Comparing MPI Passive Target Synchronization Schemes on a Non-Cache-Coherent Shared-Memory Processor

Steffen Christgau[1,(2)] Bettina Schnor[2]

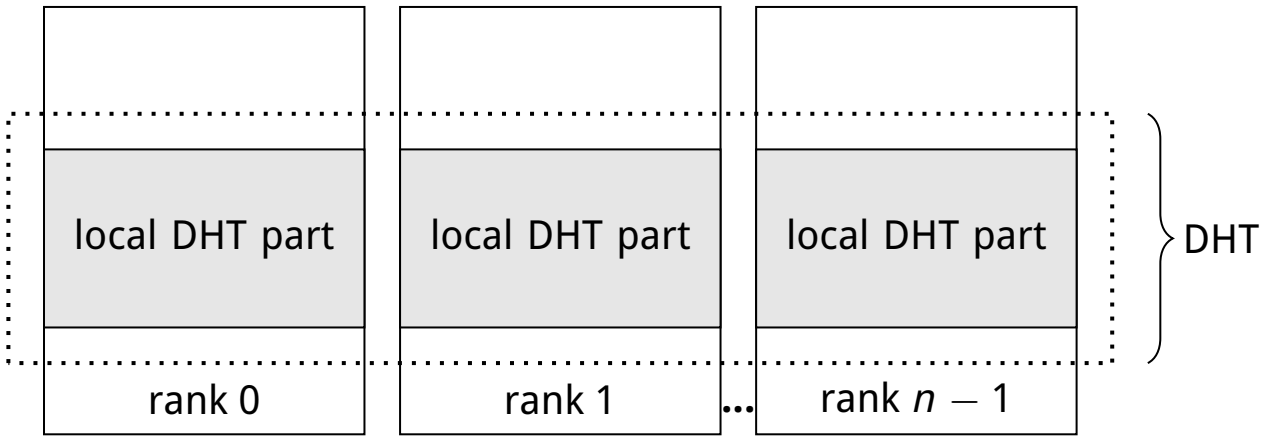
[1] Supercomputing Department
Zuse Institute Berlin

[2] Operating Systems and Distributed Systems
Institute for Computer Science, University of Potsdam

28th PARS Workshop, Berlin, Germany, March 22 2019

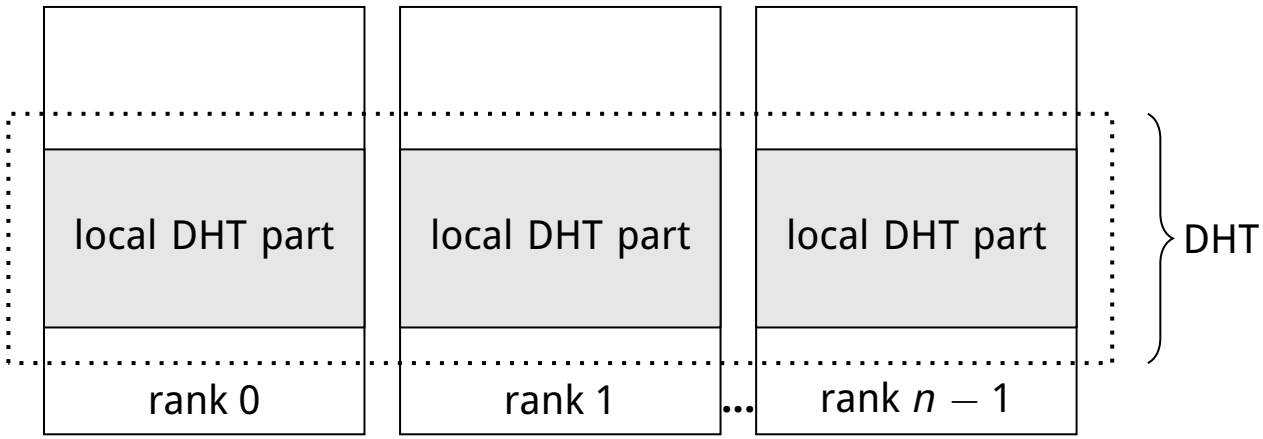
Motivation: Distributed Hash Table (DHT)

- hash table as cache for computational results in **MPI** application
- large amount of data $\rightarrow$ distribute across processes $\rightarrow$ DHT



Motivation: Distributed Hash Table (DHT)

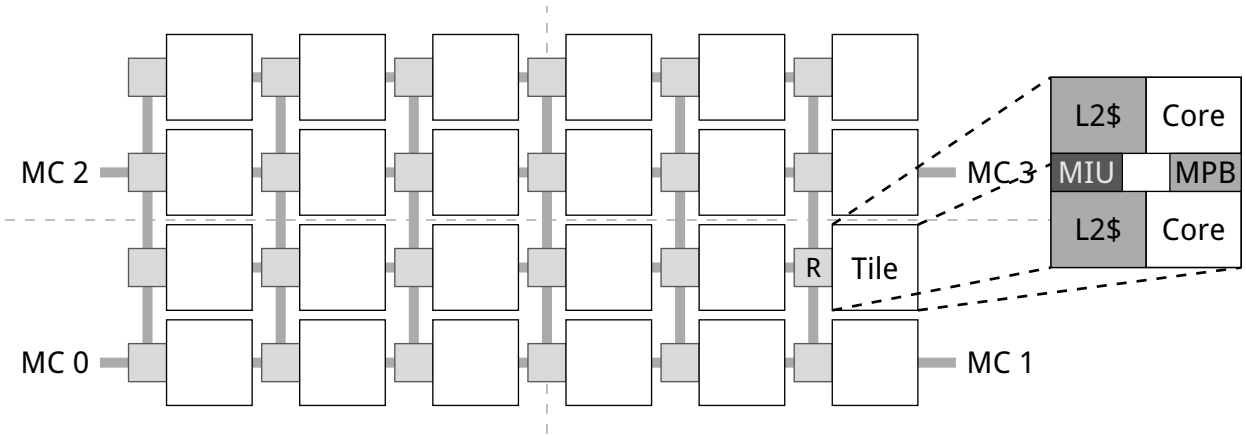
- hash table as cache for computational results in **MPI** application
- large amount of data $\rightarrow$ distribute across processes $\rightarrow$ DHT



- accessing distributed data:
  - hash function returns arbitrary process and address
  - difficult to program with two-sided message passing
  - **MPI passive target one-sided communication** to the rescue
  - **synchronization required**

Research Platform

- nCC NUMA many-core research system: Intel SCC
  - 48 Pentium cores with L1/2 caches, mesh network
  - **no HW cache coherence**



Research Platform

- nCC NUMA many-core research system: Intel SCC
  - 48 Pentium cores with L1/2 caches, mesh network
  - **no HW cache coherence**

Likely Exascale Architectures

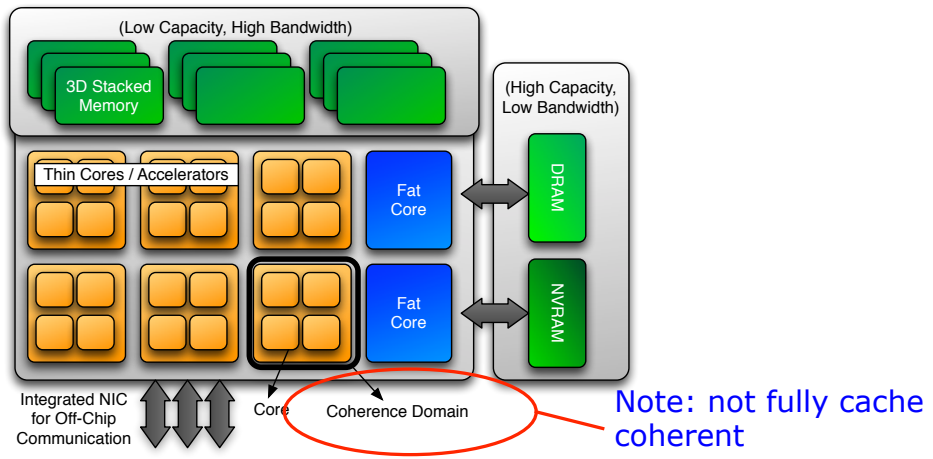


Figure 2.1: Abstract Machine Model of an exascale Node Architecture

- From "Abstract Machine Models and Proxy Architectures for Exascale Computing Rev 1.1,'

(William Gropp. MPI: The Once and Future King. EuroMPI 2016 Keynote, Edinburg)

Research Platform

- nCC NUMA many-core research system: Intel SCC
  - 48 Pentium cores with L1/2 caches, mesh network
  - **no HW cache coherence**

Likely Exascale Architectures

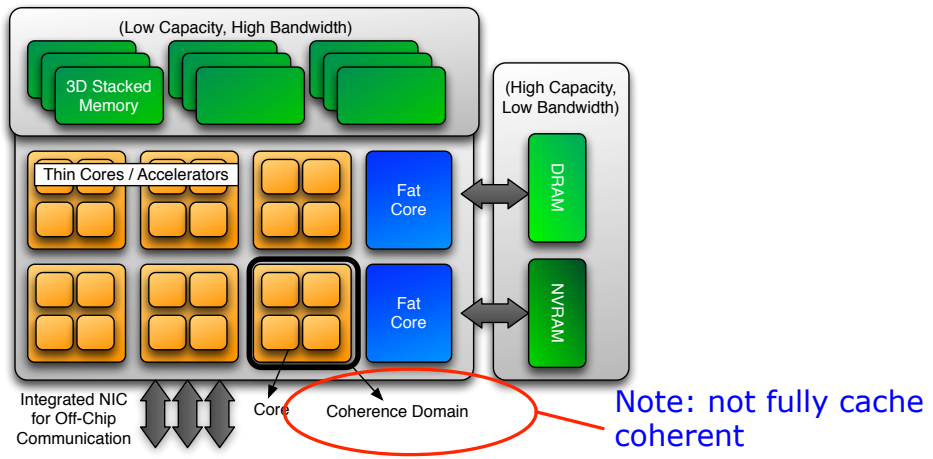


Figure 2.1: Abstract Machine Model of an exascale Node Architecture

- From "Abstract Machine Models and Proxy Architectures for Exascale Computing Rev 1.1,'

(William Gropp. MPI: The Once and Future King. EuroMPI 2016 Keynote, Edinburg)

- This talk: **comparison of synchronization schemes** for MPI passive target OSC

Outline

MPI Passive Target One-Sided Communication

origin process **target process**

```
MPI_WIN_LOCK(TYPE, target, win)
```

RMA operations


```
MPI_WIN_UNLOCK(target, win)
```

MPI Passive Target One-Sided Communication

origin process **target process**

```
MPI_WIN_LOCK(TYPE, target, win)
```

RMA operations


```
MPI_WIN_UNLOCK(target, win)
```

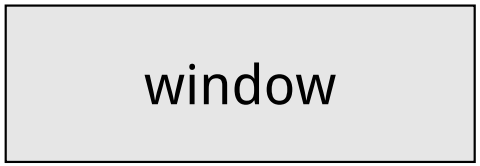
- Operations guaranteed to be finished only after `UNLOCK`.
  - Standard defines LOCK as start of accesses only $\rightarrow$ *epoch start*
  - Other processes may proceed after (exclusive) LOCK as well.

MPI Passive Target One-Sided Communication

origin process **target process**

```
MPI_WIN_LOCK(TYPE, target, win)
```

RMA operations




```
MPI_WIN_UNLOCK(target, win)
```

- Operations guaranteed to be finished only after `UNLOCK`.
  - Standard defines LOCK as start of accesses only $\rightarrow$ *epoch start*
  - Other processes may proceed after (exclusive) LOCK as well.
- Two lock types

  `EXCLUSIVE` accesses protected against concurrent access on window site

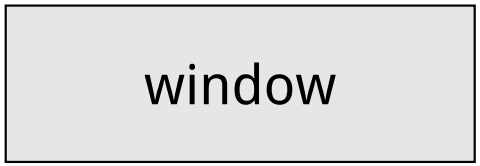
  `SHARED` no concurrent accesses protected by EXCLUSIVE lock

MPI Passive Target One-Sided Communication

origin process **target process**

```
MPI_WIN_LOCK(TYPE, target, win)
```

RMA operations




```
MPI_WIN_UNLOCK(target, win)
```

- Operations guaranteed to be finished only after `UNLOCK`.
  - Standard defines LOCK as start of accesses only $\rightarrow$ *epoch start*
  - Other processes may proceed after (exclusive) LOCK as well.
- Two lock types

  `EXCLUSIVE` accesses protected against concurrent access on window site

  `SHARED` no concurrent accesses protected by EXCLUSIVE lock
- **Implementor's freedom**: `LOCK` may block (or not)
- `LOCKALL` operation: `SHARED` lock on all processes

Using Synchronization for DHT

- for DHT: write with EXCLUSIVE lock, read with SHARED lock

`DHT_write`

```
LOCK(win, target, EXCLUSIVE)
PUT(win, target, data)
UNLOCK(win, target)
```

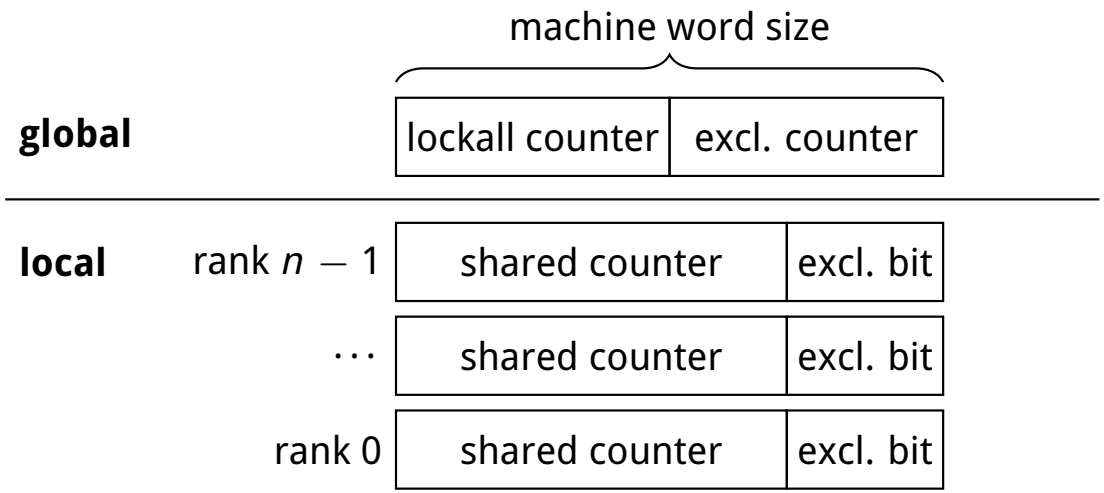
`DHT_read`

```
LOCK(win, target, SHARED)
GET(win, target, &data)
UNLOCK(win, target)
```

- desired behavior: writers get precedence $\rightarrow$ fresh data for readers
  - not enforceable through MPI standard
  - implementation may support such behavoir (via `INFO_KEY`, e.g.)

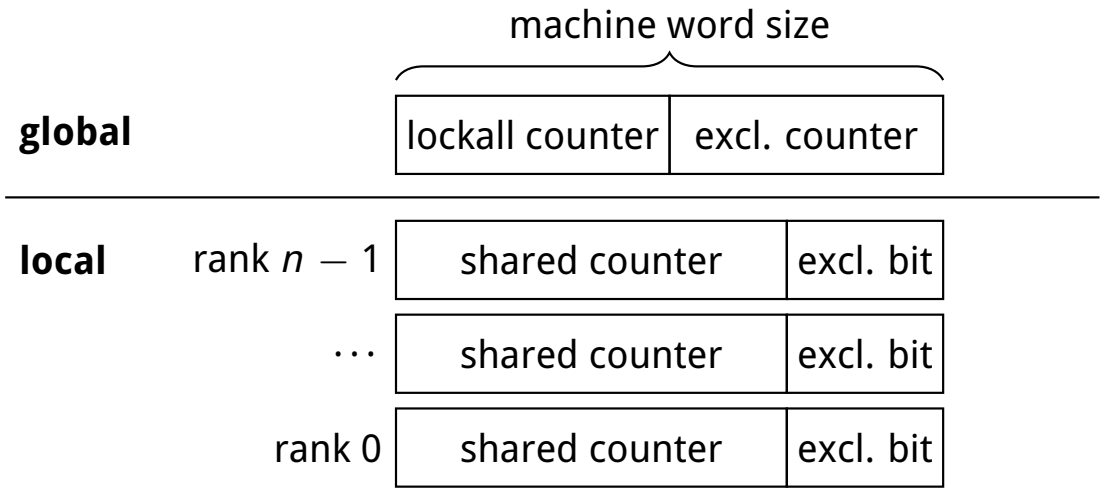
GBH: Best Effort Locks

- Design by *Gerstenberger*, *Besta*, and *Hoefler* for Cray XC super-computers, fully supports MPI passive target API
- RDMA-accessible data: **centralized global counter**, distributed local counters, but **single polling resource**



GBH: Best Effort Locks

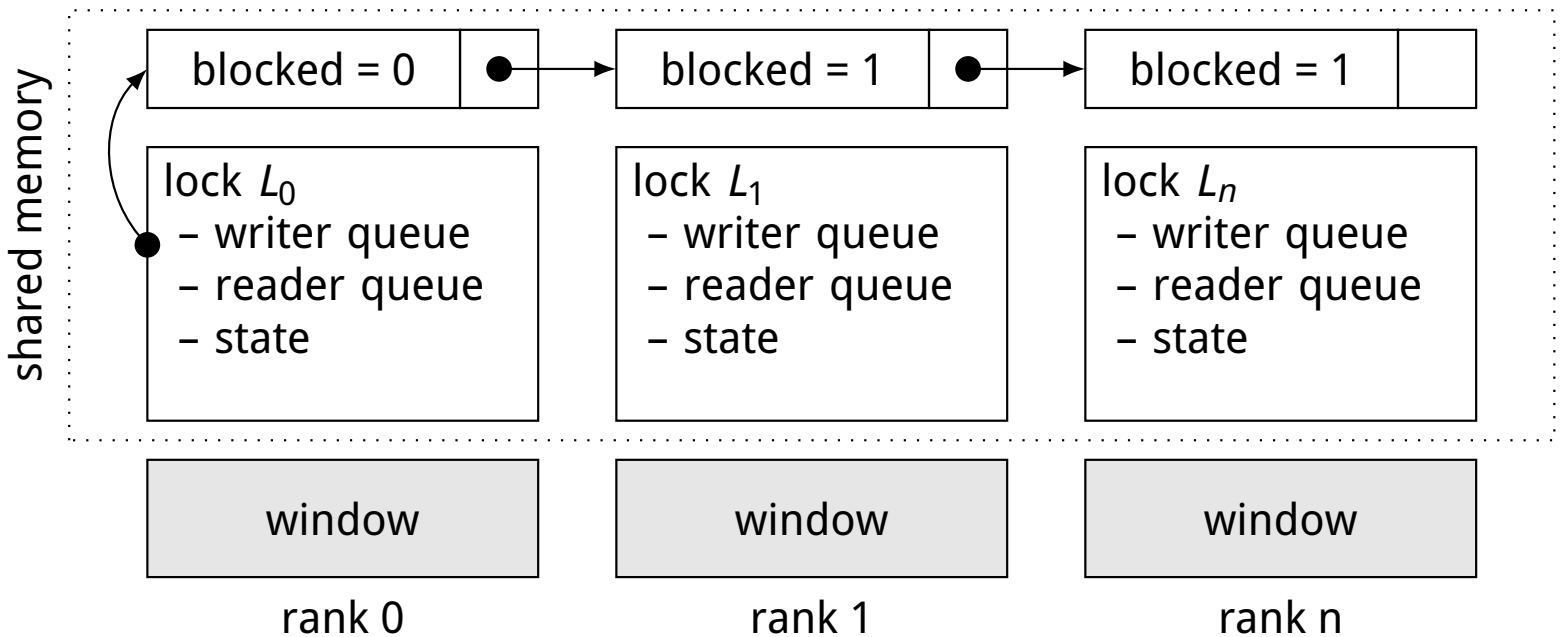
- Design by *Gerstenberger*, *Besta*, and *Hoefler* for Cray XC super-computers, fully supports MPI passive target API
- RDMA-accessible data: **centralized global counter**, distributed local counters, but **single polling resource**



- best effort: try to acquire lock, step-back if conflict detected (with exponentially increasing back-off)
- No precedence of arriving process type.

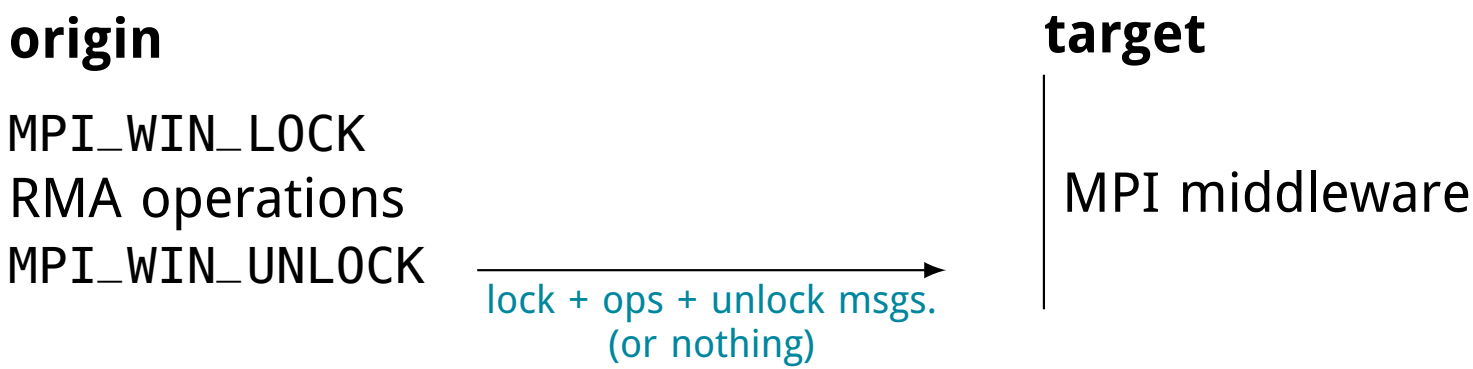
MCS-WP: Locks with Writer Precedence

- based on classical paper by Mellor-Crummey and Scott (MCS)
- state and queues for readers and writers per lock
- locally allocated (**distributed**) queue items used for spinning
- one MCS-WP lock per window/process
- ordered writer precedence, no support for LOCKALL



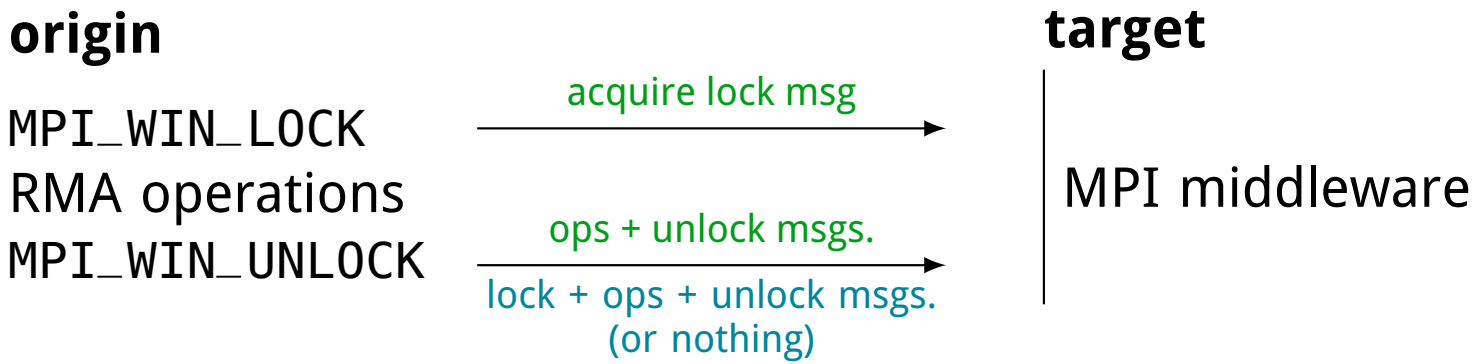
Message-based Synchronization

- used in MPICH (CH3 device)
- default behavoir: actions deferred to end of access epoch
  - lock acquisition by control message at end of epoch
  - **no message if no RMA operations** $\rightarrow$ `LOCK` + `UNLOCK` = `NO-OP`



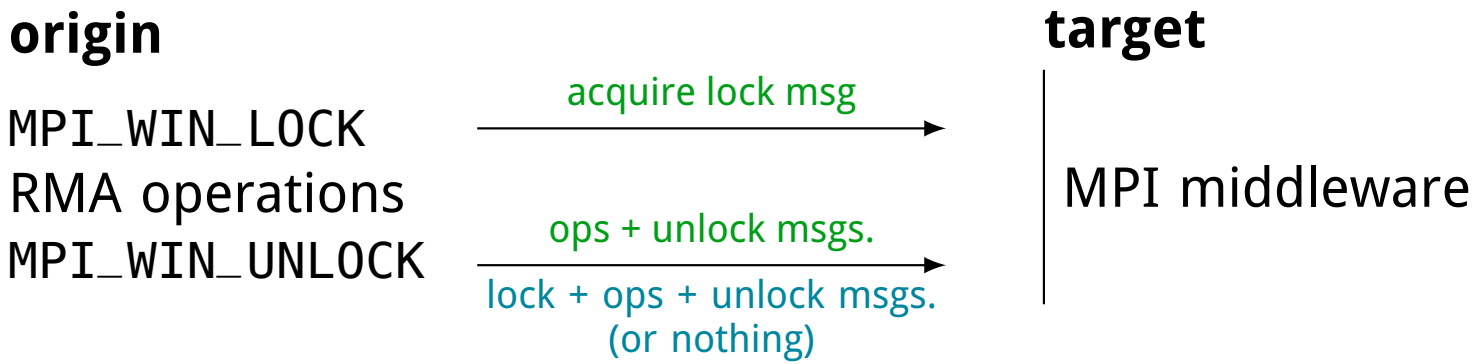
Message-based Synchronization

- used in MPICH (CH3 device)
- default behavoir: actions deferred to end of access epoch
  - lock acquisition by control message at end of epoch
  - **no message if no RMA operations** $\rightarrow$ `LOCK` + `UNLOCK` = `NO-OP`
- can be switched to immediate messaging
  - force message in `LOCK` call
  - wait for reply (*lock-granted* message) in `UNLOCK`



Message-based Synchronization

- used in MPICH (CH3 device)
- default behavoir: actions deferred to end of access epoch
  - lock acquisition by control message at end of epoch
  - **no message if no RMA operations** $\rightarrow$ `LOCK` + `UNLOCK` = `NO-OP`
- can be switched to immediate messaging
  - force message in `LOCK` call
  - wait for reply (*lock-granted* message) in `UNLOCK`
- serialization of messages at target process
  - no precedence of either process type
  - processing in target' MPI middleware



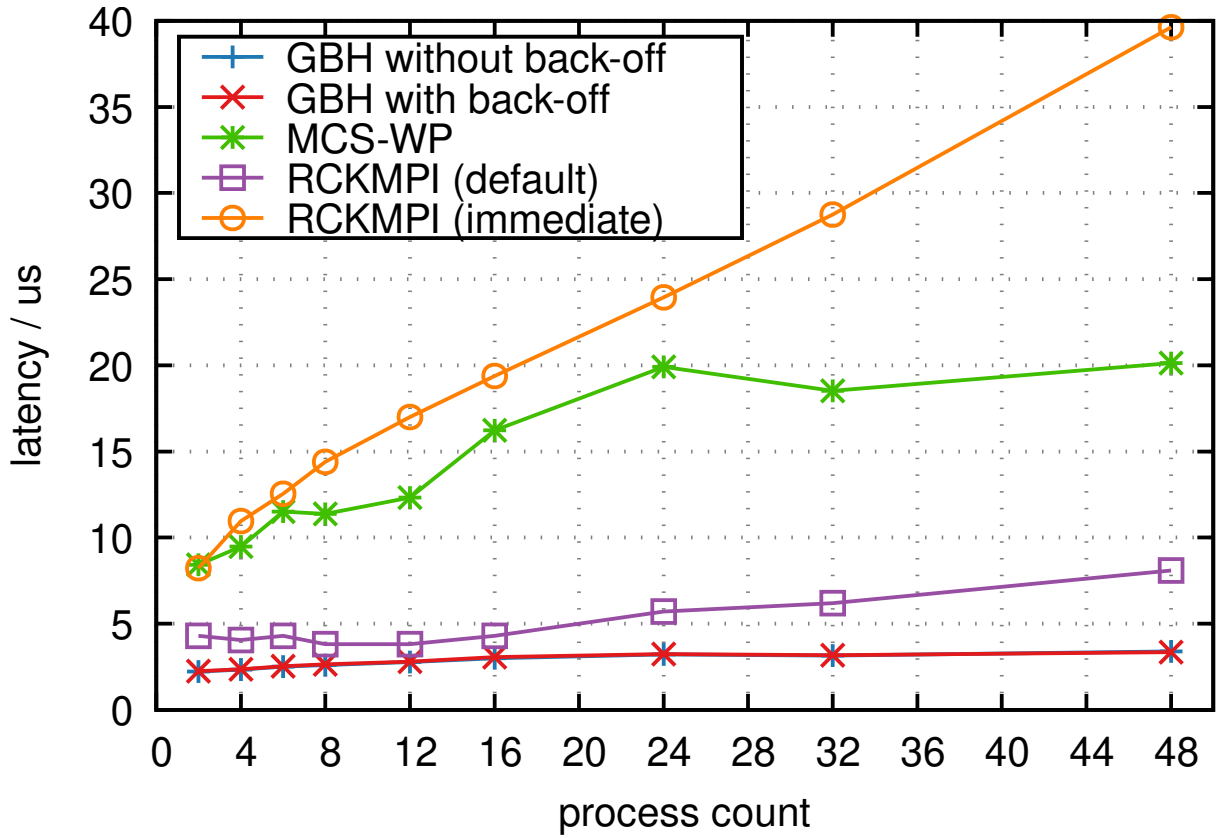
Implementation

- Implementation based on RCKMPI
  - SCC-tuned MPICH derivate, exploits hardware's message passing features
  - completely message-based communication and synchronization (inherited from CH3)
- GBH and MCS-WP implemented based on OSC modifications
  - data strutures allocated in shared memory
  - uncached memory accesses
  - OSC modification: shared memory with SW-based coherence for communication

Microbenchmark

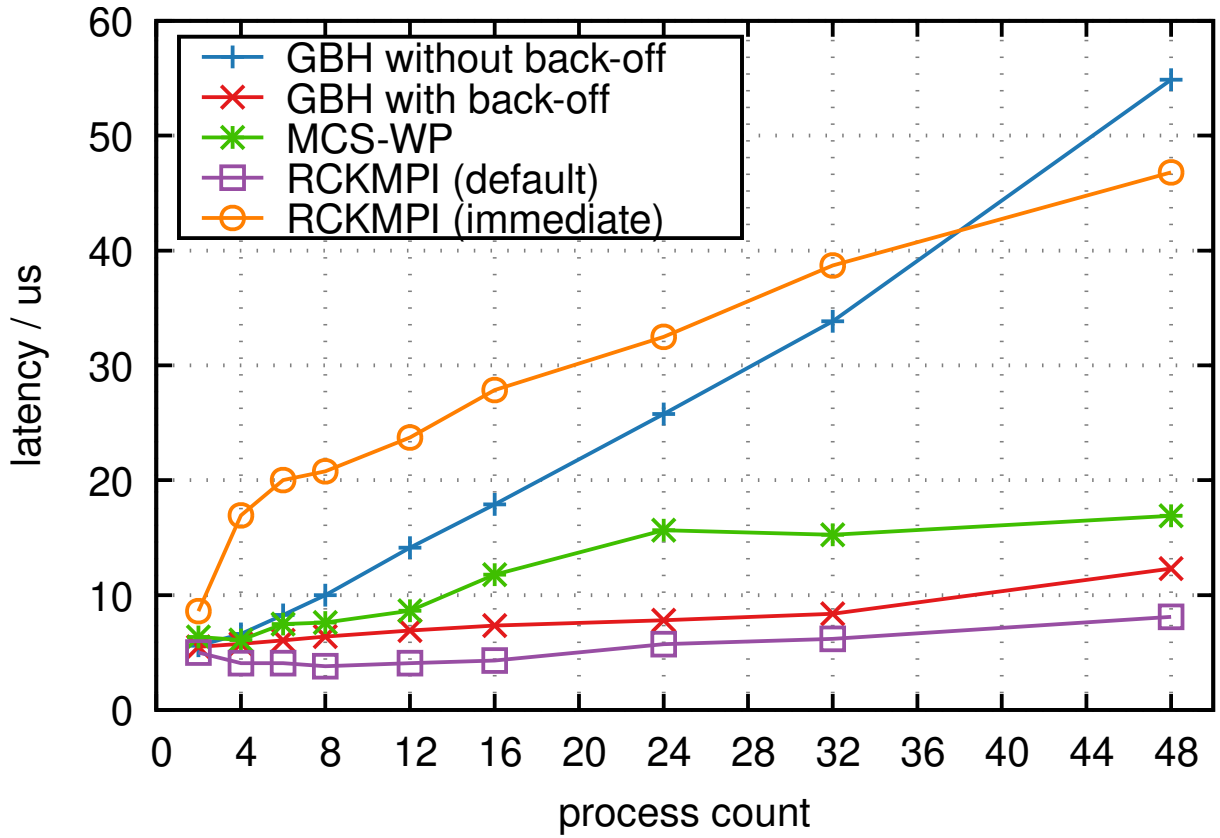
- assess latency of synchronization for different lock type mixes
- no communication involved
  - different communication implementations $\rightarrow$ unfair comparison
  - measure pure synchronization overhead only
- perform tight loop of `LOCK`/`UNLOCK` operations
  - choose between shared and exclusive lock according to given ratio $\rightarrow$ vary share of writers/readers
  - measure time t_i per iteration
  - collect all t_i from all n processes, report median
  - evaluate median latency for increasing process count

Benchmark Results: only shared accesses/locks



- RCKMPI (immediate) slow although tuned message-transfer
- constant latency for GBH $\rightarrow$ single increment
- list management overhead for MCS-WP

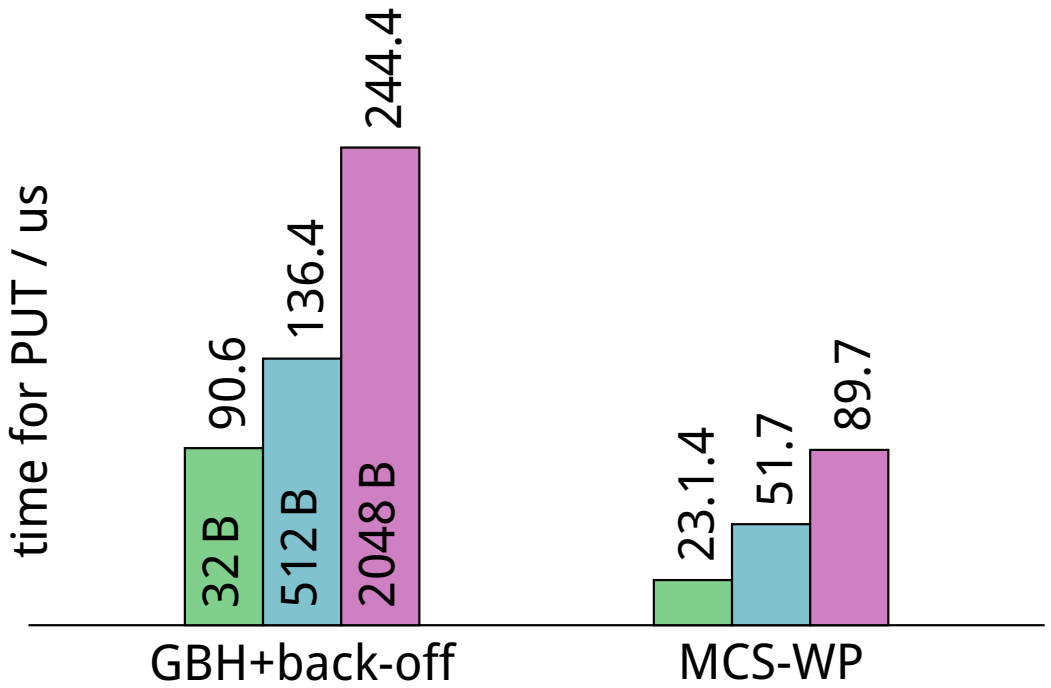
Benchmark Results: exclusive accesses/locks only



- backoff is essential for GBH performance
- consistent performance for MCS-WP

Evaluation for DHT

- 1 writer + 47 concurrent readers access same window (DHT portion)
- measure time for writer to store $k \in \{32, 512, 2048\}$ bytes
- compare GBH with back-off and MCS-WP



- GBH puts more stress on single memory controller, MCS-WP benefits from completely distributed synchronization data

Summary

- Successfully applied algorithms for RMA- and coherent shared memory systems on non-cache-cohrent one
- Superior performance compared to tuned message-based approach
- Distribution of synchronization data is critical.

Questions!?