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

Steffen Christgau¹,(2)  Bettina Schnor²

¹Supercomputing Department
  Zuse Institute Berlin

²Operating 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 → distribute across processes → DHT

![Diagram of distributed hash table with local DHT parts for ranks 0 to n-1.](image-url)
Motivation: Distributed Hash Table (DHT)

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

![Diagram of distributed hash table]

- 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


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

Note: not fully cache coherent

(William Gropp. MPI: The Once and Future King. EuroMPI 2016 Keynote, Edinburg)
MPI Passive Target One-Sided Communication

origin process

MPI_WIN_LOCK(TYPE, target, win)

RMA operations

MPI_WIN_UNLOCK(target, win)

target process

window
MPI Passive Target One-Sided Communication

origin process

MPI_WIN_LOCK(TYPE, target, win)

RMA operations

MPI_WIN_UNLOCK(target, win)

target process

• Operations guaranteed to be finished only after UNLOCK.
  ■ Standard defines LOCK as start of accesses only → *epoch start*
  ■ Other processes may proceed after (exclusive) LOCK as well.
**MPI Passive Target One-Sided Communication**

**origin process**

MPI\_WIN\_LOCK(\textbf{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 $\rightarrow$ \textit{epoch start}
  - Other processes may proceed after (exclusive) LOCK as well.

- Two lock types
  - \textbf{EXCLUSIVE} accesses protected against concurrent access on window site
  - \textbf{SHARED} no concurrent accesses protected by EXCLUSIVE lock
MPI Passive Target One-Sided Communication

**origin process**

MPI\_WIN\_LOCK\(\text{TYPE}, \text{target}, \text{win}\)

RMA operations

MPI\_WIN\_UNLOCK\(\text{target}, \text{win}\)

- 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

<table>
<thead>
<tr>
<th>DHT_write</th>
<th>DHT_read</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOCK(win, target, EXCLUSIVE)</td>
<td>LOCK(win, target, SHARED)</td>
</tr>
<tr>
<td>PUT(win, target, data)</td>
<td>GET(win, target, &amp;data)</td>
</tr>
<tr>
<td>UNLOCK(win, target)</td>
<td>UNLOCK(win, target)</td>
</tr>
</tbody>
</table>

- desired behavior: writers get precedence → fresh data for readers
  - not enforceable through MPI standard
  - implementation may support such behaviour (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**

<table>
<thead>
<tr>
<th></th>
<th>machine word size</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>global</strong></td>
<td>lockall counter</td>
</tr>
<tr>
<td><strong>local</strong></td>
<td>rank $n - 1$</td>
</tr>
<tr>
<td></td>
<td>$\cdots$</td>
</tr>
<tr>
<td></td>
<td>rank 0</td>
</tr>
</tbody>
</table>
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 behaviour: actions deferred to end of access epoch
  - lock acquisition by control message at end of epoch
  - no message if no RMA operations → LOCK + UNLOCK = NO-OP

<table>
<thead>
<tr>
<th>origin</th>
<th>target</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_WIN_LOCK</td>
<td>MPI middleware</td>
</tr>
<tr>
<td>RMA operations</td>
<td></td>
</tr>
<tr>
<td>MPI_WIN_UNLOCK</td>
<td></td>
</tr>
<tr>
<td>lock + ops + unlock msgs. (or nothing)</td>
<td></td>
</tr>
</tbody>
</table>
Message-based Synchronization

- used in MPICH (CH3 device)
- default behaviour: 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

**origin**

| MPI\_WIN\_LOCK | acquire lock msg |
| RMA operations | ops + unlock msgs. |
| MPI\_WIN\_UNLOCK | lock + ops + unlock msgs. (or nothing) |

**target**

MPI middleware
Message-based Synchronization

- used in MPICH (CH3 device)
- default behaviour: 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 (\textit{lock-granted} message) in UNLOCK
- serialization of messages at target process
  - no precedence of either process type
  - processing in target’ MPI middleware

\textbf{origin}

- \texttt{MPI\_WIN\_LOCK}
- RMA operations
- \texttt{MPI\_WIN\_UNLOCK}

\textbf{target}

- MPI middleware

\begin{align*}
\text{acquire lock msg} \\
\text{ops + unlock msgs.} \\
\text{lock + ops + unlock msgs. (or nothing)}
\end{align*}
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 → unfair comparison
  - measure pure synchronization overhead only
- perform tight loop of LOCK/UNLOCK operations
  - choose between shared and exclusive lock according to given ratio →
    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

![Bar chart showing time for PUT in us for GBH+back-off and MCS-WP for different byte sizes.]

- 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-coherent one
- Superior performance compared to tuned message-based approach
- Distribution of synchronization data is critical.

Questions!? 