Accelerating Decoupled Look-ahead to Exploit Implicit Parallelism

Raj Parihar

Advanced Computer Architecture Lab
Department of Electrical & Computer Engineering
University of Rochester, Rochester, NY

June 26, 2013
Despite the proliferation of multi-core, multi-threaded systems

- High single-thread performance is still an important design goal
- Modern programs do not lack instruction level parallelism
- Real challenge: exploit implicit parallelism without undue costs
- One effective approach: Decoupled look-ahead

![Graph showing IPC performance for various benchmarks and thread counts]
Motivation

Despite the proliferation of multi-core, multi-threaded systems
- High single-thread performance is still an important design goal
- Modern programs do not lack instruction level parallelism
- Real challenge: exploit implicit parallelism without undue costs
- One effective approach: Decoupled look-ahead
Motivation: Decoupled Look-ahead

- Decoupled look-ahead architecture targets
  - Performance hurdles: branch mispredictions, cache misses, etc.
  - Exploration of parallelization opportunities, dependence information
  - Microarchitectural complexity, energy inefficiency through decoupling

The look-ahead thread can often become a new bottleneck. We explore techniques to accelerate the look-ahead thread. Speculative parallelization is aptly suited due to increased parallelism in the look-ahead binary. Weak dependence allows weak instruction removal without affecting the quality of look-ahead.
Motivation: Decoupled Look-ahead

- Decoupled look-ahead architecture targets
  - Performance hurdles: branch mispredictions, cache misses, etc.
  - Exploration of parallelization opportunities, dependence information
  - Microarchitectural complexity, energy inefficiency through decoupling

- The look-ahead thread can often become a new bottleneck
Motivation: Decoupled Look-ahead

- Decoupled look-ahead architecture targets
  - Performance hurdles: branch mispredictions, cache misses, etc.
  - Exploration of parallelization opportunities, dependence information
  - Microarchitectural complexity, energy inefficiency through decoupling

- The look-ahead thread can often become a new bottleneck

- We explore techniques to accelerate the look-ahead thread
  - Speculative parallelization: aptly suited due to increased parallelism in the look-ahead binary
  - Weak dependence: lack of correctness constraint allows weak instruction removal w/o affecting the quality of look-ahead
Outline

Motivation

Baseline decoupled look-ahead

Look-ahead thread acceleration
  Speculative parallelization in look-ahead
  Weak dependence removal in look-ahead

Experimental analysis

Future work & 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 memory image in local L1, no writeback to shared L2
  - Sends branch outcomes through FIFO queue; also helps prefetching

A. Garg and M. Huang, “A Performance-Correctness Explicitly Decoupled Architecture”, MICRO-08
Baseline Decoupled Look-ahead

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

A. Garg and M. Huang, “A Performance-Correctness Explicitly Decoupled Architecture”, MICRO-08
Practical Advantages of Decoupled Look-ahead

- Look-ahead thread is a self-reliant agent, completely independent of main thread
  - No need for quick spawning and register communication support
  - Low management overhead on main thread
  - Easier for run-time control to disable

<table>
<thead>
<tr>
<th></th>
<th>Cache misses</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>90%</td>
<td>95%</td>
<td>90%</td>
</tr>
<tr>
<td></td>
<td>DI</td>
<td>SI</td>
<td>DI</td>
</tr>
<tr>
<td>bzip2</td>
<td>1.86</td>
<td>17</td>
<td>3.15</td>
</tr>
<tr>
<td>crafty</td>
<td>0.73</td>
<td>23</td>
<td>1.04</td>
</tr>
<tr>
<td>eon</td>
<td>2.28</td>
<td>50</td>
<td>3.34</td>
</tr>
<tr>
<td>gap</td>
<td>1.35</td>
<td>15</td>
<td>1.44</td>
</tr>
<tr>
<td>gcc</td>
<td>8.49</td>
<td>153</td>
<td>8.84</td>
</tr>
<tr>
<td>gzip</td>
<td>0.1</td>
<td>6</td>
<td>0.1</td>
</tr>
<tr>
<td>mcf</td>
<td>13.1</td>
<td>13</td>
<td>14.7</td>
</tr>
<tr>
<td>parser</td>
<td>1.31</td>
<td>41</td>
<td>1.59</td>
</tr>
<tr>
<td>pbmk</td>
<td>1.87</td>
<td>35</td>
<td>2.11</td>
</tr>
<tr>
<td>twolf</td>
<td>2.69</td>
<td>23</td>
<td>3.28</td>
</tr>
<tr>
<td>vortex</td>
<td>1.96</td>
<td>42</td>
<td>2</td>
</tr>
<tr>
<td>vpr</td>
<td>7.47</td>
<td>16</td>
<td>11.6</td>
</tr>
<tr>
<td><strong>Avg</strong></td>
<td><strong>3.60%</strong></td>
<td><strong>36</strong></td>
<td><strong>4.44%</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Branch mispredictions</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>90%</td>
<td>95%</td>
<td>90%</td>
</tr>
<tr>
<td></td>
<td>DI</td>
<td>SI</td>
<td>DI</td>
</tr>
<tr>
<td>bzip2</td>
<td>3.9</td>
<td>52</td>
<td>4.49</td>
</tr>
<tr>
<td>crafty</td>
<td>5.33</td>
<td>235</td>
<td>6.14</td>
</tr>
<tr>
<td>eon</td>
<td>2.02</td>
<td>19</td>
<td>2.31</td>
</tr>
<tr>
<td>gap</td>
<td>2.02</td>
<td>77</td>
<td>2.64</td>
</tr>
<tr>
<td>gcc</td>
<td>8.08</td>
<td>1103</td>
<td>8.41</td>
</tr>
<tr>
<td>gzip</td>
<td>8.41</td>
<td>40</td>
<td>8.66</td>
</tr>
<tr>
<td>mcf</td>
<td>9.99</td>
<td>14</td>
<td>10.2</td>
</tr>
<tr>
<td>parser</td>
<td>6.81</td>
<td>130</td>
<td>7.3</td>
</tr>
<tr>
<td>pbmk</td>
<td>2.88</td>
<td>92</td>
<td>3.21</td>
</tr>
<tr>
<td>twolf</td>
<td>5.75</td>
<td>41</td>
<td>6.48</td>
</tr>
<tr>
<td>vortex</td>
<td>1.24</td>
<td>114</td>
<td>1.97</td>
</tr>
<tr>
<td>vpr</td>
<td>4.8</td>
<td>6</td>
<td>4.88</td>
</tr>
<tr>
<td><strong>Avg</strong></td>
<td><strong>5.10%</strong></td>
<td><strong>160</strong></td>
<td><strong>5.56%</strong></td>
</tr>
</tbody>
</table>
Practical Advantages of Decoupled Look-ahead

- Look-ahead thread is a self-reliant agent, completely independent of main thread
  - No need for quick spawning and register communication support
  - Low management overhead on main thread
  - Easier for run-time control to disable

- Natural throttling mechanism to prevent
  - Run-away_prefetching, cache pollution

<table>
<thead>
<tr>
<th>Cache misses</th>
<th>90%</th>
<th>95%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DI</td>
<td>S1</td>
</tr>
<tr>
<td>bzip2</td>
<td>1.86</td>
<td>17</td>
</tr>
<tr>
<td>crafty</td>
<td>0.73</td>
<td>23</td>
</tr>
<tr>
<td>eon</td>
<td>2.28</td>
<td>50</td>
</tr>
<tr>
<td>gap</td>
<td>1.35</td>
<td>15</td>
</tr>
<tr>
<td>gcc</td>
<td>8.49</td>
<td>153</td>
</tr>
<tr>
<td>gzip</td>
<td>0.1</td>
<td>6</td>
</tr>
<tr>
<td>mcf</td>
<td>13.1</td>
<td>13</td>
</tr>
<tr>
<td>parser</td>
<td>1.31</td>
<td>41</td>
</tr>
<tr>
<td>pbmk</td>
<td>1.87</td>
<td>35</td>
</tr>
<tr>
<td>twolf</td>
<td>2.69</td>
<td>23</td>
</tr>
<tr>
<td>vortex</td>
<td>1.96</td>
<td>42</td>
</tr>
<tr>
<td>vpr</td>
<td>7.47</td>
<td>16</td>
</tr>
<tr>
<td><strong>Avg</strong></td>
<td><strong>3.60%</strong></td>
<td><strong>36</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Branch mispredictions</th>
<th>90%</th>
<th>95%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DI</td>
<td>S1</td>
</tr>
<tr>
<td>bzip2</td>
<td>3.9</td>
<td>52</td>
</tr>
<tr>
<td>crafty</td>
<td>5.33</td>
<td>235</td>
</tr>
<tr>
<td>eon</td>
<td>2.02</td>
<td>19</td>
</tr>
<tr>
<td>gap</td>
<td>2.02</td>
<td>77</td>
</tr>
<tr>
<td>gcc</td>
<td>8.08</td>
<td>1103</td>
</tr>
<tr>
<td>gzip</td>
<td>8.41</td>
<td>40</td>
</tr>
<tr>
<td>mcf</td>
<td>9.99</td>
<td>14</td>
</tr>
<tr>
<td>parser</td>
<td>6.81</td>
<td>130</td>
</tr>
<tr>
<td>pbmk</td>
<td>2.88</td>
<td>92</td>
</tr>
<tr>
<td>twolf</td>
<td>5.75</td>
<td>41</td>
</tr>
<tr>
<td>vortex</td>
<td>1.24</td>
<td>114</td>
</tr>
<tr>
<td>vpr</td>
<td>4.8</td>
<td>6</td>
</tr>
<tr>
<td><strong>Avg</strong></td>
<td><strong>5.10%</strong></td>
<td><strong>160</strong></td>
</tr>
</tbody>
</table>
Practical Advantages of Decoupled Look-ahead

- Look-ahead thread is a self-reliant agent, completely independent of main thread
  - No need for quick spawning and register communication support
  - Low management overhead on main thread
  - Easier for run-time control to disable

- Natural throttling mechanism to prevent
  - Run-away prefetching, cache pollution

- Look-ahead thread size comparable to aggregation of short helper threads

<table>
<thead>
<tr>
<th></th>
<th>Cache misses</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>90%</td>
</tr>
<tr>
<td></td>
<td>DI</td>
</tr>
<tr>
<td>bzip2</td>
<td>1.86</td>
</tr>
<tr>
<td>crafty</td>
<td>0.73</td>
</tr>
<tr>
<td>eon</td>
<td>2.28</td>
</tr>
<tr>
<td>gap</td>
<td>1.35</td>
</tr>
<tr>
<td>gcc</td>
<td>8.49</td>
</tr>
<tr>
<td>gzip</td>
<td>0.1</td>
</tr>
<tr>
<td>mcf</td>
<td>13.1</td>
</tr>
<tr>
<td>parser</td>
<td>1.31</td>
</tr>
<tr>
<td>pbmk</td>
<td>1.87</td>
</tr>
<tr>
<td>twolf</td>
<td>2.69</td>
</tr>
<tr>
<td>vortex</td>
<td>1.96</td>
</tr>
<tr>
<td>vpr</td>
<td>7.47</td>
</tr>
<tr>
<td><strong>Avg</strong></td>
<td><strong>3.60%</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Branch mispredictions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>90%</td>
</tr>
<tr>
<td></td>
<td>DI</td>
</tr>
<tr>
<td>bzip2</td>
<td>3.9</td>
</tr>
<tr>
<td>crafty</td>
<td>5.33</td>
</tr>
<tr>
<td>eon</td>
<td>2.02</td>
</tr>
<tr>
<td>gap</td>
<td>2.02</td>
</tr>
<tr>
<td>gcc</td>
<td>8.08</td>
</tr>
<tr>
<td>gzip</td>
<td>8.41</td>
</tr>
<tr>
<td>mcf</td>
<td>9.99</td>
</tr>
<tr>
<td>parser</td>
<td>6.81</td>
</tr>
<tr>
<td>pbmk</td>
<td>2.88</td>
</tr>
<tr>
<td>twolf</td>
<td>5.75</td>
</tr>
<tr>
<td>vortex</td>
<td>1.24</td>
</tr>
<tr>
<td>vpr</td>
<td>4.8</td>
</tr>
<tr>
<td><strong>Avg</strong></td>
<td><strong>5.10%</strong></td>
</tr>
</tbody>
</table>
Look-ahead: A New Bottleneck

- Comparing four systems to discover new bottlenecks
  - Single-thread, decoupled look-ahead, ideal, and look-ahead alone
- Application categories:
  - Bottleneck removed or speed of look-ahead is not an issue
  - **Look-ahead thread is the new bottleneck** (right half)
Look-ahead: A New Bottleneck

- Comparing four systems to discover new bottlenecks
  - Single-thread, decoupled look-ahead, ideal, and look-ahead alone

- Application categories:
  - Bottleneck removed or speed of look-ahead is not an issue
  - **Look-ahead thread is the new bottleneck** (right half)
Look-ahead: A New Bottleneck

- Comparing four systems to discover new bottlenecks
  - Single-thread, decoupled look-ahead, ideal, and look-ahead alone
- Application categories:
  - Bottleneck removed or speed of look-ahead is not an issue
  - **Look-ahead thread is the new bottleneck** (right half)
Look-ahead: A New Bottleneck

- Comparing four systems to discover new bottlenecks
  - Single-thread, decoupled look-ahead, ideal, and look-ahead alone
  - Application categories:
    - Bottleneck removed or speed of look-ahead is not an issue
    - **Look-ahead thread is the new bottleneck** (right half)

![Graph showing IPC for different benchmarks with Look-ahead Speed Limit, Single Thread, Decoupled Look-ahead, and Ideal (cache, branch)]
Unique Opportunities for Speculative Parallelization

- Skeleton code offers more parallelism
- Certain dependencies removed during slicing for skeleton
- Short-distance dependence chains become long-distance chains, suitable for TLP exploitation

A. Garg, R. Parihar, M. Huang, “Speculative Parallelization in Decoupled Look-ahead”, PACT-11
Unique Opportunities for Speculative Parallelization

- Skeleton code offers more parallelism
  - Certain dependencies removed during slicing for skeleton
  - Short-distance dependence chains become long-distance chains, suitable for TLP exploitation

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

A. Garg, R. Parihar, M. Huang, “Speculative Parallelization in Decoupled Look-ahead”, PACT-11
Software Support

- Dependence analysis
  - Profile guided, coarse-grain at basic block level

![Diagram of dependence analysis with nodes a, b, c, d, e, f and edges showing dependence distances and spawn and target points.](image)
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 $D_{MIN}$
  - Spawned thread executes from target point
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 $D_{MIN}$
  - Spawned thread executes from target point
- Loop level parallelism is also exploited

Diagram:

- Nodes a, b, c, d, e, f connected with directed edges representing data dependencies.
- Node a is spawned at time 1 and executes before node b, which has $D_{MIN}=1$.
- Node c is the target point with $D_{MIN}=3$.
- Loop level parallelism is also shown with dependencies between nodes d and e.

Raj Parihar
Accelerating Decoupled Look-ahead to Exploit Implicit Parallelism
Parallelism Potential in Look-ahead Binary

- Available parallelism for 2 core/contexts system: $D_{MIN} = 15BB$
Parallelism Potential in Look-ahead Binary

- Available parallelism for 2 core/contexts system: \( D_{MIN} = 15BB \)
- Skeleton exhibits significant more BB level parallelism (17%)

![Graph showing approximate parallelism for different benchmarks]

**Motivation**
- Baseline decoupled look-ahead
- Look-ahead thread acceleration
- Experimental analysis
- Future work & summary

**Speculative parallelization in look-ahead**
- Weak dependence removal in look-ahead

---

**Raj Parihar**

Accelerating Decoupled Look-ahead to Exploit Implicit Parallelism
Parallelism Potential in Look-ahead Binary

- Available parallelism for 2 core/contexts system; $D_{MIN} = 15BB$
- Skeleton exhibits significant more BB level parallelism (17%)
- Loop based FP applications exhibit more BB level parallelism

![Graph showing approximate parallelism for different applications between Original binary and Skeleton]
Thread spawning and merging are very similar to regular thread spawning except:

- Spawned thread shares the same register and memory state
- Spawning thread terminates at the target PC
Hardware and Runtime Support

- Thread spawning and merging are very similar to regular thread spawning except
  - Spawned thread shares the same register and memory state
  - Spawning thread terminates 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 process]

- Lookahead thread 0
- Lookahead thread 1
- Time
- Spawn
- Duplicate rename table and set up context
- Merge
- Cleanup duplicated state
Applications in which the look-ahead thread is a bottleneck
Speedup of Speculative Parallelization

- Applications in which the look-ahead thread is a bottleneck
- Speedup of look-ahead systems over single-thread
  - Decoupled look-ahead over single-thread baseline: **1.53x**
Applications in which the look-ahead thread is a bottleneck

- Speedup of look-ahead systems over single-thread
  - Decoupled look-ahead over single-thread baseline: **1.53x**
  - Speculative parallel look-ahead over single-thread: **1.73x**
Motivation
Baseline decoupled look-ahead
Look-ahead thread acceleration
Experimental analysis
Future work & summary

Speedup of Speculative Parallelization

- Applications in which the look-ahead thread is a bottleneck
- Speedup of look-ahead systems over single-thread
  - Decoupled look-ahead over single-thread baseline: 1.53x
  - Speculative parallel look-ahead over single-thread: 1.73x
- Speculative look-ahead over decoupled look-ahead: 1.13x
Skeleton provides **more opportunities** for parallelization

![Speedup over respective baseline](chart.png)

- Speculatively parallel main: 1.65
- Speculatively parallel look-ahead: 1.07
- Crafty, eon, gzip, mcf, pbmk, twolf, vortex, vpr, ammp, art, eqk, fma3d, galgel, lucas, gmean
- Speculative parallelization in look-ahead
- Weak dependence removal in look-ahead

---

**Raj Parihar**

**Accelerating Decoupled Look-ahead to Exploit Implicit Parallelism**

---

**Motivation**
- Baseline decoupled look-ahead
- Look-ahead thread acceleration
- Experimental analysis
- Future work & summary

**Speculative Look-ahead vs Conventional TLS**
Speculative Look-ahead vs Conventional TLS

- Skeleton provides **more opportunities** for parallelization
- Speculative look-ahead over decoupled LA baseline: **1.13x**

![Graph showing speedup over respective baseline for different benchmarks (crafty, eon, gzip, mcf, pbmk, twolf, vortex, vpr, ammp, art, eqk, fma3d, galgel, lucas, gmean). The x-axis represents different benchmarks, and the y-axis represents speedup over respective baseline. The graph shows speedup values ranging from 0.8 to 1.65, with some benchmarks showing significant speedup like crafty, eon, gzip, and mcf.]
Speculative Look-ahead vs Conventional TLS

- Skeleton provides **more opportunities** for parallelization
- Speculative look-ahead over decoupled LA baseline: **1.13x**
- Speculative main thread over single thread baseline: **1.07x**
Motivation for Exploiting Weak Dependences

- Not all instructions are equally important and critical
- Example of weak instructions:
  - Inconsequential adjustments
  - Load and store instructions that are (mostly) silent
  - Dynamic NOP instructions

---

R. Parihar, M. Huang, “Accelerating Decoupled Look-ahead via Weak Dependence Removal”. Submitted to MICRO-13
Motivation for Exploiting Weak Dependences

- Not all instructions are equally important and critical
- Example of weak instructions:
  - Inconsequential adjustments
  - Load and store instructions that are (mostly) silent
  - Dynamic NOP instructions
- Plenty of weak instructions are present in programs
- Challenges involved:
  - Context-based, hard to identify and combine – much like Jenga

Comparison of Weak and Strong Instructions

- Static attributes of weak and strong insts are *remarkably* same
  - Static attributes: opcode, number of inputs
  - The correlation coefficient of the two distributions is **0.96**
Comparison of Weak and Strong Instructions

- Static attributes of weak and strong insts are *remarkably* same
  - Static attributes: opcode, number of inputs
  - The correlation coefficient of the two distributions is **0.96**
- Weakness has very poor correlation with static attributes
- Hard to identify the weak instructions through static heuristics
Genetic Algorithm based Framework

Genetic algorithm based framework to identify and eliminate weak instructions from the look-ahead skeleton.
Genetic Algorithm based Framework

- Genetic algorithm based framework to identify and eliminate weak instructions from the look-ahead skeleton
- Genetic evolution: procreation and natural selection
Genetic Algorithm based Framework

- Genetic algorithm based framework to identify and eliminate weak instructions from the look-ahead skeleton
  - Genetic evolution: procreation and natural selection
  - Chromosomes creation and hybridization
  - Baseline look-ahead skeleton construction

Diagram:
- Program Binary
- Look-ahead Binary
  - (Binary Parser)
- Multi-Instruction Genes
- Single-Instruction Genes
- Initial Chromosome Population
- Roulette Wheel
- Parent selection
- Xover & Mutation
- De-duplication
- Fitness test, Elitism
- Children Pool
- Parents Pool
- Reproduction

Motivation
- Baseline decoupled look-ahead
- Look-ahead thread acceleration
- Experimental analysis
- Future work & summary

Speculative parallelization in look-ahead
Weak dependence removal in look-ahead
Heuristic Based Solutions

- Heuristic based solutions are helpful to *jump start* the evolution
- Superposition based chromosomes
- Orthogonal subroutine based chromosomes
Optimizations to Implementation

- Fitness test optimizations
  - Sampling based fitness
  - Multi-instruction genes
  - Early termination of tests

- GA framework optimizations
  - Hybridization of solutions
  - Adaptive mutation rate
  - Unique chromosomes
  - Fusion crossover operator
  - Elitism policy
Progress of Genetic Evolution Process

- Per generation progress compared to the final best solution
  - After 2 generations, more than half of the benefits are achieved
  - After 5 generations, at least 90% of benefits are achieved

![Progress relative to best GA solution](image-url)
Experimental Setup

- **Program/binary analysis tool:** ALTO
- **Simulator:** based on heavily modified SimpleScalar
  - SMT, look-ahead and speculative parallelization support
  - True execution-driven simulation (faithfully value modeling)
- **Genetic algorithm framework**
  - Modeled as offline and online extension to the simulator

| Baseline core | Look-ahead core:
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch/Decode/Issue/Commit</td>
<td>8 / 4 / 6 / 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>Fetch Q/ Issue Q / Reg. (int,fp) (LQ,SQ)</td>
<td>(32, 32) / (32, 32) / (80, 80)</td>
</tr>
<tr>
<td>LSQ(LQ,SQ)</td>
<td>64 (32,32) 2 search ports</td>
</tr>
<tr>
<td>Branch predictor</td>
<td>Gshare – 8K entries, 13 bit history at least 7 cycles</td>
</tr>
<tr>
<td>Br. mispred. penalty</td>
<td>32KB, 4-way, 64B line, 2 cycles, 2 ports</td>
</tr>
<tr>
<td>L1 data cache (private)</td>
<td>64KB, 2-way, 128B, 2 cycles</td>
</tr>
<tr>
<td>L1 inst cache (private)</td>
<td>1MB, 8-way, 128B, 15 cycles</td>
</tr>
<tr>
<td>L2 cache (shared)</td>
<td>400 cycles</td>
</tr>
<tr>
<td>Memory access latency</td>
<td>Baseline core with only LQ, no SQ</td>
</tr>
</tbody>
</table>

| Communication: | Branch Output Queue: 512 entries |
|----------------| Reg copy latency (recovery): 64 cycles |

**Table 1: Microarchitectural configurations.**

![Genetic Algorithm Program](image-url)
Speedup of Self-tuned Look-ahead

Applications in which the look-ahead thread is a bottleneck

![Graph showing speedup comparison between baseline look-ahead and GA based look-ahead]
Speedup of Self-tuned Look-ahead

- Applications in which the look-ahead thread is a bottleneck
- Self-tuned, genetic algorithm based decoupled look-ahead
- Speedup over baseline decoupled look-ahead: **1.16x**
Applications in which the look-ahead thread is a bottleneck

Self-tuned, genetic algorithm based decoupled look-ahead

- Speedup over baseline decoupled look-ahead: **1.16x**
- Speedup over single-thread baseline: **1.78x**
Comparison with Speculative Parallel Look-ahead

- Self-tuned skeleton is used in the speculative parallel look-ahead
- In some cases, self-tuned and speculative parallel look-ahead techniques are synergistic (ammp, art)
Decoupled Look-ahead Exploration Space

- Speculative look-ahead and weak dependence can be combined synergistically to improve overall look-ahead paradigm.
- Using two cores for hard-to-parallelize SPEC applications: 1.84x

Speculative Parallelization

<table>
<thead>
<tr>
<th>Self Tuning</th>
<th>No</th>
<th>Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>Decoupled look-ahead 1.53x</td>
<td>Speculative DLA 1.73x</td>
</tr>
<tr>
<td>Yes</td>
<td>Self-tuned DLA 1.78x</td>
<td>Speculative + Weak inst 1.84x</td>
</tr>
<tr>
<td></td>
<td>Speculative + Self tuning ?</td>
<td></td>
</tr>
</tbody>
</table>
Ongoing and Future Explorations

- Load balancing through skipping non-critical branches
- Weak instruction classification, identification, and prediction
- Self-tuning based thread-level speculation in look-ahead thread
- Single-core version of decoupled look-ahead

- \textit{Shared cache management in multi-threaded systems}
  - “Coldness” metric for cache optimizations
Load Balancing in Look-ahead

- Non-critical branches can be transformed to accelerate the look-ahead thread and achieve better load balancing
  - Non-critical branches are skipped in the look-ahead thread
  - Main thread executes such branches by itself w/o any help
- We call these branches *Do-It-Yourself* or *DIY* branches

(A) Forward conditional branch (If-Than, If-Than-Else) transformations
(B) Backward conditional branch (Loop) transformations
Preliminary Speedup of Load Balancing

- Load balancing through skipping of the non-critical branches
  - Max. speedup over decoupled look-ahead: 1.76x \( (\text{art}) \)
  - Avg. speedup over decoupled look-ahead: 1.12x \( (\text{Gmean}) \)
Motivation: Coldness Metric

- A **hot** concept in program optimization is hotness
  - Program optimization (hot paths), register allocation (hot variables)

- In contrast, cache optimization has to target **cold** data
  - Hot data, due to frequent usage, tend to stay in caches
  - Majority of the misses in last level cache are caused by cold data

- In this work, we propose
  - A new metric called **coldness** to aid cache optimizations
  - To quantify the coldness variation as the cache size increases
Definition: Coldness Metric

For a cache size $c$, the minimal number of “distinct” addresses to target $r\%$ reduction in the miss ratio is defined as **coldness**

$$\text{coldness}(c, r) = (-1) \times \#\text{uniq\_addr}$$

R. Parihar, C. Ding, M. Huang, “Coldness Metric for Cache Optimizations”, *MSPC’13*
Coldness Measurement

Coldness of SPEC 2006 applications as a function of cache size

(a). Distinct addresses accounting for top 10% misses

(b). Distinct addresses accounting for top 50% misses

Classification based on coldness and # of distinct addresses:

- Less cold: h264ref, sphinx2, astar, gobmk, hmmer, dealII, namd
- Highly cold: lbm, bwaves, perlbench, mcf, soplex, sjeng, libq
Coldness Analysis and Insights

- Cache size (↑) ⇒ Coldness (↓)
  - Coldness for 10% misses: -15 (1KB) ⇒ -4630 (4MB)
  - Coldness for 90% misses: -11K (1KB) ⇒ -50K (4MB)
  - Large cache optimization ⇒ large # of distinct addresses

- Optimization target (↑) ⇒ Coldness (↓)
  - For a 4MB cache, must optimize accesses to at least 5.4MB data to reduce miss ratio by 90% (data size > cache size!).

An effective solution to colder misses is decoupled look-ahead

- Reduces primary misses by 88x and secondary misses by 38x for a 4MB cache (with >100K miss addresses)
Approximable Program Paradigm

- Weak dependence removal and speculative parallelization techniques can be applied to any approximate program.
- Few real-life examples of approximate computing:
  - Google search: does not work with a coherent, up-to-date database.
  - Map-Reduce paradigm: ignores consistently failing records.
  - Media applications: photo, audio and video have some tolerance.
- Algorithm and applications level approximations:
  - Modern benchmarks e.g., PARSEC are fundamentally approximate.
  - Applications space: clustering, predictions, optimizations, etc.
  - **BenchNN**: a neural network based alternative to PARSEC.
Summary

- Decoupled look-ahead can uncover significant implicit parallelism
  - However, look-ahead thread often becomes a new bottleneck

- Fortunately, look-ahead lends itself to various optimizations:
  - Speculative parallelization is more beneficial in look-ahead thread
  - Weak instructions can be removed w/o affecting look-ahead quality

- **Intelligent** look-ahead technique is a promising solution in the era of flat frequency and modest microarchitecture scaling

- Idle cores in multicore environment will further strengthen the case of decoupled look-ahead adoption in mainstream systems
Acknowledgments

- Funding agencies: NSF, NSFC
- Prof. Michael C. Huang
- Alok Garg
- Prof. Chen Ding and his research group at URCS
- Past & current members of Advanced Computer Architecture Lab: Dulce Ponceleon, Jay Geagan (IBM Research - Almaden)
References (Partial)

- **Decoupled Access/Execute Computer Architectures**
  J. Smith, ACM TC’84

- **A Performance-Correctness Explicitly Decoupled Architecture**
  A. Garg, M. Huang, MICRO’08

- **Speculative Parallelization in Decoupled Look-ahead**
  A. Garg, R. Parihar, M. Huang, PACT’11

- **Accelerating Decoupled Look-ahead via Weak Dependence Removal: A Metaheuristic Approach**
  R. Parihar, M. Huang, (Submitted to MICRO’13)

- **Coldness metric for Cache Optimization**
  R. Parihar, C. Ding, M. Huang, ACM SIGPLAN MSPC’13
Accelerating Decoupled Look-ahead to Exploit Implicit Parallelism

Raj Parihar

Advanced Computer Architecture Lab
Department of Electrical & Computer Engineering
University of Rochester, Rochester, NY

June 26, 2013
Microthreads vs Decoupled Look-ahead

Lightweight Microthreads:

- Process
- Thread #1
- Thread #2
- Microthreads

Decoupled Look-ahead:

- Register state synchronization
- Look-ahead Core
  - Branch predictions
  - Prefetching hints
- Main Core
  - L1
  - L2
## Look-ahead Skeleton Construction

<table>
<thead>
<tr>
<th>Address</th>
<th>Skeleton</th>
<th>Original Binary</th>
<th>Skeleton</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x12001f99c:</td>
<td>\texttt{addq v0, v0, v0} \texttt{subq v0, t0, a2} \texttt{cmovvge a2, a2, v0} \texttt{addq v0, v0, v0} \texttt{subq v0, t0, a2} \ldots \ldots \ldots</td>
<td>... \ldots \ldots \texttt{bic t2, 0x3f, t2} \texttt{addl v0, 0x1, s1} \texttt{stl s1, 68(sp)} \texttt{lsl} \texttt{a2, 68(sp)} \texttt{xor} \texttt{s1, a2, a2} \texttt{bne a2, Label}</td>
<td>... \ldots \ldots \texttt{bic t2, 0x3f, t2} \texttt{NOP} \texttt{NOP} \texttt{lsl} \texttt{a2, 68(sp)} \texttt{xor} \texttt{s1, a2, a2} \texttt{bne a2, Label}</td>
</tr>
<tr>
<td>...</td>
<td>\texttt{Loop Body} Biased conditional branch</td>
<td>\texttt{Long distance load-store pair}</td>
<td></td>
</tr>
<tr>
<td>0x12001f9bc:</td>
<td>\texttt{bgt a1, 0x12001f9a0} \texttt{subq v0, t0, a2}</td>
<td>\ldots \ldots \ldots</td>
<td></td>
</tr>
<tr>
<td>0x12001f9c0:</td>
<td></td>
<td>\ldots \ldots \ldots</td>
<td></td>
</tr>
</tbody>
</table>

(A) Illustration of a biased conditional branch (loop) turned into unconditional branch in the skeleton.

(B) Illustration of a long communication store and its consumer load computation optimization in skeleton.
Speculative Parallelization: Cortex-A9 vs POWER5

<table>
<thead>
<tr>
<th></th>
<th>Cortex-A9 (3-way OoO)</th>
<th>POWER5 (4-way OoO)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipeline width</td>
<td>4 / 2 / 3 / 3</td>
<td>8 / 4 / 6 / 6</td>
</tr>
<tr>
<td>Queue size: fetch</td>
<td>16 / 8 / 8</td>
<td>32 / 32 / 32</td>
</tr>
<tr>
<td>Registers: Regs:int</td>
<td>56 / 56 / 64 / 32</td>
<td>80 / 80 / 128 / 64</td>
</tr>
<tr>
<td>Registers: Regs:fp</td>
<td>2 / 2</td>
<td>4 / 4</td>
</tr>
<tr>
<td>Caches (kB): DL0</td>
<td>16 / 32 / 512 / 32</td>
<td>32 / 64 / 1024 / 64</td>
</tr>
<tr>
<td>Caches (kB): DL1</td>
<td>1k / 512</td>
<td>8k / 2k</td>
</tr>
<tr>
<td>Caches (kB): DL2</td>
<td>2k / 4k / 32 / 1k</td>
<td>4k / 8k / 32 / 4k</td>
</tr>
<tr>
<td>Caches (kB): IL1</td>
<td>2k / 4k / 32 / 1k</td>
<td>4k / 8k / 32 / 4k</td>
</tr>
</tbody>
</table>

Graph showing speedup over single thread for various benchmarks.
Flexibility in Look-ahead Hardware Design

- Comparison of regular (partial versioning) cache support with two other alternatives
  - No cache versioning support
  - Dependence violation detection and squash
Partial Recoveries and Spawns

**Partial recoveries:**

<table>
<thead>
<tr>
<th></th>
<th>gap</th>
<th>mcf</th>
<th>pbm</th>
<th>twf</th>
<th>vor</th>
<th>vpr</th>
<th>amp</th>
<th>eqk</th>
<th>fac</th>
<th>fma</th>
<th>gal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recv-Merge</td>
<td>1116</td>
<td>1309</td>
<td>6408</td>
<td>57910</td>
<td>4862</td>
<td>12314</td>
<td>4252</td>
<td>9174</td>
<td>925</td>
<td>4062</td>
<td>291</td>
</tr>
<tr>
<td>Live 200</td>
<td>1031</td>
<td>1308</td>
<td>3642</td>
<td>39305</td>
<td>4801</td>
<td>11284</td>
<td>2970</td>
<td>4347</td>
<td>526</td>
<td>4055</td>
<td>290</td>
</tr>
<tr>
<td>Live 1000</td>
<td>917</td>
<td>1306</td>
<td>3355</td>
<td>17601</td>
<td>3774</td>
<td>10178</td>
<td>2626</td>
<td>3495</td>
<td>522</td>
<td>4041</td>
<td>290</td>
</tr>
</tbody>
</table>

**Breakdown of all the spawns:**

<table>
<thead>
<tr>
<th></th>
<th>mcf</th>
<th>pbmk</th>
<th>twolf</th>
<th>vortex</th>
<th>vpr</th>
<th>ammp</th>
<th>art</th>
<th>equake</th>
<th>fma3d</th>
<th>galgel</th>
<th>lucas</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spawns invoked under correct path</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Successful</td>
<td>2297</td>
<td>26873</td>
<td>21067</td>
<td>1273</td>
<td>42082</td>
<td>6328</td>
<td>29598</td>
<td>16676</td>
<td>9687</td>
<td>20997</td>
<td>24022</td>
</tr>
<tr>
<td>Runaway</td>
<td>257</td>
<td>245</td>
<td>1738</td>
<td>37</td>
<td>409</td>
<td>3542</td>
<td>363</td>
<td>0</td>
<td>3965</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Spawns invoked under incorrect path</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No dispatch</td>
<td>11</td>
<td>707</td>
<td>2837</td>
<td>96</td>
<td>1633</td>
<td>26</td>
<td>29</td>
<td>245</td>
<td>363</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Few dispatch</td>
<td>28</td>
<td>69</td>
<td>1803</td>
<td>6</td>
<td>273</td>
<td>45</td>
<td>116</td>
<td>10</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Wrong Path</td>
<td>11</td>
<td>184</td>
<td>2997</td>
<td>152</td>
<td>111</td>
<td>339</td>
<td>6</td>
<td>62</td>
<td>4</td>
<td>17</td>
<td>0</td>
</tr>
</tbody>
</table>
Sampling based Fitness Test

- **perlbmk**
  - Precise line
  - Sampling line

- **vpr**
  - Precise line
  - Sampling line

- **equake**
  - Precise line
  - Sampling line

- **mcf**
  - Precise line
  - Sampling line
L2 Cache Sensitivity Study

- Speedup for various L2 caches is quite stable
  - 1.161x (1 MB), 1.154x (2 MB), and 1.152x (4 MB) L2 caches
- Avg. speedups, shown in the figure, are relative to single-threaded execution with a 1 MB L2 cache
Other Details

- Energy reduction: 11% over baseline decoupled look-ahead
  - Reduced cache accesses, less stalling of main thread
- On an average, 10% of the dynamic instructions are removed from the baseline skeleton
- Offline profiling and control software overhead
  - Offline profiling time: 2 to 20 seconds on the target machine
  - Online control software: 17 million instructions for whole evolution
- Average extra recoveries: 3-4 per 100,000 instructions

<table>
<thead>
<tr>
<th></th>
<th>INT (8 apps)</th>
<th>FP (6 apps)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Max</td>
<td>Avg.</td>
</tr>
<tr>
<td>Baseline look-ahead</td>
<td>7.98</td>
<td>2.68</td>
</tr>
<tr>
<td>GA tuned look-ahead</td>
<td>11.20</td>
<td>3.14</td>
</tr>
</tbody>
</table>