Software-Managed Cache Coherence for fast One-Sided Communication

PMAM@PPoPP, Barcelona, Spain, March 12 2016

Steffen Christgau, Bettina Schnor

Operating Systems and Distributed Systems
Institute for Computer Science
University of Potsdam, Germany
Motivation

- Will future many-core systems provide hardware cache coherence?
Motivation

- Will future many-core systems provide hardware cache coherence? Not always $\rightarrow$ coherence islands in nCC systems
Motivation

- Will future many-core systems provide hardware cache coherence? Not always → coherence islands in nCC systems
- nCC many-core research system: Intel SCC

- 48 Pentium cores with L1/2 caches, no HW cache coherence
- Memory subsystem allows creation of shared memory
Motivation

- Will future many-core systems provide hardware cache coherence? Not always → coherence islands in nCC systems
- nCC many-core research system: Intel SCC

- 48 Pentium cores with L1/2 caches, **no HW cache coherence**
- memory subsystem allows creation of shared memory

- previous research: focus on message passing (used MPB)
• Will future many-core systems provide hardware cache coherence? Not always → coherence islands in nCC systems

• nCC many-core research system: Intel SCC

- 48 Pentium cores with L1/2 caches, **no HW cache coherence**
- memory subsystem allows creation of shared memory

• previous research: focus on message passing (used MPB)

• new approach: use shared memory on nCC CPU for **one-sided communication** and **manage cache coherence in software**
origin

target
MPI One-Sided Communication

- Target
- Origin
- Window (exposed by target)
- RAM

**Requirements for Software--A Managed Cache Coherence for fast OSC**
MPI One-Sided Communication

origin

window (exposed by target)

GET

PUT

target

RAM

TARGET/EXPOSURESTART: WRITE WINDOW DATA BACK TO ORIGIN/ACCESSSTART: INVALIDATE CACHED WINDOW DATA

ORIGIN/ACCESSSEND: WRITE WINDOW DATA BACK TO TARGET/EXPOSUREEND: INVALIDATE CACHED WINDOW DATA

0-2016 3. CHRISTGAU (U POTSDAM): 3W—SW-MANAGED CACHE COHERENCE FOR FAST OSC
MPI One-Sided Communication

![Diagram showing the flow of data in MPI One-Sided Communication]

- **Origin**
  - Access start
  - GET
  - PUT
  - Access end

- **Target**
  - Exposure start
  - Exposure end

- **Window** (exposed by target)
  - RAM

- **Requirements**
  - TARGET/EXPOSURE_START: WRITE_WINDOW_DATA_BACK_TO_ORIGIN
  - ORIGIN/ACCESS_START: INVALIDATE_CACHED_WINDOW_DATA
  - ORIGIN/ACCESS_END: WRITE_WINDOW_DATA_BACK_TO_ORIGIN
  - TARGET/EXPOSURE_END: INVALIDATE_CACHED_WINDOW_DATA
MPI One-Sided Communication

- **Origin**
  - Access start
  - GET
  - PUT
  - Access end

- **Target**
  - Exposure start
  - Exposure end

- **Window** (exposed by target)
  - RAM

- **Requirements for Software—Managed Cache Coherence**
  - Target/Exposure Start: Write window data back to origin
  - Origin/Access Start: Invalidate cached window data
  - Origin/Access End: Write window data back to target
  - Target/Exposure End: Invalidate cached window data


- OUTDATED COPY
- \( \text{GET} \)
- \( \text{PUT} \)
- \( \text{IN\textit{V}\textit{A}L\textit{.IDATE}} \)
- \( \text{PUT} \)
- \( \text{\textit{I\textit{N\textit{V\textit{A}L\textit{IDATE}}} CURRENT DATA} \)
- \( \text{\textit{I\textit{N\textit{V\textit{A}L\textit{IDATE}}} OUTDATED COPY}} \)
- \( \text{\textit{I\textit{N\textit{V\textit{A}L\textit{IDATE}}} PUT}} \)
- \( \text{\textit{I\textit{N\textit{V\textit{A}L\textit{IDATE}}} CURRENT DATA} \)
- \( \text{\textit{I\textit{N\textit{V\textit{A}L\textit{IDATE}}} OUTDATED COPY}} \)

- \( \text{\textit{I\textit{N\textit{V\textit{A}L\textit{IDATE}}} accEssWindoWdaTa} \)
- \( \text{\textit{I\textit{N\textit{V\textit{A}L\textit{IDATE}}} accEssWindoWdaTa} \)
MPI One-Sided Communication

Requirements for Software-Managed Cache Coherence

**target/exposure start:** write window data back to RAM
MPI One-Sided Communication

Requirements for Software-Managed Cache Coherence

- **target/exposure start**: write window data back to RAM
- **origin/access start**: invalidate cached window data
Requirements for Software-Managed Cache Coherence

**target/exposure start:** write window data back to RAM

**origin/access start:** invalidate cached window data

**origin/access end:** write window data back to RAM
MPI One-Sided Communication

**Requirements for Software-Managed Cache Coherence**

- **target/exposure start**: write window data back to RAM
- **origin/access start**: invalidate cached window data
- **origin/access end**: write window data back to RAM
- **target/exposure end**: invalidate cached window data
Implementation on the SCC: SCOSCo

- conventional cacheable/uncacheable memory types unsuited
Implementation on the SCC: SCOSCo

- conventional cacheable/uncacheable memory types unsuited
- new memory type on SCC
  - L1-cacheable, cache lines marked with special bit
  - new instruction: invalidate marked cache lines in few cycles
  - configurable L1 behavior: write-back or write-through
  - write-combine buffer

- Requirement 1: Write window data back to \( \overrightarrow{\text{WRITE-THROUGH}} \) cache on guarantion for window
- Requirement 2: Invalidate cache data \( \overrightarrow{\text{ISSUE FAST INVALIDATE INSTRUCTION}} \)
- But: SCC's WRITE-THROUGH has uncacheable memory performance
  - Use write-back configuration as substitute for benchmarking: ensure cache miss on WRITE-0-2016
Implementation on the SCC: SCOSCo

- conventional cacheable/uncacheable memory types unsuited
- new memory type on SCC
  - L1-cacheable, cache lines marked with special bit
  - new instruction: invalidate marked cache lines in few cycles
  - configurable L1 behavior: write-back or write-through
  - write-combine buffer
- use new memory type + write-through for window
Implementation on the SCC: SCOSCo

- conventional cacheable/uncacheable memory types unsuited
- new memory type on SCC
  - L1-cacheable, cache lines marked with special bit
  - new instruction: invalidate marked cache lines in few cycles
  - configurable L1 behavior: write-back or write-through
  - write-combine buffer
- use new memory type + write-through for window
- requirement 1: write window data back to RAM
  $\rightarrow$ write-through cache configuration for window
Implementation on the SCC: SCOSCo

- conventional cacheable/uncacheable memory types unsuited
- new memory type on SCC
  - L1-cacheable, cache lines marked with special bit
  - new instruction: invalidate marked cache lines in few cycles
  - configurable L1 behavior: write-back or write-through
  - write-combine buffer
- use new memory type + write-through for window
- requirement 1: write window data back to RAM
  → write-through cache configuration for window
- requirement 2: invalidate cached data
  → issue fast invalidate instruction
Implementation on the SCC: SCOSCo

- conventional cacheable/uncacheable memory types unsuited
- new memory type on SCC
  - L1-cacheable, cache lines marked with special bit
  - new instruction: invalidate marked cache lines in few cycles
  - configurable L1 behavior: write-back or write-through
  - write-combine buffer
- use new memory type + write-through for window
- requirement 1: write window data back to RAM
  \[\text{write-through cache configuration}\] for window
- requirement 2: invalidate cached data
  \[\text{issue fast invalidate instruction}\]
- but: SCC’s write-through has uncached memory performance
  - use write-back configuration as substitute
  - for benchmarks: ensure cache miss on write
Experimental Results: 3D FFT

- MPI implementation of NPB-FT
  - origins only PUT → window gets updated correctly
  - assume similar performance as working write-through memory
Experimental Results: 3D FFT

- MPI implementation of NPB-FT
  - origins only PUT → window gets updated correctly
  - assume similar performance as working write-through memory
potentials for further improvements
Wishful Thinking for Fast OSC on nCC Systems

Recommendations for Future Systems

potentials for further improvements

- slow flush to RAM
  - slow performance of working write-through memory (?)
  - (full) cache flush slow anyway
Recommendations for Future Systems

potentials for further improvements

- slow flush to RAM
- wasteful invalidation
  - removes all window data (not only modified)
  - affects other cached data of same type
Wishful Thinking for Fast OSC on nCC Systems

Recommendations for Future Systems

potentials for further improvements
  • slow flush to RAM
  • wasteful invalidation

desireable beneficial hardware features in future nCC systems:
Wishful Thinking for Fast OSC on nCC Systems

Recommendations for Future Systems

potentials for further improvements

• slow flush to RAM
• wasteful invalidation

desireable beneficial hardware features in future nCC systems:

• **flushable write-combine buffer** for write-through memory
  ■ alternative: address range-based flush of modified cache lines
Wishful Thinking for Fast OSC on nCC Systems

Recommendations for Future Systems

- slow flush to RAM
- wasteful invalidation

Desireable beneficial hardware features in future nCC systems:
- **flushable write-combine buffer** for write-through memory
- address **range-based cache invalidation** (for GETs)
Wishful Thinking for Fast OSC on nCC Systems

Recommendations for Future Systems

potentials for further improvements
  • slow flush to RAM
  • wasteful invalidation

desirable beneficial hardware features in future nCC systems:
  • flushable write-combine buffer for write-through memory
  • address range-based cache invalidation (for GETs)
  • remote cache invalidation by address range (for PUTs)
Wishful Thinking for Fast OSC on nCC Systems

Recommendations for Future Systems

potentials for further improvements

- slow flush to RAM
- wasteful invalidation

desireable beneficial hardware features in future nCC systems:

- **flushable write-combine buffer** for write-through memory
- address **range-based cache invalidation** (for GETs)
- **remote cache invalidation** by address range (for PUTs)

→ more explicit control over caches for software
Summary

- demonstrated implementation for software-based cache coherence on nCC many-core CPU
- shared memory approach outperforms message-based solution
  - 4–5x less communication time (40–45% app. speedup) for FFT
- identified beneficial features for future systems
  - local and remote address-based cache invalidation
Summary

- demonstrated implementation for software-based cache coherence on nCC many-core CPU
- shared memory approach outperforms message-based solution
  - 4–5x less communication time (40–45% app. speedup) for FFT
- identified beneficial features for future systems
  - local and remote address-based cache invalidation

Thanks for your attention!