Removing Architectural Bottlenecks
to the Scalability of Speculative Parallelization

Milos Prvulovic, María Jesús Garzarán, Lawrence Rauchwerger and Josep Torrellas

Department of Computer Science
University of Illinois at Urbana-Champaign
Motivation

- Codes with access patterns not analyzable by compiler
- Speculative parallelization can extract parallelism
- Several designs (CMPs and DSM)
- Poor speedups on scaleable systems
  - Even when lots of parallelism available
Summary of Findings

- Key bottlenecks identified
- Architectural Solutions for
  - Commit Serialization
  - Overflow of Speculative Buffers
  - Speculation-Induced Traffic
- Avg. speedup for 64 CPUs
  - 31.7 (Opt) vs. 8.7 (No Opt)
Outline

- Motivation
- Background
- Bottlenecks & Solutions
- Evaluation
- Conclusions
Speculative Thread-Level Parallelization

- Extract tasks from sequential code
- Assume no dependences and execute tasks in parallel
- Track data accesses
- Detect dependence violations
- Squash offending tasks and restart them

for(i=1;i<N;i++){
    ...
    = A[L[i]]+...
    A[K[i]] = ...
}
Outline

- Motivation
- Background
- Bottlenecks & Solutions
  - Task Commit Serialization
  - Overflow of Speculative Buffers
  - Speculation-Induced Traffic
- Evaluation
- Conclusions
Task Commit Serialization

E  Execute

C  Commit

Critical Path

C 0
E 4
C 0
E 0

C 1
E 5
C 1
E 1

C 2
E 6
C 2
E 2

C 3
E 7
C 3
E 3

C 4

Low-Complexity Commit in Constant Time

- Commit in Constant Time, Merge State Later
  - Commit only involves passing status to successor
  - Committed data stays in caches until displaced
Low-Complexity Commit in Constant Time

- Multi-Version Multiple-Writer Protocol

- Prevent out-of-order updates
  - Combine and destroy older versions on displacement
Low-Complexity Commit in Constant Time

Version Combining Register
Outline

- Motivation
- Background
- Bottlenecks & Solutions
  - Task Commit Serialization
  - Overflow of Speculative Buffers
  - Speculation-Induced Traffic
- Evaluation
- Conclusions
Overflow of Speculative Buffers
Overflow of Speculative Data into Memory

- Expandable Victim Buffer in Local Memory
  - Contains any overflow from L2 cache
  - Accessed only when L2 set actually overflows

- Managed as a set-associative cache

- Main memory still updated only with committed data
  - Simple, local squashing and recovery
Outline

- Motivation
- Background
- Bottlenecks & Solutions
  - Task Commit Serialization
  - Overflow of Speculative Buffers
  - Speculation-Induced Traffic
- Evaluation
- Conclusions
Speculation-Induced Traffic

- Two sources of additional traffic due to speculation
  - Dependence checking
    - Sent by writers so premature successor reads can squash
  - Data forwarding
    - Reads search for correct version
Per-Line State: Unnecessary Squashes

- Checking per line ⇒ false-sharing causes squashes

I'm writing; check for dependences

I'm reading; find correct version

W

R

5

7

Data

HOME DIRECTORY

Per-Word State: High Traffic

- Checking per word $\Rightarrow$ MUCH more traffic

I’m writing; check for dependences

ACK

HOME DIRECTORY
Exploiting High-Level Access Patterns

- Check per word to avoid squashes
- Eliminate traffic when violations not possible:
  - No Exposed Reader
    - No task reads before writing (privatization), can not cause flow dependences
  - No Writer
    - All sharers only read the line, cannot cause dependences
  - No Sharing
    - Single node reads/writes the line, cannot cause dependences
Exploiting High-Level Access Patterns

I’m writing; check for dependences

ACK, no readers

HOME DIRECTORY
Outline

- Motivation
- Background
- Bottlenecks & Solutions
- Evaluation
  - Setup and Application Characteristics
  - Simulation Results
- Conclusions
Evaluation Set-Up

- 1GHz 4-issue ooo processor, 64kB L1, 512kB L2
- CC-NUMA directory-based system, up to 128 nodes
- Polaris parallelizing compiler
- Network: 2D-torus, virtual cut-through
- No-contention latencies (CPU cycles)
  - 2 (L1 hit), 8 (L2 hit), 57 (local mem), 137 (neighbor mem),
  - 4 cycles per additional hop
  - Detailed simulation: contention, routing, (un)marshalling
Applications

- Speedups reported only for non-analyzable sections
Outline

- Motivation
- Background
- Bottlenecks & Solutions
- Evaluation
  - Setup and Application Characteristics
  - Simulation Results
- Conclusions
Optimized vs. Base

- Speedup curves diverge for >16 CPUs
Optimized vs. Base (Euler)

- Low impact if true dependences dominate
Without Commit in Constant Time

- Large improvement for >16 CPUs
Without Overflow Optimization

- L2 overflow a problem for >32 CPUs
Without Patterns Optimization

- Naïve per-word protocol a problem
- Aggressive loads [Cintra00] not enough
Performance Results Summary

Average for 64 processors: 31.7 (Opt) vs 8.7 (NoOpt)
Outline

- Motivation
- Background
- Bottlenecks & Solutions
- Evaluation
- Conclusions
Conclusions

- Need all three optimizations for scalability
  - Low-complexity commit in constant time
  - Overflow of speculative data into memory
  - Exploiting high-level access patterns

- Significant speedups:
  - Up to 64 for 128 processors
  - Average for 64 processors: 31.7 (Opt) vs. 8.7 (NoOpt)
Removing Architectural Bottlenecks to the Scalability of Speculative Parallelization

Milos Prvulovic,
María Jesús Garzarán, Lawrence Rauchwerger and Josep Torrellas

prvulovi@cs.uiuc.edu
http://iacoma.cs.uiuc.edu
Related Work: Cintra00 and Steffan00

- Expensive commit serialization
  - Cintra00: Flush all dirty data at commit time
  - Steffan00: Get ownership of all written data at commit time
- Stall on speculative buffer overflow
  - Stall on L1 overflow
  - Cintra00: Speculative clean data need not be buffered
- High speculation-induced traffic
  - Cintra00: per-word protocol
  - Steffan00: some per-line support, squashes due to false sharing
Future Work

- Some optimizations can apply to CMPs as well
  - E.g. reducing traffic helps
- Parallel apps can benefit as well
  - E.g., on-demand privatization
- Optimizations for mostly sequential codes
  - Fast squashing, etc.
Applications

High-Level Access Patterns

Track

Apsi

Commit in Constant Time

Overflow into Memory

Tree

Bdna

Euler

No Opt

Comparison with Cintra et al. (ISCA00)

- **Commit Serialization in Cintra et al.**
  - Flush the task’s dirty data at commit time

- **Speculative Buffer Overflow in Cintra et al.**
  - 4 CPUs per chip, share L2, private L1
  - Only one of the four can use L2 as speculative buffer
  - Other three use L1
  - Speculative clean data need not be buffered

- **Cintra et al. is a full per-word protocol**
  - Traffic on first load and on first store to each word