MPI-3 One Sided

- ☺ some simple changes ☺ -

- half-baked -

Torsten Hoefler

Open Systems Lab

Indiana University

Bloomington, IN, USA

Introduction to MPI-2 One Sided

- 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 (likely 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 req. interf.)
- 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 (P2P Exposure)

- MPI-2 Memory windows expose collectively
  - P2P exposure can increase scalability -> local windows
- MPI_Win_create_local(base, size, displ, info, comm, win)
  - Local call to create a local window object
  - We might want to add a test for the type of win (not included yet)
  - MPI_Win_free() semantics change with type of win (coll. or not)

- Memory registration vs. registered allocation
  - Window creation takes existing memory
  - Some underlying APIs have strange requirements
    - posix shmat() wants page-aligned addresses or rounds addr. up
  - MPI RMA is not transparent → CPU can translate addresses
    - however, addr-translation can be simplified with special allocators

Proposal (AM Emulation)

- Allow user-defined ops in MPI_Accumulate()
  - requires **either** sending of MPI_Ops **or**
  - collective MPI_Op_create()
  - remember: ops *may have side effects*
- Differences to “real” AMs
  - can not trigger new messages (one-level)
  - have fixed-size vector input and output
    - access to more complex structures, e.g., lists or queues, needs to be “emulated” (hacked via side effects ;-))
  - MPI guarantees atomicity (this is good and bad)
  - no guaranteed asynchronous progr. (epochs)

Proposal (MPI_Get_accumulate)

- To implement lock-free remote atomics
  - (test&set, test&accumulate)
- Avoids unnecessary synchronization
- Allows concurrent accesses to same mem
  - unlike MPI_Put(), MPI_Get() which have undefined outcome in the current proposal
- Can play tricks with MPI_REPLACE or a new op MPI_OP_NULL (? -- simple get)
- Can easily be optimized for predefined ops!
  - Many interconnects (IB) do so already

What are Epochs (good for)?

- Epochs enable:
  - BSP-like bulk synchronicity
  - implementation on top of Message Passing (S/R)
  - batch 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)

Bonachea's and Duell's criticism

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

1. Window creation is collective
   - hinders efficient exposure for local objects
   - no "sparse" communication
   - adding local windows

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
   - force better guarantees on MPI_Alloc_mem?
   - static mem and stack variables remain problematic (is ok?)

Bonachea's and Duell's criticism ff

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
   - conflicting accesses are undefined not erroneous (no atomicity!)

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
   - conflicting accesses are undefined not erroneous (no atomicity!)

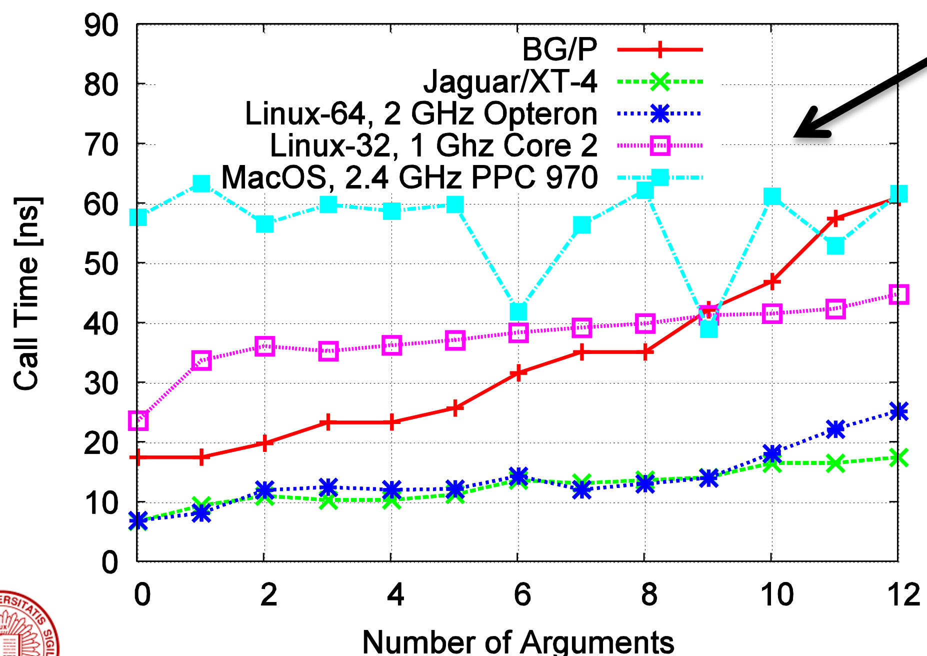
Bonachea's and Duell's criticism ff

5. Overlapping memory regions of multiple windows can be created but not be used
   - "concurrent communications may lead to erroneous results"
   - ➢ conflicting accesses are undefined not erroneous (no atomicity!)

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
   - ➢ use one local memory window for each process

What about the access calls?

- Put() and Get() are simple/sufficient
  - Lots of arguments though
    - Put()/Get(): 8; Acc(): 9; Get_acc(): 9 or 12
    - test: call a function 10^5 times with n pointer-args



MacOS results are reproducible!

System	Time
BlueGene/P	3.61 ns
Jaguar (XT-4)	0.78 ns
Odin (2 GHz x86-64)	1.15 ns
Laptop (1 GHz x86)	0.89 ns

Table: Costs per Argument

How to be as fast as SHMEM/ARMCI?

- The Put Calls:
  - shmem_put64(tgt, addr, size, dst)
  - ARMCI_Put(src_addr, dst_addr, count, tgt) **but**
  - ARMCI_PutS(src_addr, src_stride[], dst_addr, dst_stride[], count, stride_levels, tgt)
  - MPI_Put(src_addr, src_cnt, src_type, tgt_rank, tgt_disp, tgt_cnt, tgt_type, win)
- MPI "Overheads" (what makes MPI MPI)
  - datatypes (arbitrary patterns, heterogenity)
  - tgt_displs (heterogeneity)
  - win (collective **or** local window)
- If the Forum endorses a fast-path (advice to impl/users)
  - e.g., if (src_type==tgt_type==MPI_BYTE) then bypass datatype processing (heterogeneity and access patterns)
  - overheads: window lookup (might mean cache miss); tgt_displs might be expensive (could mean cache miss or remote action); make all displs==0 a special case!
  - if no cache miss: only two branches more!

Synchronization and Consistency?

- ARMCI:
  - remote fence blocks until all ops completed at dest
  - collective version: AllFence or Barrier
- GASNet:
  - AM synchronization by user (poll on local variable)
  - RMA functions block until memory is consistent
    - Nonblocking versions available (with and without explicit handles)!
- SHMEM:
  - might require shmem_udcflush() on target (Cray T90)
  - synch with shmem_wait(), shmem_barrier() or polling
    - not 100% defined (publicly) as it seems

Synchronization Models in MPI

- Collective windows
  - MPI_Win_fence() works well
  - post/start/complete/wait is nice but hard to use
  - passive target mode has consistency problems
- Local windows:
  - MPI_Win_fence() would need to be called by both processes (doesn't seem useful)
  - post/start/complete/wait would be funny
  - passive target mode could be useful on systems with strong memory consistency
    - how do we know the memory consistency?

Querying memory consistency

- MPI_RMA_query(optype, model)
  - optype – selects put, get, acc, or get_acc
  - model:
    - MPI_RMA_SEPARATE – not cache coherent/weak semantics
    - MPI_RMA_ONE – cache coherent/strict semantics
- Memory consistency explained:
  - SEPARATE: behaves like MPI-2, needs target needs to call MPI to make progress
  - ONE: behaves like InfiniBand, ARMCI, SHMEM, target memory is updated asynchronously (eventually)

What’s different with strong consistency?

- Active target mode:
  - fence: pretty much nothing (the user can assume better overlap behavior ☺)
  - post/start/complete/wait: post-start and complete-wait dependencies persists
- Passive target mode:
  - becomes similar to ARMCI/SHMEM
  - user might choose to ignore lock/unlock
    - access epochs vanish
  - synchronization via polling (ugs)
    - we might want to add MPI_Target_wait() (?)

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)

Questions?