Accelerating Approximable Programs through Evolutionary Algorithms

Raj Parihar

Advisor: Prof. Michael C. Huang

Advanced Computer Architecture Lab
Department of Electrical & Computer Engineering

April 19, 2013
Motivation

- Many real-life applications can tolerate approximate results
- Few examples of approximate computing
  - Google search: does not work with a coherent, up-to-date database
  - Map-Reduce paradigm: ignores consistently failing records
  - Path-finding techniques: often provide good solutions, not the "best"
  - Media applications: photo, audio and video have some tolerance
Motivation

Many real-life applications can tolerate approximate results

Few examples of approximate computing

- Google search: does not work with a coherent, up-to-date database
- Map-Reduce paradigm: ignores consistently failing records
- Path-finding techniques: often provide good solutions, not the “best”
- Media applications: photo, audio and video have some tolerance

Applications that have **ephemeral** and **statistical** output tend to tolerate approximations and occasional errors

Approximate computing in which we are interested

- Helper threading and **look-ahead** architectures
- A secondary thread provides high quality hints to main thread
Abstractions of Approximate Computing

- **Algorithmic level**
  - Modern benchmarks *e.g.*, PARSEC are fundamentally approximate
  - Applications space: clustering, predictions, optimizations, etc.
  - **BenchNN**: a neural network based alternative to PARSEC
    - Replaces computational kernel with an equivalent neural network
    - Still achieves more than 99% accuracy at reasonable speedup
    - [http://www.benchnn.org/about.php](http://www.benchnn.org/about.php)

- **Architecture level**
  - Neural network based accelerator to offload the approximable code
  - **Parrot** transformation: A neural network learns the behavior of a code region and replaces it with equivalent neural network
Outline

Motivation

Baseline decoupled look-ahead

Look-ahead thread acceleration

Weak dependence removal 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 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
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 various applications]

Raj Parihar

Accelerating Approximable Programs through Evolutionary Algorithms
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 systems and workloads]
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)
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

Motivation
Baseline decoupled look-ahead
Look-ahead thread acceleration
Experimental analysis
Summary

Weak dependence removal in look-ahead
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 insts 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
Heuristic Based Solutions

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

![Diagram showing single-instruction and multi-instruction genes in initial chromosomes](image-url)
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
Experimental Setup

- Program analysis: ALTO based
- Simulator: SimpleScalar based
  - SMT, look-ahead support
- Genetic algorithm framework
  - A hyper-visor for the simulator

Hybrid Genetic Algorithm

1. Initial Seeds
2. Remove Genes
3. Notify HGA
4. Launch fitness test
5. Collect fitness score

Binary Parser
Fitness Evaluator

Microarchitectural config:

<table>
<thead>
<tr>
<th></th>
<th>Baseline core</th>
<th>Look-ahead core: Baseline core with L0 cache (16KB, 4-way, 32B line, 2 cycle, 2 ports). Round trip latency to L1 is 6 cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch/Decode/Commit ROB</td>
<td>8 / 4 / 6</td>
<td>8 / 4 / 6</td>
</tr>
<tr>
<td>Functional units</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>Issue Q / Reg. (int,fp)</td>
<td>(32, 32) / (120, 120)</td>
<td>(32, 32) / (120, 120)</td>
</tr>
<tr>
<td>LSQ(LQ,SQ)</td>
<td>64 (32,32) 2 search ports</td>
<td>64 (32,32) 2 search ports</td>
</tr>
<tr>
<td>Branch predictor</td>
<td>Bimodal + Gshare</td>
<td>Bimodal + Gshare</td>
</tr>
<tr>
<td>- Gshare</td>
<td>8K entries, 13 bit history</td>
<td>8K entries, 13 bit history</td>
</tr>
<tr>
<td>- Bimodal/Meta/ BTB</td>
<td>4K/8K/4K (4-way) entries</td>
<td>4K/8K/4K (4-way) entries</td>
</tr>
<tr>
<td>Br. mispred. penalty</td>
<td>at least 7 cycles</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>
<td>32KB, 4-way, 32B line, 2 cycle, 2 ports. Round trip latency to L1 is 6 cycles</td>
</tr>
<tr>
<td>L1 I cache (not shared)</td>
<td>64KB, 1-way, 128B, 2 cyc</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>
<td>1MB, 8-way, 128B, 15 cyc</td>
</tr>
<tr>
<td>Memory access latency</td>
<td>400 cycles</td>
<td>32 cycles</td>
</tr>
</tbody>
</table>

Communication: BOQ: 512 entries; PAB: 256 entries; register copy latency (during recovery): 32 cycles
Speedup of Self-tuned Look-ahead

- Applications in which the look-ahead thread is a bottleneck
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
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
  - 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*)
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:
  - 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.
- Weak dependence idea can be applied to other approximable programs without degrading the overall quality.
References

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

- A Study of Slipstream Processors
  Z. Purser, K. Sundaramoorthy, E. Rotenberg, MICRO’00

- Master/Slave Speculative Parallelization
  C. Zilles, G. Sohi, MICRO’02

- 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
  R. Parihar, M. Huang, (Submitted)