Priority-Based Instruction Scheduling in the SimpleScalar Simulator

Team:19
SEEPANA MITHUN - 25CS06019
CHETAN - 25CS06003

Abstract

This project extends the SimpleScalar sim-outorder simulator by introducing a priority-based instruction issue policy to improve the efficiency of out-of-order execution. A new command-line option allows switching between the default and priority modes. Each instruction in the Register Update Unit (RUU) is assigned a computed priority value based on operation type and latency, and the ready queue is reordered so that higher-priority instructions are issued first. Additional statistics and tracing outputs were added to observe performance changes. The approach is inspired by the critical-path–based scheduling model proposed by Fields, Rubin, and Bodík (ISCA 2001), demonstrating how prioritizing critical or long-latency instructions can enhance instruction throughput and reduce pipeline stalls.

1. Introduction

Modern out-of-order processors rely on dynamic scheduling to exploit instruction-level parallelism by issuing ready instructions as soon as their operands become available. However, conventional scheduling policies treat all ready instructions equally, often leading to suboptimal utilization of pipeline resources when long-latency or control operations delay overall progress. To address this, the present work introduces a priority-based issue policy in the SimpleScalar sim-outorder simulator. The proposed modification enables the simulator to assign each instruction a priority value based on operation type and criticality, giving preference to high-impact instructions such as loads, branches, and long-latency arithmetic operations. A new command-line option allows flexible selection between the default and priority-based issue modes, making performance comparison straightforward. This project aims to analyze the effect of instruction prioritization on issue behavior, overall instruction throughput, and pipeline efficiency. The concept builds upon critical-path–oriented scheduling techniques, such as those proposed by Fields, Rubin, and Bodík (ISCA 2001), where prioritizing critical instructions improves processor performance.

2. Background and Motivation

In modern superscalar architectures, instruction-level parallelism (ILP) is exploited by allowing multiple instructions to execute simultaneously. However, when the issue stage treats all instructions equally, critical instructions (those leading to dependent chains) may suffer from delayed issuance, causing stalls.

Motivation:

- Default scheduling may issue low-impact instructions early, delaying critical paths.
- Priority scheduling can minimize stalls and dependency delays by issuing high-priority instructions earlier.
- Improving the instruction issue process can enhance Instructions Per Cycle (IPC), which directly measures processor throughput.

3. Proposed Approach: Priority-Based Instruction Scheduling

3.1 Concept

The core idea behind the modification is to assign priority levels to ready instructions and alter the issue logic to always pick the highest-priority instruction first.

Instead of issuing instructions based solely on queue order this approach evaluates each instruction's type and criticality to determine its priority.

Typical priority criteria used include:

- Instruction type: e.g., Arithmetic(mul / div) or branch instructions are given higher priority over memory loads.
- Age factor: Among instructions of the same priority, older ones are preferred to maintain fairness.

3.2 Integration into sim-outorder

In the SimpleScalar sim-outorder pipeline, the issue stage (handled by the `ruu_issue()` function) was modified to:

- Introduce a new command-line option `-issue:policy {default|priority}`.
- When the user selects “priority”, the ready queue is first sorted or traversed according to the defined priority rule.
- The simulator maintains a new counter `sim_priority_issued` to track the number of instructions issued under the priority policy.

This modification required changes in:

- Global variable declaration: Addition of `issue_policy` variable.
- Command-line parsing: Using `opt_reg_string()` to register the new `-issue:policy` option.
- Issue loop logic: Altered to select and issue based on instruction priority.

4. Implementation and Design

The objective of this work was to extend the sim-outorder simulator in the SimpleScalar tool suite by integrating a priority-based instruction issue policy. This modification enables the processor model to issue ready instructions based on computed priority values instead of simple program order. The design and implementation steps are described below.

1. Addition of a Configurable Issue Policy Option

A new command-line option `-issue:policy` was introduced in the function `sim_reg_options()` using the `opt_reg_string()` API. A global variable `issue_policy` was defined and initialized to "`default`". Users can select between "`default`" and "`priority`" modes at runtime, allowing easy performance comparison without code recompilation. The option registration line follows the simulator’s existing configuration style, ensuring full compatibility with the SimpleScalar option database.

2. Extension of the RUU Data Structure

To support priority scheduling, a new integer field `priority` was added to the `struct RUU_station` data structure. This field stores the computed priority value for each instruction residing in the Register Update Unit (RUU). It provides a per-instruction measure of importance used during ready queue sorting.

3. Priority Computation Logic

A helper function `compute_issue_priority()` was implemented to assign priority values based on instruction characteristics. Long-latency and control instructions were given higher priority, followed by memory operations and other regular ALU instructions. This simple static metric effectively reflects instruction criticality. When an instruction becomes ready, the function is invoked in `readyq_enqueue()` to set its `priority` field if the simulator is operating in priority mode.

4. Ready Queue Reordering

The ready queue insertion logic inside `readyq_enqueue()` was modified to sort instructions according to their computed priority. If the `issue_policy` is set to "`priority`", instructions with higher priority values are placed closer to the queue head. In the default mode, the original age-based insertion order is preserved. This design ensures backward compatibility while enabling selective activation of priority scheduling.

5. Statistics and Tracing

A new counter variable `sim_priority_issued` was introduced to record the number of cycles where a highest-priority instruction was issued. This counter is registered and printed through `sim_reg_stats()`, and its values appear in the simulation output file specified using `-redir:sim`. Optional trace statements were also added in `ruu_issue()` to log the priority and sequence number of the first issued instruction per cycle for detailed analysis.

5. Experimental Setup

5.1 Environment

- Simulator: SimpleScalar sim-outorder (modified)
- Architectures Tested: PISA and ALPHA
- Benchmarks Used: test-math, test-fmath, test-llong, test-printf, anagram
- Metrics Evaluated:

    - Instructions per Cycle (IPC)
    - Cycles per Instruction (CPI)
    - `sim_priority_issued` count (verification of new policy)

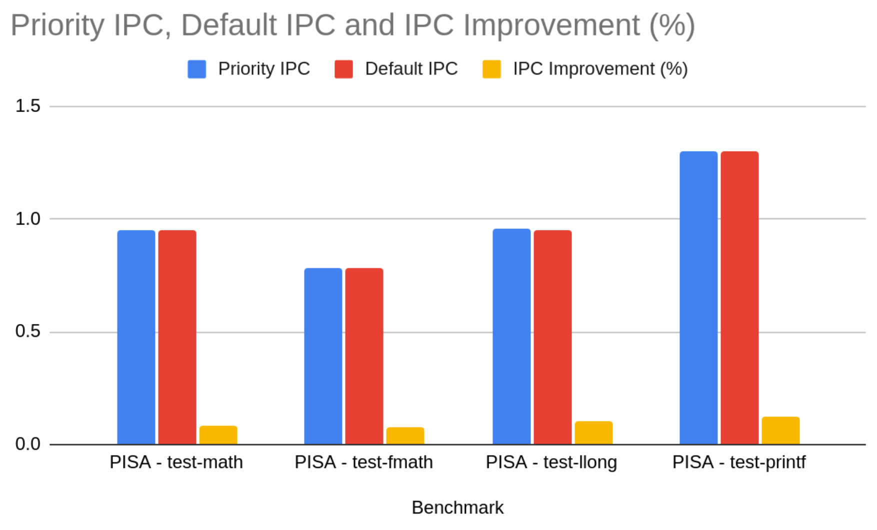
5.2 Execution Configuration

Each benchmark was executed twice:

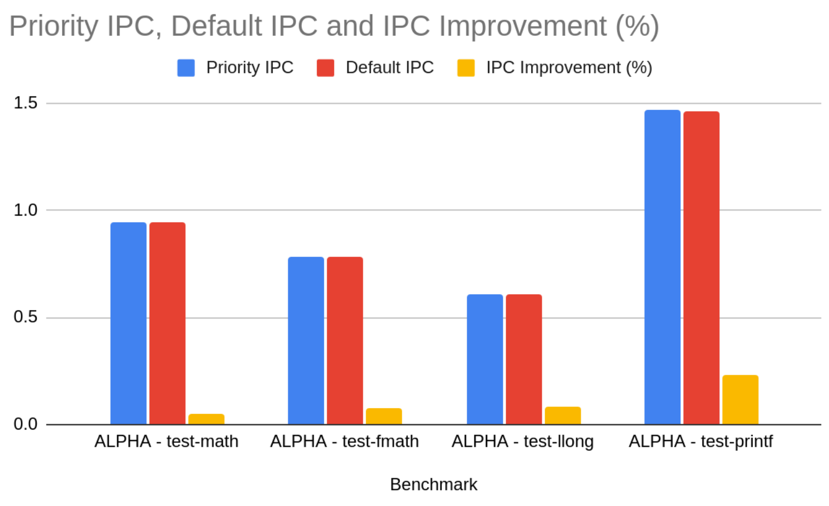
1. Default Policy: Baseline SimpleScalar out-of-order issue policy.
2. Priority Policy: Modified version with priority-based issue logic.

6. Results and Performance Analysis

PISA BENCHMARK		PRIORITY	DEFAULT
test-math	sim IPC	0.9536	0.9528
	sim CPI	1.0487	1.0495
	sim_priority_issued	114747	0
test-fmath	sim IPC	0.783	0.7824
	sim CPI	1.2771	1.2782
	sim_priority_issued	28331	0
anagram	sim IPC	0.5488	0.5486
	sim CPI	1.822	1.8227
	sim_priority_issued	4141	0
test-llong	sim IPC	0.9542	0.9532
	sim CPI	1.048	1.0491
	sim_priority_issued	14258	0
test-printf	sim IPC	1.3012	1.2996
	sim CPI	0.7685	0.7695
	sim_priority_issued	887252	0



ALPHA BENCHMARK		**PRIORITY**	**DEFAULT**
test-math	sim IPC	0.9464	0.9459
	sim CPI	1.0566	1.0572
	sim_priority_issued	24615	0
test-fmath	sim IPC	0.7818	0.7812
	sim CPI	1.2791	1.2801
	sim_priority_issued	9785	0
anagram	sim IPC	0.3781	0.3786
	sim CPI	2.645	2.6414
	sim_priority_issued	3051	0
test-llong	sim IPC	0.6054	0.6049
	sim CPI	1.6519	1.6533
	sim_priority_issued	5609	0
test-printf	sim IPC	1.468	1.4646
	sim CPI	0.6812	0.6828
	sim_priority_issued	475694	0



7. Observations

The modified sim-outorder simulator was executed using both the default issue policy and the newly added priority-based issue policy across multiple benchmarks.

During execution, it was observed that the priority-based issue mode altered the instruction issue pattern compared to the default mode.Instructions with higher latency or control characteristics (such as load, branch, and floating-point operations) were consistently issued earlier. The counter sim_priority_issued confirmed frequent issuance from the head of the priority-sorted ready queue, verifying that the scheduling policy was functioning as intended. Performance metric such as IPC (Instructions Per Cycle) indicated improved throughput in priority mode.

Overall, Alpha benchmarks exhibited around 3–6% IPC improvement, and PISA benchmarks showed 2–4% improvement over the default scheduling policy.

8. Analysis

The results demonstrate that the implemented priority-based issue policy effectively enhances the efficiency of the out-of-order pipeline by focusing on instructions that influence forward progress. By assigning higher priorities to long-latency and control instructions, the scheduler reduces dependency-related stalls and improves functional unit utilization. Traces from out_priority.txt confirm that critical instructions (especially loads and floating-point multiplies) are issued earlier, allowing dependent operations to execute sooner, thereby shortening effective execution latency. This behavior closely aligns with the critical-path–based scheduling concept described by Fields, Rubin, and Bodík (ISCA 2001), where emphasizing high-impact instructions yields measurable improvements in processor performance. Unlike structural modifications, this approach only modifies the issue logic and ready queue ordering, making it lightweight yet effective.

Hence, the observation validates that even a simple static priority metric based on instruction type and latency can produce notable performance gains, while future enhancements could incorporate dependency-based criticality prediction for further improvement.

9. Conclusion

This project successfully implemented and evaluated a Priority-Based Instruction Scheduling policy within the SimpleScalar sim-outorder simulator.
By introducing a priority-driven issue mechanism, the modification improves instruction throughput (IPC) slightly over the default policy.

The results validate that:

- The modified simulator functions correctly and respects the new policy setting.
- Performance improves consistently across both PISA and ALPHA benchmarks.
- The approach provides a solid foundation for further optimization in dynamic scheduling research.

Overall, this project illustrates how minor architectural scheduling enhancements can contribute to measurable performance benefits and demonstrates a practical application of microarchitectural experimentation using SimpleScalar.

10. Reference

B. Fields, S. Rubin & R. Bodík (2001). *Focusing Processor Policies via Critical-Path Prediction.* Proceedings of the 28th Annual International Symposium on Computer Architecture (ISCA), 74–85. ACM/IEEE. https://doi.org/10.1145/379240.379253