Tradeoffs in Buffering Memory State (for Thread-Level Speculation in Multiprocessors)

Christopher Tice
CSC 258
4/2/2007
What is speculative parallelization?

- Optimistically running a thread in parallel with incomplete program information

- Why?

- Pointers, interprocedural dependencies, or input-dependent access patterns
Major Challenges with Speculation

- Detection of violations
  - data is tagged to enforce sequential order (e.g. bitmap)

- State repair
  - buffer the unsafe memory state
  - many ways to do this
Major Challenges with Buffering

- Separation of task state
- Multiple versions of the same variable
- Multiple speculative tasks per processor
- Multiple versions of the same variable in a single processors
- Merging of task state
Taxonomy of Approaches

- Separation of Task State
- Merging of Task State
- Architectural Main Memory (AMM)
- Future Main Memory (FMM)

- Eager
- Lazy

- Multiple Spec Tasks
  - Multiple Versions of Same (MultiT&MV)
  - Single Version (MultiT&SV)
- Single Spec Task
  - Single Version (SingleT)

- Memory-System Reorder Buffer (MROB)
- Memory-System History Buffer (MHB)
Merging of Task States

- Eager AMM – at commit time
  - Problem: large commits

- Lazy AMM – after commit time

- FMM – any time
AMM vs. FMM

Architectural Main Memory

Future Main Memory

Younger versions

Memory–System Reorder Buffer (MROB)

Architectural memory system state

Variables

(a)

Future memory system state

Memory–System History Buffer (MHB)

Older versions

Variables

(b)

Legend:
- Last speculative version
- Non–last speculative version
- Committed version
- Unmodified
Analysis of Benefits and Complexity

- start simple
- add complex features
- consider performance benefits and hardware support required (complexity)
Single Task Eager AMM

- simplest scheme
- each task stores its state in a buffer (e.g. cache)
- state is merged with main memory for commit
- invalidation required for state repair
Multiple Tasks

- Benefits: tolerate load imbalance
- Requires: cache task ID
Multiple Tasks and Versions

- Benefits: tolerate load imbalance and mostly-privatization patterns
- Requires: cache task ID and cache retrieval logic
Example Execution

- **SingleT**
- **MultiT&SV**
- **MultiT&MV**
Lazy Merging with AMM

- Benefits: remove commit wavefront from critical path
- Requires: cache task ID and version combining logic (or memory task ID)
Example Execution

(a) MultiT&MV Eager AMM  (b) MultiT&MV Lazy AMM  (c) SingleT Eager AMM  (d) SingleT Lazy AMM
Future Main Memory

- Benefits: faster version commit but slower recovery
- Requires: cache task ID, memory task ID, and undo log
Complexity Comparison

- SingleT --> MultiT&SV
  - tolerate load imbalance

- MultiT&SV --> MultiT&MV
  - tolerate load imbalance with private variables

- Eager AMM --> Lazy AMM
  - remove commit wavefront from critical path

- Lazy AMM --> FMM
  - faster version commit but slower version recovery
Complexity Comparison Diagram

- Architectural Main Memory (AMM)
- Future Main Memory (FMM)

- Eager
- Lazy

- Multiple Versions of Same
  - MultiT&MV
- Single Version
  - MultiT&SV

- Single Spec Task (SingleT)

- Task Load Imbalance + Mostly-Privatization Patterns
- Task Load Imbalance

- Task Commit Wavefront in Critical Path
- Cache Overflow due to Capacity or Conflicts

- Frequent Recoveries from Dependence Violations
Benchmark Results

- **MultiT&MV**
  - 32% faster than SingleT

- **Lazy AMM**
  - 30% faster for SingleT and MultiT&SV
  - 24% faster for MultiT&MV

- **FMM**
  - similar to performance of lazy AMM

- **CMP ~ NUMA**
Aside: Process-Level Speculation

- Behavior-oriented parallelization (BOP) led by Chen Ding

- Advantages:
  - no extra hardware, buffering is free

- Disadvantages:
  - overhead, false sharing
Aside: Process-Level Speculation

- where does BOP fit in?
Summary

- novel taxonomy of approaches to buffering
- complexity-benefit tradeoff
Conclusion

Q & A