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

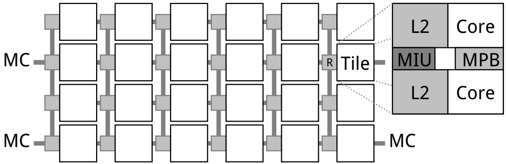
- Will future many-core systems provide hardware cache coherence?

Motivation

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

Motivation

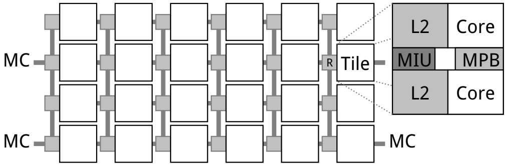
- Will future many-core systems provide hardware cache coherence? Not always → coherence islands in nCC systems
- nCC many-core research system: Intel SCC



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

Motivation

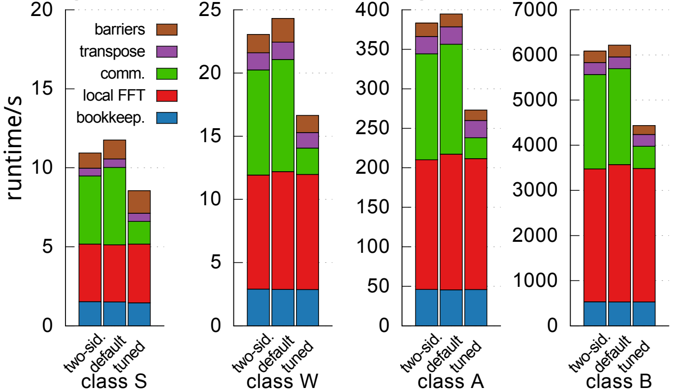
- Will future many-core systems provide hardware cache coherence? Not always $\rightarrow$ coherence islands in nCC systems
- nCC many-core research system: Intel SCC



  - 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

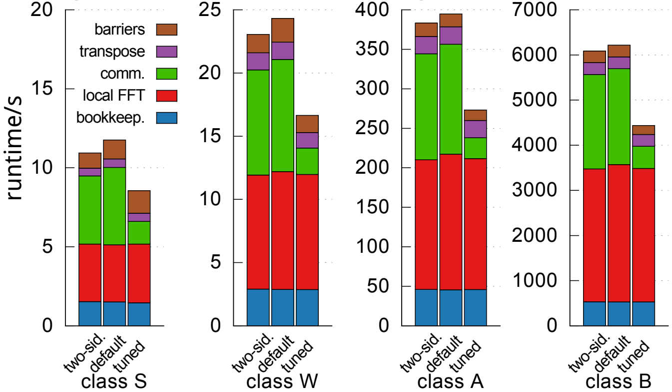
- previous results: reduce communication costs by factor of 4 – 5 when using OSC with software-managed cache coherence



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

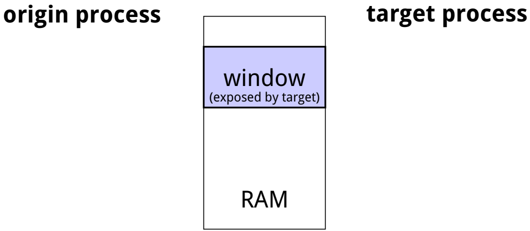
- sender *and* receiver must know communication parameters (buffer address, data type, tag, sender/receiver)
- implicit synchronization

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



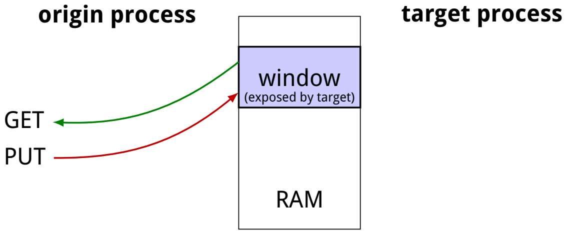
Background: MPI Communication Styles

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)



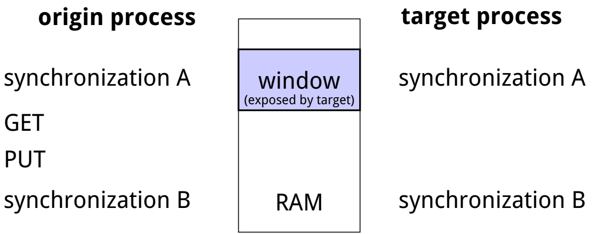
Background: MPI Communication Styles

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)



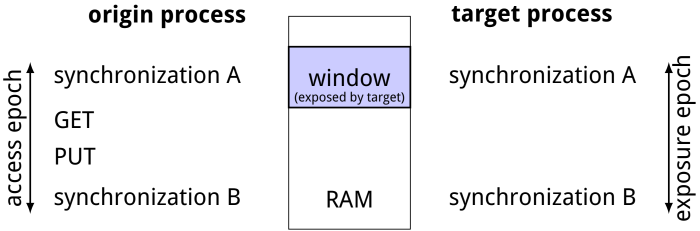
Background: MPI Communication Styles

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)



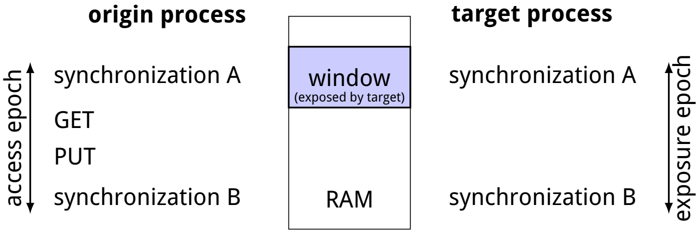
Background: MPI Communication Styles

two-sided communication

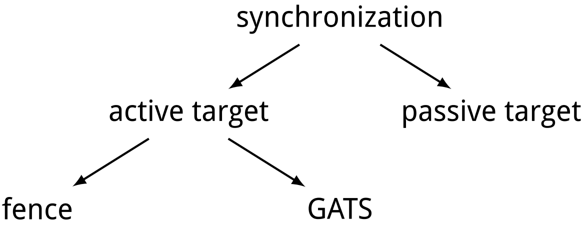
- sender *and* receiver must know communication parameters (buffer address, data type, tag, sender/receiver)
- implicit synchronization

one-sided communication

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

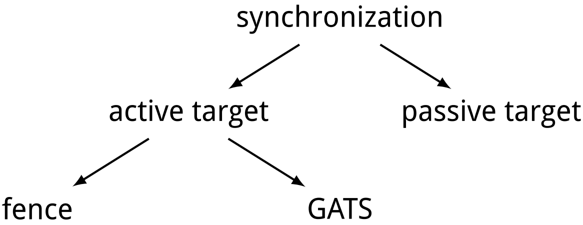


MPI Synchronization Types



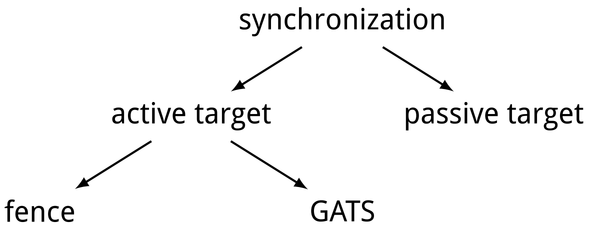
- active target: both sides involved

MPI Synchronization Types



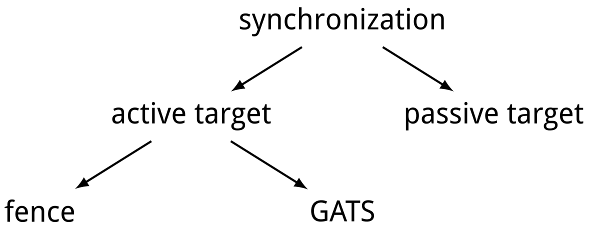
- active target: both sides involved
  - fence: global synchronization between all window sharers (similar to barrier)

MPI Synchronization Types



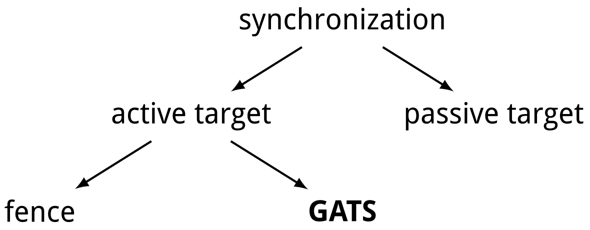
- active target: both sides involved
  - fence: global synchronization between all window sharers (similar to barrier)
  - general active target synchronization (GATS) synchronize subset only + specify role

MPI Synchronization Types



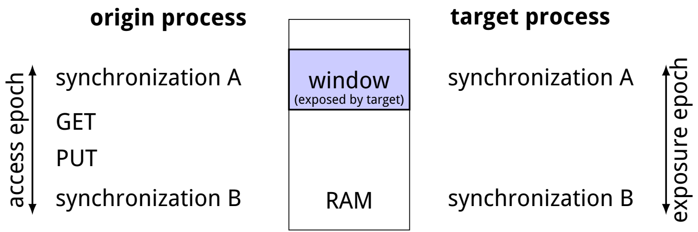
- active target: both sides involved
  - fence: global synchronization between all window sharers (similar to barrier)
  - general active target synchronization (GATS) synchronize subset only + specify role
- passive target: only origins synchronize

MPI Synchronization Types

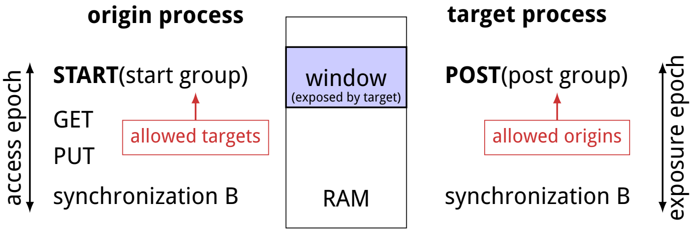


- active target: both sides involved
  - fence: global synchronization between all window sharers (similar to barrier)
  - **general active target synchronization (GATS)** synchronize subset only + specify role
- passive target: only origins synchronize

General Active Target Synchronization (GATS)



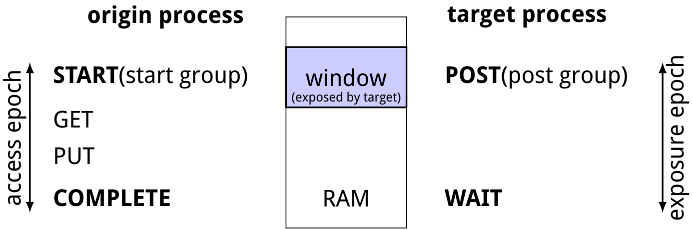
General Active Target Synchronization (GATS)



POST allow communication on own window

START open access epoch (allow communication calls)

General Active Target Synchronization (GATS)



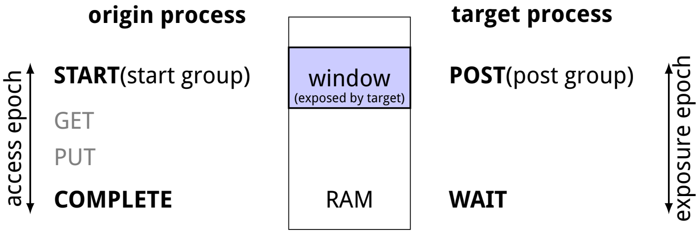
POST allow communication on own window

START open access epoch (allow communication calls)

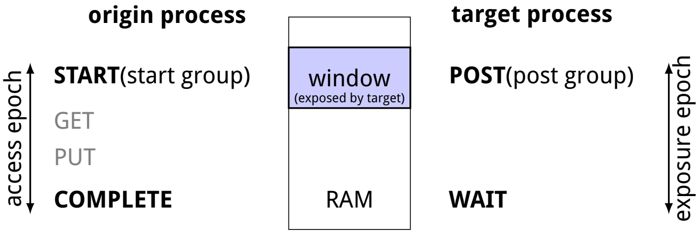
COMPLETE finish communication and notify targets

WAIT wait for all origin notifications

Classification of Implementations

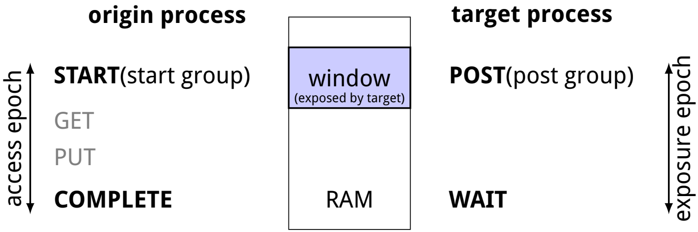


Classification of Implementations



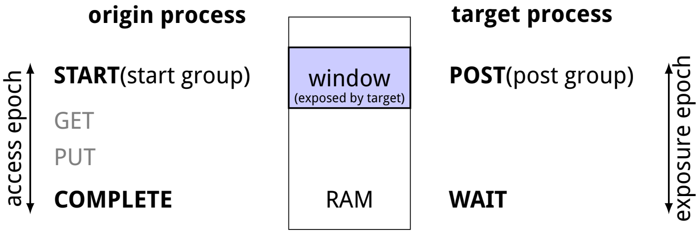
class	epoch start	communication	overlap
immediate	blocking	prompt	yes

Classification of Implementations



class	epoch start	communication	overlap
immediate	blocking	prompt	yes
deferred	non-blocking	delayed	no

Classification of Implementations



class	epoch start	communication	overlap
immediate	blocking	prompt	yes
deferred	non-blocking	delayed	no
trigger-only	non-blocking	prompt	yes

Implementation Survey

summary of conducted source code survey

class	epoch start	communication	overlap
immediate	blocking	prompt	yes
deferred	non-blocking	delayed	no
trigger-only	non-blocking	prompt	yes

Implementation Survey

summary of conducted source code survey

- MPICH: deferred + message-based

class	epoch start	communication	overlap
immediate	blocking	prompt	yes
deferred	non-blocking	delayed	no
trigger-only	non-blocking	prompt	yes

Implementation Survey

summary of conducted source code survey

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

class	epoch start	communication	overlap
immediate	blocking	prompt	yes
deferred	non-blocking	delayed	no
trigger-only	non-blocking	prompt	yes

Implementation Survey

summary of conducted source code survey

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

class	epoch start	communication	overlap
immediate	blocking	prompt	yes
deferred	non-blocking	delayed	no
trigger-only	non-blocking	prompt	yes

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

class	epoch start	communication	overlap
immediate	blocking	prompt	yes
deferred	non-blocking	delayed	no
trigger-only	non-blocking	prompt	yes

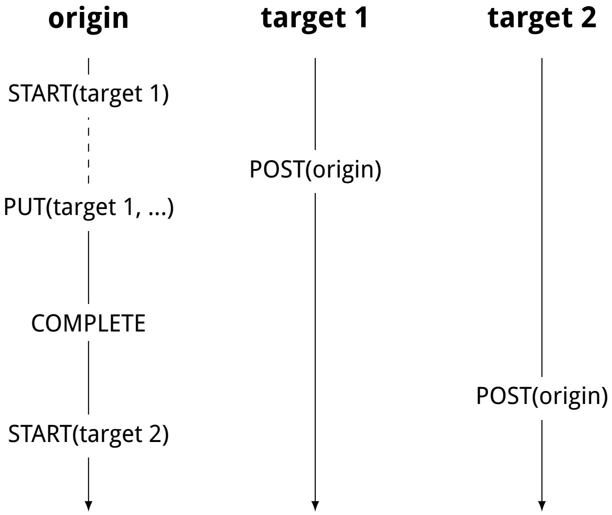
Implementation Survey

summary of conducted source code survey

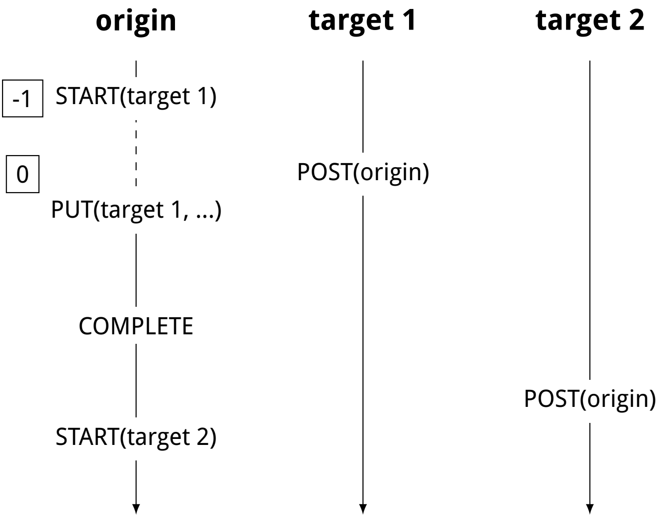
- MPICH: deferred + message-based
- MVAPICH/InfiniBand: immediate + InfiniBand RDMA
- Open MPI/InfiniBand: deferred + message-based
- **MVAPICH/shared memory: trigger-only + shared memory**
- **Open MPI/shared memory: immediate + shared memory**
  - two shared atomic counters per process (epoch start/end)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

class	epoch start	communication	overlap
immediate	blocking	prompt	yes
deferred	non-blocking	delayed	no
trigger-only	non-blocking	prompt	yes

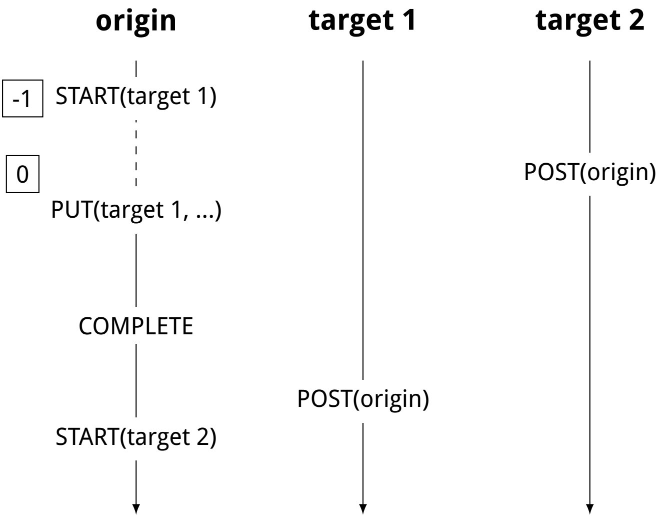
OpenMPI's GATS Bug



OpenMPI's GATS Bug



OpenMPI’s GATS Bug



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

class	epoch start	communication	overlap
immediate	blocking	prompt	yes
deferred	non-blocking	delayed	no
trigger-only	non-blocking	prompt	yes

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 $\rightarrow$ now MVAPICH-like

class	epoch start	communication	overlap
immediate	blocking	prompt	yes
deferred	non-blocking	delayed	no
trigger-only	non-blocking	prompt	yes

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

class	epoch start	communication	overlap
immediate	blocking	prompt	yes
deferred	non-blocking	delayed	no
trigger-only	non-blocking	prompt	yes

SCOSCo: Implementation on the SCC

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)

Data Structures

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

Data Structures

match vector = bit vector

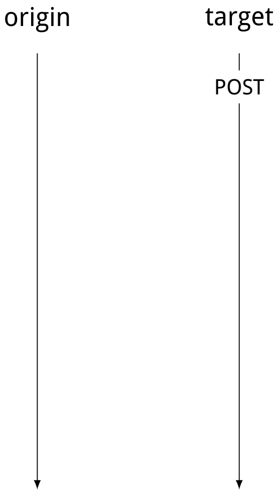
- for synchronization at epoch start:
- one vector per process, placed in shared memory
- dedicated to origin processes
- k-th bit set $\Rightarrow$ target process with ID/rank k synchronized

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

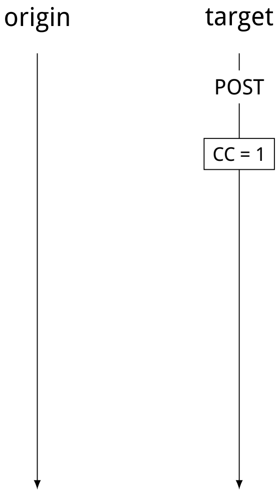
POST(post_group)



Synchronization: Epoch Start

POST(post_group)

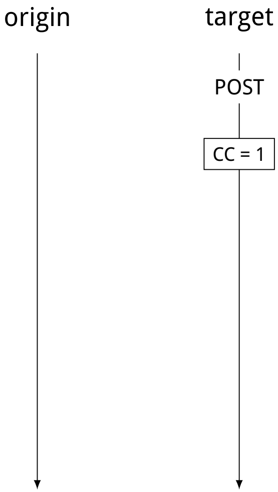
- CC $\leftarrow$ number of given origins



Synchronization: Epoch Start

POST(post_group)

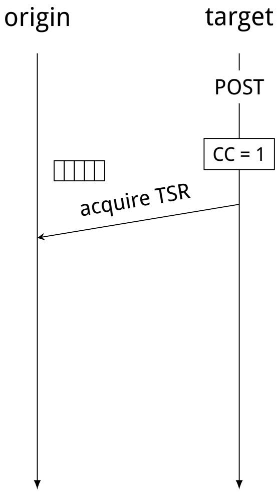
- CC $\leftarrow$ number of given origins
- signal all origins $\in$ post group:



Synchronization: Epoch Start

POST(post_group)

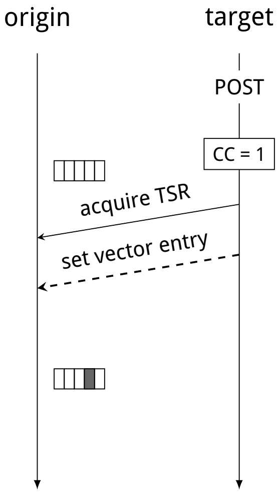
- CC $\leftarrow$ number of given origins
- signal all origins $\in$ post group:
  - acquire origins's Test&Set Register



Synchronization: Epoch Start

POST(post_group)

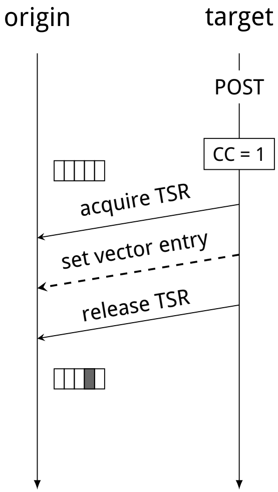
- CC $\leftarrow$ number of given origins
- signal all origins $\in$ post group:
  - acquire origins's Test&Set Register
  - set entry in match vector



Synchronization: Epoch Start

POST(post_group)

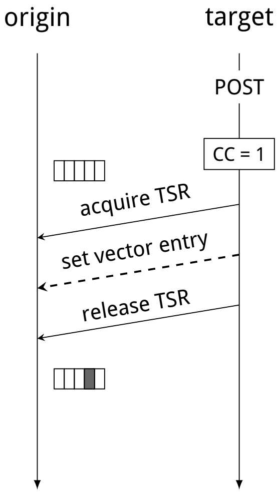
- CC $\leftarrow$ number of given origins
- signal all origins $\in$ post group:
  - acquire origins's Test&Set Register
  - set entry in match vector
  - release origins's TSR



Synchronization: Epoch Start

POST(post_group)

- CC $\leftarrow$ number of given origins
- signal all origins $\in$ 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**



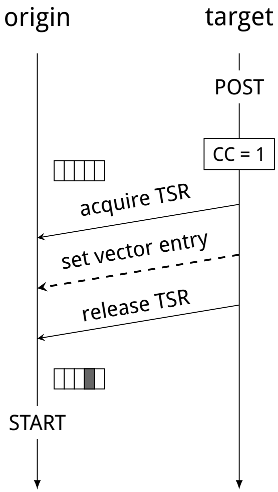
Synchronization: Epoch Start

POST(post_group)

- CC $\leftarrow$ number of given origins
- signal all origins $\in$ 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



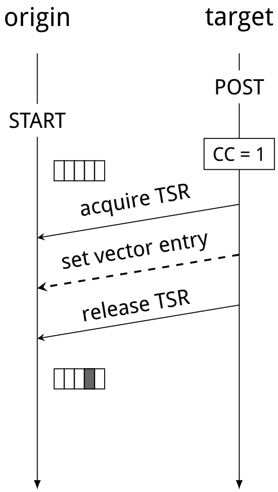
Synchronization: Epoch Start

POST(post_group)

- CC $\leftarrow$ number of given origins
- signal all origins $\in$ 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**

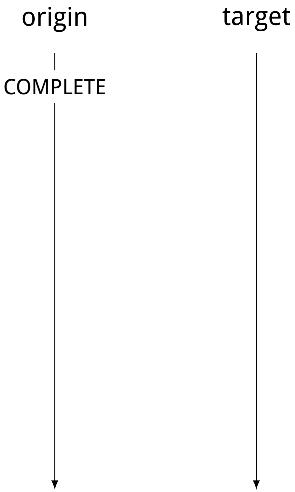
START(start_group)

- save list of target processes



Synchronization: Epoch End

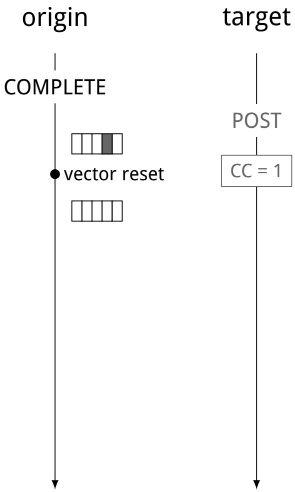
COMPLETE



Synchronization: Epoch End

COMPLETE

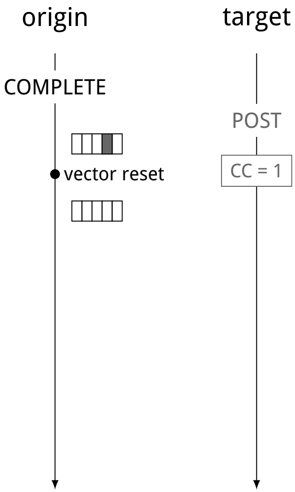
- wait for all (remaining) targets
- reset match vector entries



Synchronization: Epoch End

COMPLETE

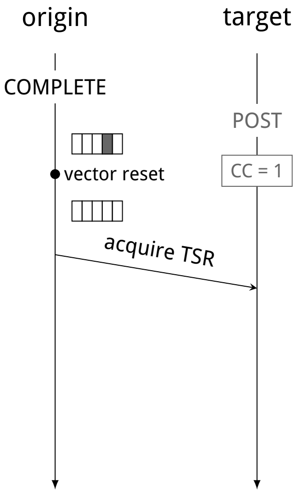
- wait for all (remaining) targets
- reset match vector entries
- for all saved targets:



Synchronization: Epoch End

COMPLETE

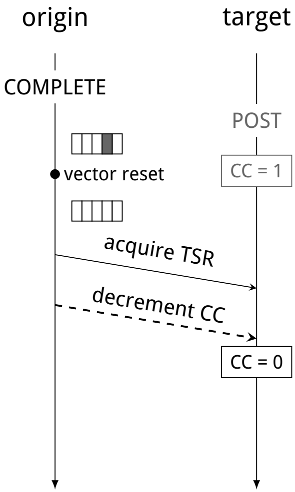
- wait for all (remaining) targets
- reset match vector entries
- for all saved targets:
  - acquire target's TSR



Synchronization: Epoch End

COMPLETE

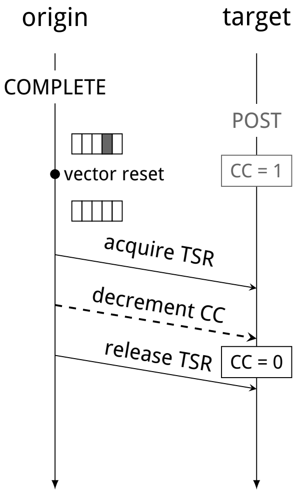
- wait for all (remaining) targets
- reset match vector entries
- for all saved targets:
  - acquire target's TSR
  - **uncached CC decrement**



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



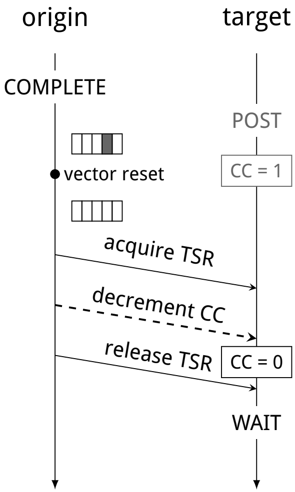
Synchronization: Epoch End

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



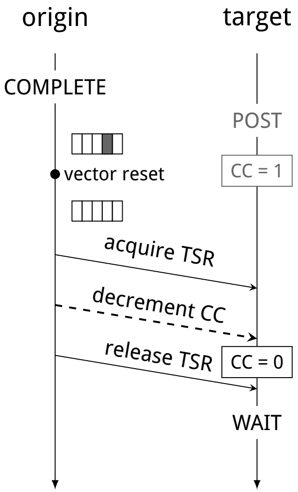
Synchronization: Epoch End

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 $\rightarrow$ create own one
- no communication $\rightarrow$ fair comparison of synchronization

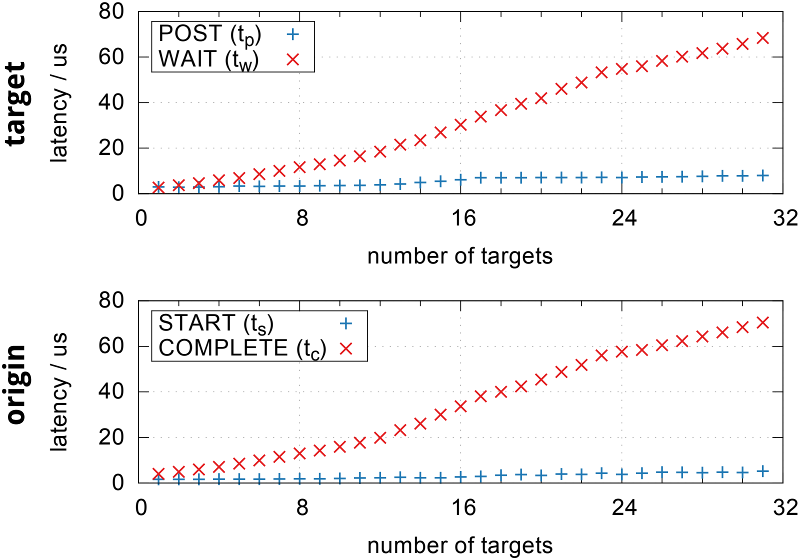
Microbenchmark

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

for $i = 0 \ldots 1000$ **do**
  if proc is origin **then**
    $\textsc{time}(\textsc{Start}(G_s = \{1 \ldots n-1\}))$
    $\textsc{time}(\textsc{Complete})$
  else
    $\textsc{time}(\textsc{Post}(G_p = \{0\}))$
    $\textsc{time}(\textsc{Wait})$
  end if
end for

Scaling

Increasing Number of Targets

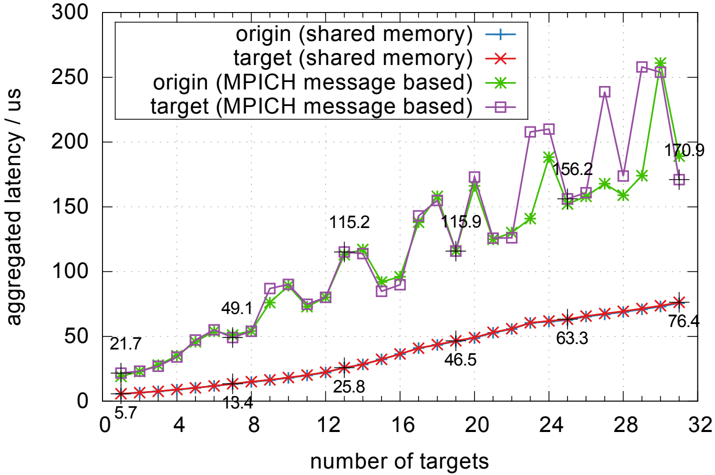


Comparison with RCKMPI

- 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

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

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…