Accelerating Approximable Programs through Genetic Algorithms

Raj Parihar

Advisor: Prof. Michael C. Huang

Advanced Computer Architecture Lab
Department of Electrical & Computer Engineering

April 18, 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
  - Path-finding techniques: often provide good solutions, not the “best”
  - Media applications: photo, audio and video have some tolerance
- Approximate computing in which we are interested
  - Helper threading and **look-ahead** architectures
  - A secondary thread provides high quality hints to main thread
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 Agent: A New Bottleneck

- Comparing four systems to discover new bottlenecks
  - Single-thread, decoupled look-ahead, ideal, and look-ahead alone
  - Look-ahead thread is the new bottleneck (right half)
  - “Help the helper thread” w/o compromising look-ahead quality
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
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
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
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 dependences can be removed to accelerate approximable programs without degrading the overall quality