass network primarily affect the latency with which data dependencies are resolved. If no bypass path is available, consumer instructions must wait until operands are written back to and subsequently read from the register file. When clustering extends to the register file itself, additional inter-cluster communication mechanisms are required to allow instructions in one cluster to consume values produced in another. These mechanisms may be implemented explicitly through inter-cluster communication buffers [11, 12] or implicitly via specialized register-copy instructions [14].

In this work, we assume a single centralized register file for all evaluated configurations. Under this assumption, the absence of certain bypass paths manifests solely as additional latency penalties in specific cases. We evaluate both an idealized implementation featuring a single-cycle, fully connected bypass network and a more realistic design in which bypass latency gradually increases.

To the best of our knowledge, existing literature does not provide a systematic design-space exploration of backend organizations that contrasts unified and partitioned scheduling scopes in contemporary out-of-order processors, which is the focus of this study.

3 Evaluation Framework

3.1 Methodology

We use gem5 [7, 24] as the primary tool for our evaluation, forked from commit 7a2b0e4 (Release v25.1.0.0) of the official repository. During the course of this study, we identified a limitation in the default steering policy used for distributed-queue backends, which led to suboptimal instruction-to-port assignment. We implemented a corrected steering mechanism to address this issue, improving performance by approximately 5% on average (and up to ~10% for SPECint workloads). To ensure reproducibility, the modified source code required to replicate all experiments is publicly available [4].

We evaluate workloads drawn from three benchmark suites selected to exercise both the processor core and the memory subsystem: SPEC CPU2017 [1] (SINT and SFP labels), SPLASH-4 [19] (SH4 label), and the NAS Parallel Benchmarks (NPB label) [22]. SPEC CPU2017 is an industry-standard benchmark suite composed of 23 applications (rate mode), divided into integer and floating-point subsets, and representative of a broad range of workloads spanning desktop, server, and scientific computing domains. SPLASH-4 was originally designed to characterize shared-memory multiprocessor workloads; although we execute 14 applications in single-threaded mode, they remain representative of scientific, engineering, and graphics workloads. Finally, we evaluate 9 applications from the NAS Parallel Benchmarks, all derived from computational fluid dynamics kernels. For each application, execution is fast-forwarded to its region of interest (ROI), after which detailed cycle-accurate simulation is performed for a fixed number of instructions.

To enable a realistic and representative evaluation, we consider two baseline architectural configurations. The first one, referred as BigO3, is an ultra-wide out-of-order core, comparable to recent high-performance RISC-V designs such as those discussed in Section 1 [2, 13, 38]. This configuration features a frontend width of six instructions per cycle and a generously provisioned backend with multiple execution ports. Its main microarchitectural characteristics, including the execution port distribution, are summarized in Tables 1 and 2.

Table 1: Processor configuration under test.

Parameter	BigO3	SmallO3
L1 Icache	64Kb, 4-w	32Kb, 4-w
Branch Pred.	TAGE-SC-L	TAGE-SC-L
Dec/Ren/Exe/Ret	6/6/12/8	3/3/6/4
ROB size	700 entries	192 entries
Register File	228 (int), 240 (FP)	128 (int), 119 (FP)
LSQ size	196 (LD) / 64 (ST)	36 (LD) / 18 (ST)
L1 Dcache	64Kb, 8-w, 4-cyc hit	32Kb, 4-w. 4-cyc hit
L2 Dcache	1Mb, 8-w, 12-cyc hit	256Kb, 8-w, 12-cyc hit
LLC	16Mb, 16-w, 40-cyc hit	2Mb, 16-w, 40-cyc hit

Table 2: FU organization for each processor configuration.

Port	Type	BigO3	SmallO3
1	INT	ALU, Div	ALU, Div
2	INT	ALU, Mul	ALU, Mul
3	INT	ALU, Mul	ALU, Br
4	INT	ALU	–
5	INT	Br, ALU	–
6	INT	Br, ALU	–
7	FP	Add, Mul, MAC, Div, Sqrt	Add, Mul, MAC, Div, Sqrt
8	FP	Add, Mul, MAC	–
9	FP	Add, Mul, MAC	–
10	MEM	AGU, Ld	AGU, Ld
11	MEM	AGU, Ld, St	AGU, Ld, St
12	MEM	AGU, Ld, St	–

In contrast to the high-performance focus of this configuration, we define a second, more frugal out-of-order core with fewer execution resources, representing a contemporary design optimized for reduced complexity and power consumption. This configuration, referred to as SmallO3, is inspired by the SiFive P550 microarchitecture [32]. Its design emphasizes efficiency and lower degrees of instruction-level parallelism, making it representative of general-purpose, energy-efficient processor cores. Table 1 and 2 also summarize its main microarchitectural characteristics and port distribution. Additionally, to provide a complete characterization of the execution backend, Table 3 details the execution latencies and pipelining capabilities of the functional units used across both baseline configurations.

3.2 Backend Organizations under test

As discussed in the Introduction, contemporary processors exhibit a wide range of backend organizations. Motivated by this diversity, we group the evaluated alternatives into three backend organizations that correspond to the three design philosophies introduced

Table 3: Execution latency and pipelining characteristics of Functional Units.

Class	Operation	Latency (cycles)	Pipelined
INT	ALU, Br, Mul	1	Yes
INT	Div	3	No
FP	Add, Mul	3	Yes
FP	MAC	5	Yes
FP	Div, Sqrt	5	No

earlier. Each organization, sketched in Figure 2, is defined below as it is instantiated in our evaluation framework.

- Single-IQ + Distributed Ports: Inspired by the centralized scheduling philosophy adopted in many high-performance commercial processors, this backend model employs a single global issue queue, where the scheduling scope of each execution port spans the entire queue. Functional units are distributed across execution ports to maximize execution bandwidth, and the scheduler may dispatch ready instructions to any eligible port.
- Sliced-IQ + Distributed Ports: Representative of the organization commonly adopted in contemporary RISC-V processors, this configuration implements an independent issue queue per execution port. The queue selection policy is intentionally simple: instructions are first routed based on their operation type, and when multiple ports provide the required functional unit, an occupancy-based policy is employed, selecting among the ports with required functionality.
- Sliced-IQ + Replicated Ports: The final configuration models the backend organization of the Cuzco processor [17]. It preserves the sliced-IQ structure of the previous design but modifies the functional unit organization. In this case, a complete set of functional units is replicated at each execution port. As a result, port selection depends solely on queue occupancy, since instruction type no longer constrains port eligibility.

Each backend organization presents clear advantages and disadvantages. Distributed issue queues reduce the complexity and cost of the scheduling logic but may limit performance by restricting the effective scheduling scope. Conversely, replicating functional units across ports simplifies scheduling decisions by eliminating instruction-type constraints during port selection. However, full functional unit replication incurs significant area overhead and may also increase the cost and complexity of other backend structures, such as bypass and forwarding networks. The goal of our evaluation is to quantitatively characterize these trade-offs and provide numerical insight into their practical impact across a broad set of workloads.

4 Results

4.1 Issue Queue Parametrization

This section presents the experimental evaluation of the backend organizations introduced earlier. We analyze the impact of issue queue parametrization, scheduling scope, and execution resource

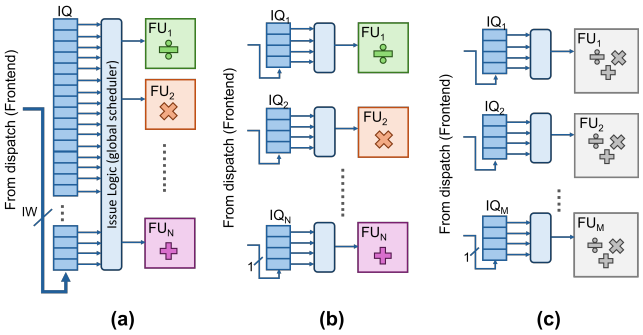


Figure 2: Backend organizations under test. (a) Single IQ + Distributed Ports, (b) Sliced IQ + Distributed Ports, (c) Sliced IQ + Replicated Ports.

organization on performance. Each subsection explores a specific dimension of the design space, first under idealized assumptions and subsequently incorporating cost-oriented considerations, in order to understand the robustness of the trends.

A solid and well-justified baseline is essential for any evaluation based on detailed simulation. Although cycle-accurate simulation provides good fidelity, it may introduce small discrepancies with respect to measurements obtained from RTL or real hardware. For this reason, our first step is to determine appropriate issue queue (IQ) sizes for the evaluated out-of-order processor implementations and baseline configurations described in Section 3. To this end, we progressively increase the IQ size and analyze the resulting evolution of instructions per cycle (IPC). Our objective is to identify the point beyond which increasing issue queue capacity yields only marginal IPC improvements. The resulting curve exhibits three characteristic regions: (i) an initial regime where IPC increases rapidly with queue size, (ii) a transition region in which performance gains progressively diminish, and (iii) a saturation region where further increases in IQ capacity produce negligible performance improvements. To capture these operating regimes, we select three representative IQ sizes located near the end of the initial growth region, within the transition knee, and at the onset of saturation, where all benchmarks converge to ~99% of peak IPC. These three reference points are used throughout the remainder of the evaluation and can be considered near-optimal from a performance–cost perspective, capturing different trade-offs between queue capacity and achieved performance. The rationale for selecting multiple points rather than a single configuration is to ensure that subsequent conclusions are robust across a reasonable range of IQ sizes and not tied to a specific design point.

Figures 3 and 4 present the results obtained for the two evaluated baseline configurations, SmallO3 and BigO3, respectively. To maintain readability, IPC values are grouped into four data series, each representing the average IPC of the applications belonging to a given benchmark suite. For SPEC CPU2017, results are further divided into integer and floating-point workloads. Individual application results are also overlaid in light tones (without labels) to highlight the consistency of the overall trends while exposing the variability across workloads that averages alone may conceal. In

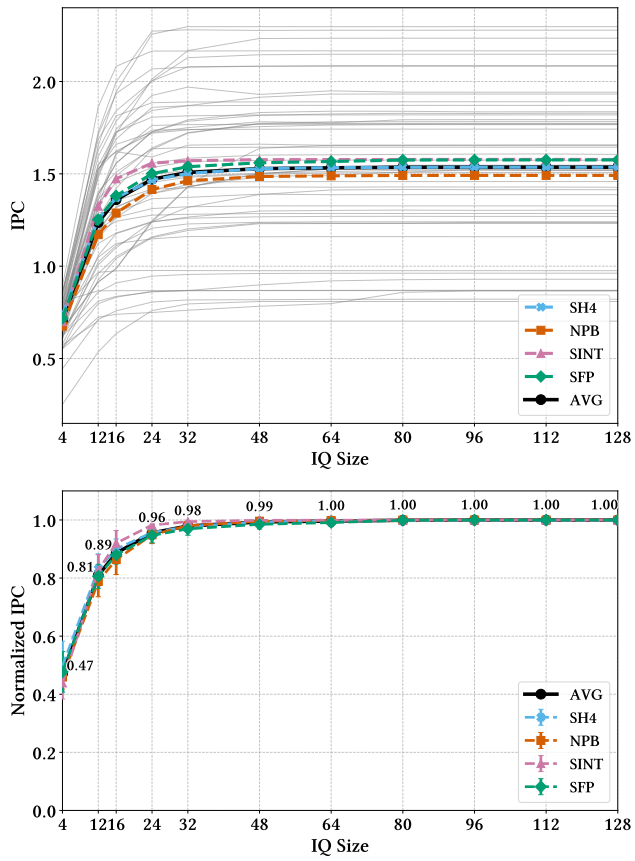


Figure 3: Single-IQ Performance evolution with IQ Size, SmallO3 configuration. (up) Raw IPC value. (down) IPC normalized to maximum.

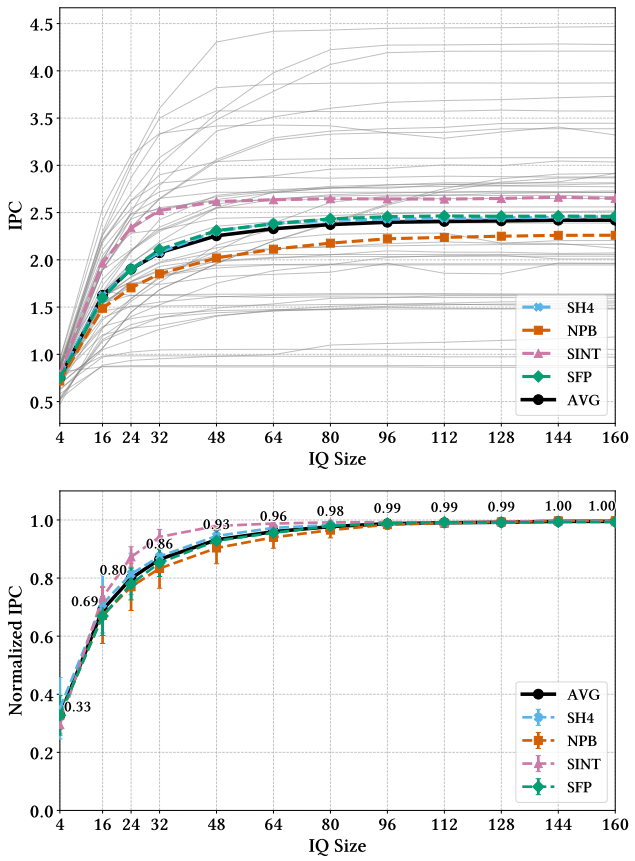


Figure 4: Single-IQ Performance evolution with IQ Size, BigO3 configuration. (up) Raw IPC value. (down) IPC normalized to maximum.

each figure, we report both absolute IPC values (top plots) and IPC normalized to the maximum observed performance (bottom plots). For the normalized data, error bars corresponding to a 95% confidence interval among applications of that benchmark are included.

The results confirm several expected trends. First, they highlight the critical role of the issue queue depth in determining overall performance. Second, they illustrate the higher absolute performance achieved by wider, more aggressive microarchitectures (BigO3 versus SmallO3). Finally, they reveal significant variability in sensitivity to IQ size across benchmark suites. In particular, applications from the NAS benchmark suite exhibit a stronger dependence on IQ capacity and have the highest variability among them. This behavior is likely due to their lower baseline IPC and numerical nature, where a larger instruction window modestly expands the effective scheduling scope and enables additional instruction-level parallelism to be exploited. On the opposite side, SPEC2017-INT applications get to their maximum IPC with smaller IQ sizes. A possible explanation could be the relatively lower number of data dependencies in these applications, which increases the chances to find an instruction to schedule even with the smaller planification scope.

Beyond these effects, our results also reveal an interesting divergence from the issue queue sizes adopted in commercial processors used as reference points in this study. Specifically, the IQ capacities suggested by our IPC–IQ size curves tend to be somewhat smaller than those used in practice. Rather than undermining the validity of the analysis, this observation hints at additional architectural factors influencing real-world design choices. As discussed in subsequent sections, differences in backend organization and scheduling scope play a key role in explaining this apparent discrepancy. This observation provides additional confidence in the representativeness of our simulation framework and baseline configurations. Following the selection policy described above, the IQ sizes used in the remainder of the evaluation are as follows:

- For the BigO3 configuration, we select IQ sizes of 96, 48, and 24 entries, corresponding to approximately 99%, 93%, and 80% of the maximum observed IPC, respectively.
- For the SmallO3 configuration, we select IQ sizes of 48, 24, and 12 entries, corresponding to approximately 99%, 96%, and 81% of the maximum observed IPC, respectively.

Using these parametrized IQ sizes, the next section analyzes the performance loss incurred when transitioning from an idealized

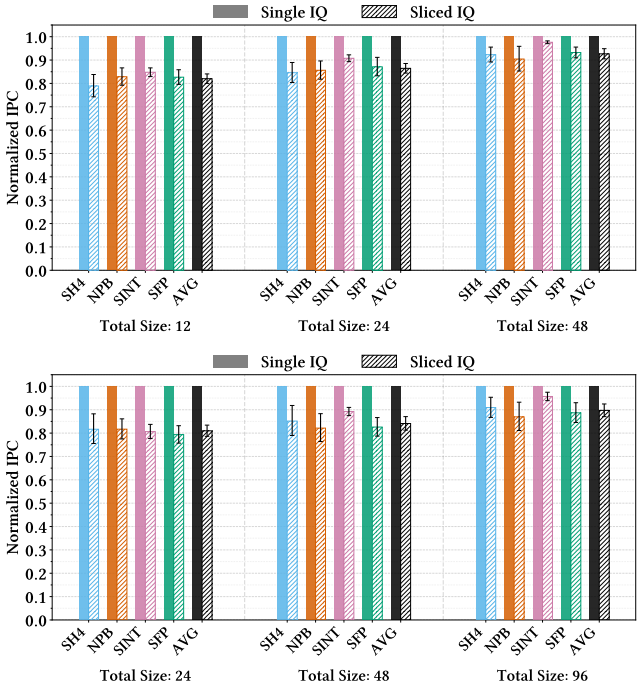


Figure 5: IPC degradation for distributed Issue Queue implementations. (up) SmallO3 configuration. (down) BigO3 configuration.

unified issue queue to a distributed scheduling model with one issue queue per execution port, corresponding to the Sliced-IQ + Distributed Ports organization described in Section 3.2.

4.2 Issue Queue Evaluation (Global vs. Sliced)

As discussed in Section 4.1, we use the global issue queue (Global-IQ) configuration as an upper bound on achievable performance, ignoring physical implications. The first experiment in this section aims to quantify the performance loss caused solely by partitioning the issue queue into independent segments, each associated with a backend execution port. Although practical designs may assign slightly different queue capacities depending on functional specialization, our exploratory analysis showed that such fine-grained tuning yields relatively minor improvements compared to the structural effects evaluated here. For this reason and in order to gain a better understanding of performance trends, we restrict our analysis to uniform partitions of the total issue queue capacity across all per-port queues. We repeat the processor configurations from the previous section and collect performance metrics for the three IQ sizes identified in Section 4.1. Figure 5 reports the results for the SmallO3 (up) and BigO3 (down) configurations. Each bar represents the average performance of a benchmark suite, and all results are normalized to the performance obtained with a global issue queue.

A consistent and expected trend emerges across all benchmarks and IQ sizes: partitioning the issue queue into per-port structures results in a measurable performance loss relative to the unified queue. For both processor configurations, this degradation follows a similar pattern, being more pronounced at smaller IQ sizes, where issue queue capacity plays a more critical role in sustaining performance. As the total IQ size increases, the performance gap between the global and distributed organizations gradually narrows, decreasing from approximately 20% for the smallest evaluated configuration (2 entries per queue) to around 10% for the largest one.

Interestingly, SPECint nearly matches the performance of the unified queue once the distributed queues reach sufficient capacity, for the same reason these applications saturate their IPC with smaller queue sizes.

As anticipated in the previous section, this performance loss provides insight into the discrepancy between the IQ saturation sizes observed in simulation and those implemented in commercial processors. One possible interpretation is that real-world designs compensate for the reduced scheduling scope of distributed queues by overprovisioning their capacity to recover part of the lost performance. To evaluate this hypothesis, we conducted an additional experiment to quantify the degree of overprovisioning required to close the gap. To reach the 99% peak IPC saturation point, the distributed queues require 18 entries per port for both the BigO3 and SmallO3 configurations. This represents a overprovisioning factor of 125% compared to the ideal unified baseline (with 96 total entries across 12 ports, and 48 total across 6 ports). Consequently, the resulting total backend capacities of our overprovisioned distributed models (216 in BigO3, 108 in SmallO3) closely align with the specifications of contemporary real-world processors. For instance, the SiFive P870 and XiangShan Kunminghu, feature highly comparable capacities of over 200 total entries (12-30 entries per port). Similarly, SiFive P550 allocates 104 total entries, similar to SmallO3.

The previous experiment shows that, when implementation costs are ignored, the broader scheduling scope of a global issue queue can deliver performance improvements of up to 10% when the issue queue is sized near the saturation point of the IPC versus queue-size curve. However, the scheduling logic is among the most timing-critical components of an out-of-order processor [27], and issue queue size is a primary factor affecting its latency. As a result, assuming similar scheduling costs (in terms of latency) for global and distributed issue queues is unlikely to be realistic.

At the same time, accurately estimating the implementation overhead of a global issue queue is inherently complex, as scheduling latency strongly depends on low-level design choices (low-level optimizations, place-and-route, technology constraints, etc.) which are beyond the scope of this work. Rather than attempting to predict a specific cost, we will next conduct a sensitivity analysis to explore how performance evolves as scheduling latency increases for the global issue queue. This allows us to identify how quickly its performance advantage erodes and to determine the latency ranges over which a unified scheduling organization remains beneficial.

In the following experiment, we therefore model progressively increasing scheduling latencies in the global issue queue. First, we assume identical scheduling latencies of one and two cycles for both the global and distributed configurations. Then, we incrementally add extra cycles to the global-queue scheduling pipeline only. This allows us to determine when its performance will converge to that of the distributed issue queue.

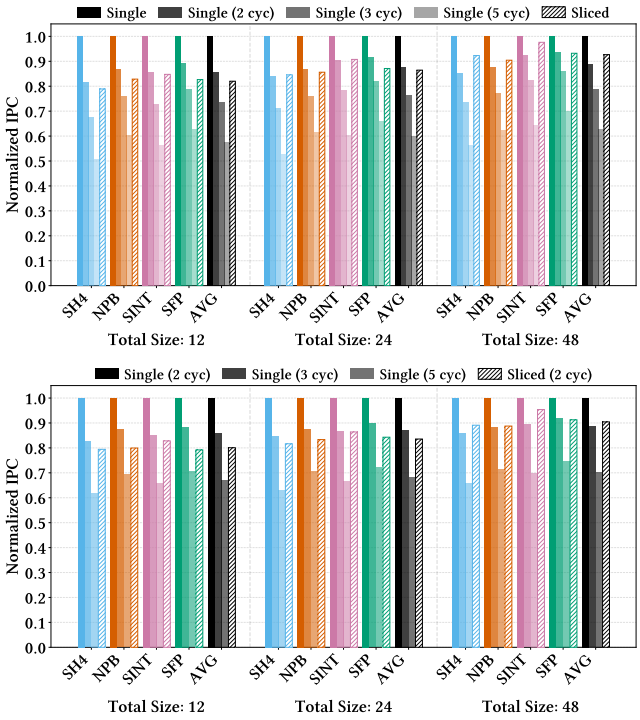


Figure 6: IPC degradation for global Issue Queue with increasing delays. SmallO3 configuration, (up) 1-cycle reference, (down) 2-cycle reference.

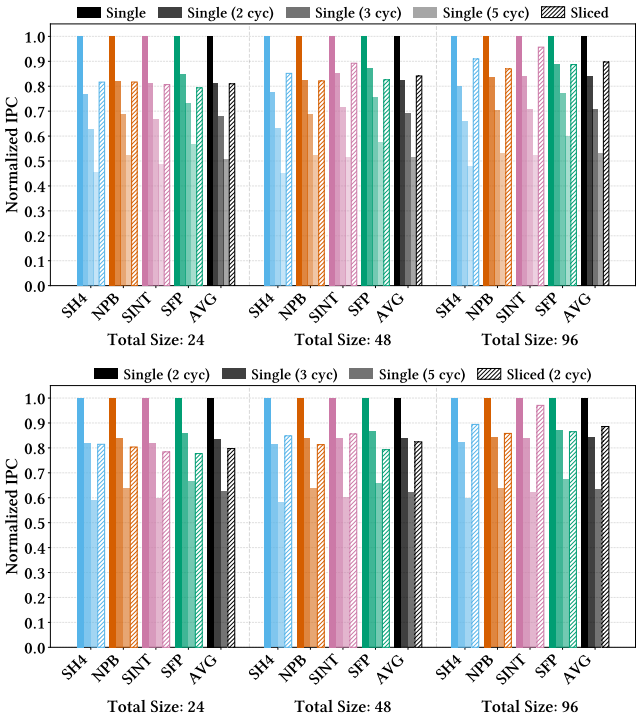


Figure 7: IPC degradation for global Issue Queue with increasing delays. BigO3 configuration, (up) 1-cycle reference, (down) 2-cycle reference.

Figures 6 and 7 report the results for the SmallO3 and BigO3 configurations, respectively. Performance is normalized to the best-performing design in each scenario, namely the global issue queue with minimum scheduling latency (1 cycle in the upper plots and 2 cycles in the lower plots). Starting with SmallO3, the results clearly illustrate the critical impact of scheduling latency on performance. Introducing a single additional cycle of scheduling delay in the global issue queue almost completely eliminates its advantage over the distributed design. For the smallest IQ size (12 entries), the broader scheduling scope of the unified queue still compensates partially for the added latency, preserving roughly a 5% performance gain. However, this benefit rapidly diminishes as the IQ size increases. At the intermediate size (24 entries), performance between both organizations becomes nearly indistinguishable, with the distributed queue even outperforming the unified configuration for certain workload suites, including Splash and SPECint. Finally, at 48 entries, this trend reversal becomes consistent across all benchmark suites, with the distributed organization achieving slightly higher performance than a global scheduler operating under a one-cycle latency penalty. As the assumed scheduling penalty continues to increase, performance degrades further, and the unified design quickly becomes non-competitive. Importantly, these trends are consistent regardless of the baseline scheduling latency assumed (1 or 2 cycles), indicating that the sensitivity to additional delay is intrinsic to the scheduling scope rather than an artifact of a particular starting point.

The impact is slightly more pronounced in BigO3, a wider and more aggressively provisioned core. In the smallest IQ sizes, the unified queue hardly exploits its theoretical advantage in scheduling scope once a single cycle of additional latency is introduced. From the lowest IQ size onward, the trend becomes more pronounced than the one observed in SmallO3. A one-cycle penalty already results in a slight average performance degradation for the unified queue at the intermediate IQ size, exceeding 5% in the largest queue configuration. Additional latency penalties render the unified scheduler clearly inferior. As in the SmallO3 case, these trends are largely independent of the initial scheduling latency assumption.

Taken together, these results clearly show that within the explored design space, a unified issue queue provides a consistent performance advantage of up to ~10% over distributed queues when the issue queue is sized near the saturation point and the implementation cost is assumed equal. However, this advantage is fragile, and even a minimal increase in scheduling latency largely offsets the gains from the expanded scheduling scope. This finding highlights a fundamental tradeoff in backend design. While global scheduling maximizes theoretical instruction-level parallelism, its practical benefit depends critically on maintaining tight timing in the wakeup–select loop. In design environments where low-level circuit optimization resources are limited, pursuing a unified scheduling organization may introduce significant implementation risk and provide an uncertain net performance benefit.

Since distributed queues emerge as a robust and competitive alternative under realistic timing assumptions, as architectural transitions like ARM Neoverse V1 to V2 [10] suggest, it is natural to explore whether alternative functional unit organizations can further improve the performance–complexity balance.

4.3 Execution Ports (Replication vs. Distribution)

This section extends previous analysis to sliced backend organizations with replicated functional units [17, 29, 34], examining whether they could offer a more favorable tradeoff between scheduling scope, latency sensitivity, and implementation cost. As in the previous section, we divide the design-space exploration in two stages. First, we evaluate performance without explicitly accounting for implementation implications of increasing the slices. This will allow to quantify how many replicated backend slices are required for a processor with a replicated organization to match the performance of a conventional heterogeneous backend. Second, we incorporate cost-oriented considerations to assess how performance evolves under different implementation-effort assumptions. Increasing the number of slices will have an impact on the backend area and the writeback and bypass networks. The final cost depends on low-level implementation decisions, constraints, and the steering policy. To account for complexity variations in the backend, we will analyze the sensitivity of performance to forwarding and writeback delays.

Figure 8 shows the results for the two processor configurations, normalized to the performance of the reference design (Sliced IQ + Distributed Ports). As can be seen, both SmallO3 and BigO3 exhibit similar qualitative trends. For the smallest IQ sizes, where the distributed organization results in very shallow per-queue availability, the replicated configuration reaches the reference performance level with a relatively small number of slices. This behavior is likely explained by two factors: (i) functional-unit replication reduces contention, and (ii) with fewer slices than ports, each slice retains a deeper local issue queue (for a given aggregate storage), increasing effective scheduling capacity per slice. Beyond this critical region, as total IQ capacity increases the advantage associated with deeper per-slice queues diminishes, and a pattern arises: the replicated backend outperforms the distributed baseline when the number of slices matches the frontend width (i.e., the fetch/decode width). For the configurations evaluated, this corresponds to three slices in SmallO3 and six slices in BigO3, both of which achieve performance levels 5-10% higher than the reference organization. It is important to notice that these values are smaller than the Retiring width of the reference backend which is 4 and 8 for SmallO3 and BigO3 respectively, and substantially smaller than the number of execution ports (6 and 12 respectively). A possible explanation is that conventional distributed backends provision a large number of execution ports to minimize structural hazards and avoid making the execution stage the performance bottleneck. In contrast, full functional-unit replication inherently suppress contention within the slice. Once the number of slices matches the instruction supply rate from the frontend, the backend can sustain a comparable throughput without requiring the higher port counts and retiring width typically used in distributed organizations. Finally, as expected, when the

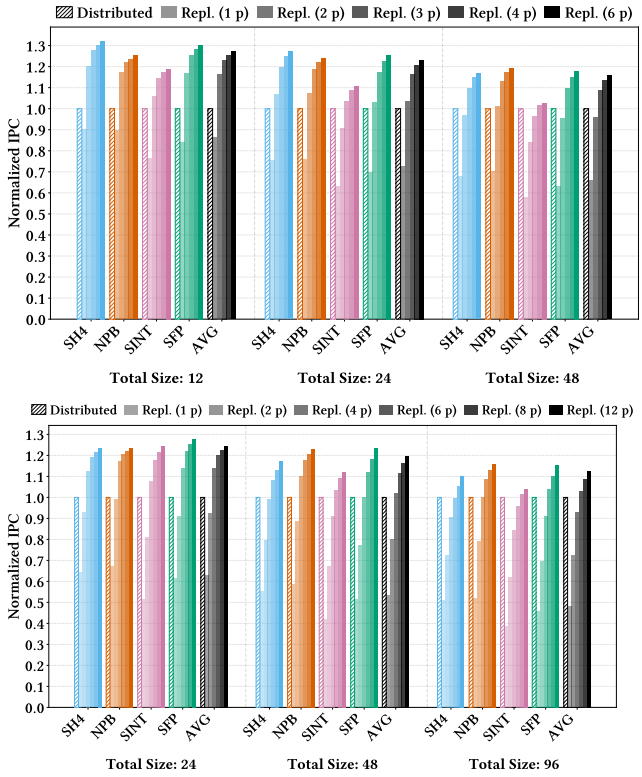


Figure 8: IPC evolution as the number of replicated backend slices grows. SmallO3 configuration (up) and BigO3 configuration (down).

number of replicated slices equals the number of ports in the distributed baseline, the Sliced IQ + Replicated Ports configuration benefits from strictly greater functional-unit availability. This leads to performance improvements in the 15-25% range, attributable to reduced contention for execution resources.

As in the previous section, implementation cost must be considered when interpreting these results. Unlike the global-versus-distributed issue queue comparison where additional cost is clearly associated with the unified scheduler, the cost balance here is less clear. On one hand, the Sliced IQ + Replicated Ports organization incurs significant area overhead due to the replication of backend structures. On the other hand, matching the reference performance often requires fewer slices than ports, potentially reducing the size and complexity of writeback and bypass networks, whose cost typically grows quadratically with the number of ports. Because precise cost estimation would require a detailed physical implementation, we again adopt a design-space exploration approach, evaluating the performance impact under different cost assumptions for the replicated backend organization.

For the final step in the evaluation process, we conduct the following experiment. In the Sliced IQ + Replicated Ports configuration, we consider three execution widths:

- Number of slices equal to the decode/rename width (see Table 1: six for BigO3 and three for SmallO3).

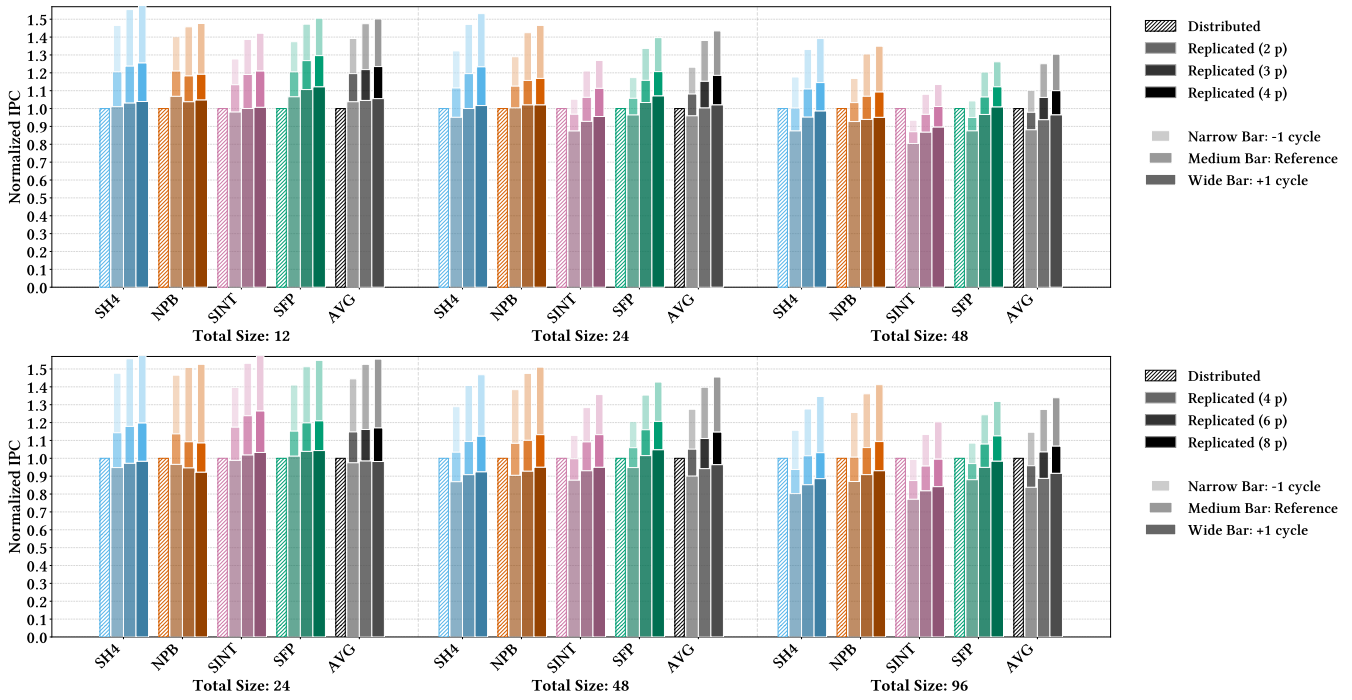


Figure 9: IPC evolution with variable writeback delays for Sliced IQ + Replicated Ports. SmallO3 configuration (up) and BigO3 configuration (down).

- The immediately smaller configuration: four slices in BigO3 and two slices in SmallO3.
- Number of slices equivalent to the retire width (see Table 1: eight for BigO3 and four for SmallO3).

For these setups, we analyze performance evolution under two opposite latency assumptions:

(1) The writeback and forwarding delay is higher in the replicated backend (assuming slice area dominates latency).
(2) The writeback and forwarding delay is lower in the replicated backend (assuming bypass and forwarding network complexity dominates latency, which is higher in the distributed organization due to its larger number of ports).

To model these scenarios, we vary the writeback and forwarding latency of the Sliced IQ + Replicated Ports configuration by ±1 cycle, while keeping the distributed backend fixed. This represents a conservative, worst-case approximation, as the added latency is applied uniformly to all instructions, whereas in practice only a subset would be affected.

The results are shown in Figure 9 and are normalized to the conventional distributed backend. For the Sliced IQ + Replicated Ports configuration, results are shown using three overlaid bars of different widths: the widest bar corresponds to the implementation with one additional writeback cycle, the medium-width bar to the implementation with the same writeback latency, and the narrowest bar to the implementation with one fewer writeback cycle. A first observation is that the absolute performance sensitivity to writeback latency is similar to that previously observed when penalizing scheduling latency. This is expected, as in both cases dependency resolution is effectively delayed by one cycle, either at execution start (scheduling) or upon result broadcast (writeback).

We analyze the results separately for each of the evaluated slice configurations. Starting with the smallest number of slices (i.e., below the fetch/decode width), it is noteworthy that the replicated backend remains competitive as long as its writeback latency does not exceed that of the conventional organization. Furthermore, if the replicated configuration achieves a lower writeback latency, this reduced implementation can deliver higher performance even at larger IQ sizes, where the sliced organization is otherwise less competitive. These observations suggest that execution-resource replication alters the backend scaling dynamics more fundamentally than a simple comparison based solely on execution port count would indicate.

For the configuration where the number of slices matches the frontend width, Figure 8 already showed that the replicated backend achieves comparable performance to the baseline when writeback latency is equal. As expected, if writeback can be completed one cycle earlier, the replicated organization attains substantial gains, in the 25–50% range. Although this represents an optimistic bound, it underscores the strong quadratic impact of port scaling on backend critical paths. The more informative case is the opposite assumption, where the replicated backend incurs an additional one-cycle delay due to the increased slice area. At small IQ sizes, the replicated configuration remains competitive: the combination of deeper per-slice queues and reduced functional-unit contention enables it to match the distributed baseline despite the added latency. As total IQ capacity increases and the queue-depth advantage diminishes,

the performance of the replicated design correspondingly degrades. In the SmallO3 configuration, the performance loss remains below 5% even at the largest IQ size. The impact is more pronounced for BigO3, where the degradation reaches approximately 5% at the intermediate queue size and slightly larger than 10% in the largest configuration.

Finally, when the number of replicated slices equals the retire width of the distributed backend (4 ports in SmallO3 and 8 ports in BigO3), the performance of the Sliced IQ + Replicated design remains competitive even when considering the higher writeback latency, with average performance losses remaining below 5% for the SmallO3 configuration and 10% for BigO3, including for the largest IQ configurations.

Taken together, these results indicate that a Sliced IQ + Replicated Ports organization can constitute a viable and competitive alternative to the conventional distributed backend, even when moderate implementation overhead is assumed. While replication could increase structural area, it reshapes critical-path behavior and contention patterns in ways that may offset part of that cost. Beyond raw performance considerations, replicated backends offer additional architectural advantages, including structural regularity and potential simplification of scheduling logic. In the context of evolving RISC-V microarchitectures seeking scalable high-performance designs, backend replication could therefore be regarded as a credible path for continued performance scaling.

5 Conclusions

This work analyzed the impact of scheduling scope and execution-resource organization on performance scalability in wide superscalar out-of-order backends. Motivated by the contrast between unified scheduling structures and the distributed organizations increasingly adopted in high-performance RISC-V cores, we explored a design space spanning monolithic, distributed, and fully replicated sliced backends.

Our results show that while unified issue queues can provide a performance advantage under equal-latency assumptions, this benefit is highly sensitive to additional wakeup–select delay. Even modest timing penalties largely offset the gains of expanded scheduling scope, highlighting the practical limitations of aggressively centralized designs. In contrast, replicated sliced backends exhibit a more balanced tradeoff between resource availability and latency sensitivity. By reducing execution contention and maintaining shorter critical paths, replication can match, or be close to, the performance of wider distributed designs, even when moderate writeback penalties are assumed.

Overall, the results suggest that backend replication is a viable scaling strategy, especially if timing and physical constraints make large centralized schedulers unfeasible.

Acknowledgments

We would like to thank José Ángel Gregorio for his valuable comments and suggestions. This work was supported by the Spanish Government (Ministerio de Ciencia, Innovación y Universidades / Agencia Estatal de Investigación) under grant PID2022-139664NB-I00.

References

[1] 2017. SPEC CPU 2017. https://www.spec.org/
[2] 2023. P870 High-Performance RISC-V Processor. In *2023 IEEE Hot Chips 35 Symposium, HCS 2023*. doi:10.1109/HCS59251.2023.10254712
[3] Jaume Abella, Ramon Canal, and Antonio González. 2003. Power- and Complexity-Aware Issue Queue Designs. *IEEE Micro* 23 (2003). Issue 5. doi:10.1109/MM.2003.1240212
[4] Esther Alonso, Pablo Abad, Pablo Prieto, and Valentin Puente. 2026. *Source code for Design Tradeoffs in Backend Organization*. https://github.com/ceunican/gem5-riscv-backend-tradeoffs
[5] Hideki Ando. 2018. Performance improvement by prioritizing the issue of the instructions in unconfident branch slices. In *Proceedings of the Annual International Symposium on Microarchitecture, MICRO*, Vol. 2018-October. doi:10.1109/MICRO.2018.00016
[6] Hideki Ando. 2022. Segmenting Age Matrices to Improve Instruction Scheduling without Increasing Delay and Area. In *Proceedings - IEEE International Conference on Computer Design: VLSI in Computers and Processors*, Vol. 2022-October. doi:10.1109/ICCD56317.2022.00059
[7] Nathan L. Binkert, Bradford M. Beckmann, Gabriel Black, et al. 2011. The gem5 simulator. *SIGARCH Comput. Archit. News* 39, 2 (2011), 1–7. doi:10.1145/2024716.2024718
[8] Nadav Bonen, Arik Gihon, Leon Polishuk, Yoni Aizik, Yulia Okunev, Tsvika Kurts, and Nithiyanandan Bashyam. 2025. Lunar Lake an Intel Mobile Processor: SoC Architecture Overview (2024). *IEEE Micro* 45 (2025). Issue 3. doi:10.1109/MM.2025.3558407
[9] Mary D. Brown, Jared Stark, and Yale N. Patt. 2001. Select-free instruction scheduling logic. In *Proceedings of the Annual International Symposium on Microarchitecture*. doi:10.1109/micro.2001.991119
[10] Magnus Bruce. 2023. Arm Neoverse V2 platform: Leadership Performance and Power Efficiency for Next-Generation Cloud Computing, ML and HPC Workloads. In *2023 IEEE Hot Chips 35 Symposium, HCS 2023*. doi:10.1109/HCS59251.2023.10254718
[11] Ramon Canal, Joan Manuel Parcerisa, and Antonio Gonzalez. 1999. Cost-effective clustered architecture. *Parallel Architectures and Compilation Techniques - Conference Proceedings, PACT* (1999). doi:10.1109/pact.1999.807517
[12] Ramon Canal, Joan Manuel Parcerisa, and Antonio González. 2000. Dynamic cluster assignment mechanisms. In *IEEE High-Performance Computer Architecture Symposium Proceedings*. doi:10.1109/HPCA.2000.824345
[13] Chen Chen, Xiaoyan Xiang, Chang Liu, Yunhai Shang, Ren Guo, Dongqi Liu, Yimin Lu, Ziyi Hao, Jiahui Luo, Zhijian Chen, Chunqiang Li, Yu Pu, Jianyi Meng, Xiaolang Yan, Yuan Xie, and Xiaoning Qi. 2020. Xuantie-910: A Commercial Multi-Core 12-Stage Pipeline Out-of-Order 64-bit High Performance RISC-V Processor with Vector Extension : Industrial Product. In *Proceedings - International Symposium on Computer Architecture*, Vol. 2020-May. doi:10.1109/ISCA45697.2020.00016
[14] Chandana S. Deshpande, Arthur Perais, and Frédéric Pétrot. 2025. Address/Data Instruction Steering in Clustered General Purpose Processors. *ACM Transactions on Architecture and Code Optimization* 22 (2025). Issue 3. doi:10.1145/3744908
[15] S. Dropsho, A. Buyuktosunoglu, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas, G. Semeraro, G. Magklis, and M. L. Scottt. 2002. Integrating adaptive on-chip storage structures for reduced dynamic power. In *Parallel Architectures and Compilation Techniques - Conference Proceedings, PACT*, Vol. 2002-January. doi:10.1109/PACT.2002.1106013
[16] D. Folegnani and A. González. 2001. Energy-effective issue logic. In *Conference Proceedings - Annual International Symposium on Computer Architecture, ISCA*. doi:10.1145/379240.379266
[17] Ty Garibay and Shashank Nemawarkar. 2025. Cuzco: A High-Performance RISC-V RVA23 Compatible CPU IP. doi:10.1109/hcs66204.2025.11154418
[18] Wilfred Gomes, Slade Morgan, Boyd Phelps, Tim Wilson, and Erik Hallnor. 2022. Meteor Lake and Arrow Lake Intel Next-Gen 3D Client Architecture Platform with Foveros. In *2022 IEEE Hot Chips 34 Symposium, HCS 2022*. doi:10.1109/HCS55958.2022.9895532
[19] Eduardo Jose Gomez-Hernandez, Juan M. Cebrian, Stefanos Kaxiras, and Alberto Ros. 2022. Splash-4: A Modern Benchmark Suite with Lock-Free Constructs. In *Proceedings - 2022 IEEE International Symposium on Workload Characterization, IISWC 2022*. doi:10.1109/IISWC55918.2022.00015
[20] Masahiro Goshima, Kengo Nishino, Yasuhiko Nakashima, Shin Ichiro Mori, Toshiaki Kitamura, and Shinji Tomita. 2001. A high-speed dynamic instruction scheduling scheme for superscalar processors. In *Proceedings of the Annual International Symposium on Microarchitecture*. doi:10.1109/micro.2001.991121
[21] Ipoom Jeong, Jiwon Lee, Myung Kuk Yoon, and Won Woo Ro. 2022. Reconstructing Out-of-Order Issue Queue. In *Proceedings of the Annual International Symposium on Microarchitecture, MICRO*, Vol. 2022-October. doi:10.1109/MICRO56248.2022.00023

[22] H Jin, M Frumkin, and J Yan. 1999. The OpenMP implementation of NAS parallel benchmarks and its performance. *National Aeronautics and Space Administration (NASA), Technical Report NAS-99-011, Moffett Field, USA* (1999). Issue October. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.65.1321&rep=rep1&type=pdf

[23] Yunsup Lee, Andrew Waterman, Henry Cook, Brian Zimmer, Ben Keller, Alberto Puggelli, Jaehwa Kwak, Ruzica Jevtic, Stevo Bailey, Milovan Blagojevic, Pi Feng Chiu, Rimas Avizienis, Brian Richards, Jonathan Bachrach, David Patterson, Elad Alon, Bora Nikolic, and Krste Asanovic. 2016. An agile approach to building RISC-V microprocessors. *IEEE Micro* 36 (2016). Issue 2. doi:10.1109/MM.2016.11

[24] Jason Lowe-Power, Abdul Mutaal Ahmad, Ayaz Akram, et al. 2020. The gem5 Simulator: Version 20.0+. *CoRR* abs/2007.03152 (2020). arXiv:2007.03152 https://arxiv.org/abs/2007.03152

[25] Tali Moreshet and R. Iris Bahar. 2004. Effects of speculation on performance and issue queue design. In *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 12. Issue 10. doi:10.1109/TVLSI.2004.834226

[26] Kenichiro Mori, Sota Kosugi, Hiroto Yoshida, Hajime Shimada, and Hideki Ando. 2024. Localizing the Tag Comparisons in the Wakeup Logic to Reduce Energy Consumption of the Issue Queue. In *Proceedings of the Annual International Symposium on Microarchitecture, MICRO*. doi:10.1109/MICRO61859.2024.00044

[27] Subbarao Palacharla, Norman P. Jouppi, and J. E. Smith. 1997. Complexity-effective superscalar processors. In *Conference Proceedings - Annual International Symposium on Computer Architecture, ISCA*. doi:10.1145/264107.264201

[28] Pablo Prieto, Pablo Abad, Jose Angel Gregorio, and Valentin Puente. 2023. Performance Characterization of Popular DNN Models on Out-of-Order CPUs. In *Parallel Architectures and Compilation Techniques - Conference Proceedings, PACT*. doi:10.1109/PACT58117.2023.00025

[29] Satish Kumar Sadasivam, Brian W. Thompto, Ron Kalla, and William J. Starke. 2017. IBM Power9 Processor Architecture. *IEEE Micro* 37 (2017). Issue 2. doi:10.1109/MM.2017.40

[30] Shinji Sakai, Taishi Suenaga, Ryota Shioya, and Hideki Ando. 2019. Rearranging Random Issue Queue with High IPC and Short Delay. In *Proceedings - 2018 IEEE 36th International Conference on Computer Design, ICCD 2018*. doi:10.1109/ICCD.2018.00027

[31] André Seznec. 2016. TAGE-SC-L branch predictors Again. *JWAC-4: Championship Branch Prediction* (2016).

[32] SiFive. [n. d.]. SiFive Hifive Premier P550 development board (2025).

[33] Teja Singh, Spence Oliver, Sundar Rangarajan, Shane Southard, Carson Henrion, Alex Schaefer, Brett Johnson, Sarah Bartaszewicz Tower, Kathy Hoover, Deepesh John, Ted Antoniadis, Shravan Lakshman, Vibhor Mittal, Brian Kasprzyk, Ross McCoy, Kurt Mohlman, Anitha Mohan, Hon Hin Wong, Daryl Lieu, Russell Schreiber, Sahilpreet Singh, Nick Lance, Darryl Prudich, Justin Coppin, Tim Jackson, Anita Karegar, Ryan Miller, Sabeesh Balagangadharan, James Pistole, Wilson Li, and Michael McCabe. 2025. 'Zen 5': The AMD High-Performance 4nm x86-64 Microprocessor Core. In *Digest of Technical Papers - IEEE International Solid-State Circuits Conference*. doi:10.1109/ISSCC49661.2025.10904529

[34] William J. Starke, Brian W. Thompto, Jeff A. Stuecheli, and Jose E. Moreira. 2021. IBM's POWER10 Processor. *IEEE Micro* 41 (2021). Issue 2. doi:10.1109/MM.2021.3058632

[35] David Suggs, Mahesh Subramony, and Dan Bouvier. 2020. The AMD 'Zen 2' processor. *IEEE Micro* 40 (2020). Issue 2. doi:10.1109/MM.2020.2974217

[36] Andrew Waterman, Yunsup Lee, David Patterson, Krste Asanovic, Volume I User level Isa, Andrew Waterman, Yunsup Lee, and David Patterson. 2014. The RISC-V instruction set manual. *Volume I: User-Level ISA', version* 2 (2014).

[37] Henry Wong, Vaughn Betz, and Jonathan Rose. 2016. High Performance Instruction Scheduling Circuits for Out-of-Order Soft Processors. In *Proceedings - 24th IEEE International Symposium on Field-Programmable Custom Computing Machines, FCCM 2016*. doi:10.1109/FCCM.2016.11

[38] Yinan Xu, Zihao Yu, Dan Tang, Guokai Chen, Lu Chen, Lingrui Gou, Yue Jin, Qianruo Li, Xin Li, Zuojun Li, Jiawei Lin, Tong Liu, Zhigang Liu, Jiazhan Tan, Huaqiang Wang, Huizhe Wang, Kaifan Wang, Chuanqi Zhang, Fawang Zhang, Linjuan Zhang, Zifei Zhang, Yangyang Zhao, Yaoyang Zhou, Yike Zhou, Jiangrui Zou, Ye Cai, Dandan Huan, Zusong Li, Jiye Zhao, Zihao Chen, Wei He, Qiyuan Quan, Xingwu Liu, Sa Wang, Kan Shi, Ninghui Sun, and Yungang Bao. 2022. Towards Developing High Performance RISC-V Processors Using Agile Methodology. In *Proceedings of the Annual International Symposium on Microarchitecture, MICRO*, Vol. 2022-October. doi:10.1109/MICRO56248.2022.00080