Is MPI+X Enough for Exascale?

William Gropp

www.cs.illinois.edu/~wgropp
Likely Exascale Architectures

Figure 2.1: Abstract Machine Model of an exascale Node Architecture

More Details: Processor

- Memory bandwidths from 1200GB/s (core-L1 cache) to 60GB/s (off-chip conventional DRAM)
- 64-256 cores/chip; 2-64 threads/core; 4-8 wide SIMD, 8-128 outstanding references per core
- Atomic ops, including transactional memory
- Documents silent on memory latencies
  - Probably because numbers are uninspiring
More Details: Memory

- **Multilevel**
  - DRAM on chip (64GB), off chip (2TB)
  - NVRAM (higher density – 16TB, but requires larger access units (≥1KB))
  - Stacked memory

- **Compute near memory**
  - “Extended memory semantics”
  - Full/empty bits; gather/scatter; stream compute; ...
More Details: Network

- 100-400 GB/s injection BW
- Topology anyone’s guess (SlimFly, perhaps?)
- 250M message/s two-sided
- 1000M messages/s one-sided
- Latency:
  - 0.5-1.4 µsec two-sided nearest neighbor
  - 0.4-0.6 µsec one-sided nearest neighbor
  - 3-5 µsec cross machine
- Note: about the same as current systems
Another estimate, from “CFD Vision 2030 Study: A Path to Revolutionary Computational Aerosciences,” Slotnick et al, 2013
What About a Homogeneous System?

- IBM BlueGene was the only homogeneous* system at this scale, but ...
  
  - “Both CORAL awards leverage the IBM Power Architecture, NVIDIA’s Volta GPU and Mellanox’s Interconnected technologies to advance key research initiatives ...”

- * Try to use the very wide SIMD on BlueGenes. Homogeneously heterogeneous
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
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
- Answer 0: Programmers will need help with performance issues, whatever parallel programming system is used
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 Intel Phi)</td>
</tr>
<tr>
<td>Sequoia BG/Q</td>
<td>1,572,864</td>
</tr>
<tr>
<td>Blue Waters</td>
<td>792,064* + 1/6 acc (59,136 GPU stream proc)</td>
</tr>
<tr>
<td>Mira</td>
<td>786,432</td>
</tr>
<tr>
<td>K computer</td>
<td>705,024</td>
</tr>
<tr>
<td>Stampede</td>
<td>462,462 (most in Intel Phi)</td>
</tr>
<tr>
<td>Julich BG/Q</td>
<td>458,752</td>
</tr>
<tr>
<td>Vulcan BG/Q</td>
<td>393,216</td>
</tr>
<tr>
<td>Titan</td>
<td>299,008* + acc (261,632 GPU stream proc)</td>
</tr>
</tbody>
</table>

* 2 cores share a wide FP unit
Some Experiments over 1M MPI Processes

- ROSS Parallel Discrete Event Simulator
  - Used over 7.8M MPI processes on 2 combined BG/Q systems at LLNL, 4 ranks per core
  - “Warp Speed: Executing Time Warp on 1,966,080 Cores,” Barnes, Carothers Jefferson, LaPre, PADS 2013

- FG-MPI implements MPI ranks as coroutines
  - Wagner at UBC
  - Over 100M MPI ranks on 6,480 cores
MPI+X

• Many reasons to consider MPI+X
  ♦ Major: We always have:
    • MPI+C, MPI+Fortran
  ♦ Both C11 and Fortran include support of parallelism (shared and distributed memory)

• 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
X = MPI (or X = $\phi$)

- MPI 3.1 features esp. important for Exascale
  - Generalize collectives to encourage post BSP programming:
    - Nonblocking collectives
    - Neighbor – including nonblocking – collectives
  - Enhanced one-sided (recall AMM targets)
    - Precisely specified (see “Remote Memory Access Programming in MPI-3,” Hoefler et al, to appear in ACM TOPC)
    - Many more operations including RMW
  - Enhanced thread safety
X = Programming with Threads

- Many choices, different user targets and performance goals
  - Libraries: Pthreads, TBB
  - Languages: OpenMP 4, C11/C++11
- C11 provides an adequate (and thus complex) memory model to write portable thread code
  - Also needed for MPI-3 shared memory
X=UPC (or CAF or ...) 

- MPI Processes are UPC programs (not threads), spanning multiple coherence domains. This model is the closest counterpart to the MPI + OpenMP model, using PGAS to extend the “process” beyond a single coherence domain.

- Could be PGAS across chip
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
Answer 1: 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
Some Resources to Negotiate

- **CPU resources**
  - Threads and contexts
  - Cores (incl placement)
  - Cache

- **Memory resources**
  - Prefetch, outstanding load/stores
  - Pinned pages or equivalent NIC needs
  - Transactional memory regions
  - Memory use (buffers)

- **NIC resources**
  - Collective groups
  - Routes
  - Power

- **OS resources**
  - Synchronization hardware
  - Scheduling
  - Virtual memory
  - Cores (dark silicon)
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
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)
    • 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

• Unsolved problem – impact of faults on X (and +) in MPI+X
Fault Tolerance

• Most effort in MPI Forum is on process fail-stop faults
  ♦ Complication: process fails while communicating
  ♦ Confusion: between MPI the standard and MPI an implementation

• Other faults may be more important
  ♦ I/O failover faults. How long should an I/O operation wait before failing, and should the operation be safely restartable? Who is responsible?
    • I/O software errors the dominant source of system downtime
  ♦ Silent data corruption.
    • Data in numeric values. Often easy to define restart. State of program is correct, except for the affected data (and tainted data)
    • Data in code, pointers, key data structures. State of program may be unknown. Restart needed from known good state
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 one of several “Xs” that supports this

- MPI designed for this case
  - Despite common practice, MPI definition of MPI_Get_address supports, for example, segmented address spaces

- 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; followups at CCGrid’15
  - Streams
- Tighter integration with threads
  - Endpoints
- Data centric
  - More flexible datatypes
  - Faster datatype implementations
- Unified address space handling
  - E.g., GPU memory to GPU memory without CPU processing
MPI and Execution Models

• MPI’s Execution model is...
  ♦ Blissfully simple: Communicating Sequential Processes
    • Some complexity in communication, esp. MPI-3 one-sided
  ♦ Process operations are copy, elementwise arithmetic/logic/bit, read/write (I/O)
  ♦ MPI adds two-party and group synchronization and operations
  ♦ No performance guarantees
  ♦ Deliberately vague on progress
  ♦ Deliberately vague on faults
MPI and Exascale Execution Models

• End of Dennard scaling, end of Moore’s law (which is about feature scaling at fixed cost), forcing new, more complex execution models
  ♦ Some can be buried in the “X”, e.g., stream programming
  ♦ Some can be buried in the “+”, e.g., limited resources for implementing runtimes and programming systems
  ♦ Some may need to be exposed to the MPI programmer
MPI is not a BSP system

- BSP = Bulk Synchronous Programming
  - Programmers like the BSP model, adopting it even when not necessary (see FIB)
  - 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 the 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.
Summary

• MPI a viable component in an Exascale software stack
• But addresses only part of the problem
• More work is needed on effective combination of systems (the “+”)
• More work is needed on automation for performance and for performance portability
• MPI must be augmented by higher level approaches, as originally intended