BLUE WATERS

SUSTAINED PETASCALE COMPUTING

New and old Features in MPI-3.0: The Past, the Standard, and the Future

Torsten Hoefler

With contributions from the MPI Forum

GREAT LAKES CONSORTIUM FOR PETASCALE COMPUTATION

All used images belong to the owner/creator!

What is MPI – Message Passing Interface?

- An open standard library interface for message passing, ratified by the MPI Forum
  - Versions: 1.0 ('94), 1.1 ('95), 1.2 ('97), 2.0 ('97), 1.3 ('08), 2.1 ('08), 2.2 ('09), 3.0 (probably '12)
- Common misconceptions:
  - MPI parallelizes your application
  - MPI is for distributed memory only
  - MPI (a library interface) is not scalable
  - MPI is fundamentally slower then PGAS etc.
- Really, if you don't know what MPI is, you won't enjoy this talk ☺

What is this MPI Forum?

- An open Forum to discuss MPI
  - You can join! No membership fee, no perks either
- Since 2008 meetings every two months for three days (switching to four months and four days)
  - 5x in the US, once in Europe (with EuroMPI)
- Votes by organization, eligible after attending two of the three last meetings, often unanimously
- Everything is voted twice in two distinct meetings
  - Tickets as well as chapters

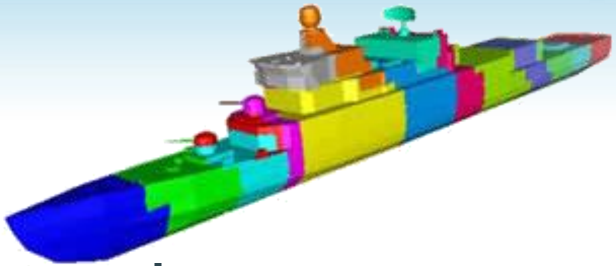
How does the MPI-3.0 process work

- Organization and Mantras:
  - Chapter chairs (convener) and (sub)committees
  - Avoid the "Designed by a Committee" phenomenon → standardize common practice
  - 99.5% backwards compatible
- Adding new things:
  - Review and discuss early proposals in chapter
  - Bring proposals to the forum (discussion)
  - Plenary formal reading (usually word by word)
  - Two votes on each ticket (distinct meetings)
  - Final vote on each chapter (finalizing MPI-3.0)

Now to the technical part ☺

- Topology Mapping (MPI-2.2)
- Nonblocking and Neighborhood Collectives
- Matched Probe
- MPI Tool interface
- New One Sided Functions and Semantics
- New Communicator Creation Functions
- Improvements in Language Bindings
- Fault Tolerance/Resiliency

Topology Mapping in MPI-2.2

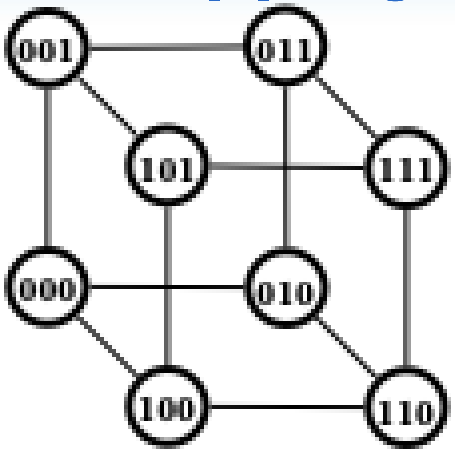


- Specify application/algorithm communication topology via virtual topology creation functions (since MPI-1.0)
  - MPI_Cart_create() – builds a k-dimensional Cartesian application topology, very scalable
  - MPI_Dist_graph_create() – replaces non-scalable ~~MPI_Graph_create()~~ with a scalable version
  - MPI_Dist_graph_create_adjacent() – even more scalable but **all** processes specify **all** neighbors
- How does it map to a topology?

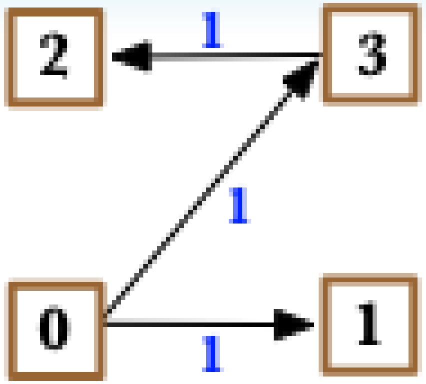
Hoefler et al.: The Scalable Process Topology Interface of MPI 2.2, CCPE Journal 2010

Example Mappings

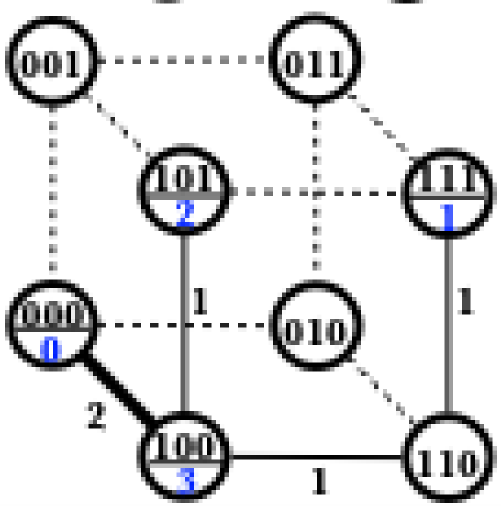
Physical Topology:



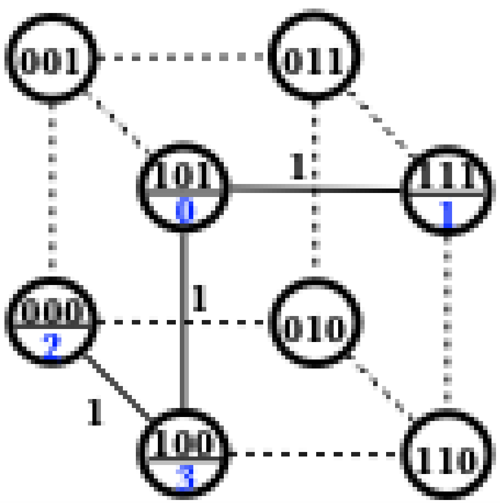
Application Topology:



Mapping 1:



Mapping 2:



Why do I care?

- Increase performance or decrease energy consumption!
  - Performance: reduce maximum congestion
  - Energy: reduce average dilation
- The general problem is NP-complete (ND17)
- Heuristics are known, algorithms for special cases to be discovered!
- Portable research-quality implementation in LibTopoMap [1]

[1]: Hoefler and Snir: Generic Topology Mapping Strategies for Large-scale Parallel Architectures ICS’11

Nonblocking Collective Operations

- E.g., MPI_Ibcast(…, &req); MPI_Wait(&req);
- Simple to understand, some things to note:
  - Requests are normal MPI_Requests, can be mixed
  - Progress is not guaranteed!
  - The init call must return independently of remote procs
  - All buffers (including arrays for vector colls) shall not be modified (or accessed) until the op completes
  - No matching with blocking collectives
  - Collectives must be called in order (as for threading)

Hoefler et al.: Implementation and Performance Analysis of Non-Blocking Collective Operations for MPI, SC07

Why do I care?

- Easy availability (LibNBC and MPICH2)
- Overlapping communication and computation
  - Improved performance (≤2x though)
  - Sometimes tricky, see [1] (will change)
- Decoupling start and synchronization of collectives
  - Enhanced system noise resiliency
- Interesting synchronization semantics when mixed with point-to-point operations!
  - E.g., limited-depth termination detection [2]

[1]: Hoefler, Lumsdaine: Message Progression in Parallel Computing - To Thread or not to Thread?, Cluster 2008
[2]: Hoefler et al.: Scalable Communication Protocols for Dynamic Sparse Data Exchange, PPoPP'10

Neighborhood Collective Operations

- Many applications are written in a BSP-like model (compute, communicate, compute, …)
  - High temporal locality in communication patterns!
- Specify the communication pattern statically
  - "User-defined collective communication"
  - Cf. MPI Datatypes (who's using them?)
- Communication along a virtual topology
  - MPI_Neighbor_allgather() – same buffer to all
  - MPI_Neighbor_alltoall() – personalized send buffer

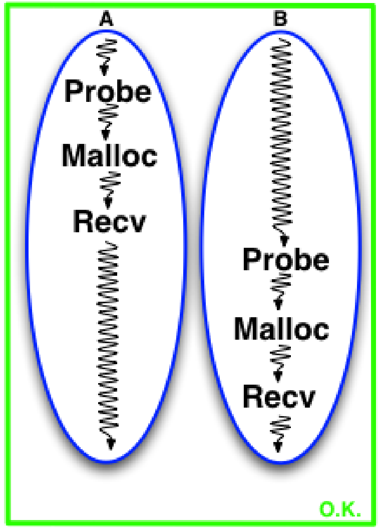
Why do I care?

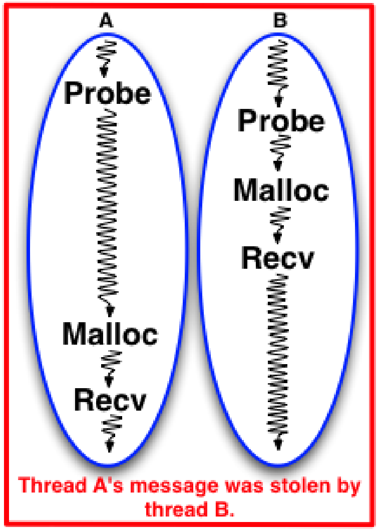
- Simplified programming
  - MPI stores the communication partners for you.
  - Simple intuitive interface (from an MPI perspective)
- Optimization possibilities (in addition to mapping!)
  - Message scheduling
  - Needs additional information (e.g., comm. volumes)
  - Standard leaves options open (MPI_Info)
- Many applications fit this scheme!
  - All stencil codes on Cartesian grids

More info: Hoefler, Traeff: Sparse Collective Operations for MPI, HIPS’09

Matched Probe

- MPI-2.2 point-to-point communication is not thread safe!






```
MPI_Probe(..., status)
size = get_count(status)*size_of(datatype)
buffer = malloc(size)
MPI_Recv(buffer, ...)
```

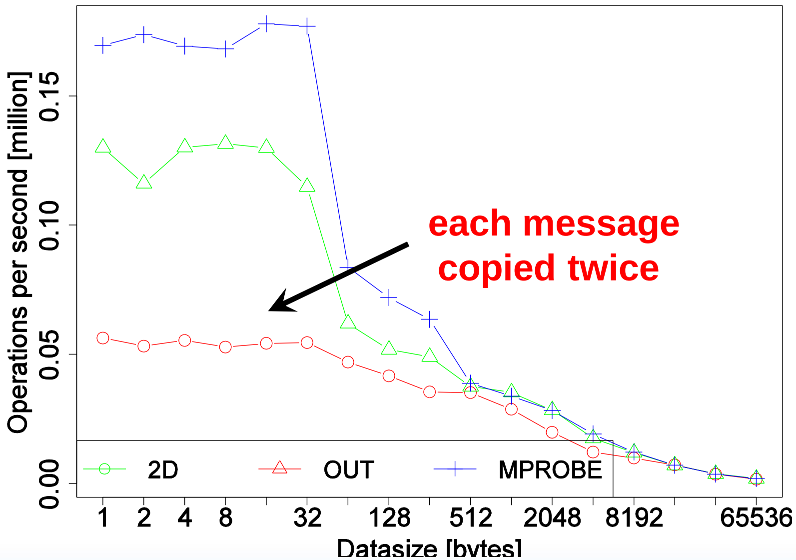
- Easy to fix: return a message handle from probe!
  - Receive this message only through the handle

More info: Hoefler et al.: Efficient MPI Support for Advanced Hybrid Programming Models, EuroMPI'10

Why do I care?

- Did you try writing a threaded MPI library which is called by a threaded code?
  - It's a mess!
  - Mprobe cleans this up (a bit)
- Mprobe is actually faster than user-level hacks
  - And much easier to use

message rate




```
MPI_Mprobe(..., msg, status)
size = get_count(status)*size_of(datatype)
buffer = malloc(size)
MPI_Mrecv(buffer, …, msg, ...)
```

More info: Hoefler et al.: Efficient MPI Support for Advanced Hybrid Programming Models, EuroMPI'10

MPI Tool Interface

- Query (and set) internal MPI variables and counters
  - Variables are not prescribed but queried
  - Control variables (prefix c): behavior
  - Performance variables (prefix p): performance

- Query number of variables MPI_T_cvar_get_num() and a description with MPI_T_cvar_get_info()
  - Returns a string (similar to PAPI native events)
- Read and write variables MPI_T_cvar_read() and MPI_T_cvar_write()

Why do I care?

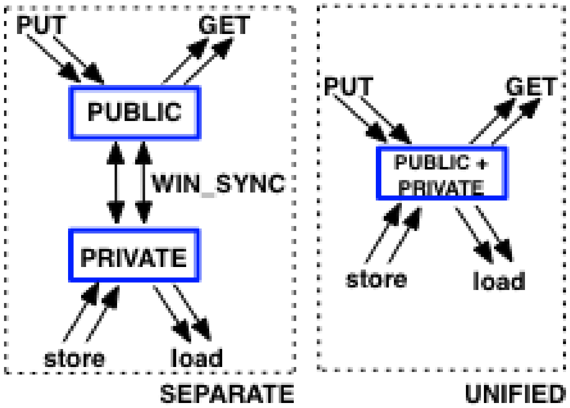
- You probably don't care unless you are a tool developer – or a fine-tuner ☺
- Query (or change) behavior of MPI implementations
  - E.g., eager limit (auto-tuning?)
- Tools (Periscope, Vampir, Scalasca and friends) can query internal counters
  - Recv queue length, blocking time for rendezvous

One Sided – Remote Memory Access

- Probably the most complex change in MPI-3.0
  - Long history
  - First attempt: re-write it from scratch (ICPP'09)
    - Failed (no support for non-cache coherence)
  - Second attempt: extend MPI-2.0
    - MPI-2.0 is very elegant for non-coherent systems
    - Hard to use and slow on coherent systems
  - Also extend for lock-free programming
    - Atomics (CAS, F&A, F&S), no CAS2
    - No locks! (MPI_Lock is not really a lock)

The Memory Models

- MPI defines a window as an exposed memory region with a public and private copy



- MPI_RMA_SEPARATE
  - Like MPI-2.0, windows can have different values!
- MPI_RMA_UNIFIED
  - Cache-coherent → windows cannot differ

New Window Types (I)

- Allocated Windows: MPI_Win_allocate()
  - MPI library allocates memory, collectively
  - Lower address translation overhead
    - Cf. symmetric heap in SHMEM
- Dynamic Windows: MPI_Win_dynamic()
  - No memory by default, can attach memory locally (MPI_Win_attach()/MPI_Win_detach())
  - Cf. memory registration

New Window Types (II)

- MPI_Win_allocate_shared() – collectively allocate shared memory (communicator must allow that!)
  - Fast communication in shared memory (direct access) → be careful, potentially big mess!
  - Allows to reduce memory consumption (share large static structures, e.g., tables)
  - Returns simple memory layout by default, info option to request more complex (but NUMA-aware layout)

MPI RMA Atomics

- Cf. ISA atomics for shared memory
- MPI_Get_accumulate() – MPI look and feel, complex argument set, full datatype support
- MPI_Fetch_and_op() – only for single elements, maps to low-level directives
- MPI_Compart_and_swap() – only single elements, maps to low-level directives

New Completion/Synchronization Semantics

- MPI_Win_flush{_all}() – bulk completes all operations to the specified (all) target(s)
- MPI_Win_flush_local{_all}() – bulk completes all operations to the specified (all) target(s)
- MPI_Win_sync() – synchronize private and public windows
- E.g., MPI_Rget(…, &req) returns a request
  - Completion of the request only indicates local completion! (cf. MPI_Rput())
  - Only valid in passive target epochs

Accumulate Ordering and Memory Semantics

- Conflicting put/get accesses are undefined (not erroneous)
- Conflicting accumulates are defined:
  - No order between different pairs of processes
  - Strict order between the same processes
    - Can be relaxed with info argument! (recommended)
- I wish I had the time to talk about semantics ☺
  - Simple rule (C++0x-like): avoid races, they will lead to undefined outcome on the window

Why do I care?

- It’s amazing! (and amazingly complex) ☺
  - It opens a lot of opportunity
  - Think real PGAS algorithms in MPI
- Shared memory windows offer a portable way to shared memory
  - On-node memory savings
- An interesting base for algorithm research
  - Is PGAS really better than message passing?

New Communicator Creation Functions

- Noncollective communicator creation
  - Allows to create communicators without involving all processes in the parent communicator
  - Very useful for some applications (dynamic sub-grouping) or fault tolerance (dead processes)
- Nonblocking communicator duplication
  - MPI_Comm_idup(…, req) – like it sounds
  - Similar semantics to nonblocking collectives
  - Enables the implementation of nonblocking libraries

J. Dinan et al.: Noncollective Communicator Creation in MPI, EuroMPI’11
T. Hoefler: Writing Parallel Libraries with MPI - Common Practice, Issues, and Extensions, Keynote, IMUDI’11

Language Bindings

- Enhanced Fortran Language bindings:
  - Comply with Fortran standard (void * type)
  - Type safety (type-safe handles, not all integers)
  - Enable correct asynchrony (disallow temp copies, code movement etc.)
  - F08 interface to C
- Deprecated C++ bindings
  - Make C++ optional
  - Remove the deprecated bindings (any users?)

Fault Tolerance and Resiliency

- Focus on user-level failure notification
  - No magic at all – enables ABFT
  - Requires robust MPI library
- Management through communicators
  - comm_invalidate, comm_shrink, comm_failure_ack
- Still somewhat in flux
  - Very hard to define and little existing practice

The Future

- Tickets for MPI-<next> plannes:
  - Scalable vector collectives
  - Request completion callbacks
  - Timed requests (complete after timeout)
  - New communicator creation routines (hierarchical)
  - …
- Many cleanups (including errata items)
- No timeline yet

Summary and Questions?

- MPI-3.0 is coming quickly!
- Use-cases are being defined
- For more details and training:

June 17th ISC'12 Tutorial
Hoefler and Schulz: "Next Generation MPI Programming: Advanced MPI-2 and New Features in MPI-3"

- And I will be available for questions today ☺