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

Ali Hajiabadi\textsuperscript{+}, Andreas Diavastos\textsuperscript{+}, Trevor E. Carlson\textsuperscript{+}

\textsuperscript{+} National University of Singapore
\textsuperscript{*} Universitat Politècnica de Catalunya

ASPLOS ‘21
NOREBA: Goal

- Current processors hold on to resources longer than necessary
- **NOREBA implements an intelligent resource management technique based on true branch dependencies**

→ Performance Improvement
→ Low Power and Area Overheads
General-Purpose Out-of-Order Processors

- End of Moore’s law requires efficient computing

- However, general-purpose CPUs still have a significant impact on the overall performance; Hard-to-parallelize work is left for the CPU

- Re-thinking the traditional design to unlock efficiency:
  - Co-design the different layers of the system

---

ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8

Frontend

<table>
<thead>
<tr>
<th>Inst9</th>
<th>Inst10</th>
<th>Inst11</th>
<th>Inst12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inst5</td>
<td>Branch</td>
<td>Inst7</td>
<td>Inst8</td>
</tr>
<tr>
<td>Inst1</td>
<td>Inst2</td>
<td>Inst3</td>
<td>Load</td>
</tr>
</tbody>
</table>

Commit

Cycle Count

0

Program Order

Reorder Buffer

Head of ROB
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8

Frontend

<table>
<thead>
<tr>
<th>Inst9</th>
<th>Inst10</th>
<th>Inst11</th>
<th>Inst12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inst5</td>
<td>Branch</td>
<td>Inst7</td>
<td>Inst8</td>
</tr>
<tr>
<td>Inst2</td>
<td>Inst3</td>
<td>Load</td>
<td></td>
</tr>
</tbody>
</table>

Commit

Cycle Count

0

Program Order

Reorder Buffer

| Inst1 |

Head of ROB
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8

Frontend

Commit

Program Order

Reorder Buffer
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8

Frontend

Commit

Cycle Count
3

Program Order

Reorder Buffer

Head of ROB
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8

Frontend

Commit

Program Order

Reorder Buffer

Head of ROB
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8

Frontend

Commit

Program Order

Reorder Buffer

Head of ROB
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8

Frontend

Commit

Program Order

Reorder Buffer

Head of ROB
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8

Frontend

Commit

Program Order

Reorder Buffer

Head of ROB
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8

**Frontend**

- Inst11
- Inst12

**Commit**

- Inst3
- Inst2
- Inst1

**Reorder Buffer**

- Inst10
- Inst9
- Inst8
- Inst7
- Branch
- Inst5
- Load

- Cycle Count: 9

Program Order

Head of ROB
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8

Reorder Buffer

Head of ROB
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8

Frontend

Commit

Reorder Buffer

ROB Full

Cycle Count

Program Order

Head of ROB

Inst12

Inst3

Inst2

Inst1

Inst11

Inst10

Inst9

Inst8

Inst7

Branch

Inst5

Load

Blocked

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8
**ROB in Traditional General-Purpose Processors**

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8

![Diagram showing the ROB in a general-purpose processor](image-url)
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8

Frontend

Commit

Program Order

Reorder Buffer

Head of ROB
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8

Frontend

Commit

Cycle Count

14

Program Order

Reorder Buffer

Inst12 Inst11 Inst10 Inst9 Inst8 Inst7 Branch Inst5

Load Inst3 Inst2 Inst1

Head of ROB
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8

Frontend

Commit

Reorder Buffer

Cycle Count

Program Order

Load

Inst3

Inst2

Inst1

Branch

Inst5

Head of ROB

Inst12

Inst11

Inst10

Inst9

Inst8

Inst7

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8

Frontend

Commit

Program Order

Reorder Buffer

Cycle Count
17

Head of ROB

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8

Frontend

Commit

Program Order

Reorder Buffer

Cycle Count
17

Head of ROB
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8

Frontend

Commit

Reorder Buffer

Cycle Count

18

Program Order

Inst12 Inst11 Inst10 Inst9

Load Inst3 Inst2 Inst1

Inst8 Inst7 Branch Inst5

Head of ROB
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8

Frontend

Commit

Program Order

Cycle Count

20

Load

Inst3

Inst2

Inst1

Inst8

Inst7

Branch

Inst10

Inst5

Inst9

Inst12

Inst11

Reorder Buffer

Head of ROB

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8
ROB in Traditional General-Purpose Processors

Execution Time = 1 cycle | Long Latency Load = 10 cycles | Fetch/Commit Width = 1 | ROB Size = 8

Frontend

In-Order Commit

Reorder Buffer

Program Order

Cycle Count

22

Load
Inst8
Inst12
Inst3
Inst7
Branch
Inst11
Inst10
Inst5
Inst9

Head of ROB
Alternative Approach

In-Order Commit is conservative
What if Inst9, Inst10, Inst11, and Inst12 are independent from Branch?

Frontend

Commit

Reorder Buffer
Alternative Approach

In-Order Commit is conservative
What if Inst9, Inst10, Inst11, and Inst12 are independent from Branch?

Frontend

Commit

Fast-forwarding 7 cycles

Program Order

Reorder Buffer
In-Order Commit is conservative

What if Inst9, Inst10, Inst11, and Inst12 are independent from Branch?

Frontend

Commit

Program Order

Reorder Buffer
Alternative Approach

In-Order Commit is conservative
What if Inst9, Inst10, Inst11, and Inst12 are independent from Branch?

Frontend

Commit

Reorder Buffer
Alternative Approach

In-Order Commit is conservative
What if Inst9, Inst10, Inst11, and Inst12 are independent from Branch?

Frontend

Commit

Reorder Buffer

Inst12

Cycle Count
10

Inst9
Inst3
Inst2
Inst1
Inst10

Inst11
Inst8
Inst7
Branch
Inst5
Load

Program Order

In-Order Commit is conservative
What if Inst9, Inst10, Inst11, and Inst12 are independent from Branch?
Alternative Approach

In-Order Commit is conservative
What if Inst9, Inst10, Inst11, and Inst12 are independent from Branch?

Frontend

Commit

Reorder Buffer

Program Order

Cycle Count

11

Inst9 Inst3 Inst2 Inst1

Inst11 Inst10

Inst12 Inst8 Inst7 Branch Inst5 Load
Alternative Approach

In-Order Commit is conservative
What if Inst9, Inst10, Inst11, and Inst12 are independent from Branch?

Frontend

Commit

Program Order

Reorder Buffer
In-Order Commit is conservative
What if Inst9, Inst10, Inst11, and Inst12 are independent from Branch?
In-Order Commit is conservative
What if Inst9, Inst10, Inst11, and Inst12 are independent from Branch?

Program Order:
- Inst8
- Inst7
- Branch
- Inst5

Cycle Count:
- 14

Commit:
- Inst9
- Inst3
- Inst2
- Inst1
- Load
- Inst12
- Inst11
- Inst10

Reorder Buffer
Alternative Approach

In-Order Commit is conservative
What if Inst9, Inst10, Inst11, and Inst12 are independent from Branch?

In-Order Commit is conservative
What if Inst9, Inst10, Inst11, and Inst12 are independent from Branch?
Alternative Approach

In-Order Commit is conservative
What if Inst9, Inst10, Inst11, and Inst12 are independent from Branch?

Frontend

Commit

Cycle Count
16

Program Order

Reorder Buffer
Alternative Approach

In-Order Commit is conservative
What if Inst9, Inst10, Inst11, and Inst12 are independent from Branch?

Frontend

Commit

Cycle Count
17

Program Order

Reorder Buffer
Alternative Approach

In-Order Commit is conservative
What if Inst9, Inst10, Inst11, and Inst12 are independent from Branch?

Frontend

Commit

Reorder Buffer
Alternative Approach

In-Order Commit is conservative
What if Inst9, Inst10, Inst11, and Inst12 are independent from Branch?

Frontend

Commit

Cycle Count
18

Saving 4 cycles compared to in-order commit

Program Order

Reorder Buffer
Alternative Approach

In-Order Commit is conservative
What if Inst9, Inst10, Inst11, and Inst12 are independent from Branch?

Out-of-Order Commit

Cycle Count
18

Saving 4 cycles compared to in-order commit
Program Order

Reorder Buffer
Questions

- How to detect branch dependencies non-speculatively?
- How to implement OoO-commit efficiently?
- How to handle exceptions and context switches?

Compiler
Static Compiler Analysis

Architecture
Multi-Queue Selective ROB

Operating System

Communicating the OoO-Commit State

Branch dependencies information
NOREBA: Static Compiler Analysis

1. Detecting the branch reconvergence point
2. Detecting control dependent instructions
3. Detecting data dependent instructions
4. Marking branch dependent regions

BB1

setBranchId BR1
breqz a5, L1

BB2

L1:
setDependency 8 BR1
lw a4, -40(s0)
lw a5, -36(s0)
subw a5, a4, a5
sw a5, -20(s0)
lw a4, -40(s0)
lw a5, -36(s0)
addw a5, a4, a5
sw a5, -24(s0)

BB3

setDependency 9 BR1
lw a4, -40(s0)
lw a5, -36(s0)
addw a5, a4, a5
sw a5, -20(s0)
lw a4, -40(s0)
lw a5, -36(s0)
subw a5, a4, a5
sw a5, -24(s0)
j L2

BB4

L2:
setDependency 4 BR0
lw a4, -40(s0)
lw a5, -36(s0)
xor a5, a5, a4
sw a5, -52(s0)

setDependency 6 BR1
lw a5, -20(s0)
xor a5, a5, a4
sw a5, -48(s0)
lw a5, -24(s0)
xor a5, a5, a4
sw a5, -56(s0)

setBranchId Branch_ID
Assigns an ID to the branch

setDependency #NUM Branch_ID
Specifies the branch dependency of the next #NUM instructions
NOREBA: Microarchitecture

Frontend
- Tracking true branch dependencies informed by the compiler for each instruction

Backend
- A lightweight implementation for OoO-commit that provides more opportunities to release resources
NOREBA: Challenge in Exception Handling

- Need to save and restore the state of the OoO-committed instructions

**Backend of NOREBA Core**

- **Selective ROB**
  - Old
  - ROB'
  - New

- **Commit Queue Table**
  - Commit Q1
  - Commit Q2
  - ... Commit QN

- **Commit**

- **Committed Instructions Table**

- **setCITEntry and getCITEntry** instructions for communicating with the OS

**Correct Path**

**Branch**

**Misprediction**

**Exception**

**Reconvergence Point**

**OoO-Committed**

**OS**
Evaluation: Setup

- Simulation: gem5
- Compiler: LLVM-10
- Benchmarks:
  - *SPEC CPU2006*: C/C++ programs, running single 1B instruction representatives (using SimPoint)
  - *MiBench*: entire program runs

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>L1d/i size</td>
<td>32KB, 4 clk</td>
</tr>
<tr>
<td>L2 size</td>
<td>256KB, 12 clk</td>
</tr>
<tr>
<td>L3 size</td>
<td>1MB, 36 clk</td>
</tr>
<tr>
<td>Fetch/dispatch/commit width</td>
<td>4/4/4</td>
</tr>
<tr>
<td>Branch Predictor</td>
<td>TAGE-SC-L-8KB</td>
</tr>
<tr>
<td>Prefetcher</td>
<td>DCPT</td>
</tr>
</tbody>
</table>

**Selective ROB**

- ROB'                           | 224 entries   |
- BR-CQ                          | 2 × 8 entries |
- PR-CQ                          | 8 entries     |

| BIT/CQT size                   | 8             |
| CIT size                       | 128           |
| Baseline ROB                   | 224 entries   |
| IQ/LQ/SQ/RF                    | 68/72/56/168  |
Evaluation: Performance

In-Order Commit
All commit conditions preserved

This Work
This Work w/ unlimited queues
Speculating on Branch commit condition

~22% performance improvement on average
Up to 230% performance improvement
Reaches ~95% of a fully branch speculative OoO-commit implementation
Evaluation: Critical Branches

More dependent instructions and more critical ➔ More opportunities

~ 2.3X improvement for mcf

More dependent instructions per branch

More dependent instructions and less critical ➔ Fewer opportunities

~ 1.1X improvement for bzip2
Evaluation: Size of Resources

We are close to aggressive and branch speculative OoO-commit (~95% of SpeculativeBR OoO-C)

Higher performance for bigger cores with more resources

NOREBA continues to scale

Small Core
- ROB = 128
- IQ = 56
- LQ/SQ = 48/36
- RF = 64

Medium Core
- ROB = 192
- IQ = 60
- LQ/SQ = 72/42
- RF = 128

Big Core
- ROB = 224
- IQ = 68
- LQ/SQ = 72/56
- RF = 168
Evaluation: NOREBA and Prefetchers

Normalized to NHM InO-C

Additive effect of combining NOREBA and prefetchers

Higher Performance using both

Prefetching allows continuing execution, but NOREBA allows continuing committing instructions
Evaluation: Power and Area Overhead

4% power overhead, 8% area overhead

Low overhead for the extra performance (~22% on average, up to 230%)
NOREBA: Overview of the Design

Implementation

Detecting branch dependencies *non-speculatively*
Implementing OoO-commit *efficiently*
Able to handle *exceptions* and context switches

- **Compiler**
  - Static Compiler Analysis

- **Architecture**
  - Multi-Queue
  - Selective ROB

- **Operating System**

  Communicating the OoO-Commit State

Branch dependencies information
Conclusion

• Efficient interaction between different layers of the system unlocks efficiency and performance for general-purpose processors

• NOREBA provides a HW/SW co-design solution that enables OoO-commit and better resource management
  • 22% performance improvement over the baseline and achieving 95% of the aggressive branch speculative OoO-commit implementation
  • Low power and area overheads (~4% power, and ~8% area overhead)
NOREBA: A Compiler-Informed Non-speculative Out-of-Order Commit Processor

Thanks for your attention

Ali Hajiabadi, Andreas Diavastos, Trevor E. Carlson

National University of Singapore
Universitat Politècnica de Catalunya

ASPLOS ‘21