CAP: Criticality Analysis for Power-Efficient Speculative Multithreading

James Tuck, Wei Liu, Josep Torrellas
University of Illinois at Urbana-Champaign

Motivation

- Speculative Multithreading (SM) is important for CMPs
  - It can speedup hard-to-parallelize programs
- Power inefficiency of SM is a serious concern
Propositions

- We can improve power efficiency using criticality analysis
  - Some threads matter more for performance than others
- Dynamically construct a graph of SM execution and calculate criticality
- Schedule tasks on a CMP using criticality
  - DVFS per-core for power-efficiency
  - Schedule critical tasks to higher V-f cores, non-critical to lower V-f
Contributions

- Novel, widely-applicable task critical model for Speculative Multithreading
- CAP architecture for a CMP that implements our proposed model
- Evaluation of SPECint2000
  - We reduce average power by Geo.Mean of 22%
  - Average slow down of 2.6%
  - ED^2 reduced on average 15%
- Characterize task criticality composition of different applications
Task-Level Criticality Model

- Model execution at the level of task events
  - Events of interest: spawn, commit, squash
  - Keeps overhead low compared to instruction-level schemes

- Seamlessly handle a variety of SM systems
  - In-order vs. out-of-order spawns
  - Scheduling mechanism
    - Round robin
    - First available core

- Efficient hardware implementation
Lifetime of SM Task
Criticality Graph Summary

- **Nodes**
  - Stages of the task's execution
    - Start, Execute, Finish/Spawn, Commit

- **Edges**
  - Transitioning between states in a single task
  - Events between tasks
    - Spawn, squash, commit, freeing a resource, wait to become safe
CAP Architecture

- Build criticality graph in hardware using our model
- Dynamically analyze critical path of graph
- Make predictions and schedule tasks
CAP in a Multiprocessor System

- Task Controller
  - Tracks running tasks and their context
- Novel components of CAP
  - Critical path builder
    - Builds path and analyzes graph
  - Critical path predictor
CAP Overview

- To collect the graph
  - TC sends summary of task execution to builder after task commits
  - Summary contains summary of important edges
    - Who spawned it, who squashed it, etc.
  - The CPB creates a node in the graph and adds the edges

- Analyzing the graph
  - Store nodes such that critical path calculation is easy
  - Walk graph in reverse to find critical path
Critical Path Predictor

- Train using calculated critical path
- Record edge-centric information
  - Spawn edges
  - Squash edges
- Use strongly biased edges to control scheduling decisions. For example:
  - When Task(A) spawns Task(B), B is likely critical
  - When Task(A) squashes Task(C), C becomes critical
Scheduling on a CMP

- Assume DVFS per core on a CMP
- CMP is statically configured among Voltage-frequency (V-f) pairs
- Promote critical tasks to high V-f cores
- Demote non-critical tasks to low V-f cores
Evaluation Setup

- SPECint2000 applications
  - Optimized for SM using POSH compiler [PPoPP'06]
- Two V-f settings
- 3 Static CMP configurations
  - 3-Crit, 2-Crit, 1-Crit
Moving to fewer fast cores has small impact on performance

- Only 2.2% for 2-Crit!
Best ED^2 is obtained for 2-Crit
- Average reduction of 16.2%
- Max reduction of 57.5%
Conclusions

- SM can be power efficient
- Efficiently modeled task-level criticality in hardware
- Criticality analysis successfully schedules tasks for power efficiency
  - Average performance loss of only 2.2%
  - ED^2 reduction of 16.6% on average