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

Universität Potsdam

Motivation: Distributed Hash Table (DHT)

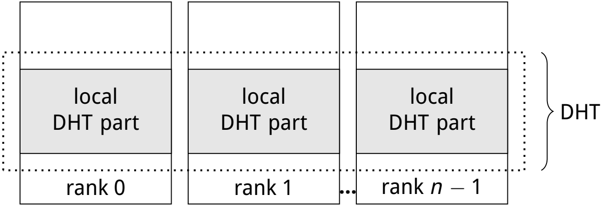
- hash table as cache for computational results in **MPI** application

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

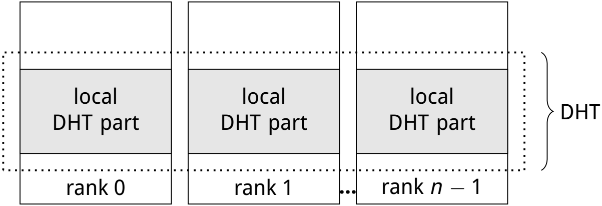
Motivation: Distributed Hash Table (DHT)

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



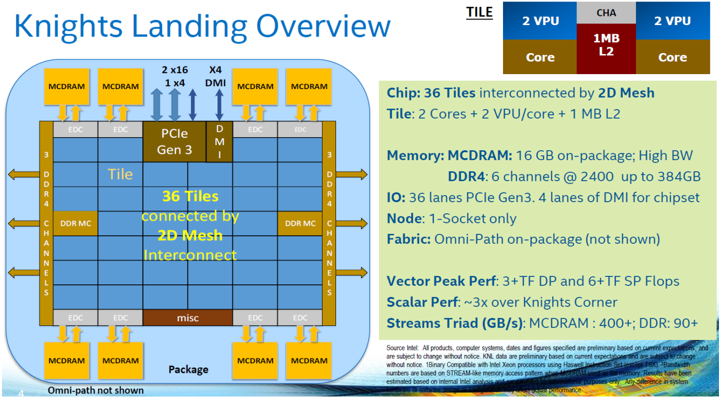
- accessing distributed data:
  - hash function returns arbitrary process and address
  - difficult to program with two-sided message passing
  - MPI passive target one-sided communication to the rescue
  - **synchronization required**

Motivation: nCC Systems

- Future many-cores may not provide (global) cache coherence.

Motivation: nCC Systems

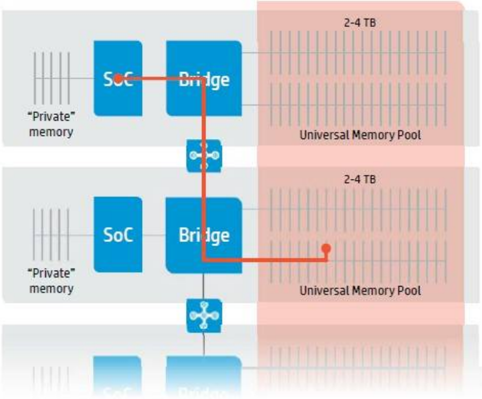
- Future many-cores may not provide (global) cache coherence.
  - Intel Knights Landing: coherent multi-socket systems not feasible



https://www.extremetech.com/wp-content/uploads/2016/04/KnightsLanding.png

Motivation: nCC Systems

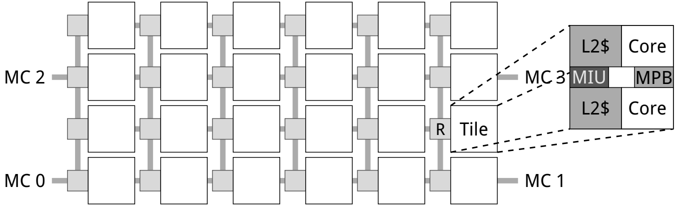
- Future many-cores may not provide (global) cache coherence.
  - Intel Knights Landing: coherent multi-socket systems not feasible
  - HPE "The Machine", EuroServer: coherence islands



https://regmedia.co.uk/2016/11/22/the_machine_universal_memory_pool_access.jpg

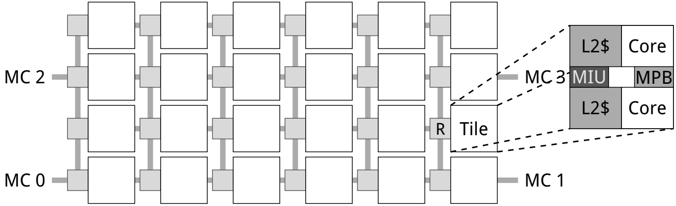
Research Platform

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



Research Platform

- 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

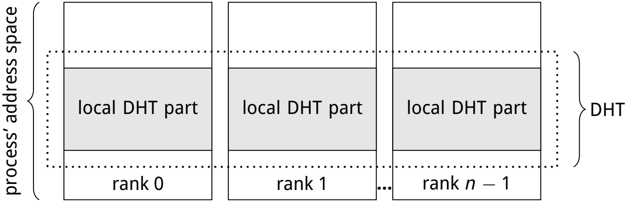
MPI Passive Target One-Sided Communication

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

Outlook and Future Work

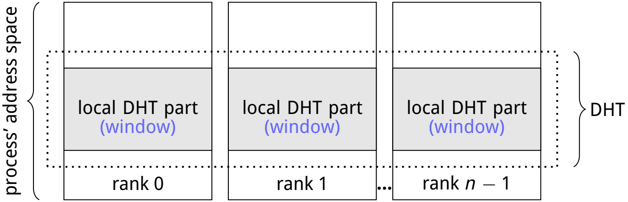
MPI One-Sided Communication

- process memory exposed via **windows**



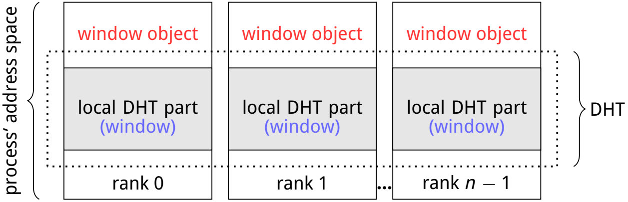
MPI One-Sided Communication

- process memory exposed via **windows**



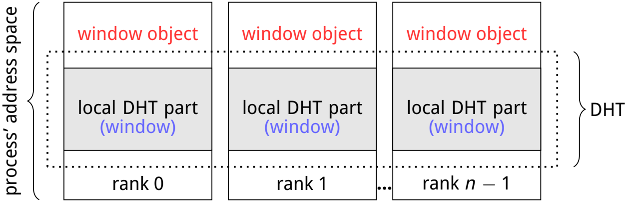
MPI One-Sided Communication

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



MPI One-Sided Communication

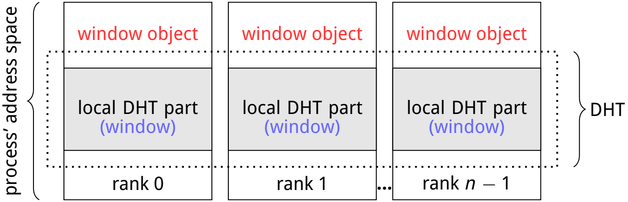
- process memory exposed via **windows**
- access to windows with **window object** (handle)



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

MPI One-Sided Communication

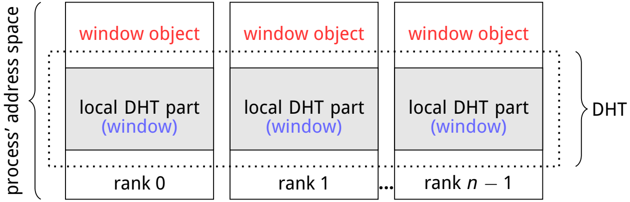
- process memory exposed via **windows**
- access to windows with **window object** (handle)



- **key concept**: only one communication partner issues communication operations
  - **origin** processes issue communication operations

MPI One-Sided Communication

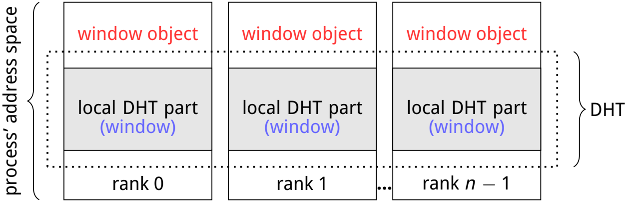
- process memory exposed via **windows**
- access to windows with **window object** (handle)



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

MPI One-Sided Communication

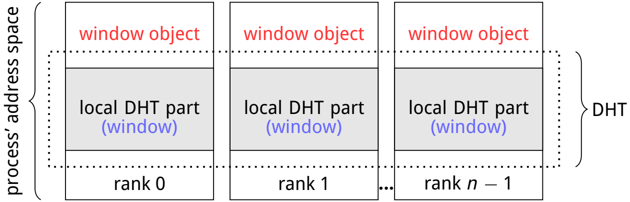
- process memory exposed via **windows**
- access to windows with **window object** (handle)



- **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)



- **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)

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 `WIN_LOCK(win, rank, ...)`
  2. perform operations `PUT(win, rank, ...)`
  3. release lock `WIN_UNLOCK(win, rank)`

MPI Passive Target Synchronization

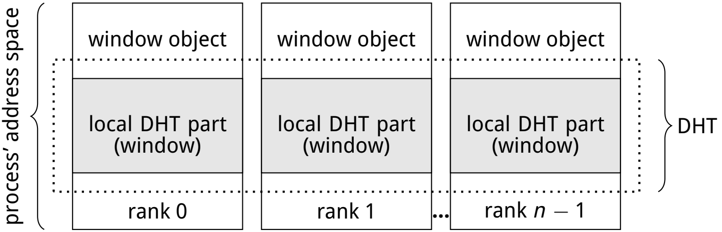
- **locks** as means for synchronization, used by origins only
- no participation of targets in synchronization (passive targets)
- usage similar to shared memory locks
  1. acquire lock for target window `WIN_LOCK(win, rank, ...)`
  2. perform operations `PUT(win, rank, ...)`
  3. release lock `WIN_UNLOCK(win, rank)`

MPI defines two lock types:

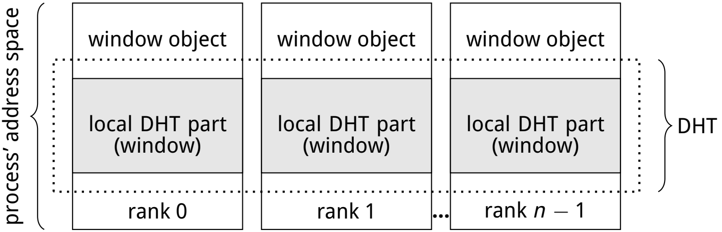
shared concurrent accesses on target window allowed

exclusive prevent concurrent accesses on same target window

Distributed Hash Table with MPI OSC

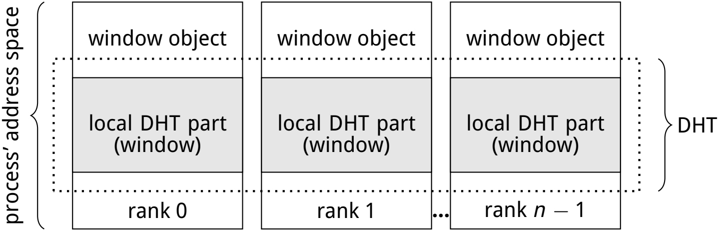


Distributed Hash Table with MPI OSC



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 $\rightarrow$ recent data for readers

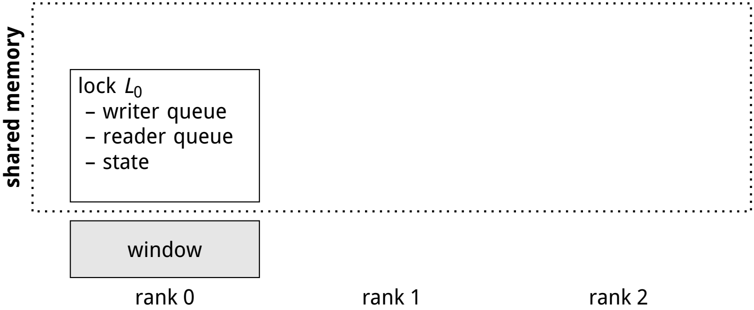
Synchronization for the DHT

- 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

Data Structures for Synchronization

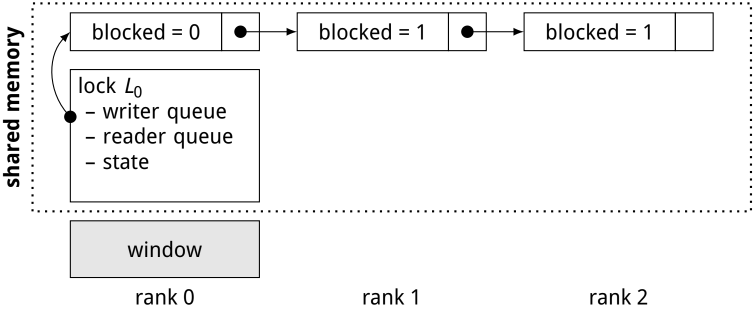
use lock data structure as proposed by Mellor-Crummey/Scott ('91)



Data Structures for Synchronization

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

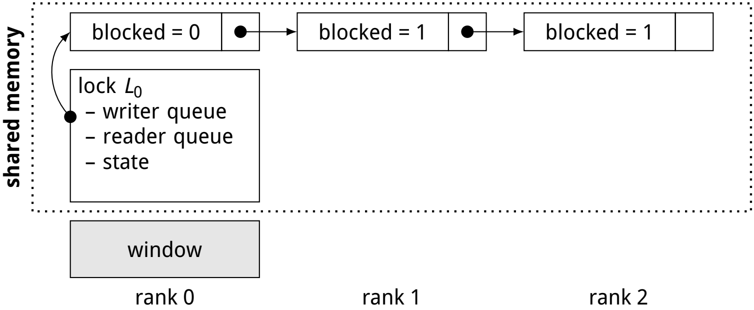
- distributed lists of waiting readers and writers



Data Structures for Synchronization

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

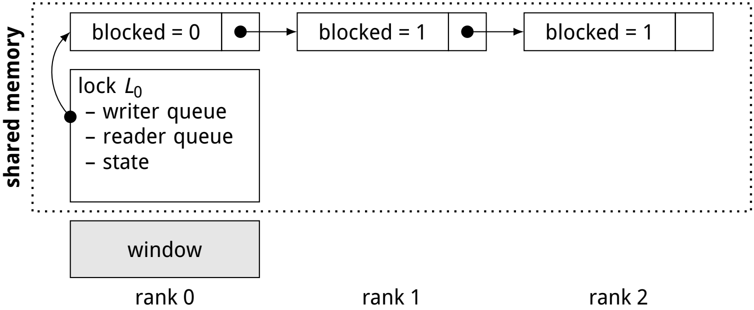
- distributed lists of waiting readers and writers
  - no centralized object to spin on (avoid memory contention)
  - instead: per-process list entry for spinning



Data Structures for Synchronization

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

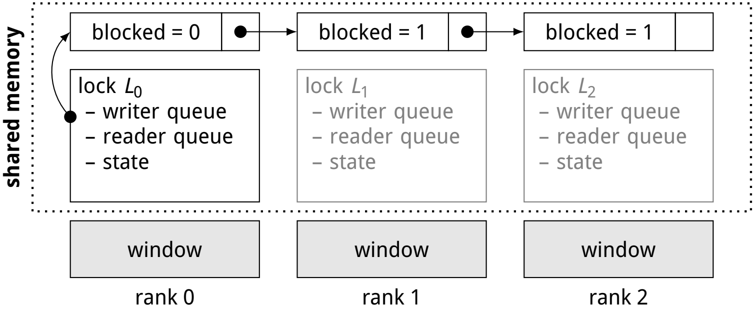
- distributed lists of waiting readers and writers
  - no centralized object to spin on (avoid memory contention)
  - instead: per-process list entry for spinning
- state variable: counts active/interested readers/writers



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

R&W Synchronization inside MPI Library

```
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);
}
```

R&W Synchronization inside MPI Library

```
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

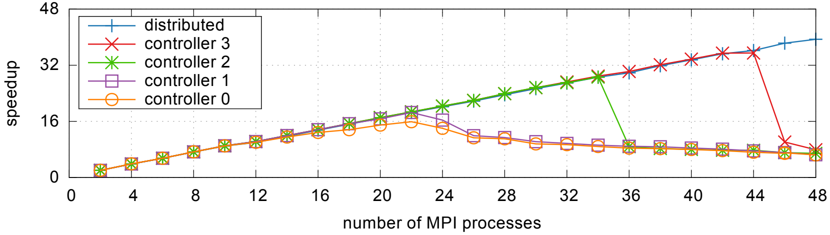
synchronization data located in shared memory

- danger of contention on memory interface

Data Placement

synchronization data located in shared memory

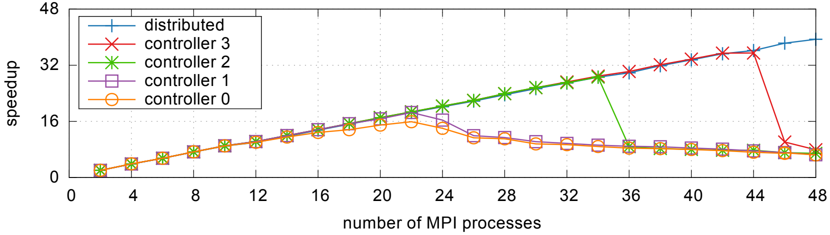
- danger of contention on memory interface
- speedup of memory-bound application with different synchronization data locations:



Data Placement

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
  Christgau, Schnor: Exploring One-Sided Communication and Synchronization on a non-Cache-Coherent Many-Core Architecture. Concurrency and Computation: Practice and Experience. 2017

Summary and Outlook

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 and Outlook

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 and Outlook

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!?