An Analysis of MPI-2 OneSided

- not like it seemed -

Torsten Hoefler

Open Systems Lab

Indiana University

Bloomington, IN, USA

pervasivetechnologylabs
AT INDIANA UNIVERSITY

Introduction

- Two concepts and three calls
  - Central concepts: Window + Epochs
  - Data manipulation calls:
    - MPI_Put(src_addr, src_cnt, src_type, tgt_rank, tgt_disp, tgt_cnt, tgt_type, win)
    - MPI_Get(src_addr, src_cnt, src_type, tgt_rank, tgt_disp, tgt_cnt, tgt_type, win)
    - MPI_Accumulate(src_addr, src_cnt, src_type, tgt_rank, tgt_disp, tgt_cnt, tgt_type, op, win)
- Looks very similar to Active Messages
  - Possible implementation:
    - Put and accumulate are special handlers
    - Get triggers a put message on remote end
  - Interface needs to support hardware
    - RDMA and Shared Memory

Do we need Memory Windows?

- Windows enable:
  - Protection
  - Process-local offsets (heterogeneity)
  - Group contexts (cf. Communicators)
  - Explicit memory exposure helps hw support
- Windows require:
  - $\Omega(P)$ offset translation tables (if RDMA is used)
  - dense collective semantics
    - dense access is assumed
    - lazy translation entry fetching?
    - scalability problems

What do we really need?

- Do we want to (can we) give this up?
  - Shared memory needs mapping for direct access
    - shm_open(), mmap()
  - RDMA interconnects are not first class citizens (yet)
    - Need memory registration (pinning, explicit addr. transl.)
    - Future developments might use IOMMUs (page requ. intf.)
- We need memory exposure operation
  - often referred to as “memory registration”
  - GASNet+ARMCI also require special memory
- We don’t need this collectively though
  - Some apps might benefit from collective semantics?

Proposal I (Exposure)

- MPI_Expose_mem(addr, cnt, type, *hndl)
  - hndl could be MPI_Mem; hndl must be communicated to processes who want to access the region
    - Obvious scalability problem if all processes want to access/expose a region which has a global address then we might want an MPI_Win
    - Windows are suboptimal for objects/tasks or distributed data structures
  - independent of MPI_Alloc_mem()
    - Process memory can be exposed more than once!
  - see it as a “statically defined receive region”
  - could also specify contiguous region with bytes (like Win_create); datatypes seem more MPI-style though (?)
  - datatype is encoded in hndl
- MPI_Put(src_addr,src_cnt,src_type,tgt_rank,tgt_disp,tgt_cnt,hndl)
  - no target type! This is encoded in handle (static optimization)
  - data must fit in exposed buffer!
  - same for Get/Accumulate

pervasivetechnologylabs
AT INDIANA UNIVERSITY

Alternative Proposal (II) (Exposure)

- MPI_Expose_mem(addr, cnt, type, *hndl)
  - Datatype not encoded in hndl, could use MPI_Aint?
- MPI_Put(…, hndl)
  - Like original put, just uses hndl instead of win
- Differences between two versions:
  - Sender specifies data layout vs. receiver
  - Some implementation needs to send type information in both cases
    - Implicit in MPI-2 interface
    - Explicit in proposal version I

Do we need Epochs?

- Epochs enable:
  - BSP-like bulk synchronicity
  - Implementation on top of Message Passing (S/R)
  - Multiple nonblocking accesses without requests
- Synchronization modes:
  - Define consistency and scope of epochs
  - They seem rather complex in MPI-2
- Three modes:
  - Fence – BSP-like global synch (dynamic patterns)
  - start-complete/post-wait – p2p synch (fixed patterns)
  - Lock-unlock – target-specific epoch between lock/unlock (passive target)

pervasivetechnologylabs
AT INDIANA UNIVERSITY

Bonachea's and Duell's criticism

- Do not need collective semantics
  - passive target mode (lock/unlock)

1. Window creation is collective
   - hinders efficient exposure for local objects
   - no "sparse" communication
2. Exposed memory must be MPI_Alloc_mem()'d
   - no exposure of static memory or stack-variables
   - alloc_mem might be limited by the implementation
3. 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

Bonachea's and Duell's criticism ff

4. 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
5. Overlapping memory regions of multiple windows can be created but not be used
   - "concurrent communications may lead to erroneous results"
6. 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

Can we lift those restrictions?

1. Add a new point-to-point exposure (proposal I/II)
   - Would need remote pointer instead of window
   - MPI_Aint as target address?
2. Seems reasonable (see rationale in MPI-2)
   - Force better guarantees on Alloc_mem?
   - Fixed in proposal I/II (w/o Alloc_mem)
3. Could just be allowed (undefined outcome)
   - guarantee that any one of the accesses succeeds?
4. See 3)
5. See 3)
6. Needs new p2p synchronization mode

Extending the Calls?

- Put() and Get() are simple/sufficient
  - Lots of arguments though
    - supply destination information in a separate struct?
- Accumulate() seems very limited
  - Allow user-defined operations (must be able to send local op handles or register them globally)
- New call: Accumulate_get()
  - Implement read-modify-write
  - Arguments similar to Get (+ MPI_Op)
  - Very mighty together with user-defined ops

Synchronization and Consistency?

- ARMCI:
  - remote fence calls block until all ops issued to the target proc are completed at the dest
  - collective version: AllFence or Barrier
  - (New MPI RMA proposal seems identical)
- GASNet:
  - AM synchronization explicit by user (poll until a local condition is true)
  - RMA functions block until memory is consistent
    - Nonblocking versions available (with and without explicit handles)!

Synchronization Models contd.

- Collective epochs for BSP-like apps
  - MPI_Win_fence() works
- Point-to-point epochs
  - Maybe remote completion is not a bad idea
  - Wait/test for remote completion on a particular rank (with and without explicit handles)
  - Explicit synchronization (p2p) by user instead of start-complete-post-wait
- Which application would need passive target?

Proposal III (Synchronization)

- MPI_RMA_Wait(rank), MPI_RMA_Test(rank, *flag)
  - Waits/Tests for (remote) completion of **all** operations (from calling proc) on remote proc
  - Test() is kind of problematic because it is **not** local! Test() might block!
- Extend MPI_Put(…, *req)
  - one parameter more ☹ ☹ ☹
  - only for small numbers of relatively large transfers!
  - *req might be NULL (then only MPI_RMA_xxx() synch)
  - Test() is not allowed to block (can't progress)!
    - (small) problem with progress rule if target never calls MPI!
  - Wait() might block (poll target)

Optimizing the Interface (possibilities!)

- RDMA hardware (OFED, DCMF) and SM:
  - MPI_RMA_xxx() will just return success ☺
  - Win_fence() is simply an MPI_Barrier()
  - Ops might need extra message (see below)
- HW-supported AM (LAPI, DCMF):
  - send memory+op+local flag to target
  - Win_fence() could use termination detection algs.
  - AM handler on target triggers message to set local flag
  - or counter mechanism (cf. LAPI)
- Message Passing (Send/Recv):
  - emulate active messaging (send reply message after op is finished)

And Applications?

- Scalable RMA Put seems tricky
  - How does one ensure consistency?
  - If memory regions are assigned to each peer, what is the advantage to message passing?
    - Is pre-posting receives that hard?
    - Depends on application
- RMA Get seems useful
  - Accessing a globally distributed structure
- Are there algorithms or applications?
  - Any examples?

Questions?

pervasivetechnologylabs
AT INDIANA UNIVERSITY