MPI Past and Future

William Gropp
wgropp.cs.Illinois.edu

NCSA

Rolf and The MPI Forum

- The MPI Forum is an ad hoc group of volunteers passionate about providing a practical, effective method for programming massively parallel computers
- Rolf has been a key member of the Forum
  - A strong advocate for Fortran – and precision in use of and conformance to the standard
- Rolf has also been a strong advocate for the use of language features to make the MPI library more “user friendly”, including catching usage errors at compile time

Some Context

- Before MPI, there was chaos – many systems, but mostly different names for similar functions.
  - Even worse – similar but not identical semantics
- Same time(ish) as attack of the killer micros
  - Single core per node for almost all systems
- Era of rapid performance increases due to Dennard scaling
  - Most users could just wait for their codes to get faster on the next generation hardware
  - MPI benefitted from a stable software environment
    - Node programming changed slowly, mostly due to slow quantitative changes in cache, instruction sets (e.g., new vector instructions)
- The end of Dennard scaling unleashed architectural innovation
  - And imperatives – more performance requires exploiting parallelism or specialized architectures
  - (Finally) innovation in memory – at least for bandwidth

Why Was MPI Successful?

- It addresses all of the following issues:
  - Portability
  - Performance
  - Simplicity and Symmetry
  - Modularity
  - Composability
  - Completeness
- For a more complete discussion, see "Learning from the Success of MPI",
- https://link.springer.com/chapter/10.1007/3-540-45307-5_8

Performance vs. Productivity

- MPI gives the tools for achieving performance
  - In large part by not getting in the way of locality management
- But that very feature impacts productivity
  - User has no choice but to manage locality, which is both hard and tricky
- In addition, as Marc Snir has noted, MPI is neither high nor low level
- But is that part of MPI's ***success*** – it does both high and low level, and the tradeoff in greater use (mostly) makes up for loss of performance/function
- Any programming system will need to consider the tradeoffs of
  - Latency vs. Bandwidth vs. Convenience vs. Modularity (among others)

But What about the Programming Crisis?

- Use the right tools
- MPI tries to satisfy everyone, but the real strengths are in
  - Attention to performance and scalability
  - Support for libraries and tools
- Many computational scientists use frameworks and libraries built upon MPI
  - This is the right answer for most people
  - Saying that MPI is the problem is like saying C (or C++) is the problem, and if we just eliminated MPI (or C or C++) in favor of a high productivity framework *everyone's* problems would be solved
  - In some ways, MPI is *too* usable – many people can get their work done with it, which has reduced the market for other tools
    - Particularly when those tools don't satisfy the 6 features in the success of MPI

What Might Be Next

- Intranode considerations
  - SMPs (but with multiple coherence domains); new memory architectures
  - Accelerators, customized processors (custom probably necessary for power efficiency)
  - MPI can be used (MPI+MPI or MPI everywhere), but somewhat tortured
    - No implementation built to support SIMD on SMP, no sharing of data structures or coordinated use of the interconnect
- Internode considerations
  - Networks supporting RDMA, remote atomics, even message matching (partially supported in MPI now – but what's next?)
  - Overheads of ordering
  - Reliability (who is best positioned to recover from an error)

What Might Be Next

- MPI is both high and low level – can we resolve this?
- Challenges and Directions
  - Scaling at fixed (or declining) memory per node
    - How many MPI processes per node is "right"?
  - Realistic fault model that doesn't guarantee state after a fault
  - Support for complex memory models (MPI_Get_address ☺)
  - Support for applications requiring strong scaling
    - Implies very low latency interface and overheads
    - Low latency means paying close attention to the implementation
      - RMA latencies sometimes 10-100x point-to-point in implementations (!)
  - MPI performance in MPI_THREAD_MULTIPLE mode
  - Integration with code re-writing and JIT systems as an alternative to a full language

Adapt to Innovation in Architecture

- Complex nodes
  - MPI + X, for X such as OpenMP, CUDA, OpenACC, etc. often effective
  - But challenges in the "**+**": sharing of resources such as cores, memory, …
- Implementation of MPI on complex nodes
  - Sharing information between MPI processes on the same node that must share resources, such as memory, network, accelerators, …
  - Optimize data movement
- Some can be hidden from the user (shared memory for intranode message passing)
- Some requires user action – e.g., node-aware algorithms and methods

NCSA

Adapt to New Language Models – And to Their Rapid Evolution

- Is (long-term) backward compatibility still important?
  - Many newer languages and systems don't think so – 5 years is long for them
- How does the value of backward compatibility change with age?
  - As older codes become less important (or more modern codes become available), what is the tradeoff in making newer codes more capable/flexible/etc. or the environment more productive?
- What is the cost to future applications and usage from providing backward compatibility?
  - Many of us started careers when long-term backward compatibility was expected. Is this still the right thing?
- What does all of this mean for MPI?

Adapt to New Application Domains and User Communities

- Adapt to new application domains and user communities, as well as expectations about software
- MPI is still for HPC – but new domains such as bioinformatics, Health, AI+X, ….

One Sided/Remote Memory Access History

- MPI-2 added RMA in 1997 (25 years ago!)
  - Some practice, but semantics before MPI 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
- 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
- MPI-4 further updated RMA in 2021 (only minor changes)

Synchronization

- 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
    - 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 – and hence may not perform well
- 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?

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.
- 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?

Progress

- One-sided nature of RMA requires some progress guarantee
- But TANSTAAFL (There Aint 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

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 “<u>common</u>”
- Significant performance impact when abstraction is far from what is supported in hardware – but hardware operations still evolving
  - Some systems handle by giving up on precision in the specification (!!)
- Is high performance low latency or high bandwidth? What if you can’t have both?

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.
  - What are the right use cases for MPI-5 RMA?
  - What is the right audience?

Thoughts for RMA in MPI 5.0

- One-sided hardware acceleration remains in flux
  - Unclear what are the right abstractions
  - Suggests: Don't require greatest common denominator for RMA synchronization. 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
- "Progress" may be solved, at least to first order
  - Can we assume that there are enough cores/execution contexts to ensure some progress?
  - 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?

Summary

- MPI has been very successful, but faces challenges as computing changes
- What is the balance between innovation (change) and stability (backward compatibility)?
- Specification vs. implementation
- MPI and X – how can be better compose programs that use programming systems (languages, libraries, tools) optimized to each part of the application?
- Become part of the conversation!
  - Join the MPI Forum
  - Participate in discussions
  - Provide challenges