MPI in 2020: Opportunities and Challenges

William Gropp
www.cs.illinois.edu/~wgropp
The Message Passing Interface (MPI) has been amazingly successful

- First released in 1992, it is still the dominant programming system used to program the world’s fastest computers
- The most recent version, MPI 3.1, released in June 2015, contains many features to support systems with >100K processes and state-of-the-art networks

Supercomputing (and computing) is reaching a critical point as the end of Dennard scaling has forced major changes in processor architecture.

This talk looks at the future of MPI from the point of view of Extreme scale systems
- That technology will also be used in single rack systems
Likely Exascale Architectures

- Note that I/O is not part of this (maybe hung off the NIC)
What This (Might) Mean for MPI

- Lots of innovation in the processor and the node
- More complex memory hierarchy; no chip-wide cache coherence
- Tightly integrated NIC
- Execution model becoming more complex
  - Achieving performance, reliability targets requires exploiting new features
- Node programming changing
  - OpenMP/OpenACC/CUDA; shared memory features in C11/C++11
What This (Might) Mean for Applications

• Weak scaling limits the range of problems
  ♦ Latency may be critical (also, some applications nearing limits of spatial parallelism)

• Rich execution model makes performance portability unrealistic
  ♦ Applications will need to be flexible with both their use of abstractions and their implementation of those abstractions

• One Answer: Programmers will need help with performance issues, whatever parallel programming system is used
  ♦ Much of this is independent of the internode parallelism, and can use DSLs, annotations, source-to-source transformations.
Where Is MPI Today?

- Applications already running at large scale:

<table>
<thead>
<tr>
<th>System</th>
<th>Cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tianhe-2</td>
<td>3,120,000 (most in Phi)</td>
</tr>
<tr>
<td>Sequoia</td>
<td>1,572,864</td>
</tr>
<tr>
<td>Blue Waters</td>
<td>792,064* + 59,136smx</td>
</tr>
<tr>
<td>Mira</td>
<td>786,432</td>
</tr>
<tr>
<td>K computer</td>
<td>705,024</td>
</tr>
<tr>
<td>Julich BG/Q</td>
<td>458,752</td>
</tr>
<tr>
<td>Titan</td>
<td>299,008* + 261,632smx</td>
</tr>
</tbody>
</table>

* 2 cores share a wide FP unit
Many reasons to consider MPI+X

- **Major**: We always have:
  - MPI+C, MPI+Fortran
- Both C11 and Fortran include support of parallelism (shared (C) and distributed memory (Fortran))

Abstract execution models becoming more complex

- Experience has shown that the programmer must be given some access to performance features
- Options are (a) add support to MPI and (b) let X support some aspects
Many Possible Values of X

- X = MPI (or $X = \phi$)
  - MPI 3 has many features esp. important for Extreme scale
  - Nonblocking collectives, neighbor collectives,…
  - MPI 4 looking at additional features (e.g., RMA with notify; come to the MPI BoF today!)

- X = threads (OpenMP/pthreads/C11)
  - C11 provides an adequate (and thus complex) memory model for writing portable thread code

- X = CAF or UPC or other (A)PGAS
  - Think of as an extension of a thread model
What are the Issues?

- Isn’t the beauty of MPI + X that MPI and X can be learned (by users) and implemented (by developers) independently?
  - Yes (sort of) for users
  - No for developers

- MPI and X must either partition or share resources
  - User must not blindly oversubscribe
  - Developers must negotiate
More Effort needed on the “+”

• MPI+X won’t be enough for Exascale if the work for “+” is not done very well
  ♦ Some of this may be language specification:
    • User-provided guidance on resource allocation, e.g., MPI_Info hints; thread-based endpoints
  ♦ Some is developer-level standardization
    • A simple example is the MPI ABI specification – users should ignore but benefit from developers supporting
Which MPI?

- Many new features in MPI-3
  - Many programs still use subsets of MPI-1
- MPI implementations still improving
  - A long process – harmed by non-standard shortcuts
- MPI Forum is active and considering new features relevant for Exascale
  - MPI 3.1 released June 2015
  - See the MPI BoF Today for more info!
Fault Tolerance

• Often raised as a major issue for Exascale systems
  ♦ Experience has shown systems more reliable than simple extrapolations assumed
    • Hardly surprising – reliability is costly, so systems engineered only to the reliability needed

• Major question: What is the fault model?
  ♦ Process failure (why is this the common model?)
    • Software – then program is buggy. Recovery may not make sense
    • Hardware – Where (CPU/Memory/NIC/Cables)? Recovery may be easy or impossible
  ♦ Silent data corruption (SDC)

• Most effort in MPI Forum is on process fail-stop faults
Separate Coherence Domains and Address Spaces

• Already many systems without cache coherence and with separate address spaces
  ♦ GPUs best example; unlikely to change even when integrated on chip
  ♦ OpenACC an “X” that supports this
• MPI designed for this case
  ♦ Despite common practice, MPI definition of MPI_Get_address supports, for example, segmented address spaces; MPI_Aint_add etc. provides portable address arithmetic
• MPI RMA “separate” memory model also fits this case
  ♦ “Separate” model defined in MPI-2 to support the World’s fastest machines, including NEC SX series and Earth Simulator
Towards MPI-4

- Many extensions being considered, either by the Forum or as Research, including
  - Other communication paradigms
    - Active messages
      - Toward Asynchronous and MPI-Interoperable Active Messages, Zhao et al, CCGrid’13
    - Streams
  - Tighter integration with threads
    - Endpoints
  - Data centric
    - More flexible datatypes
    - Faster datatype implementations (see, e.g., Prabhu & Gropp, EuroMPI’15)
    - Better parallel file systems (match the MPI I/O semantics)
  - Unified address space handling
    - E.g., GPU memory to GPU memory without CPU processing
MPI is not a BSP system

- **BSP = Bulk Synchronous Programming**
  - Programmers like the BSP model, adopting it even when not necessary (see “functionally irrelevant barriers”)
  - Unlike most programming models, designed with a performance model to encourage quantitative design in programs

- **MPI makes it easy to emulate a BSP system**
  - Rich set of collectives, barriers, blocking operations

- **MPI (even MPI-1) sufficient for dynamic adaptive programming**
  - The main issues are performance and “progress”
  - Improving implementations and better HW support for integrated CPU/NIC coordination is the right answer
Some Remaining Issues

• Latency and overheads
  ♦ Libraries add overheads
    • Several groups working on applying compiler techniques to MPI and to using annotations to transform user’s code; can address some issues

• Execution model mismatch
  ♦ How to make it easy for the programmer to express operations in a way that makes it easy to exploit innovative hardware or runtime features?
  ♦ Especially important for Exascale, as innovation essential in meeting 20MW, MTBF, total memory, etc.
What Are The Real Problems?

- Support for application-specific, distributed data structures
  - Not an MPI problem
  - Very hard to solve in general
  - Data-structure Specific Language (often called “domain” specific language) a better solution

- A practical execution model with a performance model

- Greater attention to latency
  - Directly relates to programmability
MPI in 2020

- Alive and well, using C11, C++11, Fortran 2008 (or later)
- Node programming uses locality-aware, autotuning programming systems
- More use of RMA features
  - Depends on better MPI implementations, continued co-evolution of MPI and RMA hardware to add new features (notification?)
- (Partial?) solution of the “+” problem
  - At least an ad hoc implementers standard for sharing most critical resources
- Some support for fault tolerance
  - Probably not at the level needed for reliable systems but ok for simulations
- Better I/O support, including higher level libraries
  - But only if the underlying system implements something better than POSIX I/O