MPI RMA
Is It Time To Start Over?

William Gropp
wgropp.cs.Illinois.edu

Different Perspectives on RMA

- Different communities have different goals for RMA
  - Express algorithms with different synchronization patterns
    - Improves (perhaps significantly) programmability for some algorithms
    - Better match to algorithm; e.g., avoid extra messages/data transfers/synchronization
  - Access higher performance
    - Reduce CPU overhead (e.g., tag matching, message ordering)
    - Minimize memory motion, control messages
    - Exploit hardware support for communication, overlap with computation, especially for large block transfers
    - Low-latency for short data transfers
    - Remote atomic memory operations or other hardware assist
      - But tricky – e.g., atomics not always faster than lock/ops/unlock for sequences of operations
  - Optimize for important special cases
    - E.g., symmetric allocation

Thinking about RMA performance

- Different applications have different needs
  - Some need low overhead/latency on short messages; common for strong scaling case
  - Some need high bandwidth for large messages; overlap with computation; common for weak scaling case
- These can be in tension, with performance tradeoffs
- MPI “look and feel” adds additional tradeoffs
  - Datatypes can provide optimization opportunities (e.g., bundle move, from strided to general scatter/gather, together) but add complexity (increase latency for simple cases)
  - Rich capabilities can force reliance on remote agent, impacting performance

What Can an RMA Design Assume?

- MPI has succeeded by embracing a “Greatest Common Denominator” approach
  - Take advantage of community consensus on hardware features
  - Ensure portability with good performance
- But the “Common” part stifles innovation
  - Features not already adopted are hard to fully exploit from within the standard
  - The tyranny of “Common” forces the standard to sacrifice both performance and programmability for portability and precise semantics
- Q: How can MPI encourage innovation in RMA interfaces?
- Q: Is portability (with performance) a hard or soft constraint?

MPI One Sided/Remote Memory Access History

- MPI-2 added RMA in 1997 (25 years ago!)
  - Some practice, but semantics of other systems often imprecise
  - Matched hardware capabilities of high-end systems of the time (Cray T3D/T3E; NEC Earth Simulator)
  - Expected support in network NIC with local memory (hence memory model)
  - Only collective association of memory with MPI_Win
- Both Fence and PSCW defined to exploit the hardware of the day
  - Fence could be implemented in hardware and was *very* fast on some systems (e.g., T3D/T3E)
  - PSCW described halo exchange well (though hard to exploit)
- Active target and passive target captured different application styles
  - Passive target limitations an attempt to ensure portable performance

MPI One Sided/Remote Memory Access History

- MPI-2 RMA had limited adoption
  - Complex memory model hard to explain
  - Limitations on passive target memory limit usefulness
  - Limitations on operations, memory, etc.
  - Poor performance of implementations – often unnecessarily so
  - Even with that, some apps and implementations did very well
- MPI-3 substantially revised and enhanced RMA in 2012
  - Address overly strong correctness semantics (undefined rather than erroneous) and additional use cases for applications
  - Add “unified” memory model – HW support for coherency now widespread
  - Add additional ways to associate memory, describe data transfers, complete operations, and extend to processes sharing memory
  - But added to MPI-2 RMA – keeping all features from (then) 15 years before

Relevance

- Is MPI RMA too complex, portable, limited, constrained, etc. to be useful?
  - Consider challenges in using MPI RMA for implementing other one-sided programming systems and libraries
- MPI-2 RMA, for all of its limitations, was driven by use examples of the time. MPI-3 also driven by different use examples.
  - What are the right use cases for MPI-5 RMA?
  - Who is the right audience?
- Is MPI RMA a high-level interface, expected to be used freely within user applications, or a low-level interface, used to implement core abstractions in an application framework?
  - Like much of MPI, as Marc Snir points out, it is both – and that is likely a bad choice

Synchronization and Notification

- Moving data is the easy part. Synchronization/notification is the hard part
  - This is the biggest area where RMA has struggled, with many different mechanisms for completing RMA, both locally and remotely
    - Example: Fence – with hardware support, can be incredibly fast – but imposes a "BSP"-like structure. More general semantics (groups != WORLD) may not have same hardware support, but this is difficult for the programmer to determine
- Notification is both more powerful and harder/more demanding to implement and meet user performance expectations
- Small changes in semantics can have large performance impact if they change what can be done in hardware and what requires CPU assist
  - This applies to *both* the MPI specification and the capabilities of hardware

MPI RMA Synchronization

- MPI RMA Synchronization is complex
- Trying to keep things simple for programmer (all sync methods available at all times) makes implementation complex and adds overhead
- Q: How important is this level of interoperability?
- Q: Are these the right ways to synchronize?

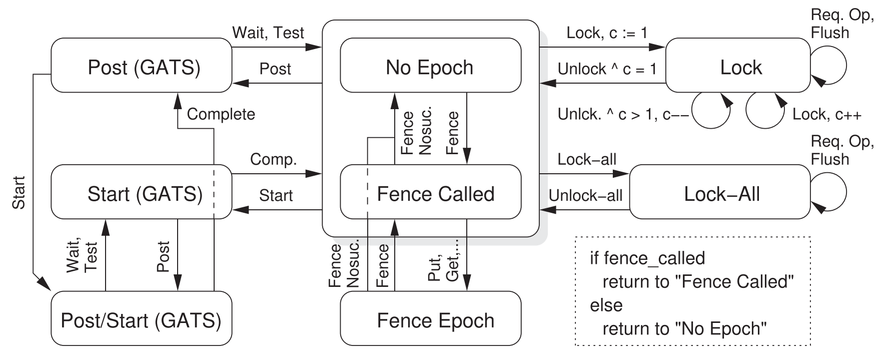

Figure 3. Remote memory access synchronization state tracking diagram. Dashed lines indicate that a particular state is bypassed, depending on the fence state of the process.

An implementation and evaluation of the MPI 3.0 one-sided communication interface, James, Dinan, Pavan Balaji, Darius Buntinas, David Goodell, William Gropp,, and Rajeev Thakur, *Concurrency and Computation: Practice and Experience*, 28, 17, 4385–4404, cpe.3758, 2016.

Innovation and Stability

- How can MPI RMA stay current with technology when there isn't consensus?
  - It can't – so we'll need to make some compromises
    - We're currently accepting lower performance and capability to get portability and stability of code. Is that the right choice?
- Q: Should MPI RMA provide access to features that can't be efficiently supported on all platforms?
  - Should implementations be required to support all features, even if they are inefficient, or should the features not be supported?
  - How can users reliably determine what is supported, and what is supported with adequate performance (and how do you define that)?

Audience

- Who is expected to use MPI RMA? End users? Tool developers? Compiler writers?
  - More precisely, *which* parts of RMA are for each of these groups?
  - What is the role of libraries?
  - For end users, how expert are the users? Shared memory issues are very tricky; RMA shares many of these hazards.
- The MPI RMA specification is aimed at MPI implementors, and (correctly) worries about all the edge cases
  - Users (mostly) don't want that – they want a subset that is easy to use and understand, even if that might mean missing some optimizations
- Q: Can we describe RMA for a single audience, or should the be different user profiles?
- Q: Can/Should MPI RMA define easier-to-understand subsets? Or is this the obligation of training materials?

Feature Lifetime

- What is the lifetime required? Do RMA codes need to run without change in 20 years? 10? 5? At what cost in potential performance?
  - This impacts how we approach hardware innovation
  - Many modern software systems expect to break backward compatibility – is it time for MPI to do the same, at least in some places?
- Note we’re leaving an era of over 3 decades of architectural stability – which has been of great benefit to MPI
  - But we’re leaving that era – what made sense while architectures were (mostly) stable may no longer make sense
- Q: Is backward compatibility essential?
- Q: Could the standard offer some features with an explicit limited lifetime?

Progress

- One-sided nature of RMA requires some progress guarantee
- But TANSTAAFL (There Ain't No Such Thing As A Free Lunch)
  - Many tradeoffs – e.g., more frequent/responsive progress *may* increase latency, lower performance. Or increase latency, but increase performance. Or increase performance, because you found a good use for an idle core…
- Many changing technical tradeoffs (dark silicon, "extra" cores, …)
  - Tradeoffs that made sense with < 1core/chip may not with > 100 cores/chip
- Rather than all-or-nothing progress, is there something in the middle?
  - Note that MPI-2 permitted restricting passive target operations to special memory – something many did not like, but made sense at the time
- "Functional progress," e.g., a guarantee of eventual progress, does not meet most users' expectation for "timely progress"

Performance and Generality

- MPI is a greatest common denominator approach
  - Often described insultingly as "least common denominator" – which is a nonsense phrase
  - But even "greatest common" is limited to "common"
- Significant performance impact when abstraction is far from what is supported in hardware – but hardware operations esp. for RDMA are still evolving
  - Some systems handle this by giving up on precision in the specification (!!)
- Is high performance low latency or high bandwidth? What if you can't have both?
- The goal of "performance portability" without code change opposes making "optional" features available
  - But what if a framework (e.g., PETSc) insulates the user from those changes?
  - Q: A what level (if any) do we need performance portability?

Thoughts for RMA in MPI 5.0

- RMA in MPI-2 was driven by the hardware of the day, including limitations
  - Examples: Fence in hardware, limited memory in NIC (separate memory model, passive target memory restrictions)
  - 25+ years is too long to simply tweek the programming model to match the hardware – MPI RMA should be rethought from the ground up to meet current hardware
- One-sided hardware acceleration remains in flux
  - No consensus on what are the right abstractions (though some are clear)
  - Suggests:
    - Don’t require greatest common denominator for RMA.
    - Provide a way to access extensions and query for capabilities.
    - Define a likely subset where portability (in time and across vendors) is important as a trade off in performance
  - Applications are likely to define communication abstractions – and can provide implementations that can exploit optional features without imposing a large burden on the programmer
    - Many do this today

Thoughts for RMA in MPI 5.0

- Progress may be solved, at least to first order
  - Can we assume that there are enough cores/execution contexts to ensure some progress?
    - Or is this the wrong direction? Should we be looking at progress with no CPU involvement, at least with the right hardware?
  - As above, are there intermediate levels of progress, as there are for thread support?
- Evolution should be driven by use cases
  - Where do we want to see MPI RMA used? How do we engage that community?
  - But a warning: an RMA interface that is the Union of all features may satisfy no-one.
    - A strength of MPI is its support for tools and higher-level interfaces
    - Can we ensure that users are served by these tools without requiring MPI to directly support everyone?

Conclusion

- MPI RMA originally designed to exploit (limited) RDMA hardware capabilities
- Use cases driven by limited set of examples
- Focus on “Greatest Common Denominator” for HW and over-constrained design has limited innovation
- For the rest of this workshop:
  - Be clear about your audience, goals, and tradeoffs
  - **Everything is fair game**
  - But be constructive – just because a design decision isn’t the one you would make doesn’t make it wrong – but maybe wrong for you!