MPI-3 RMA Draft

RMA Goals

- Extend the applicability of MPI RMA to additional (but not necessarily *all*) applications
- Address documented shortcomings in using MPI RMA
- Maintain look and feel of MPI; maintain relevance to future platforms

2

New Features at a Glance

- Additional ways to create MPI_Win
  - Allocate memory for window (more scalable)
  - Dynamically attach/detach memory to an existing window
- Additional remote operations
  - Including read-modify-write, absent in MPI-2
- Extensions to passive target RMA
  - Win_lock_all locks all processes in MPI_Win
  - Local and remote completion
- RMA operations with requests

3

New Features Continued

- Many operations previously erroneous now undefined
  - Enables different programming models
- Ordering:
  - Any combination of read/write ordering
  - rr, rw, wr, ww as info key value string
- Extended info keys
  - Relax ordering (see above)
  - Info query mechanism suggested (for misc WG, will be discussed later)

4

New Routines at a Glance

- MPI_Win_allocate
- MPI_Win_create_dynamic
  - MPI_Win_register
  - MPI_Win_deregister
- MPI_Get_accumulate
  - Generic, similar to MPI_Accumulate
- MPI_Fetch_and_op
  - Single datatype, single item specialized (fast)
- MPI_Compare_and_swap

5

New Routines Continued

- MPI_Win_lock_all
  - MPI_Win_unlock_all
- MPI_Win_flush
  - MPI_Win_flush_all
- MPI_Win_flush_local
  - MPI_Win_flush_local_all
- MPI_Win_sync
- MPI_Win_query

6

New Routines Continued

- MPI_Rget, MPI_Rget_acumulate, MPI_Rput, MPI_Raccumulate

7

Win Allocate

- Collective Window Allocation
  - Enables symmetric allocation for simple offset calculation
- MPI_WIN_ALLOCATE(size, disp_unit, info, comm, base, win)
  - int MPI_Win_allocate(MPI_Aint size, int disp_unit, MPI_Info info, MPI_Comm comm, void *base, MPI_Win *win)

8

Win Create Dynamic

- Creates a window that allows local registration
  - Memory needs to be registered before being accessed remotely
- int MPI_Win_create_dynamic(MPI_Info info, MPI_Comm comm, MPI_Win *win)
  - int MPI_Win_register(MPI_Win win, void *base, MPI_Aint size)
  - int MPI_Win_deregister(MPI_Win win, void *base)
  - Overlapping registrations are erroneous!

9

Get Accumulate

- Similar to fetch&add
- int MPI_Get_accumulate(void *origin_addr, int *origin_count, MPI_Datatype origin_datatype, void *result_addr, int *result_count, MPI_Datatype result_datatype, int target_rank, MPI_Aint target_disp, int *target_count, MPI_Datatype target_datatype, MPI_Op op, MPI_Win win)
  - Supports MPI_NO_OP

10

Fetch and Op

- Specialized Get Accumulate for performance

- MPI_Fetch_and_op(void *origin_addr, void *result_addr, MPI_Datatype datatype, int target_rank, MPI_Aint target_disp, MPI_Op op, MPI_Win win)
  - ♦ No count (count = 1)
  - ♦ Only one datatype

11

Compare and Swap

- int MPI_Compare_and_swap(void *origin_addr, void *compare_addr, void *result_addr, MPI_Datatype datatype, int target_rank, MPI_Aint target_disp, MPI_Win win)
  - ♦ Datatypes are limited to some predefined datatypes
  - ♦ C integer, Fortran integer, Logical, Complex, Byte, MPI_AINT, MPI_OFFSET

12

Win Lock All

- Locks all target processes in a window
  - ♦ Obviously a shared lock ☺
  - ♦ Optimization for a simple loop
- int MPI_Win_lock_all(int assert, MPI_Win win)
  - ♦ int MPI_Win_unlock_all(MPI_Win win)
- Did not include MPI_Win_lock_group
  - ♦ No use-case, can implement as loop over Win_lock

13

Win Flush

- Blocks until all operations completed remotely
- int MPI_Win_flush(int rank, MPI_Win win)
  - ♦ Flush a specific rank
- int MPI_Win_flush_all(MPI_Win win)
  - ♦ Flush all ranks
- Applies only to passive target RMA

14

Win Flush Local

- Blocks until all operations completed locally
  - Local buffers can be re-used
  - Might imply remote completion (e.g., Get)

- int MPI_Win_flush_local(int rank, MPI_Win win)
  - Operations targeted to a specific rank
- int MPI_Win_flush_local_all(MPI_Win win)
  - All previous operations on window win

15

RMA Query

- Query remote memory consistency
- int MPI_RMA_query(MPI_Win win, int *model)
  - Returns either MPI_RMA_SEPARATE (MPI-2) or MPI_RMA_UNIFIED (public window = private window)
  - MPI_RMA_UNIFIED changes semantic rules 5+6

16

RMA With Request

- Allows process to wait for local completion
  - ♦ Permits reuse of buffers on Rput, Raccumulate
  - ♦ Permits use of data on Rget, Rget_accumulate
  - ♦ Uses an MPI_Request (can use MPI_Wait/Test)

20

Bonachea's and Duell's criticism

- Window creation is collective
  - ♦ hinders efficient exposure for local objects
  - ♦ no "sparse" communication
- MPI_Win_create_dynamic
  - ♦ Well suited for automatic memory management
  - ♦ Good as compilation target

21

21

Bonachea's and Duell's criticism

- Exposed memory must be MPI_Alloc_mem()'d
  - no exposure of static memory or stack-variables
  - alloc_mem might be limited by the implementation
- Also addressed in MPI_Win_create_dynamic
  - Can register stack
  - Dangerous though! Undefined results if stack is out of scope.

22

22

Bonachea's and Duell's criticism

- Forbids conflicting get/put (or local load/store) accesses to same memory
  - really hard to track for compilers (halting problem?)
  - Easy source of bugs in user codes
- Outcome is undefined (does not change the behavior of *correct* MPI-2 programs)

23

23

Bonachea's and Duell's criticism

- Window's memory may not be updated by remote gets and local stores concurrently
  - simplifies MPI implementation significantly
  - seems very artificial and suboptimal from user's perspective
- Outcome is undefined
  - MPI_Win_sync provides way to make outcome defined and as user expects
  - MPI_Win_query with result MPI_RMA_UNIFIED also makes outcome defined and as user expects

24

Bonachea's and Duell's criticism

- Overlapping memory regions of multiple windows can be created but not be used
  - "concurrent communications may lead to erroneous results"
- Now also "undefined"
  - [is this much better than erroneous?]
  - But these applications are likely to use a single dynamic window, so problem avoided

Bonachea's and Duell's criticism

- Passive target RMA ops only lock a single process during an epoch
  - ops from one source to different targets are serialized
  - one window for each target to enable concurrent access?
    - scalability limitation
- MPI_Win_lockall, may use multiple MPI_Win_lock

26

Comparison to ARMCI

- Similar to MPI_RMA_UNIFIED (if available)
  - ARMCI does not support non-CC systems without an additional activity (thread/process/callback)
- Local completion is different
  - Either blocking, implicit handles or Test/Wait(all)
  - Similar to "implicit handles"
- Remote completion similar
  - (all)Fence == Flush(_all)
- Ordering with collectives
  - ARMCI_Barrier combines barrier + allfence

27

Comparison to ARMCI cont.

- ARMCI_Malloc for exposed memory
  - Collective and local; no register (in an undocumented version)
- ARMCI_RMW
  - Similar to Get_accumulate
  - No compare-and-swap
- ARMCI_Lock/Unlock
  - Lock special synch. objects
  - MPI RMA still doesn't include user-defined locks
- Limited set of collectives
  - Supports MPI collectives

1867

28

Co-Array Fortran

- RMA through co-arrays
  - Explicit collective allocation
- Execution divided in segments (delimited by "image control statements")
  - (non-volatile) operations in a segment are unordered
  - Exceptions are values with "atom" or volatile argument
- Image control statements:
  - Sync all, sync images, sync memory, lock/unlock
- Can (almost?) be implemented with MPI-3 RMA

1867

29

Unified Parallel C

- Explicit collective allocation
- Relaxed and strict access
  - ♦ Relaxed accesses are unordered
    - Exception are conflicting accesses to the same memory which appear in program order
  - ♦ Strict accesses are ordered (program order)
- Sync operations:
  - ♦ upc_fence, upc_(notify,wait), upc_barrier, upc_(un)lock
  - ♦ flush, (nonblocking) barrier, barrier, <no user locks>

30

Comparison Summary

- MPI-3 RMA covers most features from existing RMA systems
  - ♦ No user-defined locks
    - Can be emulated with compare and swap
- MPI-3 RMA can efficiently simulate many other RMA systems
  - ♦ Note more RMA programming models do not include an active message interface
  - ♦ Likely due to the complexity in defining the detailed semantics of AM for users
  - ♦ Some RMA features in other systems may require emulation, e.g., with MPI pt-2-pt

31

Executive Summary

- Fixes known deficits of MPI-2 RMA and can be implemented on non-CC architectures
    - Part of the complexity is pushed to the user
- ♦ Enables the use of MPI-RMA for new important application domains
- ♦ Does not handle everything but that was not a goal (e.g., no active messages)

32