Discerning the Dominant Out-of-Order Performance Advantage

D. McFarlin, C. Tucker, C. Zilles

ASPLOS - 2013

CMU       UIUC

August 1, 2013
Motivation

- Despite the power & energy inefficiency and increased complexity of OOO designs, they are still predominantly used
  - Mostly due to high single-thread performance compared to IO core

- This paper tries to quantify the performance contributions of various sources in an OOO engine
  - Also tries to figure out whether an equally provisioned IO core has any hope of matching the performance of OOO core

- Two dominant sources of OOO performance compared to IO
  - The improved schedules enabled by OOO hw speculation support
  - Ability to generate different schedules on different occurrences of the same insts based on operand and functional unit availability
Out-of-order vs. In-order in a Nutshell

- On an average, an OOO design achieves about 53% speedup over a similarly provisioned in-order machine.
- 88% of the speedup can be achieved by using a single “best” static schedule obtained from observing OOO schedules.
  - Rest of the speedup comes from the *dynamism*.

- Major benefit of dynamism can be achieved by targeting:
  - Load instructions that miss in the cache only part of the time.
  - Branch mispredictions.

- This is a limit study as opposed to any real design study:
  - Oblivious to any specific microarchitectural configurations.
Commonly used static schedules overlap: 3.3 - 5 basic blocks

Even the most sophisticated static schedule is hobbled by the stall-on-use / head-of-line blocking problem inherent to cache misses in in-order designs
Basic Block Overlapping Ability of OOO Designs

- Overlapping the execution of insts from 4 BBs is only sufficient to account for about 70% of the OOO execution cycles.
- Upto 6 BBs (best static schedule) accounts only 82% of OOO cycles.

![Graph showing the percentage of execution cycles vs. consecutive basic blocks](image)
Scheduling vs. Dynamism

- **Path Specificity**
  - OOO gets the benefit of scheduling based on the predicted path

- **Variable Latency**
  - OOO easily adopts a different schedule in the presence of variable latency insts

- **Early Branch Resolution**
  - OOO machine has the potential to overlap branch misprediction latency with other execution

- **Branch Misprediction**
  - ILP “hoisted” insts further delay the critical inst after recovery
Path Specificity

- Static schedules can “hoist” instructions from down the stream.
- An ILP compiler achieves good overlap in proportion with the bias of the branch.
- The OOO engine achieves good overlap in proportion to the predictability of the branch.

The OOO schedule beats ILP compiler schedule because
- Branch predictability is almost always higher than branch bias.
Tolerance of long latency operations is an often touted strength of OOO processors.

A scenario where the critical path is a function of latency:

\[ z = p1\rightarrow x\rightarrow z + p2\rightarrow x\rightarrow z \]

How the second pair or loads should be scheduled depends on which loads miss, so that we can overlap misses.

This is an example where there is no substitute for dynamism.
Early Branch Resolution

- Overlapping branch misprediction latency with other execution
  - `p->y ++; if (b) { ... }`
  - The branch on `b` does not depend on the computation before it
  - Branch can be resolved in parallel with the computation

- Achieving this overlap in IO machines can be challenging
  - Branches generally must be placed at the end of their basic block to demarcate which insts belong in the block
  - Also constrains branches place in the schedule

- Two solutions to the above mentioned problem
  - Software soln: pre-branch computation replicated on both paths
  - Hw soln: “prepare-to-branch” separate the computation of a branch outcome from the branch’s role as a BB boundary; IBM’s Cell
The static schedule that enters the window after the branch misprediction recovery consists of instructions that were hoisted to cover the L1-hit latency but this latency has already been covered by the mispredict latency.
Branch Misprediction

- The static schedule that enters the window after the branch misprediction recovery consists of instructions that were hoisted to cover the L1-hit latency but this latency has already been covered by the mispredict latency.

![Diagram](image-url)

**Figure 4.** Misprediction recovery latency covers L1-hit latency
Branch Misprediction (cont...) 

- Load chain: \( r_0 = \text{LD} (A) ; \ r_1 = \text{LD}(r_0) \) from \( eon \)

Figure 5. Misprediction recovery latency covers L1-hit latencies of a Load chain
Experimental Method

- Experimental Setup: simulator based limit study
- Benchmarks
  - SPECint 2000 applications, compiled for alpha with peak flag
  - Warm-ups: 2 million insts; Simulation: 10 million insts
- Harvesting Schedules
  - Collect OOO trace of dominant schedule for various regions and stitch them together
- Replying Dominant Schedules
  - To measure the enhanced IO performance
- OOO Core Model
  - Processor pipeline: 4-wide fetch, execute and commit
  - ROB: 128 entry; DRAM latency: 150 cycle
Results: Speculation vs. Dynamism

- IO machine achieves only 47% of the performance of OOO

![Baseline In-Order Vs. In-Order Enhanced With Speculation Support](image-url)

[performance normalized to Out-of-Order]
In-order Continual Flow Pipelining (iCFP) exploits a good fraction of dynamism and achieves near OOO performance benefit.

![Bar chart comparing baseline in-order and in-order enhanced with speculation support versus in-order enhanced with speculation support and iCFP. Performance is normalized to OOO.](chart.png)
Conclusions

- OOO processors generally attain higher performance on control intensive integer code compared to in-order designs.
- Conventional wisdom for the disparity in performance is the OOO ability to avoid head-of-line blocking and exploit far-flung MLP.
- The major enabler for dynamism is the OOO general-purpose, well provisioned speculation support mechanisms:
  - Very large ROB, Renamer, Load/Store Queue, etc.
  - These can rollback speculative state at lower cost than even the most advanced in-order designs.
- 88% of the speedup attained by the OOO over an IO design can be attributed to the speculation support mechanisms alone.