Techniques for Efficient Processing in Runahead Execution Engines

Onur Mutlu
Hyesoon Kim
Yale N. Patt
Talk Outline

- Background on Runahead Execution
- The Problem
- Causes of Inefficiency and Eliminating Them
- Evaluation
- Performance Optimizations to Increase Efficiency
- Combined Results
- Conclusions
Background on Runahead Execution

- A technique to obtain the memory-level parallelism benefits of a large instruction window

- When the oldest instruction is an L2 miss:
  - Checkpoint architectural state and enter runahead mode

- In runahead mode:
  - Instructions are speculatively pre-executed
  - The purpose of pre-execution is to generate prefetches
  - L2-miss dependent instructions are marked INV and dropped

- Runahead mode ends when the original L2 miss returns
  - Checkpoint is restored and normal execution resumes
Runahead Example

Small Window:

- Load 1 Miss
- Load 2 Miss

Runahead:

- Load 1 Miss
- Load 2 Miss
- Load 1 Hit
- Load 2 Hit

Saved Cycles
The Problem

- A runahead processor pre-executes some instructions speculatively
- Each pre-executed instruction consumes energy

- Runahead execution significantly increases the number of executed instructions, *sometimes* without providing significant performance improvement
The Problem (cont.)

Efficient Runahead Execution
Efficiency of Runahead Execution

\[
\text{Efficiency} = \frac{\% \text{ Increase in IPC}}{\% \text{ Increase in Executed Instructions}}
\]

- **Goals:**
  - Reduce the number of executed instructions without reducing the IPC improvement
  - Increase the IPC improvement without increasing the number of executed instructions
Talk Outline

- Background on Runahead Execution
- The Problem
- Causes of Inefficiency and Eliminating Them
- Evaluation
- Performance Optimizations to Increase Efficiency
- Combined Results
- Conclusions
Causes of Inefficiency

- Short runahead periods
- Overlapping runahead periods
- Useless runahead periods
Short Runahead Periods

- Processor can initiate runahead mode due to an already in-flight L2 miss generated by
  - the prefetcher, wrong-path, or a previous runahead period

- Short periods
  - are less likely to generate useful L2 misses
  - have high overhead due to the flush penalty at runahead exit
Eliminating Short Runahead Periods

- Mechanism to eliminate short periods:
  - Record the number of cycles C an L2-miss has been in flight
  - If C is greater than a threshold T for an L2 miss, disable entry into runahead mode due to that miss
  - T can be determined statically (at design time) or dynamically

- T=400 for a minimum main memory latency of 500 cycles works well
Overlapping Runahead Periods

- Two runahead periods that execute the same instructions

- Second period is inefficient
Overlapping Runahead Periods (cont.)

- Overlapping periods are not necessarily useless
  - The availability of a new data value can result in the generation of useful L2 misses
- But, this does not happen often enough

- Mechanism to eliminate overlapping periods:
  - Keep track of the number of pseudo-retired instructions $R$ during a runahead period
  - Keep track of the number of fetched instructions $N$ since the exit from last runahead period
  - If $N < R$, do not enter runahead mode
Useless Runahead Periods

- Periods that do not result in prefetches for normal mode

- They exist due to the lack of memory-level parallelism

- Mechanism to eliminate useless periods:
  - Predict if a period will generate useful L2 misses
  - Estimate a period to be useful if it generated an L2 miss that cannot be captured by the instruction window
    - Useless period predictors are trained based on this estimation
Predicting Useless Runahead Periods

- Prediction based on the past usefulness of runahead periods caused by the same static load instruction
  - A 2-bit state machine records the past usefulness of a load

- Prediction based on too many INV loads
  - If the fraction of INV loads in a runahead period is greater than T, exit runahead mode

- Sampling (phase) based prediction
  - If last N runahead periods generated fewer than T L2 misses, do not enter runahead for the next M runahead opportunities

- Compile-time profile-based prediction
  - If runahead modes caused by a load were not useful in the profiling run, mark it as non-runahead load
Talk Outline

- Background on Runahead Execution
- The Problem
- Causes of Inefficiency and Eliminating Them
- Evaluation
- Performance Optimizations to Increase Efficiency
- Combined Results
- Conclusions
Baseline Processor

- Execution-driven Alpha simulator
- 8-wide superscalar processor
- **128-entry instruction window**, 20-stage pipeline
- 64 KB, 4-way, 2-cycle L1 data and instruction caches
- 1 MB, 32-way, 10-cycle unified L2 cache

- 500-cycle minimum main memory latency
- Aggressive stream-based prefetcher
- 32 DRAM banks, 32-byte wide processor-memory bus (4:1 frequency ratio), 128 outstanding misses
  - Detailed memory model
Impact on Efficiency

Increase Over Baseline OOO

- baseline runahead: 26.5%
- short: 26.5%
- overlapping: 26.5%
- useless
- short+overlapping+useless: 22.6%
- Executed Instructions: 15.3%
- IPC: 6.7%

Efficient Runahead Execution
Performance Optimizations for Efficiency

- Both efficiency AND performance can be increased by increasing the usefulness of runahead periods

- Three optimizations:
  - Turning off the Floating Point Unit (FPU) in runahead mode
  - Optimizing the update policy of the hardware prefetcher (HWP) in runahead mode
  - Early wake-up of INV instructions (in paper)
Turning Off the FPU in Runahead Mode

- FP instructions do not contribute to the generation of load addresses
- FP instructions can be dropped after decode
  - Spares processor resources for more useful instructions
  - Increases performance by enabling faster progress
  - Enables dynamic/static energy savings

- Results in an unresolvable branch misprediction if a mispredicted branch depends on an FP operation (rare)

- Overall – increases IPC and reduces executed instructions
HWP Update Policy in Runahead Mode

- Aggressive hardware prefetching in runahead mode may hurt performance, if the prefetcher accuracy is low
- Runahead requests more accurate than prefetcher requests
- Three policies:
  - Do not update the prefetcher state
  - Update the prefetcher state just like in normal mode
  - Only train existing streams, but do not create new streams

- Runahead mode improves the *timeliness* of the prefetcher in many benchmarks
- Only training the existing streams is the best policy
Talk Outline

- Background on Runahead Execution
- The Problem
- Causes of Inefficiency and Eliminating Them
- Evaluation
- Performance Optimizations to Increase Efficiency
- Combined Results
- Conclusions
Efficient Runtimed Execution

Overall Impact on Executed Instructions

Increase in Executed Instructions:
- AVG: 6.2%
- Other applications: 26.5%
Overall Impact on IPC

- Increase in IPC
- Baseline runahead
- All techniques

Efficient Runahead Execution
Conclusions

- Three major causes of inefficiency in runahead execution: *short, overlapping, and useless runahead periods*
- Simple efficiency techniques can effectively reduce the three causes of inefficiency
- Simple performance optimizations can increase efficiency by increasing the usefulness of runahead periods

- Proposed techniques:
  - reduce the extra instructions from 26.5% to 6.2%, without significantly affecting performance
  - are effective for a variety of memory latencies ranging from 100 to 900 cycles
Backup Slides
Memory Latency (Executed Instructions)

[Bar chart showing the increase in executed instructions for different memory latencies. The chart compares baseline runahead and all techniques.]

Efficient Runahead Execution
Memory Latency (IPC Delta)
Cache Sizes (Executed Instructions)

Increase in Executed Instructions

- 512 KB
- 1 MB
- 2 MB
- 4 MB

- baseline runahead
- all techniques
Cache Sizes (IPC Delta)

Increase in IPC

<table>
<thead>
<tr>
<th>Cache Size</th>
<th>Baseline Runahead</th>
<th>All Techniques</th>
</tr>
</thead>
<tbody>
<tr>
<td>512 KB</td>
<td>25%</td>
<td>20%</td>
</tr>
<tr>
<td>1 MB</td>
<td>20%</td>
<td>20%</td>
</tr>
<tr>
<td>2 MB</td>
<td>10%</td>
<td>10%</td>
</tr>
<tr>
<td>4 MB</td>
<td>5%</td>
<td>5%</td>
</tr>
</tbody>
</table>
INT (Executed Instructions)

![Bar chart showing the increase in executed instructions versus memory latency for runahead (INT) and all techniques (INT).]
INT (IPC Delta)

![Bar chart showing increase in IPC with memory latency.]

- Blue bars represent runahead (INT).
- Red bars represent all techniques (INT).

Efficient Runahead Execution
FP (Executed Instructions)

Increase in Executed Instructions

- runahead (FP)
- all techniques (FP)

Memory Latency

- 100
- 300
- 500
- 700
- 900

Efficient Runahead Execution
FP (IPC Delta)

![Bar chart showing increase in IPC with memory latency.]

- **runahead (FP)**
- **all techniques (FP)**

Efficient Runahead Execution
Early INV Wake-up

- Keep track of INV status of an instruction in the scheduler.
- Scheduler wakes up the instruction if \textit{any} source is INV.

  + Enables faster progress during runahead mode by removing the useless INV instructions faster.

  - Increases the number of executed instructions.
  - Increases the complexity of the scheduling logic.

- Not worth implementing due to small IPC gain