

# RUNAHEAD EXECUTION

Fuat Onur ERGÜN

Reference: Paper by Onur Mutlu, Jared Stark, Chris Wilkerson, Yale Patt

## Outline

- Motivation
- Overview
- Mechanism
- Experimental Evaluation
- Conclusions

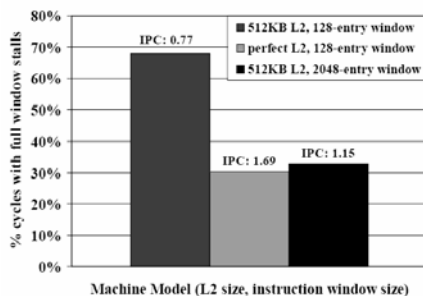
## Motivation

- Out-of-order processors require very large instruction windows to tolerate today's main memory latencies.
  - Even in the presence of caches and prefetchers
- As main memory latency (in terms of processor cycles) increases, instruction window size should also increase to fully tolerate the memory latency.
- Building a very large instruction window is not an easy task.

## Small Windows: Full-window Stalls

- Instructions are retired in-order from the instruction window to support precise exceptions.
- When a very long-latency instruction is not complete, it blocks retirement and incoming instructions fill the instruction window if the window is not large enough.
- Processor cannot place new instructions into the window if the window is already full. This is called a full-window stall.
- L2 misses are responsible for most full-window stalls.

## Impact of Full-window Stalls



## Overview of Runahead Execution

- During a significant percentage of full-window stall cycles, no work is performed in the processor.
- Runahead execution unblocks the full window stall caused by a long-latency L2-miss instruction.
- Enter *runahead mode* when the oldest instruction is an L2-miss load and remove that load from the processor.
- While in *runahead mode*, keep processing instructions without updating architectural state and without blocking the instruction window due to L2 misses.
- When the original load miss returns back, resume normal-mode execution starting with the *runahead-causing load*

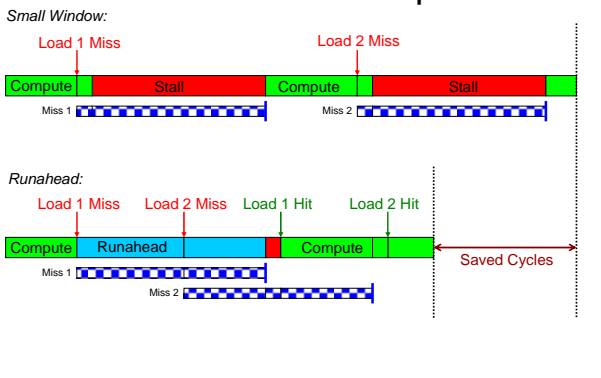
## Benefits of Runahead Execution

- Loads and stores independent of L2-miss instructions generate useful prefetch requests:
  - From main memory to L2
  - From L2 to L1
- Instructions on the predicted program path are prefetched into the trace cache and L2.
- Hardware prefetcher tables are trained using future memory access information. The prefetcher also runs ahead along with the processor.

## Mechanism

- Entry into Runahead Mode
- Processing in Runahead Mode
- Pseudo-retirement in Runahead Mode
- Runahead Cache
- Runahead Branches
- Exit from Runahead Mode

## Runahead Example



## Entry into Runahead Mode

When the instruction window is blocked by the long latency operation like L2 misses :

- Processor checkpoints; architectural register state, branch history register, return address stack.
- Processor records the address of the L2-miss load.
- Processor enters *runahead mode*.
- L2-miss load marks its destination register as *invalid* and is removed from the instruction window.

## Processing in Runahead Mode

Two types of results are produced: INV (invalid), VALID

- First INV result is produced by the L2-miss load that caused entry into runahead mode.
- An instruction produces an INV result
  - If it sources an INV result
  - If it misses in the L2 cache (A prefetch request is generated)
- INV results are marked using INV bits in the register file, store buffer, and runahead cache.
  - INV bits prevent introduction of bogus data into the pipeline.
  - Bogus values are not used for prefetching/branch resolution.

## Pseudo-retirement in Runahead Mode

- An instruction is examined for pseudo-retirement when it reaches the head of the instruction window.
- An INV instruction is removed from window immediately.
- A VALID instruction is removed when it completes execution and updates only the micro architectural state.
- Pseudo-retired instructions free their allocated resources.
- Pseudo-retired runahead stores communicate their data and INV status to dependent runahead loads.

## Runahead Cache

- An auxiliary structure that holds the data values and INV bits for memory locations modified by pseudo-retired runahead stores.
- Its purpose is memory communication during runahead mode.
- Runahead loads access store buffer, runahead cache, and L1 data cache in parallel.
- Size of runahead cache is very small (512 bytes).

## Runahead Branches

- Runahead branches use the same predictor as normal branches.
- VALID branches are resolved and trigger recovery if mispredicted.
- INV branches cannot be resolved.

## Exit from Runahead Mode

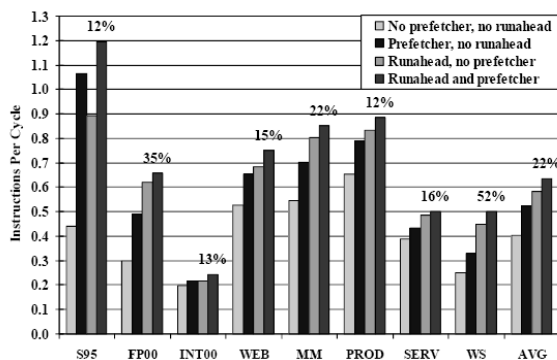
- When the data for the instruction that caused entry into runahead returns from main memory:
- All instructions in the machine are flushed.
- INV bits are reset. Runahead cache is flushed.
- Processor restores the state as it was before the runahead inducing instruction was fetched.
- Processor starts fetch beginning with the runahead-inducing L2-miss instruction.

## Experimental Evaluation

### Baseline Processor

- 3-wide fetch, 29-stage pipeline
- 128-entry instruction window
- 32 KB, 8-way, 3-cycle L1 data cache, write-back
- 512 KB, 8-way, 16-cycle L2 unified cache, write back
- Approximately 500-cycle penalty for L2 misses
- Streaming prefetcher (16 streams)

## Performance of Runahead Execution



## Conclusions

- Runahead execution results in 22% IPC increase over the baseline processor with a 128-entry window and a streaming prefetcher.
- Runahead and the streaming prefetcher interact positively.
- Store-load data communication through memory in runahead mode is vital for high performance.