Synchronization of MPI One-Sided Communication on a Non-Cache-Coherent Many-Core System

PASA@ARCS, Nuremberg, Germany, April 4 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 → 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

- new approach: use shared memory on nCC CPU for **one-sided communication** and **manage cache coherence in software**
Software-Managed Cache Coherence for OSC

- previous results: reduce communication costs by factor of $4 - 5$ when using OSC with software-managed cache coherence

S. Christgau, B. Schnor: *Software-managed Cache Coherence for fast One-Sided Communication*, PMAM 2016
Software-Managed Cache Coherence for OSC

- previous results: reduce communication costs by factor of 4 – 5 when using OSC with software-managed cache coherence

S. Christgau, B. Schnor: Software-managed Cache Coherence for fast One-Sided Communication, PMAM 2016

- today: synchronization for one-sided communication
Agenda

Background
- MPI One-Sided Communication
- Process Synchronization
- Classification and Survey

Implementation
- Data Structures
- Algorithm

Experimental Evaluation
- Scaling
- Comparison with RCKMPI

Summary
Background: MPI Communication Styles

two-sided communication

- sender *and* receiver must know communication parameters (buffer address, data type, tag, sender/receiver)
- implicit synchronization
Background: MPI Communication Styles

two-sided communication

- sender and receiver must know communication parameters (buffer address, data type, tag, sender/receiver)
- implicit synchronization

one-sided communication
Background: MPI Communication Styles

two-sided communication
- sender and receiver must know communication parameters (buffer address, data type, tag, sender/receiver)
- implicit synchronization

one-sided communication
- provide communication parameters only at one side (origin)
Background: MPI Communication Styles

two-sided communication

- sender *and* receiver must know communication parameters (buffer address, data type, tag, sender/receiver)
- implicit synchronization

one-sided communication

- provide communication parameters only at one side (origin)

---

**origin process**
- synchronization A
- GET
- PUT
- synchronization B

**target process**
- synchronization A
- RAM
- synchronization B

*window* (exposed by target)
Background: MPI Communication Styles

two-sided communication

- sender and receiver must know communication parameters (buffer address, data type, tag, sender/receiver)
- implicit synchronization

one-sided communication

- provide communication parameters only at one side (origin)
Background: MPI Communication Styles

two-sided communication

- sender *and* receiver must know communication parameters (buffer address, data type, tag, sender/receiver)
- implicit synchronization

one-sided communication

- provide communication parameters only at one side (origin)
- separation of communication and synchronization

![Diagram of two-sided communication](attachment:diagram.png)
MPI Synchronization Types

- **active target**: both sides involved

```
synchronization
  /|
 / |
active target            passive target
  |   |
  |   |
fence                   GATS
```

- active target
  - both sides involved
- passive target
  - only sources synchronize
- fence
  - global synchronization between all window holders (similar to barrier)
- GA43
  - general active target synchronization

S. Christgau (U Potsdam): MPI OSC Synchronization on nCC Many-Core CPU
MPI Synchronization Types

- **active target**: both sides involved
  - fence: global synchronization between all window sharers (similar to barrier)

- **passive target:** only origins synchronize
MPI Synchronization Types

- **active target**: both sides involved
  - fence: global synchronization between all window sharers (similar to barrier)
  - general active target synchronization (GATS) synchronize subset only + specify role
MPI Synchronization Types

- **active target**: both sides involved
  - fence: global synchronization between all window sharers (similar to barrier)
  - general active target synchronization (GATS) synchronize subset only + specify role
- **passive target**: only origins synchronize
MPI Synchronization Types

- **active target**: both sides involved
  - fence: global synchronization between all window sharers (similar to barrier)
  - **general active target synchronization (GATS)**: synchronize subset only + specify role
- **passive target**: only origins synchronize
General Active Target Synchronization (GATS)

**Origin Process**
- Synchronization A
- GET
- PUT
- Synchronization B

**Target Process**
- Synchronization A
- Synchronization B

**Window** (exposed by target)

**Access Epoch**
- OPEN ACCESS EPOCH (ALLOW COMMUNICATION CALLS)

**Exposure Epoch**
- WAIT FOR ALL ORIGIN NOTIFICATIONS
**General Active Target Synchronization (GATS)**

**origin process**

- **START** (start group)
- GET
- PUT

**target process**

- **POST** (post group)

**Window** (exposed by target)

**RAM**

**POST** allow communication on own window

**START** open access epoch (allow communication calls)
General Active Target Synchronization (GATS)

- **origin process**
  - START (start group)
  - GET
  - PUT
  - COMPLETE

- **target process**
  - POST (post group)
  - WAIT

**window** (exposed by target)

- **POST** allow communication on own window
- **START** open access epoch (allow communication calls)
- **COMPLETE** finish communication and notify targets
- **WAIT** wait for all origin notifications
Classification of Implementations

- **origin process**
  - **START** (start group)
  - GET
  - PUT
  - **COMPLETE**

- **target process**
  - **POST** (post group)
  - **WAIT**

**window** (exposed by target)

**access epoch**

**exposure epoch**

- IMMEDIATE BLOCKING
  - PROMPT YES
  - DEFERRED NON-BLOCKING
  - DELAYED NO
  - TRIGGER-ONLY NON-BLOCKING
  - PROMPT YES
Classification of Implementations

<table>
<thead>
<tr>
<th>origin process</th>
<th>window (exposed by target)</th>
<th>target process</th>
</tr>
</thead>
<tbody>
<tr>
<td>START (start group)</td>
<td>GET</td>
<td>POST (post group)</td>
</tr>
<tr>
<td>PUT</td>
<td>COMPLETE</td>
<td>WAIT</td>
</tr>
</tbody>
</table>

- Immediate: blocking communication, prompt overlap, yes

access epoch

exposure epoch
Classification of Implementations

<table>
<thead>
<tr>
<th>class</th>
<th>epoch start</th>
<th>communication</th>
<th>overlap</th>
</tr>
</thead>
<tbody>
<tr>
<td>immediate</td>
<td>blocking</td>
<td>prompt</td>
<td>yes</td>
</tr>
<tr>
<td>deferred</td>
<td>non-blocking</td>
<td>delayed</td>
<td>no</td>
</tr>
</tbody>
</table>

**Origin Process**
- **START** (start group)
- GET
- PUT
- **COMPLETE**

**Target Process**
- **POST** (post group)
- **WAIT**

Classifications:
- **Immediate**
  - Blocking
  - Prompt
  - Yes
- **Deferred**
  - Non-blocking
  - Delayed
  - No
Classification of Implementations

<table>
<thead>
<tr>
<th>class</th>
<th>epoch start</th>
<th>communication</th>
<th>overlap</th>
</tr>
</thead>
<tbody>
<tr>
<td>immediate</td>
<td>blocking</td>
<td>prompt</td>
<td>yes</td>
</tr>
<tr>
<td>deferred</td>
<td>non-blocking</td>
<td>delayed</td>
<td>no</td>
</tr>
<tr>
<td>trigger-only</td>
<td>non-blocking</td>
<td>prompt</td>
<td>yes</td>
</tr>
</tbody>
</table>
### Summary of Conducted Source Code Survey

<table>
<thead>
<tr>
<th>Class</th>
<th>Epoch Start</th>
<th>Communication</th>
<th>Overlap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Immediate</td>
<td>blocking</td>
<td>prompt</td>
<td>yes</td>
</tr>
<tr>
<td>Deferred</td>
<td>non-blocking</td>
<td>delayed</td>
<td>no</td>
</tr>
<tr>
<td>Trigger-only</td>
<td>non-blocking</td>
<td>prompt</td>
<td>yes</td>
</tr>
</tbody>
</table>
Implementation Survey

summary of conducted source code survey

- MPICH: deferred + message-based

<table>
<thead>
<tr>
<th>class</th>
<th>epoch start</th>
<th>communication</th>
<th>overlap</th>
</tr>
</thead>
<tbody>
<tr>
<td>immediate</td>
<td>blocking</td>
<td>prompt</td>
<td>yes</td>
</tr>
<tr>
<td>deferred</td>
<td>non-blocking</td>
<td>delayed</td>
<td>no</td>
</tr>
<tr>
<td>trigger-only</td>
<td>non-blocking</td>
<td>prompt</td>
<td>yes</td>
</tr>
</tbody>
</table>
summary of conducted source code survey

- MPICH: deferred + message-based
- MVAPICH/InfiniBand: immediate + InfiniBand RDMA

<table>
<thead>
<tr>
<th>class</th>
<th>epoch start</th>
<th>communication</th>
<th>overlap</th>
</tr>
</thead>
<tbody>
<tr>
<td>immediate</td>
<td>blocking</td>
<td>prompt</td>
<td>yes</td>
</tr>
<tr>
<td>deferred</td>
<td>non-blocking</td>
<td>delayed</td>
<td>no</td>
</tr>
<tr>
<td>trigger-only</td>
<td>non-blocking</td>
<td>prompt</td>
<td>yes</td>
</tr>
</tbody>
</table>
Implementations Survey

Summary of conducted source code survey

- MPICH: deferred + message-based
- MVAPICH/InfiniBand: immediate + InfiniBand RDMA
- Open MPI/InfiniBand: deferred + message-based

<table>
<thead>
<tr>
<th>class</th>
<th>epoch start</th>
<th>communication</th>
<th>overlap</th>
</tr>
</thead>
<tbody>
<tr>
<td>immediate</td>
<td>blocking</td>
<td>prompt</td>
<td>yes</td>
</tr>
<tr>
<td>deferred</td>
<td>non-blocking</td>
<td>delayed</td>
<td>no</td>
</tr>
<tr>
<td>trigger-only</td>
<td>non-blocking</td>
<td>prompt</td>
<td>yes</td>
</tr>
</tbody>
</table>
Implementation Survey

summary of conducted source code survey

- MPICH: deferred + message-based
- MVAPICH/InfiniBand: immediate + InfiniBand RDMA
- Open MPI/InfiniBand: deferred + message-based
- MVAPICH/shared memory: trigger-only + shared memory
  ▪ proposed by Potluri et al., 2011
  ▪ two bit vectors per process, one bit per process
  ▪ POST: set process’ bit in origin vectors / START: do nothing
  ▪ check for synchronization (poll vector) in communication calls
  ▪ COMPLETE: set process’ bit in target vectors / WAIT: poll vector

<table>
<thead>
<tr>
<th>class</th>
<th>epoch start</th>
<th>communication</th>
<th>overlap</th>
</tr>
</thead>
<tbody>
<tr>
<td>immediate</td>
<td>blocking</td>
<td>prompt</td>
<td>yes</td>
</tr>
<tr>
<td>deferred</td>
<td>non-blocking</td>
<td>delayed</td>
<td>no</td>
</tr>
<tr>
<td>trigger-only</td>
<td>non-blocking</td>
<td>prompt</td>
<td>yes</td>
</tr>
</tbody>
</table>
summary of conducted source code survey

- MPICH: deferred + message-based
- MVAPICH/InfiniBand: immediate + InfiniBand RDMA
- Open MPI/InfiniBand: deferred + message-based
- MVAPICH/shared memory: trigger-only + shared memory
  - two shared atomic counters per process (epoch start/end)
  - increment and poll operations

<table>
<thead>
<tr>
<th>class</th>
<th>epoch start</th>
<th>communication</th>
<th>overlap</th>
</tr>
</thead>
<tbody>
<tr>
<td>immediate</td>
<td>blocking</td>
<td>prompt</td>
<td>yes</td>
</tr>
<tr>
<td>deferred</td>
<td>non-blocking</td>
<td>delayed</td>
<td>no</td>
</tr>
<tr>
<td>trigger-only</td>
<td>non-blocking</td>
<td>prompt</td>
<td>yes</td>
</tr>
</tbody>
</table>
OpenMPI’s GATS Bug

origin

START(target 1)

PUT(target 1, ...)

COMPLETE

START(target 2)

target 1

POST(origin)

target 2

POST(origin)
OpenMPI’s GATS Bug

-1
START(target 1)

0
PUT(target 1, ...)

COMPLETE

START(target 2)

POST(origin)

POST(origin)

POST(origin)
OpenMPI’s GATS Bug

origin

-1
START(target 1)

0
PUT(target 1, ...)

COMPLETE

0
START(target 2)

target 1

POST(origin)

target 2

POST(origin)
Implementation Survey

Summary of conducted source code survey

- MPICH: deferred + message-based
- MVAPICH/InfiniBand: immediate + InfiniBand RDMA
- Open MPI/InfiniBand: deferred + message-based
- MVAPICH/shared memory: trigger-only + shared memory
- Open MPI/shared memory: immediate + shared memory
  - two shared atomic counters per process (epoch start/end)
  - increment and poll operations

<table>
<thead>
<tr>
<th>class</th>
<th>epoch start</th>
<th>communication</th>
<th>overlap</th>
</tr>
</thead>
<tbody>
<tr>
<td>immediate</td>
<td>blocking</td>
<td>prompt</td>
<td>yes</td>
</tr>
<tr>
<td>deferred</td>
<td>non-blocking</td>
<td>delayed</td>
<td>no</td>
</tr>
<tr>
<td>trigger-only</td>
<td>non-blocking</td>
<td>prompt</td>
<td>yes</td>
</tr>
</tbody>
</table>
Implementation Survey

Summary of conducted source code survey

- **MPICH**: deferred + message-based
- **MVAPICH/InfiniBand**: immediate + InfiniBand RDMA
- **Open MPI/InfiniBand**: deferred + message-based
- **MVAPICH/shared memory**: trigger-only + shared memory
- **Open MPI/shared memory**: immediate + shared memory
  - two shared atomic counters per process (epoch start/end)
  - increment and poll operations
  - error-prone ([**bug**](#)), fixed after report → now MVAPICH-like

<table>
<thead>
<tr>
<th>class</th>
<th>epoch start</th>
<th>communication</th>
<th>overlap</th>
</tr>
</thead>
<tbody>
<tr>
<td>immediate</td>
<td>blocking</td>
<td>prompt</td>
<td>yes</td>
</tr>
<tr>
<td>deferred</td>
<td>non-blocking</td>
<td>delayed</td>
<td>no</td>
</tr>
<tr>
<td>trigger-only</td>
<td>non-blocking</td>
<td>prompt</td>
<td>yes</td>
</tr>
</tbody>
</table>
Implementation Survey

Summary of conducted source code survey

- MPICH: deferred + message-based
- MVAPICH/InfiniBand: immediate + InfiniBand RDMA
- Open MPI/InfiniBand: deferred + message-based
- MVAPICH/shared memory: trigger-only + shared memory
- Open MPI/shared memory: immediate + shared memory

→ Large variety of implementations

<table>
<thead>
<tr>
<th>class</th>
<th>epoch start</th>
<th>communication</th>
<th>overlap</th>
</tr>
</thead>
<tbody>
<tr>
<td>immediate</td>
<td>blocking</td>
<td>prompt</td>
<td>yes</td>
</tr>
<tr>
<td>deferred</td>
<td>non-blocking</td>
<td>delayed</td>
<td>no</td>
</tr>
<tr>
<td>trigger-only</td>
<td>non-blocking</td>
<td>prompt</td>
<td>yes</td>
</tr>
</tbody>
</table>
preconsiderations

- outcomes of the survey:
  - for shared memory systems: bitvectors and counters
  - no nCC shared memory system support
SCOSCo: Implementation on the SCC

preconsiderations

- outcomes of the survey:
  - for shared memory systems: bitvectors and counters
  - **no nCC shared memory system support**

- Intel’s MPI for SCC: MPICH-based
  - inherently message-based + deferred synchronization style
  - incomplete implementation of one-sided communication
  - bugs removed in previous work
SCOSCo: Implementation on the SCC

preconsiderations

- outcomes of the survey:
  - for shared memory systems: bitvectors and counters
  - no nCC shared memory system support
- Intel’s MPI for SCC: MPICH-based
  - inherently message-based + deferred synchronization style
  - incomplete implementation of one-sided communication
  - bugs removed in previous work

general idea for SCC:

- bit vectors and counters for synchronization
- allocation in shared memory, but on nCC system
- software-managed cache coherence (if required)
**match vector** = bit vector

- for synchronization at epoch start:
- one vector per process, placed in shared memory
- dedicated to origin processes
- $k$-th bit set $\Rightarrow$ target process with ID/rank $k$ synchronized
**match vector** = bit vector

- for synchronization at epoch start:
- one vector per process, placed in shared memory
- dedicated to origin processes
- \( k \)-th bit set \( \Rightarrow \) target process with ID/rank \( k \) synchronized

**completion counter (CC)**

- for synchronization at epoch end
- one integer per process in shared memory
- placed at well-known offset behind match vector
- dedicated to target processes
- \( CC == 0 \Rightarrow \) all origins have completed
Synchronization: Epoch Start

POST(post_group)

- Acquire origins
- Set entry in match vector
- Release origins

No caching required
Match vector never used again could prevent vector update

S12ST (START GROUP)

- Save list of target processes
- Origin
- Target
- POST
**POST/post_group**

- $CC \leftarrow \text{number of given origins}$
**POST(post_group)**

- CC ← number of given origins
- signal all origins ∈ post group:

```
origin
POST
CC = 1
```

```
target
```
POST(post_group)

- CC ← number of given origins
- signal all origins ∈ post group:
  - acquire origins’s Test&Set Register

Synchronizing on nCC Many-Core CPU
POST(post_group)

- CC ← number of given origins
- signal all origins ∈ post group:
  - acquire origins’s Test&Set Register
  - set entry in match vector
POST(post_group)

- CC ← number of given origins
- signal all origins ∈ post group:
  - acquire origins’s Test&Set Register
  - set entry in match vector
  - release origins’s TSR
Synchronization: Epoch Start

POST(post_group)

- CC ← number of given origins
- signal all origins ∈ post group:
  - acquire origins’s Test&Set Register
  - set entry in match vector
  - release origins’s TSR
- no caching required
  - match vector never used again
  - could prevent vector update
  - use uncached writes

<table>
<thead>
<tr>
<th>origin</th>
<th>target</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>CC = 1</td>
</tr>
<tr>
<td></td>
<td>acquire TSR</td>
</tr>
<tr>
<td></td>
<td>set vector entry</td>
</tr>
<tr>
<td></td>
<td>release TSR</td>
</tr>
</tbody>
</table>

**POST(post_group)**

- CC ← number of given origins
- signal all origins ∈ post group:
  - acquire origins’s Test&Set Register
  - set entry in match vector
  - release origins’s TSR
- **no caching required**
  - match vector never used again
  - could prevent vector update
  - use **uncached writes**

**START(start_group)**

- save list of target processes
Synchronization: Epoch Start

**POST(post_group)**

- CC ← number of given origins
- signal all origins ∈ post group:
  - acquire origins’s Test&Set Register
  - set entry in match vector
  - release origins’s TSR

- no caching required
  - match vector never used again
  - could prevent vector update
  - use **uncached writes**

**START(start_group)**

- save list of target processes
**Synchronization: Epoch End**

**COMPLETE**

- Wait for all (remaining) targets
- Reset match vector entries for all saved targets:
  - Acquire target
  - Un-cache decrement
  - Release target

Poll cache until zero

Caching would prevent progression

**Origin**

**Target**

**Summary:** No caching and no coherence needed!
**Synchronization: Epoch End**

**COMPLETE**
- wait for all (remaining) targets
- reset match vector entries

![Diagram showing synchronization process]

**Summary:** No caching and no coherence needed!
**COMPLETE**

- wait for all (remaining) targets
- reset match vector entries
- for all saved targets:

- **origin**
  - COMPLETE
  - vector reset
- **target**
  - POST
  - CC = 1

**Summary:** No caching and no coherence needed!
COMPLETE

- wait for all (remaining) targets
- reset match vector entries
- for all saved targets:
  - acquire target’s TSR
**COMPLETE**

- wait for all (remaining) targets
- reset match vector entries
- for all saved targets:
  - acquire target’s TSR
  - uncached CC decrement

---

**Summary:**

- No caching and no coherence needed.
Synchronization: Epoch End

COMPLETE

- wait for all (remaining) targets
- reset match vector entries
- for all saved targets:
  - acquire target’s TSR
  - **uncached CC decrement**
  - release target’s TSR
Synchronization: Epoch End

**COMPLETE**

- wait for all (remaining) targets
- reset match vector entries
- for all saved targets:
  - acquire target’s TSR
  - **uncached CC decrement**
  - release target’s TSR

**WAIT**

- poll CC **uncached** until zero
  - caching would prevent progression
**Synchronization: Epoch End**

**COMPLETE**
- wait for all (remaining) targets
- reset match vector entries
- for all saved targets:
  - acquire target’s TSR
  - **uncached CC decrement**
  - release target’s TSR

**WAIT**
- poll **CC uncached** until zero
  - caching would prevent progression

**Summary:** No caching and no coherence needed!
Microbenchmark

- existing MPI benchmark suites, e.g. OSU: no dedicated synchronization benchmark → create own one
- no communication → fair comparison of synchronization
Microbenchmark

- existing MPI benchmark suites, e.g. OSU: no dedicated synchronization benchmark → create own one
- no communication → fair comparison of synchronization
- $n$ processes = 1 origin + $(n - 1)$ targets
- Pseudocode:

  ```plaintext
  for $i = 0 \ldots 1000$ do
    if proc is origin then
      TIME(START($G_s = \{1 \ldots n - 1\}$))
      TIME(COMPLETE)
    else
      TIME(POST($G_p = \{0\}$))
      TIME(WAIT)
    end if
  end for
  ```

MICROBENCHMARK.

• EXISTING BENCHMARKS, E.G. /35: NODE DEDICATED SYNCHRONIZATION BENCHMARK → CREATE OWN ONE
• NO COMMUNICATION → FAIR COMPARISON OF SYNCHRONIZATION
• $n$ PROCESSES = 1 ORIGIN + ($n - 1$) TARGETS
• PSEUDOCODE:

```plaintext
for $i = 0 \ldots 1000$ do
  if proc is origin then
    TIME(START($G_s = \{1 \ldots n - 1\}$))
    TIME(COMPLETE)
  else
    TIME(POST($G_p = \{0\}$))
    TIME(WAIT)
  end if
end for
```
Scaling

Increasing Number of Targets

![Graph showing latency and number of targets for POST (t_p), WAIT (t_w), START (t_s), and COMPLETE (t_c).]
Comparison with RCKMPI

- compare with SCC-tuned message-based RCKMPI/MPICH
- compare total time for both target and origin
Comparison with RCKMPI

- compare with SCC-tuned message-based RCKMPI/MPICH
- compare total time for both target and origin
Summary

- surveyed existing MPI OSC synchronization schemes
- discovered Open MPI synchronization bug
- surveyed existing MPI OSC synchronization schemes
- discovered Open MPI synchronization bug
- ported shared memory approach to nCC many-core CPU
- 4–5x faster synchronization than tuned message-based MPI
- no cache (coherence) required for OSC synchronization
Summary

- surveyed existing MPI OSC synchronization schemes
- discovered Open MPI synchronization bug
- ported shared memory approach to nCC many-core CPU
- 4–5x faster synchronization than tuned message-based MPI
- no cache (coherence) required for OSC synchronization

Thanks for your attention!

time for questions and suggestions…