MPI for the future?

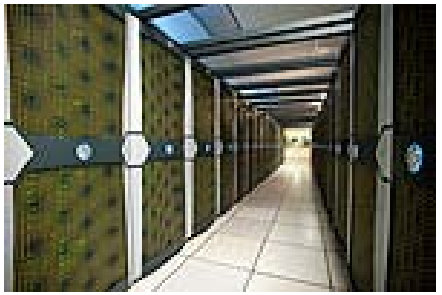
Jesper Larsson Träff
Department of Scientific Computing
University of Vienna

Outline

1. Trends in HPC Architectures
2. Scalability and other challenges to MPI
3. News in MPI 2.2 (September 2009)
4. MPI 2.x (?)
5. MPI 3.0 Activities
6. Summary/conclusion

1. Trends in HPC architectures

A look at the Top500 list (www.top500.org) – November 2009 – shows:



- many different systems
- similar general trends

MPI is (solely?) used to program these systems:

Single applications with non-trivial communication requirements

#	Site	Vendor	Peak (PF)	Cores	Interconnect	Architecture
1	ORNL	Cray XT5	2.33	224162	3d Torus	6way SMP
2	LANL	IBM RR	1.38	122400	IB 3240	Hybrid Cell+AMD
3	Tenn.	Cray XT5	1.03	96928	3d Torus	6way SMP
4	FZJ	IBM BG/P	1.00	234921	3d Torus	4way SMP
5	NUDT (China)	Intel+ATI	1.21	71680	IB (?)	Hybrid, Intel+GPU
6	NASA	SGI Altix	0.67	56320		4-8way SMP
7	LLNL	IBM BG/L	0.60	212992	3d Torus	2way-SMP
8	ANL	IBM BG/P	0.56	163840	3d Torus	4way SMP
9	Texas	Sun AMD	0.58	62976	IB 3936	16way SMP
10	Sandia	Sun Intel	0.49	41616	IB 2601	

#	Site	Vendor	Peak (PF)	Cores	Interconnect	Architecture
1	ORNL	Cray XT5	2.33	224162	3d Torus	6way SMP
2	LANL	IBM RR	1.38	122400	IB 3240	Hybrid Cell+AMD
3	Tenn.	Cray XT5	1.03	96928	3d Torus	6way SMP
4	FZJ	IBM BG/P	1.00	234921	3d Torus	4way SMP
5	NUDT (China)	Intel+ATI	1.21	71680	IB (?)	Hybrid, Intel+GPU
6	NASA	SGI Altix	0.67	56320		4-8way SMP
7	LLNL	IBM BG/L	0.60	212992	3d Torus	2way-SMP
8	ANL	IBM BG/P	0.56	163840	3d Torus	4way SMP
9	Texas	Sun AMD	0.58	62976	IB 3936	16way SMP
10	Sandia	Sun Intel	0.49	41616	IB 2601	

#	Site	Vendor	Peak (PF)	Cores	Interconnect	Architecture
1	ORNL	Cray XT5	2.33	224162	3d Torus	6way SMP
2	LANL	IBM RR	1.38	122400	IB 3240	Hybrid Cell+AMD
3	Tenn.	Cray XT5	1.03	96928	3d Torus	6way SMP
4	FZJ	IBM BG/P	1.00	234921	3d Torus	4way SMP
5	NUDT (China)	Intel+ATI	1.21	71680	IB (?)	Hybrid, Intel+GPU
6	NASA	SGI Altix	0.67	56320		4-8way SMP
7	LLNL	IBM BG/L	0.60	212992	3d Torus	2way-SMP
8	ANL	IBM BG/P	0.56	163840	3d Torus	4way SMP
9	Texas	Sun AMD	0.58	62976	IB 3936	16way SMP
10	Sandia	Sun Intel	0.49	41616	IB 2601	

#	Site	Vendor	Peak (PF)	Cores	Interconnect	Architecture
1	ORNL	Cray XT5	2.33	224162	3d Torus	6way SMP
2	LANL	IBM RR	1.38	122400	IB 3240	Hybrid Cell+AMD
3	Tenn.	Cray XT5	1.03	96928	3d Torus	6way SMP
4	FZJ	IBM BG/P	1.00	234921	3d Torus	4way SMP
5	NUDT (China)	Intel+ATI	1.21	71680	IB (?)	Hybrid, Intel+GPU
6	NASA	SGI Altix	0.67	56320		4-8way SMP
7	LLNL	IBM BG/L	0.60	212992	3d Torus	2way-SMP
8	ANL	IBM BG/P	0.56	163840	3d Torus	4way SMP
9	Texas	Sun AMD	0.58	62976	IB 3936	16way SMP
10	Sandia	Sun Intel	0.49	41616	IB 2601	

#	Site	Vendor	Peak (PF)	Cores	Interconnect	Architecture
1	ORNL	Cray XT5	2.33	224162	3d Torus	6way SMP
2	LANL	IBM RR	1.38	122400	IB 3240	Hybrid Cell+AMD
3	Tenn.	Cray XT5	1.03	96928	3d Torus	6way SMP
4	FZJ	IBM BG/P	1.00	234921	3d Torus	4way SMP
5	NUDT (China)	Intel+ATI	1.21	71680	IB (?)	Hybrid, Intel+GPU
6	NASA	SGI Altix	0.67	56320		4-8way SMP
7	LLNL	IBM BG/L	0.60	212992	3d Torus	2way-SMP
8	ANL	IBM BG/P	0.56	163840	3d Torus	4way SMP
9	Texas	Sun AMD	0.58	62976	IB 3936	16way SMP
10	Sandia	Sun Intel	0.49	41616	IB 2601	

#	Site	Vendor	Peak (PF)	Cores	Interconnect	Architecture
1	ORNL	Cray XT5	2.33	224162	3d Torus	6way SMP
2	LANL	IBM RR	1.38	122400	IB 3240	Hybrid Cell+AMD
3	Tenn.	Cray XT5	1.03	96928	3d Torus	6way SMP
4	FZJ	IBM BG/P	1.00	234921	3d Torus	4way SMP
5	NUDT (China)	Intel+ATI	1.21	71680	IB (?)	Hybrid, Intel+GPU
6	NASA	SGI Altix	0.67	56320		4-8way SMP
7	LLNL	IBM BG/L	0.60	212992	3d Torus	2way-SMP
8	ANL	IBM BG/P	0.56	163840	3d Torus	4way SMP
9	Texas	Sun AMD	0.58	62976	IB 3936	16way SMP
10	Sandia	Sun Intel	0.49	41616	IB 2601	

Summary of trends

- Peak performance into PFLOPS range
- Well beyond 100K cores

  2012: IBM/LLNL Sequoia 20PF 1.6M cores; Japanese/Fujitsu 10PF 600K cores

- Many nodes:

  Weak interconnects (large diameter, low bisection BW)

- Few Nodes

  Hierarchical systems ("fat"/hybrid processing nodes), nodes competing for BW

- Hybrid nodes: Hybrid processors (Cell), accelerators (GPU)S

European systems on #4, #13, …

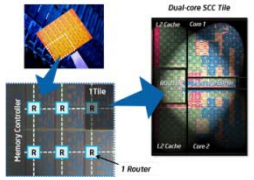
New players:
Chinese system #5, Russia #12, South Korea #14, Saudi Arabia #18, Japanese?

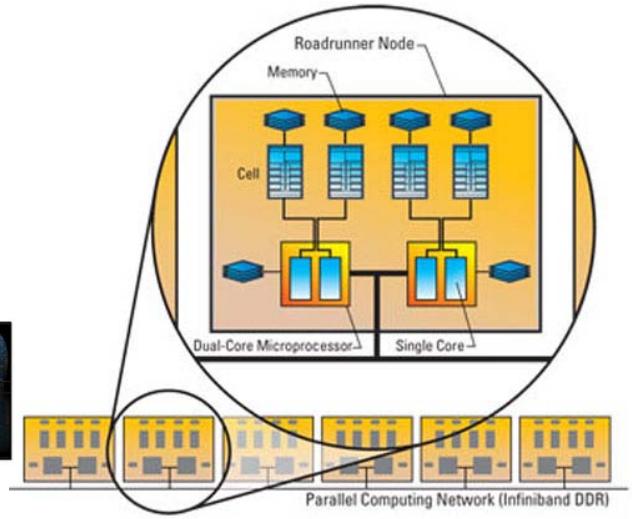
Typical, current hybrid system:

- Scalar 2 2-core AMD + 2 (1+8) core Cell
- 3240 nodes competing for IB BW
- Energy efficient:
  #4 on Green500

Cell SPU: small local memory, no HW coherency, simpler in-order processor, SIMD/vector capabilities

Difficult to program!?



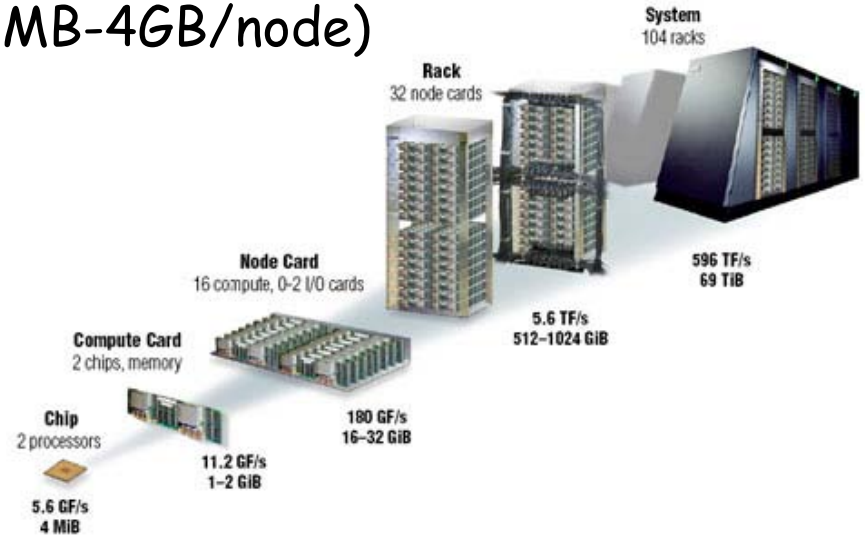


Cell processor (1+8): ~300GFLOPS

The BG systems

- "slow" processors, 700-800MHz
- Simpler processors, limited out-of-order, branch-prediction
- BG/L: 2-core, not cache-coherent
- BG/P: 4-core, cache-coherent
- Very memory constrained (512MB-4GB/node)

Energy efficient, heavily present on Green500



Possible new trend (not evident from Top500)

- Special purpose processors for communication (off-loading)
- Not only simple message passing (of consecutive data), also collective communication

Old idea (Myrinet and others), reemerging

For the applications developer/library implementer:

- Memory is not scaling linearly with cores, no global state per core
- Non-coherent memory systems, complex memory systems
- Limited bisection bandwidth
- Less powerful scalar processors
- Special purpose processors

Is MPI (or any other HPC interface) ready for this?

Can MPI (or other HPC interface) help application programmer?

2. Scalability and other challenges to MPI

MPI

Communication models

- Point-to-point: MPI_Send → MPI_Recv
- Collective: MPI_Bcast, MPI_Bcast, MPI_Bcast
- (One-sided): MPI_Put →

Communicators, datatypes, process topologies

Parallel I/O, dynamic process management

Challenge:
Will this interface scale? Can it be implemented scalably?

Definition:

An MPI construct is non-scalable, if memory or time overhead(*) is $\Omega(p)$, p number of processes (in communicator)

(*)cannot be accounted for in application

Questions:

Are there aspects of the MPI specification that are non-scalable (forces $\Omega(p)$ memory or time)?

Are there aspects of (typical) MPI implementations that are non-scalable? Which improvements are possible?

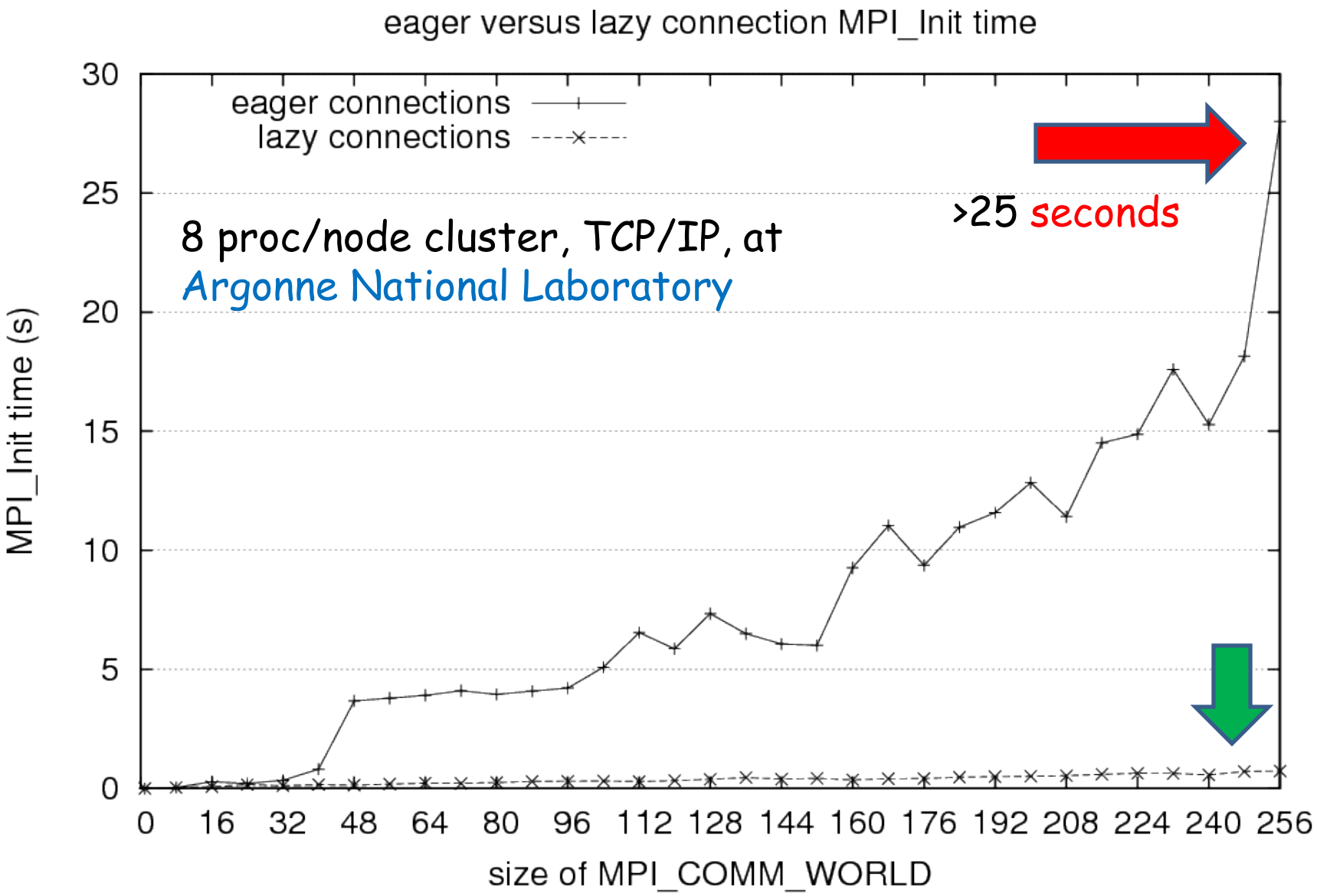
Example:

MPI_Init(argc,argv)

starts an initial set of processes, creates MPI_COMM_WORLD(*)
Non-scalable if space or time would depend linearly (or worse) on the number of processes to be started…

Unfortunately…

(*)MPI_COMM_WORLD: communication domain containing all externally started processors


eager versus lazy connection MPI_Init time
eager connections
lazy connections
8 proc/node cluster, TCP/IP, at
Argonne National Laboratory
>25 seconds
MPI_Init time (s)
size of MPI_COMM_WORLD
0
5
10
15
20
25
30
0
16
32
48
64
80
96
112
128
144
160
176
192
208
224
240
256

Problem:

Eager connection setup (TCP/IP, IB, …)

Remedy:

-Lazy connection management, set up connections when needed

-Requires collective algorithms with few communication neighbors!

-May give performance surprises

MPI_Comm_dup(comm,&newcomm)

Create duplicate communicator (communication context), with same process order (process to processor mapping)

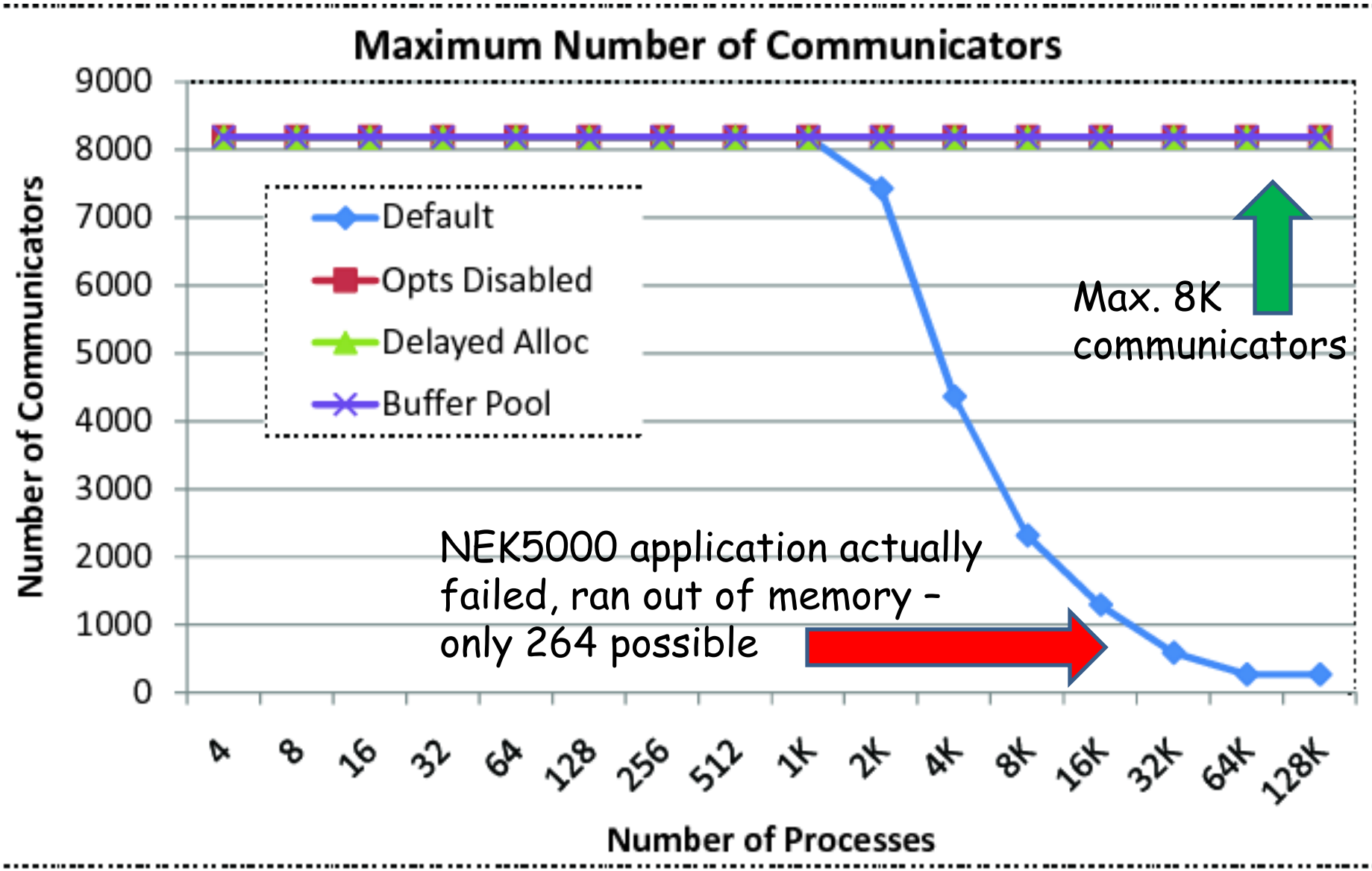
Essential for isolating communication, building safe parallel libraries.

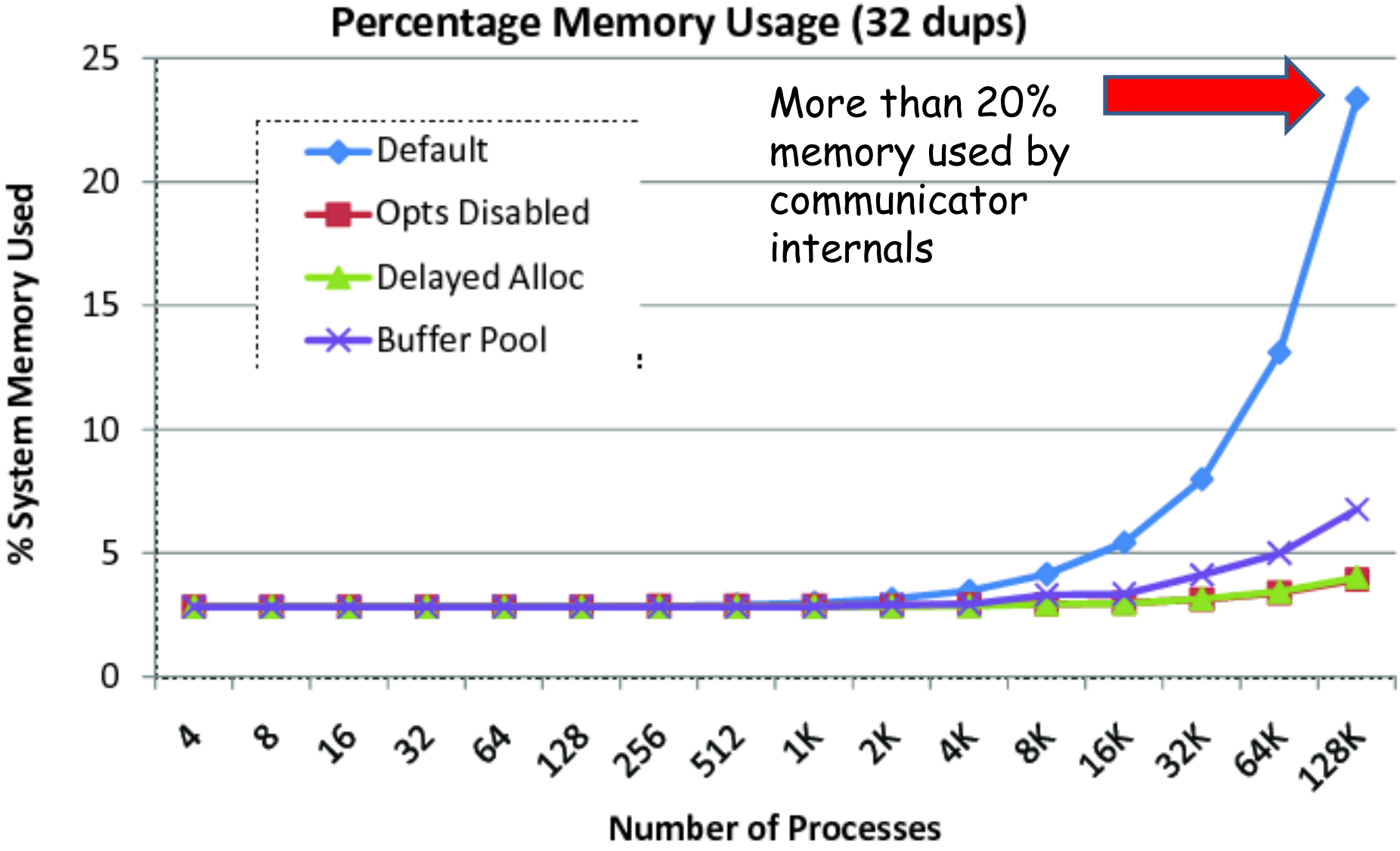
Is this important construct scalable?
Is memory consumption o(p), p number of processes in comm?

Experiment 1:
How many MPI_Comm_dups are possible on BG/P?

Experiment 2:
Memory consumption on BG/P after 32 MPI_Comm_dup's?

Setup:
IBM/MPICH MPI at Argonne National Lab, using 128K processors of #8 BG/P system


Maximum Number of Communicators
Number of Communicators
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
Default
Opts Disabled
Delayed Alloc
Buffer Pool
Max. 8K communicators
NEK5000 application actually failed, ran out of memory – only 264 possible
4
8
16
32
64
128
256
512
1K
2K
4K
8K
16K
32K
64K
128K
Number of Processes


Percentage Memory Usage (32 dups)
% System Memory Used
25
20
15
10
5
0
Default
Opts Disabled
Delayed Alloc
Buffer Pool
More than 20% memory used by communicator internals
4
8
16
32
64
128
256
512
1K
2K
4K
8K
16K
32K
64K
128K
Number of Processes

Problem:
Internal, precomputed tables (p-sized arrays) for optimizing collective performance (BG/P: in the presence of threads)

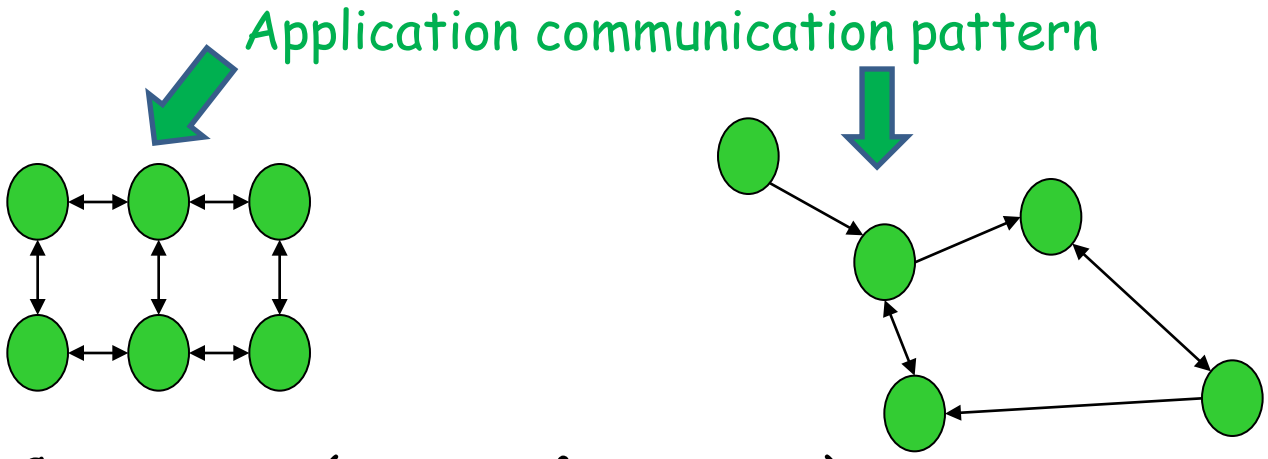
Remedy:
- Get rid of tables… (Bad)
- Better management of tables

General observation:
MPI implementations often used p-sized precomputed information (rank->processor mappings, permutations), per communicator

Need:
Better management (reuse, sharing, lazy computation)
More space-efficient data structures

Process topologies:

communicators adapted to communication pattern and architecture



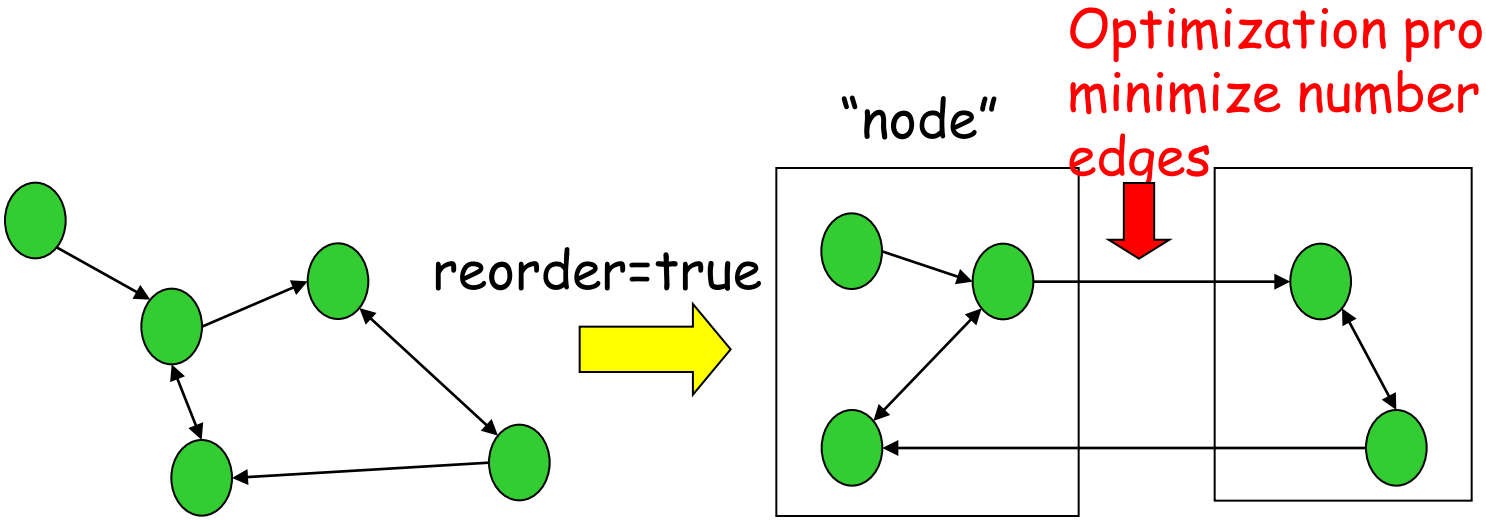
MPI_Cart_create(comm,…,&cartcomm)

MPI_Graph_create(comm,nodes,index,edges,reorder,&graphcomm)

Communication graph

Process topologies:

communicators adapted to communication pattern and architecture

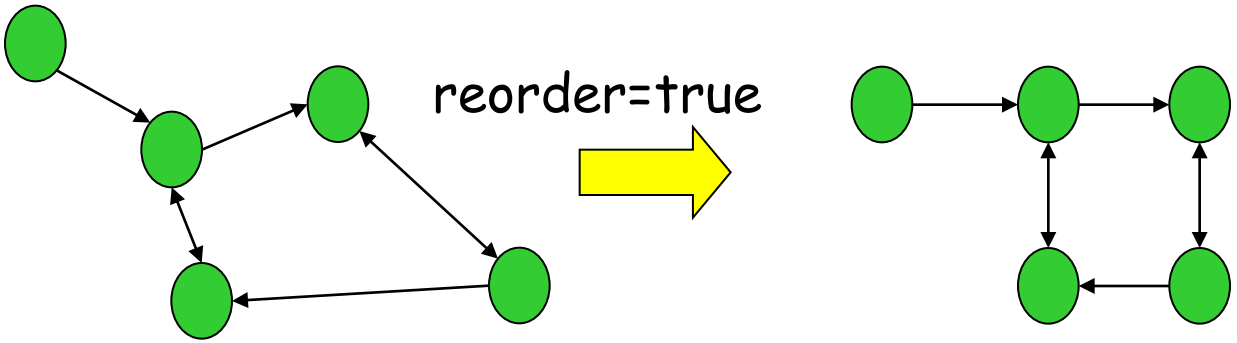


MPI_Graph_create(comm,nodes,index,edges,reorder,&graphcomm)

For hierarchical (e.g. SMP) systems:
map processes with heavy communication to "nodes" with stronger communication capabilities [Träff, Supercomputing 2002]

Process topologies:

communicators adapted to communication pattern and architecture

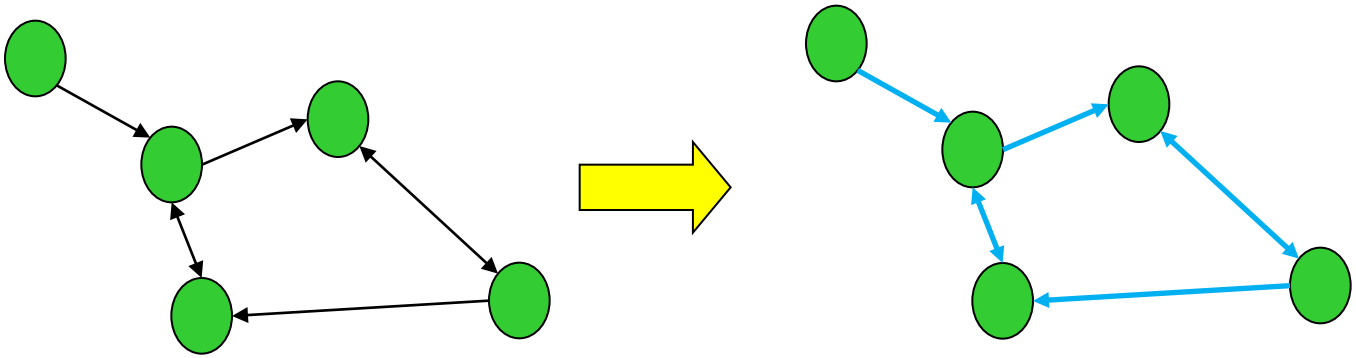


MPI_Graph_create(comm,nodes,index,edges,reorder,&graphcomm)

For specific networks (MPI_Cart_create):
embed communication graph into network, e.g. BG/P torus
[Yu,Chung,Moreira, Supercomputing 06]

Process topologies:

communicators adapted to communication pattern and architecture



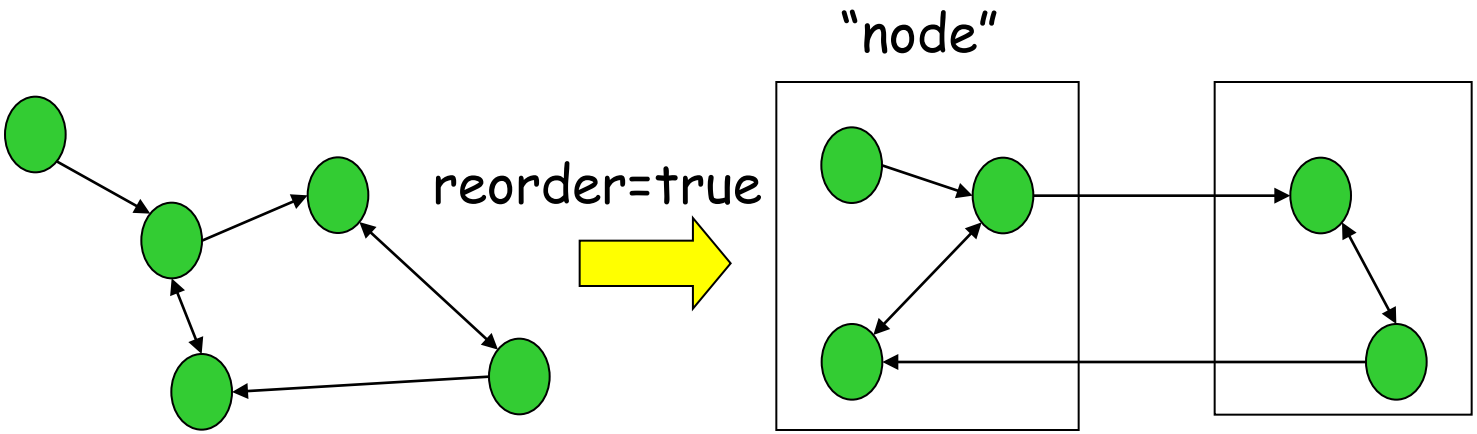
MPI_Graph_create(comm,nodes,index,edges,reorder,&graphcomm)

For connection-based networks:
Use communication graph to set up connections in advance

Instead of/addition to lazy connection management

Process topologies:

communicators adapted to communication pattern and architecture



MPI_Graph_create(comm,nodes,index,edges,reorder,&graphcomm)

Communication graph, must be provided by all processes!

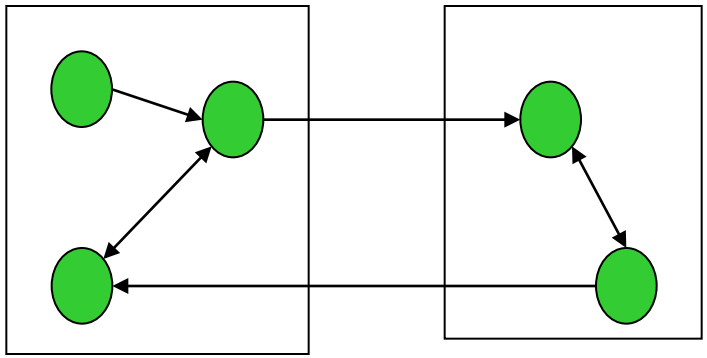
Process topologies:

communicators adapted to communication pattern and architecture

Space $\Omega(p+e)$ per process

Extremely non-scalable

Other problems:
User-unfriendly, lack of information, centralized algorithm, …



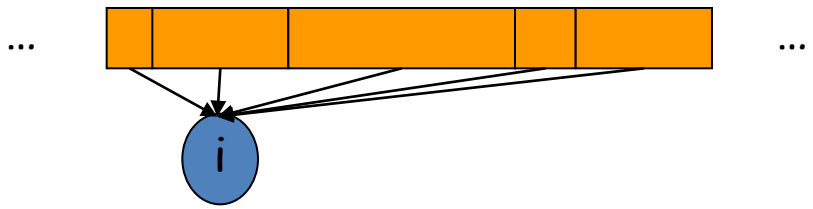
MPI_Graph_create(comm,nodes,index,edges,reorder,&graphcomm)

Communication graph, must be provided by all processes!

Collective communication:

Sparse usage of vector-collectives is not scalable

```
MPI_Alltoallv(sendbuf,sendcount[],senddisp[],…,
              recvbuf,recvcount[],recvdisp[],…,comm)
```



Proc i receives data from some neighboring ranks, rest is 0

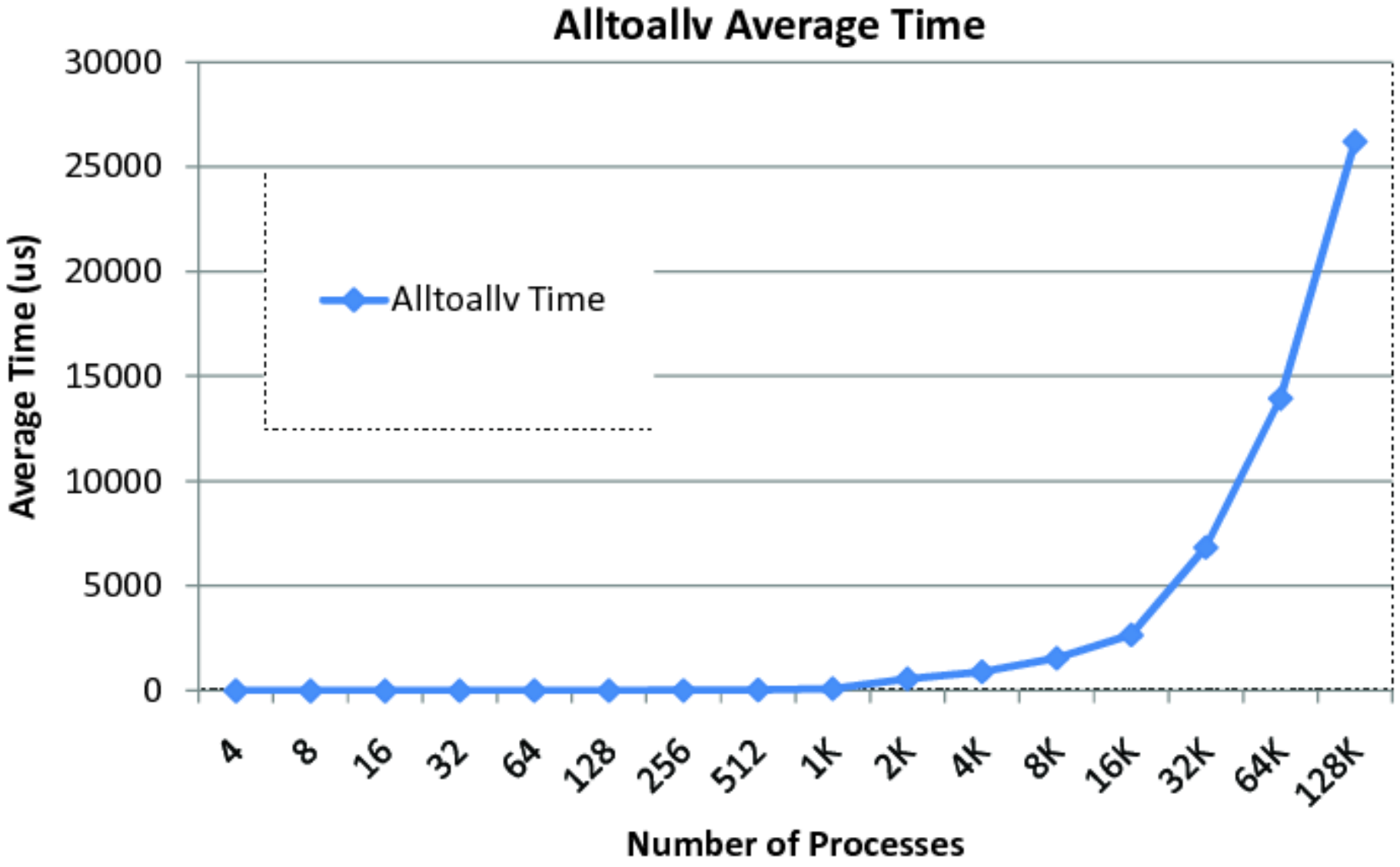
Ω(p) user-space for count, disp arrays, 4 arrays for MPI_Alltoallv, 6 arrays for MPI_Alltoallw!!

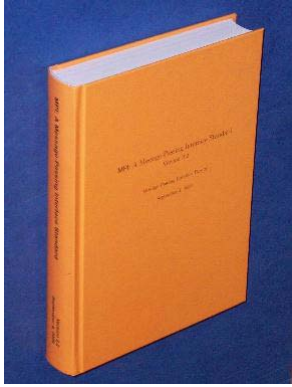
Experiment:

Degenerate case, MPI_Alltoallv with counts[i]=0 for all i

Setup:

IBM/MPICH MPI at Argonne National Lab, using 128K processors of #8 BG/P system


Alltoallv Average Time
30000
25000
20000
15000
10000
5000
0
Average Time (us)
Alltoallv Time
4
8
16
32
64
128
256
512
1K
2K
4K
8K
16K
32K
64K
128K
Number of Processes



3. News in MPI 2.2 (September 2009)

Scalable, informative graph process topology interface

MPI_Reduce_local – process local reduction function using MPI_Op, MPI_Datatype, helps building libraries with reduction-like functionalities

MPI_Reduce_scatter_block – the missing, "regular" version, all processes receive result of same size, more scalable

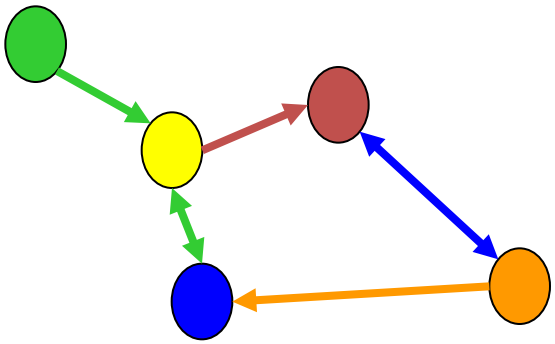
MPI_Comm_create – creates set of disjoint communicators

MPI_IN_PLACE in MPI_Alltoall, MPI_Alltoallv, MPI_Alltoallw, MPI_Exscan – good implementations will not allocate (much) extra memory

C++ bindings deprecated! – and may disappear with MPI 3.0

(Scalable) distributed graph topologies

MPI_Dist_graph_create(comm,
n,sources,degrees,destinations,weights,
info,reorder,&graphcomm)

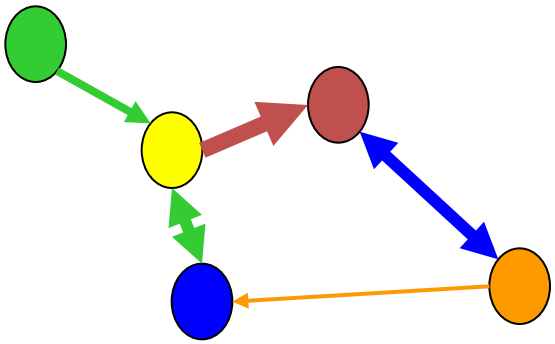


Each process contributes only edges it "knows":

n sources, for each a degree, and the corresponding destinations

(Scalable) distributed, informative graph topologies

MPI_Dist_graph_create(comm,
n,sources,degrees,destinations,weights,
info,reorder,&graphcomm)



Edges are weighted to reflect communication volume/frequency, info to control optimization process

Scalable in space: $\Omega(d(i))$ space for process i

Note:
Remapping problems are typically NP-complete – exact solution may be non-scalable (in time).

Remedies:
- Heuristics
- Use info to set limit on resources
- …

Need:
Good, easy to implement, parallel (distributed) algorithms (heuristics) for graph partitioning, graph embedding, … (see Scotch, ParMETIS, …)

4. MPI 2.x non-blocking and other collectives

Non-blocking communication: introduces slackness, has semantic advantages (deadlock avoidance!), permits overlap of communication and computation

Up to MPI 2.2: only point-to-point, one-sided, split collective MPI-I/O operations

```
MPI_Isend
MPI_Irecv
<Compute>
MPI_Wait
MPI_Wait
```

```
MPI_Isend
MPI_Irecv
<Compute>
MPI_Wait
MPI_Wait
```

Full proposal for "classical" data exchange and reduction collectives have been voted in for next MPI

1. MPI_Ibarrier(…,request)
2. MPI_Ibcast
3. MPI_Igather, MPI_Igatherv
4. MPI_Iscatter, MPI_Iscatterv
5. MPI_Iallgather, MPI_Iallgatherv
6. MPI_Ialltoall, MPI_Ialltoallv, MPI_Ialltoallw
7. MPI_Ireduce, MPI_Iallreduce, MPI_Ireduce_scatter, MPI_Ireduce_scatter_block
8. MPI_Iscan, MPI_Iexscan

Semantics naturally follows/extends point-to-point:

MPI_Wait{some,any}(request)

Discussion (for MPI 3.0):
Non-blocking versions of other functions (MPI_Comm_dup, MPI_Comm_create, MPI_Dist_graph_create, …) with collective semantics

Need:
Efficient algorithms that realize the semantic advantages

More powerful communication network support would help!

[HoeflerSiebert, ICPP 2009]

Orthogonal implementation issue: collectives that are more robust wrt. process arrival patterns

[PatarasukYuan, IPDPS 2009]

5. MPI 3.0 activities

- MPI is "maintained" by MPI Forum (www.mpi-forum.org) , active again since late 2007
- Regular meetings every 6-8 weeks, US, sometimes Europe (EuroMPI conference)
- OPEN: Consistently participating organizations can vote
- About 40 participants, representing ~30 organizations
- Vendors, MPI developers (mostly, sadly)
- MPI 2.1 June 2008, MPI 2.2 September 2009

- Avoid MPI 2.0 lack-of-experience standardization mistake!

Collectives

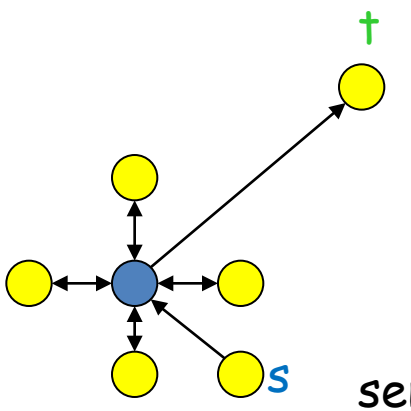
- Non-blocking collectives – also for other MPI functions will collective semantics
- Make MPI-IO non-blocking
- Sparse collectives – semantic advantages, scalability advantages (no p-sized arrays of 0'es)

Note: MPI Forum decided on explicit, non-blocking collectives instead of relying on multiple threads

Sparse collectives

[Hoefler, Träff, IPDPS-HIPS'09]

MPI_Neighbor_gather(sendbuf,…,recvbuf,recvcount,…,comm);



Calling processes receives (different) data from a set of source neighbors s

Calling process sends same data to a set of target neighbors t

sendbuf:

recvbuf:

s_0	s_1	s_2	s_3	s_4

Neighborhoods for sparse collectives specified by either

1. Process topology, or
2. Special (collective) functions

Full proposal (either version) pending, under discussion by MPI Forum

Support for hybrid programming

"million cores" doesn't necessarily mean "million MPI processes":

> On hierarchical systems (powerful SMP or heterogeneous nodes) node-internal communication/parallelization may be done by other means than MPI

Many cores/threads may live within the same MPI process

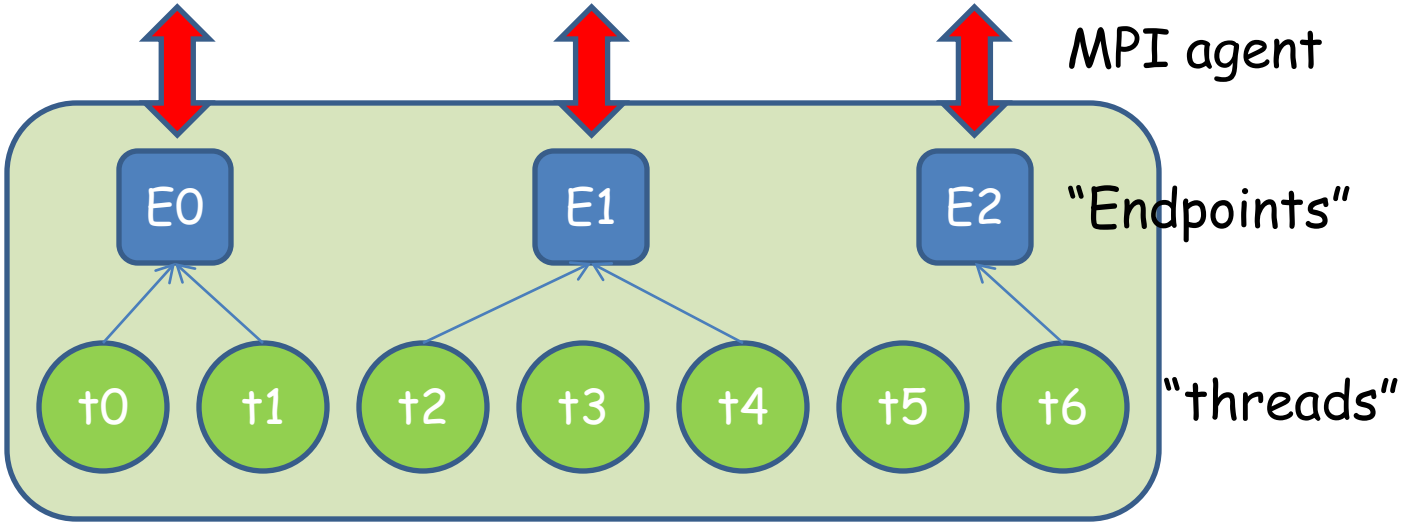
Current solution:
MPI provides a level of thread support (can be queried):
MPI_THREAD_SINGLE, MPI_THREAD_FUNNELED,
MPI_THREAD_SERIALIZED, MPI_THREAD_MULTIPLE

MPI can live with OpenMP, pthreads, …

Proposal for cleaner interaction with "threads": MPI_Endpoints

- MPI process (now: MPI agent) can have multiple "Endpoints", light-weight ranks
- Each "thread" can be associated with at most one endpoint (dynamically)
- Endpoints have ranks, successive in MPI_COMM_ENDPOINTS
- MPI_COMM_WORLD: first endpoint of each agent
- MPI_COM_PROCESS: all endpoints within an agent

Better suited to and additional support for PGAS languages



- Threads associated with endpoints can MPI communicate
- Threads do not have to be associated with an endpoint
- No races between threads associated with different endpoints
- Solves problems with thread-safety of some MPI functions (MPI_Iprobe)

One-sided communication

Important part of MPI (with MPI 2.0) that is not used much…

Portable RMA-like abstraction, semantic advantages, separation of communication and synchronization

Criticisms:

- Not consistently supported by implementations
- Not giving expected/perceived performance
- Too complicated, rigid, restrictive
- Cannot support libraries and upcoming paradigms (PGAS)

Well…

Debate:
Fix existing interface – or (deprecate and) introduce new one?

Partially new interface

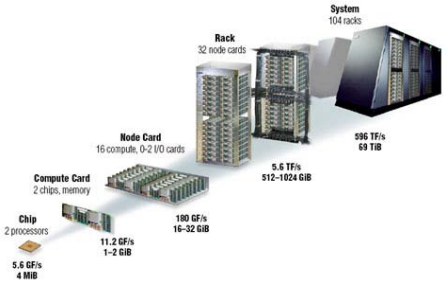
[Tipparaju et.al, ICPP 2009]

Gist:
"get rid of communication windows", "get rid of epochs", delegate synchronization/completion mechanisms to individual RMA calls (and a completion call)

Improving existing one-sided interface:

- Relieve restrictions, overlapping MPI_Put's should be undefined, not erroneous
- More "atomic", test-and-set-like operations:
  MPI_Get_accumulate, MPI_Accumulate_get
- Local completion operations inside epoch (enforce partial order)
- More powerful, less restrictive MPI_Win_lock/unlock
- More flexible, scalable communication windows
- Symmetric (collective) memory allocation

Fault tolerance



Is it an issue? …

How much can be handled at system level? What should be handled by MPI? Transparently, or with user participation? What are the costs? What cannot be handled at all?

[Cappello et al., MPICH-V, 2006]

Error behavior of MPI must be governed (current: no requirements, an MPI error typically aborts application)

Aim:
to provide support for portable, fault tolerant application programming (non-transpararent)

- Per-communicator attributes on restoration (which ranks,conditions, …)
- Validating error-free state of communicator, MPI_Comm_validate
- Restoration calls, MPI_Comm_restore
- Recovery actions by call-back functions

(building on FT-MPI and others)

Note:
Parts of current proposal are non-scalable (arrays of ranks)

Tool support

Provide better support for MPI external performance, correctness, debugging, … tools

- Provision for multiple profiling interfaces (PMPI)
- Controlled exposure of MPI certain internals

Other issues

- FORTRAN interface
- Application Binary Interface (ABI)
- Backwards compatibility(!) – ensuring consistency and coherence of standard!!
- Active messages
- Subsetting
- …

Consult: www.mpi-forum.org

6. Summary/conclusion

- Fact of life:

  MPI will stay as an/the important interface for HPC for the next decade (MPI 2.2, MPI 3.0)!

- Scaling MPI to millions of processors/cores is an issue!
- Challenges to both implementation and specification of MPI
- Implementations: new parallel algorithms, space efficient data structures, algorithms with locality and few neighbors, dedicated algorithms for interconnects
- Specification: eliminate non-scalable constructs, better support for hybrid programming, fault-tolerant programming, tools, …

Talk partly based on

- "MPI: A message-passing interface standard. Version 2.2"
- "MPI on a million cores", Balaji, Buntinas, Goodell, Gropp, Hoefler, Kumar, Lusk, Thakur, Träff, submitted 2010
- "The scalable process topology interface of MPI 2.2", Hoefler, Rabenseifner, Ritzdorf, de Supinski, Thakur, Träff, submitted 2010
- "Sparse collective operations for MPI", Hoefler, Träff, IPDPS-HIPS 2009
- MPI Forum discussions and documents, see www.mpi-forum.org
- Discussions with other MPI developers and users

All are thanked!