Design of MPI Passive Target Synchronization for a Non-Cache-Coherent Many-Core Processor

27th PARS Workshop, Hagen, Germany, May 5 2017

Steffen Christgau, Bettina Schnor

Operating Systems and Distributed Systems Institute for Computer Science University of Potsdam, Germany
Motivation: Distributed Hash Table (DHT)

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

![Diagram of DHT]

- **DHT**
- local DHT part
- rank 0
- rank 1
- rank \(n-1\)

**Accessing Distributed Data:** Hash function returns an arbitrary process and address. Distribution can be handled through two-sided messaging or one-sided communication to one process, requiring synchronization.
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](image)

- 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**
Motivation: nCC Systems

- Future many-cores may not provide (global) cache coherence.
Motivation: nCC Systems

- Future many-cores may not provide (global) cache coherence.
  - Intel Knights Landing: coherent multi-socket systems not feasible
Motivation: nCC Systems

- Future many-cores may not provide (global) cache coherence.
  - Intel Knights Landing: coherent multi-socket systems not feasible
  - HPE "The Machine", EuroServer: coherence islands

https://regmedia.co.uk/2016/11/22/the_machine_universal_memory_pool_access.jpg
• nCC many-core research system: Intel SCC
  - 48 Pentium cores with L1/2 caches
  - no HW cache coherence
• nCC many-core research system: Intel SCC
  ▪ 48 Pentium cores with L1/2 caches
  ▪ no HW cache coherence

• This talk: design of synchronization on nCC platform.
Agenda

MPI Passive Target One-Sided Communication

Design for Passive Target Synchronization on the SCC
Data Structures and Algorithms
Data Placement

Outlook and Future Work
MPI One-Sided Communication

- process memory exposed via windows
• process memory exposed via windows

MPI One-Sided Communication

process' address space

local DHT part (window)  
rank 0

local DHT part (window)  
rank 1

local DHT part (window)  
rank n – 1

DHT
MPI One-Sided Communication

- process memory exposed via **windows**
- access to windows with **window object** (handle)

![Diagram showing process address space and window objects for different ranks.](image-url)
• process memory exposed via **windows**
• access to windows with **window object** (handle)

**key concept:** only one communication partner issues communication operations
MPI One-Sided Communication

- process memory exposed via **windows**
- access to windows with **window object** (handle)

![Diagram showing process address space and DHT]

- **key concept**: only one communication partner issues communication operations
  - **origin** processes issue communication operations
• process memory exposed via **windows**
• access to windows with **window object** (handle)

**key concept**: only one communication partner issues communication operations
- **origin** processes issue communication operations
- **target** processes are addressed by operations
MPI One-Sided Communication

- process memory exposed via **windows**
- access to windows with **window object** (handle)

**key concept**: only one communication partner issues communication operations

- **origin** processes issue communication operations
- **target** processes are addressed by operations
- typical RMA operations: PUT, GET, ...
MPI One-Sided Communication

- process memory exposed via **windows**
- access to windows with **window object** (handle)

![Diagram showing process address space with window objects and local DHT parts]

**key concept**: only one communication partner issues communication operations

- **origin** processes issue communication operations
- **target** processes are addressed by operations
- typical RMA operations: PUT, GET, ...
- explicit synchronization required
MPI Passive Target Synchronization

- **locks** as means for synchronization, used by origins only
- no participation of targets in synchronization (passive targets)
• **locks** as means for synchronization, used by origins only
• no participation of targets in synchronization (passive targets)
• usage similar to shared memory locks
  1. acquire lock for target window \( \text{WIN\_LOCK}(\text{win}, \text{rank}, \ldots) \)
  2. perform operations \( \text{PUT}(\text{win}, \text{rank}, \ldots) \)
  3. release lock \( \text{WIN\_UNLOCK}(\text{win}, \text{rank}) \)
MPI Passive Target Synchronization

- **locks** as means for synchronization, used by origins only
- no participation of targets in synchronization (passive targets)
- usage similar to shared memory locks
  1. acquire lock for target window \( \text{WIN\_LOCK}(\text{win}, \text{rank}, \ldots) \)
  2. perform operations \( \text{PUT}(\text{win}, \text{rank}, \ldots) \)
  3. release lock \( \text{WIN\_UNLOCK}(\text{win}, \text{rank}) \)

MPI defines two lock types:

- **shared** concurrent accesses on target window allowed
- **exclusive** prevent concurrent accesses on same target window
Distributed Hash Table with MPI OSC

- window object
  - local DHT part (window)
    - rank 0
  - local DHT part (window)
    - rank 1
  - local DHT part (window)
    - rank \( n - 1 \)

DHT

process' address space
Distributed Hash Table with MPI OSC

DHT_read
LOCK(window_obj, target, SHARED)
GET(window_obj, target, &data)
UNLOCK(window_obj, target)
Distributed Hash Table with MPI OSC

DHT_read
LOCK(window_obj, target, SHARED)
GET(window_obj, target, &data)
UNLOCK(window_obj, target)

DHT_write
LOCK(window_obj, target, EXCLUSIVE)
PUT(window_obj, target, data)
UNLOCK(window_obj, target)
Synchronization for the DHT

- observation: high latency for synchronization in SCC’s MPI
  - previous work (PASA 2016): 5x lower latency with shared memory and uncached accesses instead of messages
Synchronization for the DHT

- observation: high latency for synchronization in SCC’s MPI
  - previous work (PASA 2016): 5x lower latency with shared memory and uncached accesses instead of messages
  - synchronization semantics undefined by MPI: "much freedom for implementors"
Synchronization for the DHT

- observation: high latency for synchronization in SCC’s MPI
  - previous work (PASA 2016): 5x lower latency with shared memory and uncached accesses instead of messages
  - synchronization semantics undefined by MPI: "much freedom for implementors"
- assumption: (far) more DHT reads than writes
  - Readers & Writers Synchronization (Courtois et al.) advantageous
  - writer precedence → recent data for readers
Synchronization for the DHT

- observation: high latency for synchronization in SCC’s MPI
  - previous work (PASA 2016): 5x lower latency with shared memory and uncached accesses instead of messages
  - synchronization semantics undefined by MPI: "much freedom for implementors"

- assumption: (far) more DHT reads than writes
  - Readers & Writers Synchronization (Courtois et al.) advantageous
  - writer precedence → recent data for readers

→ design of MPI passive target synchronization scheme with R&W semantics for SCC
use lock data structure as proposed by Mellor-Crummey/Scott ('91)
use lock data structure as proposed by Mellor-Crummey/Scott (’91)

• distributed lists of waiting readers and writers
use lock data structure as proposed by Mellor-Crummey/Scott ('91)

- distributed lists of waiting readers and writers
  - no centralized object to spin on (avoid memory contention)
  - instead: per-process list entry for spinning
use lock data structure as proposed by Mellor-Crummey/Scott (’91)

- distributed lists of waiting readers and writers
  - no centralized object to spin on (avoid memory contention)
  - instead: per-process list entry for spinning
- state variable: counts active/interested readers/writers
Data Structures for Synchronization

use lock data structure as proposed by Mellor-Crummey/Scott (’91)

• distributed lists of waiting readers and writers
  - no centralized object to spin on (avoid memory contention)
  - instead: per-process list entry for spinning

• state variable: counts active/interested readers/writers

• one lock variable per process and window
Synchronization Operations

- according to Mellor-Crummey/Scott
- processes enter either list of readers or writers

Readers

`start_read` blocks as long as writers are active or waiting, allows multiple active readers
Synchronization Operations

- according to Mellor-Crummey/Scott
- processes enter either list of readers or writers

Readers

- **start_read** blocks as long as writers are active or waiting, allows multiple active readers
- **end_read** wake first waiting writer if no active reader left
Synchronization Operations

- according to Mellor-Crummey/Scott
- processes enter either list of readers or writers

Readers
- **start_read** blocks as long as writers are active or waiting, allows multiple active readers
- **end_read** wake first waiting writer if no active reader left

Writers
- **start_write** blocks when readers are active
Synchronization Operations

- according to Mellor-Crummey/Scott
- processes enter either list of readers or writers

**Readers**

- `start_read` blocks as long as writers are active or waiting, allows multiple active readers
- `end_read` wake first waiting writer if no active reader left

**Writers**

- `start_write` blocks when readers are active
- `end_write` wake up next writer (if any) or all waiting readers
MPI_Win_lock(type, target_rank, win_obj)
{
    entry = alloc_list_entry();

    win_obj.entry[target_rank] = entry;
    win_obj.entry[target_rank].lock_type = type;

    if (type == SHARED)
        start_read(win_obj.lock[target_rank], entry);
    else
        start_write(win_obj.lock[target_rank], entry);
}
MPI_Win_lock(type, target_rank, win_obj) {
    entry = alloc_list_entry();

    win_obj.entry[target_rank] = entry;
    win_obj.entry[target_rank].lock_type = type;

    if (type == SHARED)
        start_read(win_obj.lock[target_rank], entry);
    else
        start_write(win_obj.lock[target_rank], entry);
}

unlock operation straight forward
Data Placement

synchronization data located in shared memory
  • danger of contention on memory interface
synchronization data located in shared memory

- danger of contention on memory interface
- speedup of memory-bound application with different synchronization data locations:

```
0  4  8  12  16  20  24  28  32  36  40  44  48
speedup
distributed
controller 3
controller 2
controller 1
controller 0
```

number of MPI processes
synchronization data located in shared memory

- danger of contention on memory interface
- speedup of memory-bound application with different synchronization data locations:

- bring spinning object close to process/core → allocate list entry in closest memory controller → **local uncached spinning**
Discussion

design characteristics:

- **concurrent window access**: one lock per window and process
- **per-window Readers & Writers semantic**
- **contention avoidance**: spin on local object only
- **truly passive**: no participation of the remote process in synchronization operations and communication

Summary

- presented design for implementing MPI passive target synchronization on nCC many-core
- applied concepts from Mellor-Crummey/Scott to nCC processor
- distributed data structures critical
Summary

- presented design for implementing MPI passive target synchronization on nCC many-core
- applied concepts from Mellor-Crummey/Scott to nCC processor
- distributed data structures critical

Future Work

- implement the presented scheme
- evaluate performance by comparison against message-based implementation and other designs
Summary

- presented design for implementing MPI passive target synchronization on nCC many-core
- applied concepts from Mellor-Crummey/Scott to nCC processor
- distributed data structures critical

Future Work

- implement the presented scheme
- evaluate performance by comparison against message-based implementation and other designs

Questions!?