Remote Memory Access

A deeper look at RMA

Reusing this material

Outline

- Additional MPI RMA concepts
  - Synchronization modes
  - Types of epoch
  - Memory model
- The other two MPI synchronization calls
  - Post-Start-Complete-Wait (PSCW)
  - Locking and unlocking
- Some useful MPI-3 extensions
  - Request handles
  - Dynamic windows

Additional MPI RMA concepts

Synchronization modes, epochs types and memory model

Synchronization modes

- Active target
  - Both processes are explicitly involved in the data movement. Only one process issues the data transfer call but all processes issue the synchronisation.
- Passive target
  - Only the origin process is involved in the data movement, there are no calls made on the target process. For instance two origin processes might communicate by accessing the same location in a target window, and the target process (which does not participate) might be distinct from the origin processes.

Fence is an example of active target as each process issues the fence calls

Epoch types

- Access epoch
  - RMA communication calls (get, put etc) can only be issued inside an access epoch. This is created by starting the epoch and completed by stopping the epoch.
  - I.e. it is used to access the remote memory of another process.

- Exposure epoch
  - Used in active target communication, this is required to expose memory on the target so it can be accessed by other processes' RMA operations.

Fences abstract the programmer from this as they will complete/start both access and exposure epochs automatically as required

RMA Memory model

- Public and private window copies
  - Public memory region is addressable by other processes (i.e. exposed main memory)
  - Private memory (i.e. transparent caches or communication buffers) which is only locally visible but elements from public memory might be stored.

- Coherent if updates to main memory are automatically reflected in private copy consistently
- Non-coherent if updates need to be explicitly synchronised

RMA Memory model

- MPI therefore has two models
  - Unified if public and private copies are identical – used if possible, realistic on cache coherent machines. *(This was added in MPI v3)*
  - Separate if they are not, here there is only one copy of a variable in process memory but also a distinct public copy for each window that contains it. The old model
- In the separate model a suitable synchronisation call (i.e. end of an epoch) must be issued to make these consistent. In the unified model some synchronisation calls might be omitted for performance reasons
- The window attribute tells you which model it follows

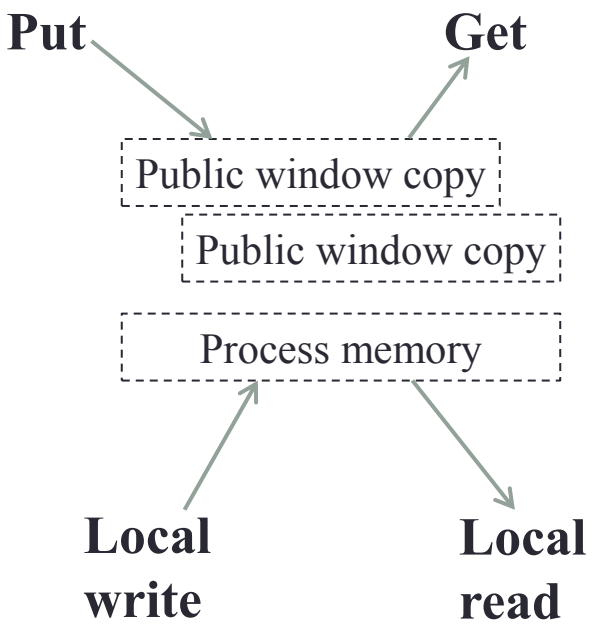


Illustration of separate model

The other two synchronization calls

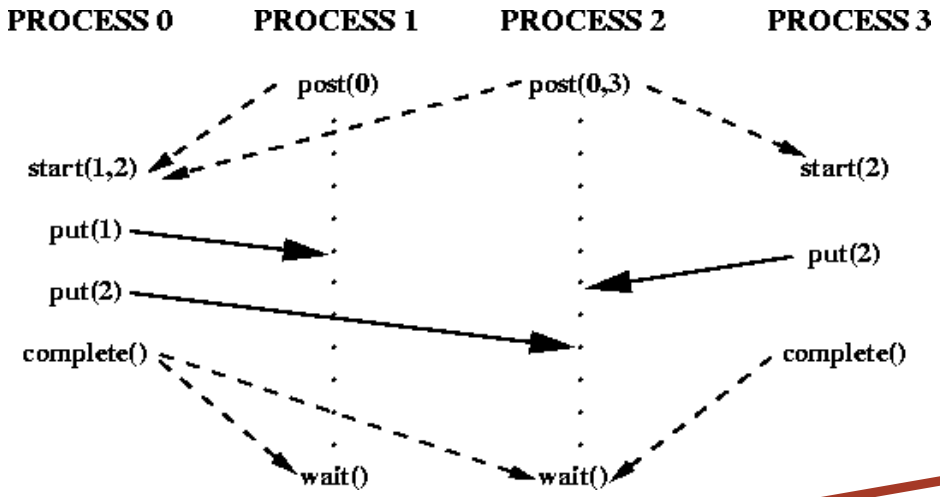
Post-start-complete-wait and locking & unlocking

Post-start-complete-wait (PSCW)

- The programmer explicitly handles different types of epoch
  - `post` creates an exposure epoch, `wait` ends an exposure epoch
  - `start` creates an access epoch, `complete` ends an access epoch
- Groups of processes are provided to `post` and `wait` calls to support synchronising over a subset of processes in the communicator. Assertions also provided if you want.

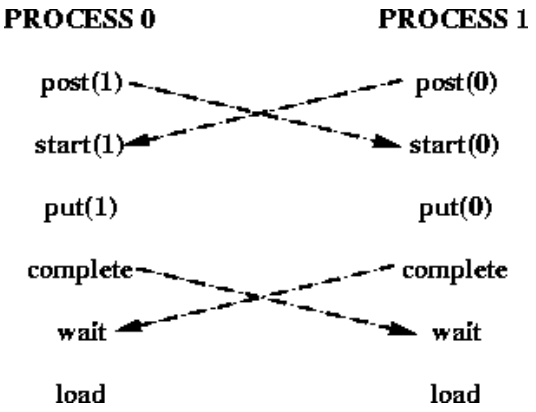
- `post` will not block, `start` may or may not block
- `wait` will block until all matching complete calls and guarantees target RMA completion
- `complete` will block until RMA communications of that epoch have completed and guarantees origin RMA completion

Post-start-complete-wait (PSCW)



Dashed arrows illustrate synchronization, solid arrows data transfer

These can overlap, i.e. you can have an exposure epoch and also create an access epoch. Remember wait will block for its matching complete, so you must have complete and then wait



PSCW example

Based on an example at
cvw.cac.cornell.edu/MPIoneSided/pscw

```
int ranks[]={0,1,2};

if (rank == 0) {
    MPI_Win_create(buf, sizeof(int)*3, sizeof(int), MPI_INFO_NULL,
        MPI_COMM_WORLD, &win);
} else {
    MPI_Win_create(NULL, 0, sizeof(int), MPI_INFO_NULL, MPI_COMM_WORLD,
        &win);
}
if (rank == 0) {
    MPI_Group_incl(comm_group, 2, ranks+1, &group);
    MPI_Win_post(group, 0, win);
    MPI_Win_wait(win);
} else {
    MPI_Group_incl(comm_group, 1, ranks, &group);
    MPI_Win_start(group, 0, win);
    MPI_Put(buf, 1, MPI_INT, 0, rank, 1, MPI_INT, win);
    MPI_Win_complete(win);
}
```

Group contains ranks 1 and 2

Start exposure epoch

Stop exposure epoch

Group contains rank 0

Start access epoch

Stop access epoch

Locks and unlocks

- PSCW is an example of active target synchronisation as the target must still explicitly create an exposure epoch
- Locks/unlocks are an example of passive synchronisation where only the origin takes part.

```
int MPI_Win_lock(int lock_type, int rank, int assert,
      MPI_Win win)

int MPI_Win_unlock(int rank, MPI_Win win)
```

Starts an access epoch

Stops the access epoch

- Inside the epoch (i.e. between lock & unlock) then RMA communication calls as normal, these complete ***for both the origin and target*** on the corresponding unlock.

Locks and unlocks

- The lock type argument to lock is either:
  - **MPI_LOCK_SHARED** where multiple processes may access the target window at any one time
  - **MPI_LOCK_EXCLUSIVE** where only one process may access the target window at any one time

- MPI 3 also added `lock_all` and `unlock_all` variants which control access to all processes associated with a window.
- There is also
  - `flush` to flush outstanding RMA operations on the window to the target rank
  - `sync` to synchronise public & private window copies (separate memory model)

Lock and unlock example

Based on an example at cvw.cac.cornell.edu/MPIoneSided/lul

```
MPI_Win win;

if (rank == 0) {
    MPI_Win_create(NULL,0,1,MPI_INFO_NULL,MPI_COMM_WORLD,&win);
    MPI_Win_lock(MPI_LOCK_SHARED,1,0,win);
    MPI_Put(buf,1,MPI_INT,1,0,1,MPI_INT,win);
    MPI_Win_unlock(1,win);
    MPI_Win_free(&win);
} else {
    MPI_Win_create(buf,2*sizeof(int),1, MPI_INFO_NULL,
        MPI_COMM_WORLD, &win);
    MPI_Win_free(&win);
}
```

Start access epoch

Communication call

Stop access epoch

Some other MPI v3 extensions

Request handles, dynamic windows

Request handles

- There is also an `R` variant of all communication calls which associate a request handle with the operation
  - i.e. `Rget`, `Rput` and `Raccumulate`
  - Importantly only valid in passive target synchronisation
- This request handle is then used as you would any other request handle (you can call `MPI_test`, `MPI_wait` etc on it.)
- Tracks origin's RMA completion only.
  - For a `put`/`accumulate` guarantees that the origin's buffer can be overwritten (but not that data has arrived)
  - For a `get` guarantees that data is available in the origin's buffer
  - Different to an `unlock` or a `flush` in this respect

Dynamic windows

- Traditional windows allocation required that memory is attached at window creation
  - But what if we don't know the amount of memory needed at that point?
- Dynamic windows allow memory can be exposed without additional synchronisation

```
int MPI_Win_create_dynamic(MPI_Info info, MPI_Comm comm,
     MPI_Win *win)
```

- Memory is then attached with

```
int MPI_Win_attach(MPI_Win win, void *base, MPI_Aint size)
```

- Detached with

```
int MPI_Win_detach(MPI_Win win, const void *base)
```

Dynamic windows

- Memory being attached can not overlap with any other memory that is already attached

- The target displacement argument of the origin's communication call is the address at the target
  - You can use `MPI_Get_Address` on the target to retrieve this

- Must ensure that the memory has been attached on the target process before the origin issues any RMA calls referencing it

Summary

- We have covered a large proportion of the RMA calls
  - All the underlying concepts
  - All three synchronisation mechanisms
  - There are some additional, less frequently used communication calls and window management functions.
  - There are a number of rules related to correctness where you can omit explicit synchronisation calls for unified windows. We have covered a “safe” approach here which works fine for both types.
- Many resources available online
  - MPI version 3 standard is very comprehensive
  - https://cvw.cac.cornell.edu/MPIoneSided is a good resource
  - https://htor.inf.ethz.ch/publications/img/mpi3-rma-overview-and-model.pdf
  - http://www.mpich.org/static/docs/v3.2/www3/
- You will get best experience of this by considering the applicability of RMA to your existing codes