NOREBA: A Compiler-Informed Non-Speculative Out-of-Order Commit Processor

Extended Abstract

Ali Hajiabadi* Andreas Diavastos† Trevor E. Carlson*

*National University of Singapore †Universitat Politècnica de Catalunya

1. Motivation

Modern superscalar processors execute instructions out-of-order, but commit them in program order to provide precise exception handling and safe instruction retirement. However, in-order instruction commit is highly conservative and holds on to critical resources far longer than necessary, severely limiting the reach of general-purpose processors, ultimately reducing performance. Solutions that allow for efficient, early reclamation of these critical resources could seize the opportunity to improve performance. One such solution is out-of-order commit, which has traditionally been challenging due to inefficient, complex hardware used to guarantee safe instruction retirement and provide precise exception handling.

In this work, we present NOREBA, a processor for **N**on-speculative **O**ut-of-order **Re**tirement via **B**ranch Reconvergence **A**nalysis. NOREBA enables non-speculative out-of-order commit and resource reclamation in a light-weight manner, improving performance and efficiency. We accomplish this through a combination of (1) automatic compiler annotation of true branch dependencies, and (2) an efficient re-design of the reorder buffer from traditional processors. By exploiting compiler branch dependency information, this system achieves 95% of the performance of aggressive, speculative solutions, without any additional speculation, and while maintaining energy efficiency.

2. Limitations of the State of the Art

Prior OoO-commit proposals tend to fall into two categories: (1) speculative [1, 4, 5, 11], and (2) non-speculative designs [2, 6, 10]. Processor designs using speculative OoO-commit typically require expensive checkpoint-and-restart mechanisms in case the speculation fails. Implementing these structures in a speculative OoO-commit processor can be expensive, reducing energy efficiency, especially for power-constrained systems [4]. On the other hand, non-speculative OoO-commit designs are more efficient as they do not require a checkpoint-and-restart mechanism. However, all prior implementations must resolve all preceding branches and memory accesses before committing out-of-order which prevents them from achieving the full potential of OoO-commit.

3. Key Insights

In-order commit of instructions and using FIFO-style reorder buffers guarantees safe instruction retirement. However, this solution requires that instructions wait for *all* preceding branches to resolve in order to release critical resources and leaves a significant amount of performance on the table. The key insight of this work is that many of these branch dependencies are artificial. Dependency analysis shows that some instructions do not always depend on the most recent branch in the ROB. The main idea of this work is to use the compiler to detect true branch dependencies of instructions and pass this information to the hardware. The benefit of using the compiler is that we can find the dependencies non-speculatively. By informing the hardware about the true non-speculative branch dependencies, the processor can commit instructions that are independent of unresolved branches without the need for expensive checkpoint-and-restart mechanisms. Precise exception handling can be enabled with proper communication between the processor and the OS by exposing the changes of recently OoO-committed instructions.

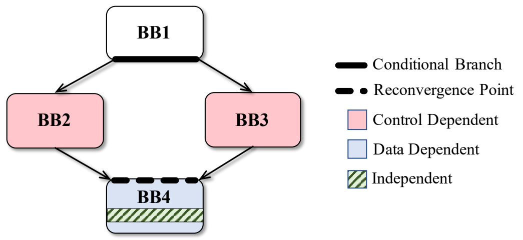

Figure 1: A simple if-then-else structure represented by basic blocks marked by branch dependent code detection pass.

4. Main Artifacts

The three main components of NOREBA are: (1) a branch dependent code detection compiler pass, (2) the OoO-commit microarchitecture, and (3) precise exception handling.

4.1. Branch Dependent Code Detection Pass

We use a compiler pass to detect branch dependencies. This pass first detects the reconvergence point for each branch. All instructions between a branch and its reconvergence point are control dependent. We then use the def-use chains of values and alias analysis to find the instructions after the reconvergence point that are data dependent to the branch and its control dependent instructions. All remaining instructions after the reconvergence point are independent of the branch (See Figure 1). When we have a set of dependent instructions for each branch, the compiler marks these regions in the program with their true branch dependencies. We introduce a new set of instructions to mark branch dependent regions.

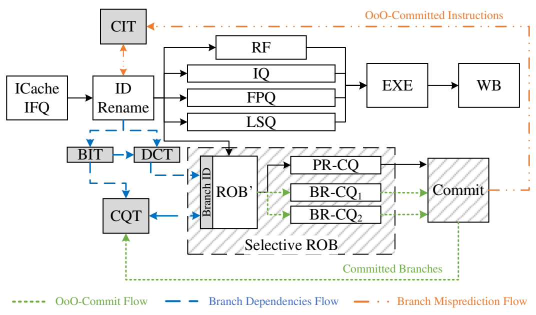

Figure 2: NOREBA Microarchitecture.

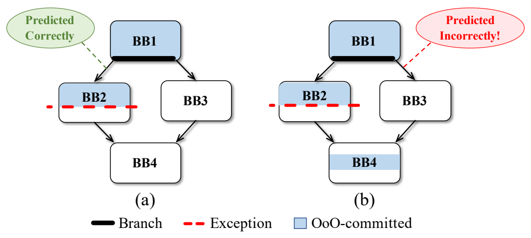

Figure 3: Two general cases of exceptions in NOREBA.

4.2. OoO-Commit Microarchitecture

Figure 2 shows the microarchitecture of NOREBA. We propose a set of new processor structures that allow for the flow of dependency information from the compiler to the hardware (Branch Dependencies Flow, See Figure 2). The Branch ID Table (BIT) stores compiler-assigned IDs of branch instructions and the Dependents Counter Table (DCT) stores the number of consecutive instructions that will be dispatched next and are dependent on a previous branch. The main innovation of the proposed architecture is the Selective ROB that replaces a traditional ROB with multiple FIFO queues that allow reordering of non-speculative instructions that can commit early and out-of-order. In addition, the Commit Queue Table (CQT) provides dependency information for steering instructions to multiple queues based on branch dependencies to improve performance. Finally, we update the commit stage to allow for out-of-order commit of instructions in a safe manner (OoO-Commit Flow, See Figure 2) and handles branch misprediction events using the Commit Instructions Table (CIT) that enables the processor to not execute re-fetched instructions that have already committed after a branch misprediction event (Branch Misprediction Flow, See Figure 2).

4.3. Precise Exception Handling

We build NOREBA processor on top of the RISC-V ISA, which limits exceptions to floating point and memory-related events. In RISC-V based systems, we can accrue floating-point exceptions in floating-point control and status register, `fcsr`, and report it at the end of execution. Memory-related exceptions can arise in both the correct path of a correct branch prediction and also the correct path of a branch misprediction (see Figure 3). Our OoO-commit implementation waits for the success of the page-table access before proceeding to out-of-order commit instructions beyond the memory accesses. However, memory exceptions arising in the correct path of a branch misprediction can be more complex for precise exception handling since we might have already committed some instructions out-of-order beyond the reconvergence point (Figure 3b). In this case, we switch to the OS to handle the exception. The OS can use the CIT information to get the exact state of the microarchitecture since the CIT contains all instructions committed out-of-order beyond the branch reconvergence point. We introduce a new set of instructions to communicate between the CIT and the OS.

4.4. Evaluation Methodology

We use LLVM [9] for our compiler analysis and the marking of branch dependencies. To evaluate the core's performance, we use gem5 [3] in Syscall Emulation mode and implement our OoO-commit design on top of the default out-of-order core. We modify McPAT v1.3 with support for our design for power and energy analysis. As a baseline core, we use a Skylake-like processor. We target SPEC CPU2006 [8] and MiBench [7] benchmark in our evaluations.

5. Key Contributions

Below are the key contributions of this work.

- We implement a low-complexity compiler pass for branch dependency detection. We communicate this information to the hardware, to allow for safe out-of-order commit of independent instructions.
- We propose a novel Selective ROB that implements the out-of-order commit of instructions using low cost hardware. The Selective ROB allows for the reordering of instructions to prioritize those that can commit early, without the additional hardware overheads typically seen in other solutions.
- We enable precise exceptions in the correct-path of branch mispredictions with an efficient recovery mechanism that exposes out-of-order-committed instructions to the OS for handling of recovery or context switching.

Why ASPLOS? This paper connects three areas related to ASPLOS, namely computer architecture, compilers, and operating systems. The goal of this work is to propose a new hardware architecture enhancement that enables high performance and efficiency (computer architecture). To enable this, we require knowledge from an LLVM pass, and propagate that to the hardware (compilers). Finally, to support context switching and precise exceptions, we require a new interface with the OS to save and restore out-of-order commit state (operating systems). The efficient interaction of the different components in our design fits ASPLOS well.

Citation for Most Influential Paper Award. NOREBA is the first efficient OoO-commit processor implementation that advances traditional out-of-order processors to reflect true instruction dependencies and manage critical resources more intelligently.

References

[1] Furat Afram, Hui Zeng, and Kanad Ghose. A group-commit mechanism for ROB-based processors implementing the x86 ISA. In *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, pages 47–58. IEEE, 2013.

[2] Gordon B Bell and Mikko H Lipasti. Deconstructing commit. In *IEEE International Symposium on-ISPASS Performance Analysis of Systems and Software, 2004*, 2004.

[3] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1–7, 2011.

[4] Adrian Cristal, Daniel Ortega, Josep Llosa, and Mateo Valero. Out-of-order commit processors. In *10th International Symposium on High Performance Computer Architecture (HPCA'04)*, pages 48–59. IEEE, 2004.

[5] Marc De Kruijf and Karthikeyan Sankaralingam. Idempotent processor architecture. In *2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 140–151. IEEE, 2011.

[6] Nam Duong and Alex V Veidenbaum. Compiler-assisted, selective out-of-order commit. *IEEE Computer Architecture Letters*, 12(1):21–24, 2012.

[7] Matthew R Guthaus, Jeffrey S Ringenberg, Dan Ernst, Todd M Austin, Trevor Mudge, and Richard B Brown. Mibench: A free, commercially representative embedded benchmark suite. In *Proceedings of the fourth annual IEEE international workshop on workload characterization. WWC-4 (Cat. No. 01EX538)*, pages 3–14. IEEE, 2001.

[8] John L Henning. Spec cpu2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News*, 34(4):1–17, 2006.

[9] Chris Lattner and Vikram Adve. Llvm: A compilation framework for lifelong program analysis & transformation. In *Proceedings of the international symposium on Code generation and optimization: feedback-directed and runtime optimization*, page 75. IEEE Computer Society, 2004.

[10] Salvador Petit Marti, Julio Sahuquillo Borras, Pedro Lopez Rodriguez, Rafael Ubal Tena, and Jose Duato Marin. A complexity-effective out-of-order retirement microarchitecture. *IEEE Transactions on computers*, 58(12):1626–1639, 2009.

[11] José F Martínez, Jose Renau, Michael C Huang, and Milos Prvulovic. Cherry: Checkpointed early resource recycling in out-of-order microprocessors. In *35th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-35). Proceedings.*, 2002.