MPI Single-sided

Advanced Parallel Programming

Dan Holmes
EPCC, University of Edinburgh

Overview

- Terms and concepts
- Windows
- Memory models
- Data-movement operations
- Synchronisation operations
- Atomic operations
- Local completion

Terms and concepts

- Called “one-sided” or “single-sided” because:
  - All communication parameters specified by a single process
- Called “RMA” or “remote memory access” because:
  - Data movement is separated from process synchronisation

- Initialise by collectively creating a “window”
  - A chunk of local memory that will be accessed by remote processes
- The “origin” process calls MPI during an “access epoch”
- Memory is accessed at the “target” process
- Synchronisation is “passive” target or “active” target
- Active target requires an “exposure epoch”

MPI Windows

- All window creation functions are collective over the specified communicator, which must be an intra-communicator
- MPI_Win_create
  - Each process specifies a chunk of pre-allocated local memory
- MPI_Win_allocate
  - MPI will allocate local memory for the window
- MPI_Win_allocate_shared
  - MPI will allocate local memory for the window and guarantee that all processes can access it as shared-memory
- MPI_Win_dynamic
  - Each process can attach and detach pre-allocated local memory

MPI_WIN_CREATE

int **MPI_Win_create**(void ***base**, MPI_Aint **size**, int **disp_unit**, MPI_Info **info**, MPI_Comm **comm**, MPI_Win ***win**)

- “base”: in C, this is a pointer to the beginning of the memory
  - In Fortran pass the (first element of) a simply-contiguous array
- “size”: is in bytes – it is not scaled by disp_unit
- “disp_unit”: is in bytes and can be 1, for no scaling
  - Supply sizeof(some_type) to use array indices in other RMA calls
- “info”: provide hints about window usage
  - For example, no_locks, same_size, same_disp_unit
- “comm”: defines which processes participate
- “win”: the output opaque handle for the created window

MPI_WIN_ALLOCATE[_SHARED]

int MPI_Win_allocate(MPI_Aint **size**, int **disp_unit**, MPI_Info **info**, MPI_Comm **comm**, void *__baseptr__, MPI_Win *__win__)

- "baseptr": output pointer to beginning of allocated memory
- "comm": for the shared variant, comm must only contain processes that can participate in a shared memory segment
- Restrictions and advice concerning MPI_ALLOC_MEM also apply to these allocating window creation functions
- Additional info key: alloc_shared_noncontig
- Memory consistency only specified for unified memory model
- MPI_Win_shared_query gets "baseptr" for other processes

MPI_WIN_CREATE_DYNAMIC

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

- “info”, “comm” and “win” have same meaning as before
- No memory is attached during creation of the window
- Memory must be attached at target before any origin uses it

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

- “win”, “base” and “size” are supplied when attaching memory
- “disp_unit” is always implicitly set to 1 for dynamic windows
- “window_base” is implicitly set to MPI_BOTTOM not “base”

Miscellaneous window operations

- MPI_Win_get_attr
  - Get values of the attributes associated with the window
  - Characteristics that were set during creation of the window
- MPI_Win_get_info and MPI_Win_set_info
  - Get and set the info keys associated with the window
- MPI_Win_get_group
  - Get the group of the communicator used to create the window
- MPI_Win_free
  - Tidies up by destroying the window and freeing MPI resources
- MPI_Win_create_errhandler and MPI_Win_set_errhandler
  - Change the default error handler for the window

Memory models

- Old “separate” memory model
  - Old model still supported, now called the “separate” memory model
- New “unified” memory model
  - Simplifies memory consistency rules on cache-coherent machines
- Discovered via the MPI_Win_model attribute on the window
- Semantics are described using “public” and “private” copies
  - Each variable has a “private” copy in process-local memory
  - Each window has a “public” copy of all variables within the window
  - In the separate model, all these copies are logically separate and may or may not be affected by updates to other copies
  - In the unified model, all these copies are logically unified and will definitely be affected by updates to other copies, eventually
- Synchronisation is always needed to guarantee consistency

Single-sided data movement operations

- All RMA communication calls are non-blocking
  - Buffers cannot be accessed until the operation has completed
- MPI_Put
  - Moves data **from** a specified buffer at the origin process **to** a specified location in a window at the target process
- MPI_Get
  - Moves data **to** a specified buffer at the origin process **from** a specified location in a window at the target process
- MPI_Accumulate
  - Like MPI_Put (data moves from origin to target) but combines the transmitted value with the value already at the target
  - Any built-in (not user-defined) MPI reduction operation can be used

RMA synchronisation

- Two methods of synchronisation: active and passive target
  - Defines whether the target is involved in the synchronisation or not
- Four different sets of synchronisation operations
  - Two are active target methods; two are passive target methods
- A process can be an origin only during an access epoch
  - “origin” means caller of MPI single-sided communication operation(s)
- Access epoch is opened by a synchronisation call at origin
  - And closed by a subsequent synchronisation call
- For active target methods, a process can be a target only during an exposure epoch
  - Exposure epoch opened and closed by synchronisation calls at target

RMA synchronisation operations

- Active target – fence
  - How to use: collective call to MPI_Win_fence
  - Closes previous, and opens new, access and exposure epochs
- General active target – PSCW or Post-Start-Complete-Wait
  - How to use at origin 1: MPI_Win_start opens access epoch
  - How to use at origin 2: MPI_Win_complete opens access epoch
  - How to use at target 1: MPI_Win_post opens exposure epoch
  - How to use at target 2: MPI_Win_wait opens exposure epoch
- Passive target – shared/exclusives 'locks'
  - How to use for shared locks: MPI_Lock, MPI_Unlock
  - How to use for exclusive locks: MPI_Lock_all, MPI_Unlock_all
  - This is badly named! It is not really locks, more like transactions
  - No concept of exposure epoch (can use MPI_Win_flush and _sync)

Atomic operations

- New atomic read-modify-write operations in MPI-3
- MPI_Get_accumulate
  - Fetches previous value of target before accumulation with origin data
- MPI_Fetch_and_op
  - Restricted version of MPI_Get_accumulate for hardware operations
- MPI_Compare_and_swap
  - Origin sends a compare value and new value
  - If the target value and the compare value are equal then
    - The target value is replaced with the new value
  - Old target value is always returned to origin process

Local completion

- New local completion semantics for one-sided operations
  - Only valid during a passive target epoch
- MPI_Rput, MPI_Rget, MPI_Raccumulate and MPI_Rgetaccumulate return a MPI_Request object handle
- Use MPI_Test or MPI_Wait to check for local completion
- Cannot call MPI_Request_free, MPI_Cancel or MPI_Start
- Only the MPI_Error field in the status object is set correctly
  - All other field return undefined values
- Must complete the request (by testing or wating)
  - Even if the operation is known to be complete, e.g. by MPI_Win_flush

Summary

- Data movement and synchronisation are separate calls
- All parameters for data movement specified by ‘one side’
- Basic operations are put/get direct into/from remote memory
- Complexity comes from consistency and synchronisation
- Two memory consistency models: separate and unified
- Two types of synchronisation: active and passive
- Atomic operations
- Local completion

- Only worth the implementation effort for some applications