CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

MPI RMA - One sided communication

Summer School 2018 – High Performance and Parallel GPU Computing

Maxime Martinasso, CSCS

July 24, 2018

Course Objectives

- Remote Memory Access or one sided MPI communication

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

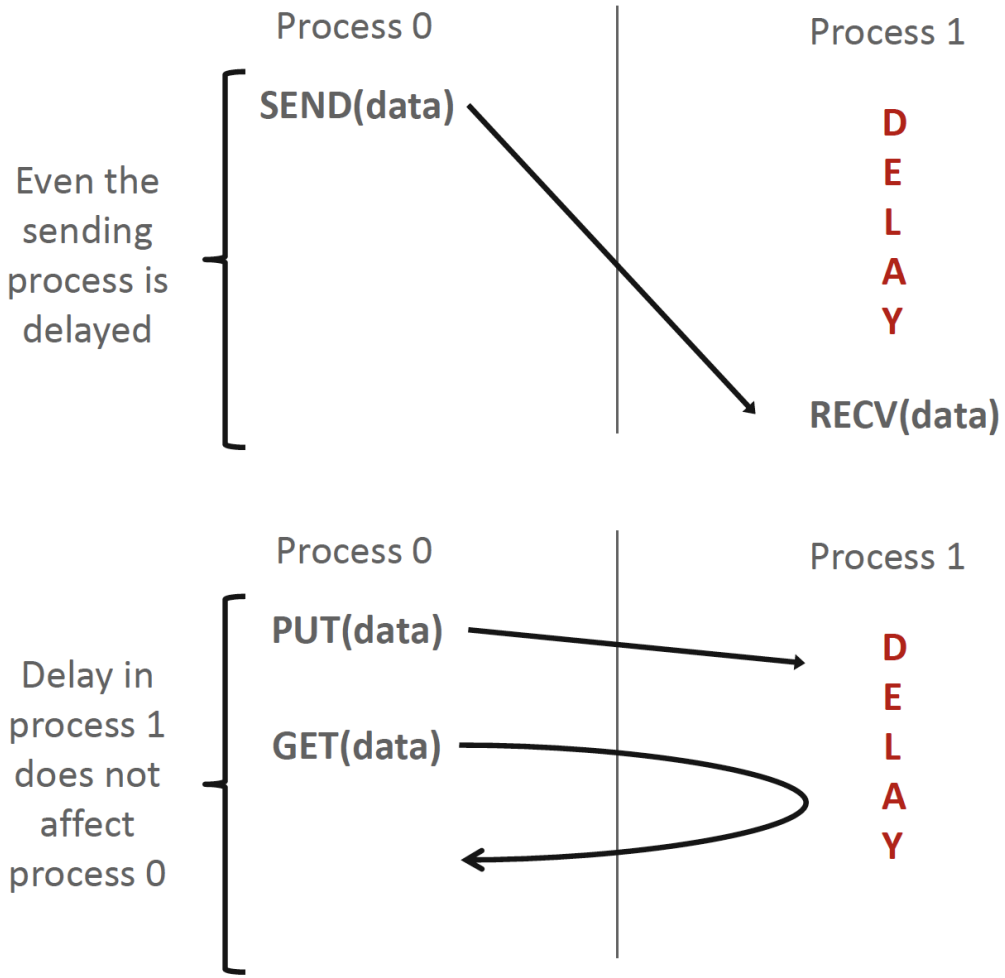
ETH zürich

One sided communication

Main concepts

- A memory region is exposed by all ranks (window)
- Each rank can read or write in a region
- Transfers and synchronisation are decoupled

2-sided vs 1-sided



Terminology

- Origin process: Process with the source buffer, initiates the operation
- Target process: Process with the destination buffer
- Epoch: Virtual time where operations are not completed. Data is consistent after new epoch is started.
- Ordering: order of message between two processes (can be relaxed compared to a strict order)

Memory region

Make a memory region remotely accessible: a windows.

Creating a window is a collective operation.

Four models exist

- `MPI_WIN_ALLOCATE`: You want to create a buffer and directly make it remotely accessible
- `MPI_WIN_CREATE`: You already have an allocated buffer that you would like to make remotely accessible
- `MPI_WIN_CREATE_DYNAMIC`: You dont have a buffer yet, but will have one in the future. You may want to dynamically add/remove buffers to/from the window
- `MPI_WIN_ALLOCATE_SHARED`: You want multiple processes on the same node share a buffer

Expose memory

Pseudo-code

```
MPI_Win_allocate(MPI_Aint size, disp_unit, MPI_Info info,
                 MPI_Comm comm, void *dest_prt, MPI_Win win)
MPI_Win_create(void *ptr, MPI_Aint size, disp_unit,
               MPI_Info info, MPI_Comm comm, MPI_Win win)
MPI_Win_free(MPI_Win win)
```

Exposes (and allocate) the memory buffer to other ranks.

It is a collective operations among all ranks.

`MPI_Info` contains user configuration.

`MPI_Aint` type that holds any valid address.

Dynamic memory

It is possible to create a window with no memory attached.

```
Pseudo-code
MPI_Win_create_dynamic(MPI_Info info, MPI_Comm comm,
                       MPI_Win win)
```

It is possible to attach/detach memory to/from a window.

```
Pseudo-code
MPI_Win_attach(MPI_Win win, void *ptr, MPI_Aint size)
MPI_Win_detach(MPI_Win win, void *ptr)
```

Data movement

MPI provides ability to read, write and atomically modify data in remotely accessible memory regions

- `MPI_PUT`
- `MPI_GET`
- `MPI_ACCUMULATE` (atomic)
- `MPI_GET_ACCUMULATE` (atomic)
- `MPI_COMPARE_AND_SWAP` (atomic)
- `MPI_FETCH_AND_OP` (atomic)

Communication

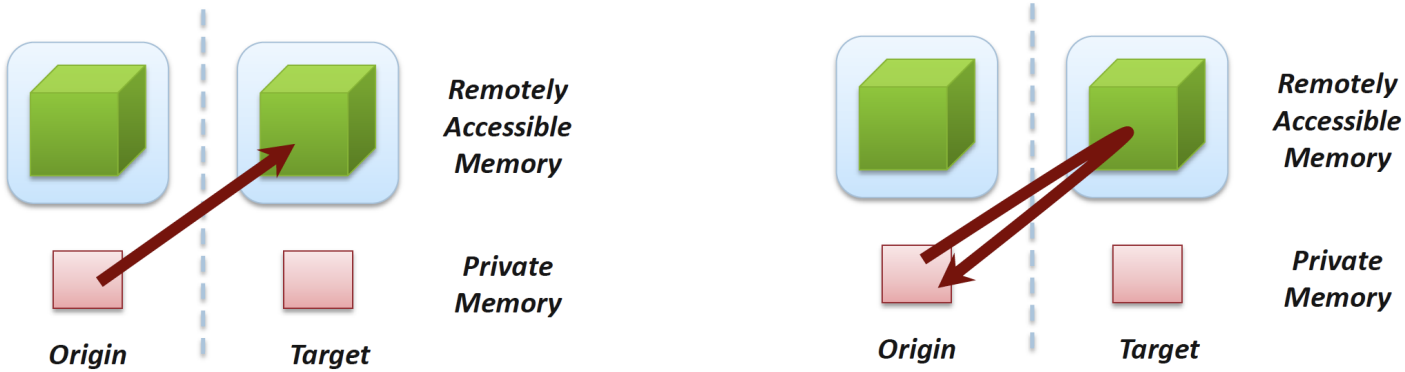
Two functions: Write data into the window (put)

```
Pseudo-code
MPI_Put(origin_addr, origin_count, origin_datatype,
        target_rank, target_disp, target_count,
            target_datatype, win)
```

Read data from the window (get)

```
Pseudo-code
MPI_Get(origin_addr, origin_count, origin_datatype,
        target_rank, target_disp, target_count,
        target_datatype, win)
```

- Non blocking communication
- Concurrent accesses give undefined behaviour (not an error).



Accumulate

Pseudo-code

```
MPI_Accumulate(origin_addr, origin_count, origin_datatype,
               target_rank, target_disp, target_count,
               target_datatype, op, win)
```

Remote accumulations, replace value in target buffer with accumulated.

- Only predefined operations

Other operations: `MPI_Fetch_and_op`, `MPI_Compare_and_swap`

Ordering of Operations

- No guaranteed ordering for Put/Get operations
- Result of concurrent Puts to the same location is undefined
- Result of Get concurrent Put/Accumulate undefined
- Result of concurrent accumulate operations to the same location are defined according to the order in which the occurred

Synchronisation

Three synchronization models provided by MPI:

- Fence (active target)
- Post-start-complete-wait (generalized active target)
- Lock/Unlock (passive target)

Data accesses occur within epochs

- Epochs define ordering and completion semantics
- Synchronization models provide mechanisms for establishing epochs

Active Target Synchronization

Collective synchronization model

```
Pseudo-code
MPI_Win_fence(int assert, MPI_Win win)
```

- Starts and ends access and exposure epochs on all processes in the window
- All processes in group of win do an `MPI_WIN_FENCE` to open an epoch
- Everyone can issue PUT/GET operations to read/write data
- Everyone does an `MPI_WIN_FENCE` to close the epoch
- All operations complete at the second fence synchronization

PSCW: Generalized Active Target Synchronization

Pseudo-code

```
MPI_Win_post/start(MPI_Group grp, int assert, MPI_Win win)
MPI_Win_complete/wait(MPI_Win win)
```

Like FENCE, but origin and target specify who they communicate with

Target: Exposure epoch

- Opened with `MPI_Win_post`
- Closed by `MPI_Win_wait`

Origin: Access epoch

- Opened by `MPI_Win_start`
- Closed by `MPI_Win_complete`

All synchronization operations may block, to enforce P-S/C-W ordering

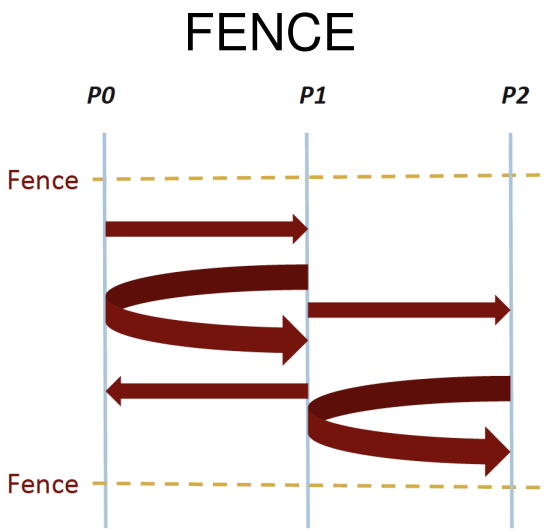
- Processes can be both origins and targets

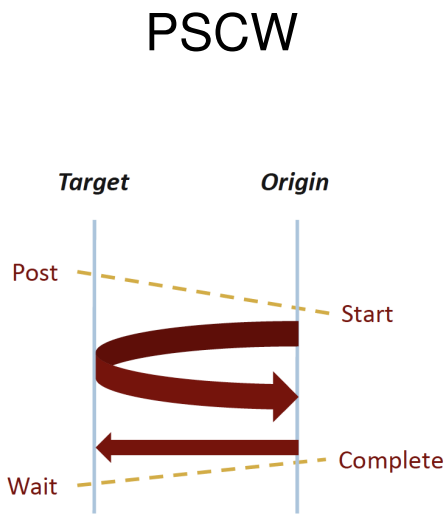
Passive Target Synchronization

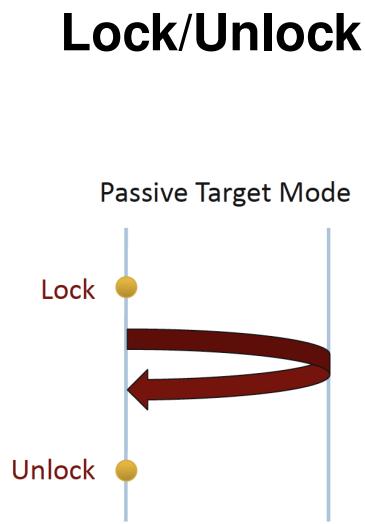
```
Pseudo-code
MPI_Win_lock(int locktype, int rank, int assert,
             MPI_Win win)
MPI_Win_unlock(int rank, MPI_Win win)
MPI_Win_flush(int rank, MPI_Win win)
```

- Lock/Unlock: Begin/end passive mode epoch
  - Target process does not make a corresponding MPI call
  - Can initiate multiple passive target epochs to different processes
  - Concurrent epochs to same process not allowed
- Lock type
  - SHARED: Other processes using shared can access concurrently
  - EXCLUSIVE: No other processes can access concurrently
- Flush: Remotely complete RMA operations to the target process
  - After completion, data can be read by target process or a different process

Synchronization summary





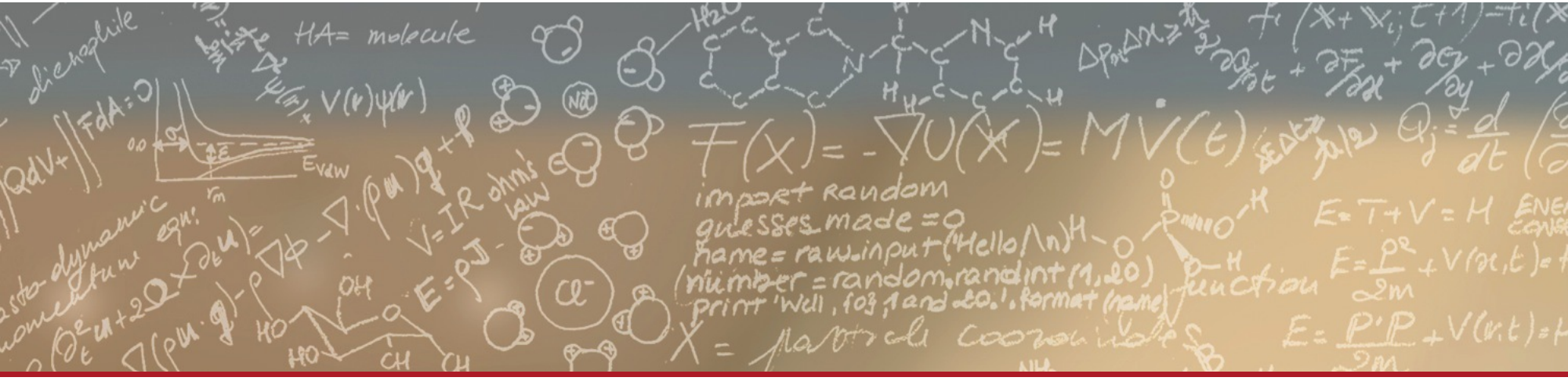


Practicals

Exercise: 07.MPI_RMA

1. `MPI_Get` + `MPI_Win_allocate` + `MPI_Win_fence`
2. `MPI_Put` + `MPI_Win_create` + `MPI_Win_lock/unlock`

Use a large number of ranks.



Thank you for your attention.