Speculative Parallelization in Decoupled Look-ahead

Alok Garg, Raj Parihar, and Michael C. Huang

Dept. of Electrical & Computer Engineering
University of Rochester, Rochester, NY
Motivation

- Single-thread performance still important design goal
- Branch mispredictions and cache misses are still important performance hurdles
- One effective approach: Decoupled Look-ahead
- Look-ahead thread can often become the new bottleneck
- Speculative parallelization aptly suited for its acceleration
  - More parallelism opportunities: Look-ahead thread does not contain all dependences
  - Simple architectural support: Look-ahead not correctness critical
Outline

• Motivation
• Baseline decoupled look-ahead
• Speculative parallelization in look-ahead
• Experimental analysis
• Summary
Baseline Decoupled Look-ahead

- Binary parser is used to generate skeleton from original program
- The skeleton runs on a separate core and
  - Maintains its own memory image in local L1, no write-back to shared L2
  - Sends all branch outcomes through FIFO queue and also helps prefetching

Register state synchronization

Practical Advantages of Decoupled Look-ahead

- Look-ahead thread is a single, self-reliant agent
  - No need for quick spawning and register communication support
  - Little management overhead on main thread
  - Easier for run-time control to disable
- Natural throttling mechanism to prevent run-away prefetching
- Look-ahead thread size comparable to aggregation of short helper threads
Look-ahead: A New Bottleneck

- Comparing four systems
  - Baseline
  - Decoupled look-ahead
  - Ideal
  - Look-ahead alone

- Application categories
  - Bottlenecks removed
  - Speed of look-ahead not the problem
  - **Look-ahead is the new bottleneck**
Outline

• Motivation
• Baseline decoupled look-ahead
• Speculative parallelization in look-ahead
• Experimental analysis
• Summary
Unique Opportunities

- Skeleton code offers more parallelism
  - Certain dependencies removed during slicing for skeleton

- Look-ahead is inherently error-tolerant
  - Can ignore dependence violations
  - Little to no support needed, unlike in conventional TLS

Assembly code from equake

1. 0x12000da84  lda a5, 744(sp)
2. 0x12000dac0  ldlt7, 0(a5)
3. 0x12000daec  lda a5, 4(a5)
4. 0x120011984  ldq a0, 80(sp)
5. 0x1200119b0  ldt $f0, 0(a0)
6. 0x120011ac  ldt $f12, 32(sp)
7. 0x120011f8  bis 0, t8, t11
8. 0x120011b04  lda a0, 8(a0)
Software Support

- Dependence analysis
  - Profile guided, coarse-grain at basic block level
- Spawn and Target points
  - Basic blocks with consistent dependence distance of more than threshold of \( D_{\text{MIN}} \)
  - Spawned thread executes from target point
- Loop level parallelism is also exploited

Parallelism at basic block level

![Diagram showing parallelism at basic block level](Diagram)

Available parallelism for 2 corecontexts system
Parallelism in Look-ahead Binary

Available theoretical parallelism for 2 core/contexts system; $D_{\text{MIN}} = 15$ BB

Parallelism potential with stables and consistent target and spawn points
Hardware and Runtime Support

- **Thread spawning and merging**
  - Not too different from regular thread spawning except
    - Spawned thread shares the same register and memory state
    - Spawning thread will terminate at the target PC

- **Value communication**
  - Register-based naturally through shared registers in SMT
  - Memory-based communication can be supported at different levels
  - *Partial* versioning in cache at line level

![Diagram showing thread spawning and merging with time progression and state transitions.](image-url)
Outline

• Motivation
• Baseline decoupled look-ahead
• Speculative parallelization in look-ahead
• Experimental analysis
• Summary
Experimental Setup

- Program/binary analysis tool: based on ALTO
- Simulator: based on heavily modified SimpleScalar
  - SMT, look-ahead and speculative parallelization support
  - True execution-driven simulation (faithfully value modeling)

<table>
<thead>
<tr>
<th>Baseline core</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch/Decode/Commit</td>
<td>8 / 4 / 6</td>
</tr>
<tr>
<td>ROB</td>
<td>128</td>
</tr>
<tr>
<td>Functional units</td>
<td>INT 2+1 mul +1 div, FP 2+1 mul +1 div</td>
</tr>
<tr>
<td>Issue Q / Reg. (int,fp)</td>
<td>(32, 32) / (120, 120)</td>
</tr>
<tr>
<td>LSQ(LQ,SQ)</td>
<td>64 (32,32) 2 search ports</td>
</tr>
<tr>
<td>Branch predictor</td>
<td>Bimodal + Gshare</td>
</tr>
<tr>
<td>- Gshare</td>
<td>8K entries, 13 bit history</td>
</tr>
<tr>
<td>- Bimodal/Meta/BTB</td>
<td>4K/8K/4K (4-way) entries</td>
</tr>
<tr>
<td>Br. mispred. penalty</td>
<td>at least 7 cycles</td>
</tr>
<tr>
<td>L1 data cache</td>
<td>32KB, 4-way, 64B line, 2 cycles, 2 ports</td>
</tr>
<tr>
<td>L1 L1 cache (not shared)</td>
<td>64KB, 1-way, 128B, 2 cyc</td>
</tr>
<tr>
<td>L2 cache (uni. shared)</td>
<td>1MB, 8-way, 128B, 15 cyc</td>
</tr>
<tr>
<td>Memory access latency</td>
<td>400 cycles</td>
</tr>
</tbody>
</table>

Microarchitectural and cache configurations
Speedup of speculative parallel decoupled look-ahead

- 14 applications in which look-ahead is bottleneck
- Speedup of look-ahead systems over single thread
  - Decoupled look-ahead over single thread baseline: 1.61x
  - Speculative look-ahead over single thread baseline: 1.81x
- Speculative look-ahead over decoupled look-ahead: 1.13x
Speculative Parallelization in Look-ahead vs Main thread

- Skeleton provides *more opportunities* for parallelization
- Speedup of speculative parallelization
  - Speculative look-ahead over decoupled LA baseline: 1.13x
  - Speculative main thread over single thread baseline: 1.07x
Flexibility in Hardware Design

- Comparison of regular (partial versioning) cache support with two other alternatives
  - No cache versioning support
  - Dependence violation detection and squash
Other Details in the Paper

- The effect of spawning look-ahead thread in preserving the lead of overall look-ahead system

- Technique to avoid spurious spawns in dispatch stage which could be subjected to branch misprediction

- Technique to mitigate the damage of runaway spawns

- Impact of speculative parallelization on overall recoveries

- Modifications to branch queue in multiple look-ahead system
Outline

• Motivation
• Baseline decoupled look-ahead
• Speculative parallelization in look-ahead
• Experimental analysis
• Summary
Summary

- Decoupled look-ahead can significantly improve single-thread performance but look-ahead thread often becomes new bottleneck.

- Look-ahead thread lends itself to TLS acceleration:
  - Skeleton construction removes dependences and increases parallelism.
  - Hardware design is flexible and can be a simple extension of SMT.

- A straightforward implementation of TLS look-ahead achieves an average 1.13x (up to 1.39x) speedup.
Speculative Parallelization in Decoupled Look-ahead

Alok Garg, Raj Parihar, and Michael C. Huang

Dept. of Electrical & Computer Engineering
University of Rochester, Rochester, NY

http://www.ece.rochester.edu/projects/acal/
References (Partial)


References (cont…)


Backup Slides

For reference only
Register Renaming Support

Primary thread

<table>
<thead>
<tr>
<th>Register</th>
<th>0</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>p6</td>
<td>p7</td>
<td>p8</td>
</tr>
</tbody>
</table>

Spawned thread

<table>
<thead>
<tr>
<th>Register</th>
<th>0</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>p6</td>
<td>p7</td>
<td>p8</td>
</tr>
</tbody>
</table>

Register mapping table

Spawning

Next dispatch instruction

```
addl r6, 0x2, r7
subl r6, 0x4, r6
```
Modified Banked Branch Queue

Global head

Bank 0 tail

Bank 0

Bank 1 tail

Bank 1

Bank 2
# Speedup of all applications

<table>
<thead>
<tr>
<th></th>
<th>bzip2</th>
<th>crafty</th>
<th>eon</th>
<th>gap</th>
<th>gcc</th>
<th>gzip</th>
<th>mcf</th>
<th>pbmk</th>
<th>twolf</th>
<th>vortex</th>
<th>vpr</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.32</td>
<td>2.30</td>
<td>2.63</td>
<td>1.92</td>
<td>2.20</td>
<td>2.14</td>
<td>0.51</td>
<td>0.89</td>
<td>0.57</td>
<td>1.93</td>
<td>1.31</td>
</tr>
<tr>
<td>2</td>
<td>1.75</td>
<td>2.47</td>
<td>2.90</td>
<td>3.35</td>
<td>4.60</td>
<td>2.34</td>
<td>0.83</td>
<td>1.14</td>
<td>0.74</td>
<td>2.24</td>
<td>1.77</td>
</tr>
<tr>
<td>3</td>
<td>1.75</td>
<td>2.48</td>
<td>2.91</td>
<td>3.36</td>
<td>4.81</td>
<td>2.36</td>
<td>0.84</td>
<td>1.35</td>
<td>1.01</td>
<td>2.27</td>
<td>2.43</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ammp</th>
<th>applu</th>
<th>apsi</th>
<th>art</th>
<th>equake</th>
<th>fac</th>
<th>fma3d</th>
<th>galgel</th>
<th>lucas</th>
<th>mesa</th>
<th>mgrid</th>
<th>six</th>
<th>swim</th>
<th>wup</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.79</td>
<td>1.81</td>
<td>1.75</td>
<td>0.27</td>
<td>1.09</td>
<td>3.03</td>
<td>2.72</td>
<td>2.35</td>
<td>0.58</td>
<td>2.99</td>
<td>3.03</td>
<td>2.84</td>
<td>1.26</td>
<td>3.77</td>
</tr>
<tr>
<td>1.92</td>
<td>2.76</td>
<td>2.34</td>
<td>1.13</td>
<td>2.73</td>
<td>3.65</td>
<td>3.12</td>
<td>3.79</td>
<td>1.75</td>
<td>3.36</td>
<td>3.83</td>
<td>3.12</td>
<td>3.78</td>
<td>4.11</td>
</tr>
<tr>
<td>1.93</td>
<td>2.75</td>
<td>2.49</td>
<td>1.57</td>
<td>2.85</td>
<td>3.68</td>
<td>3.12</td>
<td>4.17</td>
<td>2.44</td>
<td>3.37</td>
<td>3.83</td>
<td>3.12</td>
<td>3.78</td>
<td>4.11</td>
</tr>
</tbody>
</table>
## Impact on Overall Recoveries

<table>
<thead>
<tr>
<th></th>
<th>INT</th>
<th>FP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Max</td>
<td>Avg.</td>
</tr>
<tr>
<td>Baseline look-ahead</td>
<td>3.21</td>
<td>1.24</td>
</tr>
<tr>
<td>Spec. parallel look-ahead</td>
<td>6.55</td>
<td>1.21</td>
</tr>
</tbody>
</table>
“Partial” Recovery in Look-ahead

- When recovery happens in primary look-ahead
  - Do we kill the secondary look-ahead or not?
  - If we don’t kill we gain around 1% performance improvement
  - After recovery in main, secondary thread survives often (1000 insts)
- Spawning of a secondary look-ahead thread helps preserving the *slip* of overall look-ahead system
Quality of Thread Spawning

- Successful and run-away spawns: killed after certain time

- Impact of **lazy** spawning policy
  - Wait for few cycles when spawning opportunity arrives
  - Avoids spurious spawns; Saves some wrong path execution

<table>
<thead>
<tr>
<th></th>
<th>Spawns invoked under incorrect path</th>
</tr>
</thead>
<tbody>
<tr>
<td>No disp.</td>
<td>11</td>
</tr>
<tr>
<td>Some disp.</td>
<td>28</td>
</tr>
<tr>
<td>WP</td>
<td>11</td>
</tr>
</tbody>
</table>