The Evolution of MPI

William Gropp

Computer Science

www.cs.uiuc.edu/homes/wgropp

Outline

1. Why an MPI talk?
2. MPI Status: Performance, Scalability, and Functionality
3. Changes to MPI: MPI Forum activites
4. What this (should) mean for you

Why an MPI Talk?

- MPI is the common base for tools
- MPI as the application programming model
- MPI is workable at petascale, though starting to face limits. At exascale, probably a different matter
- One successful way to handle scaling and complexity is to break the problem into smaller parts
  - At Petascale and above, one solution strategy is to combine programming models

Review of Some MPI Features and Issues

- RMA
  - Also called “one-sided”, these provide put/get/accumulate
  - Some published results suggest that these perform poorly
  - Are these problems with the MPI implementation or the MPI standard (or both)?
  - How should the performance be measured?
- MPI-1
  - Point-to-point operations and process layout (topologies)
    - How important is the choice of mode? Topology?
  - Algorithms for the more general collective operations
    - Can these be simple extensions of the less general algorithms?
- Thread Safety
  - With multicore/manycore, the fad of the moment
  - What is the cost of thread safety in typical application uses?
- I/O
  - MPI I/O includes nonblocking I/O
  - MPI (the standard) provided a way to layer the I/O implementation, using “generalized requests”. Did it work?

Some Weaknesses in MPI

- Easy to write code that performs and scales poorly
  - Using blocking sends and receives
    - The attractiveness of the blocking model suggests a mismatch between the user's model and the MPI model of parallel computing
  - The right fix for this is better performance tuning tools
    - Don't change MPI, improve the environment
    - The same problem exists for C, Fortran, etc.
    - One possibility - model checking against *performance* assertions
- No easy compile-time optimizations
  - Only MPI_Wtime, MPI_Wtick, and the handler conversion functions may be macros.
  - Sophisticated analysis allows inlining
  - Does it make sense to optimize for important special cases
    - Short messages? Contiguous messages? Are there lessons from the optimizations used in MPI implementations?

Issues that are not issues (1)

- MPI and RDMA networks and programming models
  - MPI can make good use of RDMA networks
  - Comparisons with MPI sometimes compare apples and oranges
    - How do you signal completion at the target?
    - Cray SHMEM succeeded because of SHMEM_Barrier - an easy and efficiently implemented (with special hardware) way to indicate completion of RDMA operations
- Latency
  - Users often confuse Memory access times and CPU times; expect to see remote memory access times on the order of register access
  - Without overlapped access, a single memory reference is 100's to 1000's of cycles
  - A load-store model for reasoning about program performance isn't enough
    - Don't forget memory consistency issues

Issues that are not issues (2)

- MPI "Buffers" as a scalability limit
  - This is an implementation issue that existing MPI implementations for large scale systems already address
    - Buffers do not need to be preallocated
- Fault Tolerance (as an MPI problem)
  - Fault Tolerance is a property of the application; there is no magic solution
  - MPI implementations can support fault tolerance
    - RADICMPI is a nice example that includes fault recovery
  - MPI intended implementations to continue through faults when possible
    - That's why there is a sophisticated error reporting mechanism
    - What is needed is a higher standard of MPI implementation, not a change to the MPI standard
  - But - Some algorithms do need a more convenient way to manage a collection of processes that may change dynamically
    - This is not a communicator

Scalability Issues in the MPI Definition

- How should you define scalable?
  - Independent of number of processes
- Some routines do not have scalable arguments
  - E.g., MPI_Graph_create
- Some routines require O(p) arrays
  - E.g., MPI_Group_incl, MPI_Alltoall
- Group construction is explicit (no MPI_Group_split)
- Implementation challenges
  - MPI_Win definition, if you wish to use a remote memory operation by address, requires each process to have the address of each remote processes local memory window (O(p) data at each process).
  - Various ways to recover scalability, but only at additional overhead and complexity
    - Some parallel approaches require “symmetric allocation”
    - Many require Single Program Multiple Data (SPMD)
  - Representations of Communicators other than MPI_COMM_WORLD (may be represented implicitly on highly scalable systems)
    - Must not enumerate members, even internally

Performance Issues

- Library interface introduces overhead
  - ~200 instructions ?
- Hard (though not impossible) to "short cut" the MPI implementation for common cases
  - Many arguments to MPI routines
  - These are due to the attempt to limit the number of basic routines
    - You can't win --- either you have many routines (too complicated) or too few (too inefficient)
    - Is MPI for users? Library developers? Compiler writers?
- Computer hardware has changed since MPI was designed (1992 - e.g., DEC announces Alpha)
  - SMPs are more common
  - Cache-coherence (within a node) almost universal
    - MPI RMA Epochs provided (in part) to support non-coherent memory
    - May become important again - fastest single chips are *not* cache coherent
  - Interconnect networks support "0-copy" operations
  - CPU/Memory/Interconnect speed ratios
  - Note that MPI is often blamed for the poor fraction of peak performance achieved by parallel programs. (But the real culprit is often per-node memory performance)

Performance Issues (2)

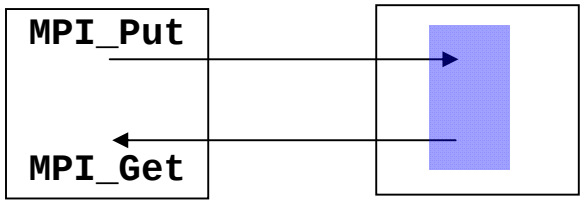
- MPI-2 RMA design supports non-cache-coherent systems
  - Good for portability to systems of the time
  - Complex rules for memory model (confuses users)
    - But note that the rules are precise and the same on all platforms
  - Performance consequences
    - Memory synchronization model
    - One example: Put requires an ack from the target process
- Missing operations
  - No Read-Modify-Write operations
  - Very difficult to implement even fetch-and-increment
    - Requires indexed datatypes to get scalable performance(!)
    - We've found bugs in vendor MPI RMA implementations when testing this algorithm
  - Challenge for any programming model
    - What operations are provided?
    - Are there building blocks, akin to the load-link/store-conditional approach to processor atomic operations?
- How fast is a good MPI RMA implementation?

MPI RMA and Proccess Topologies

- To properly evaluate RMA, particularly with respect to point-to-point communication, it is necessary to separate data transfer from synchronization
- An example application is *Halo Exchange* because it involves multiple communications per sync
- Joint work with Rajeev Thakur (Argonne), Subhash Saini (NASA Ames)
- This is also a good example for process topologies, because it involves communication between many neighboring processes

MPI One-Sided Communication

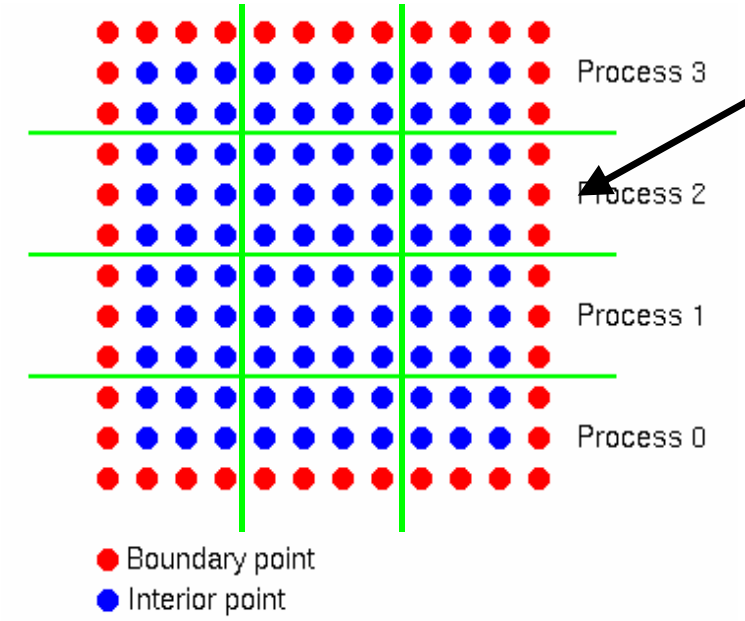
- Three data transfer functions
  - Put, get, accumulate

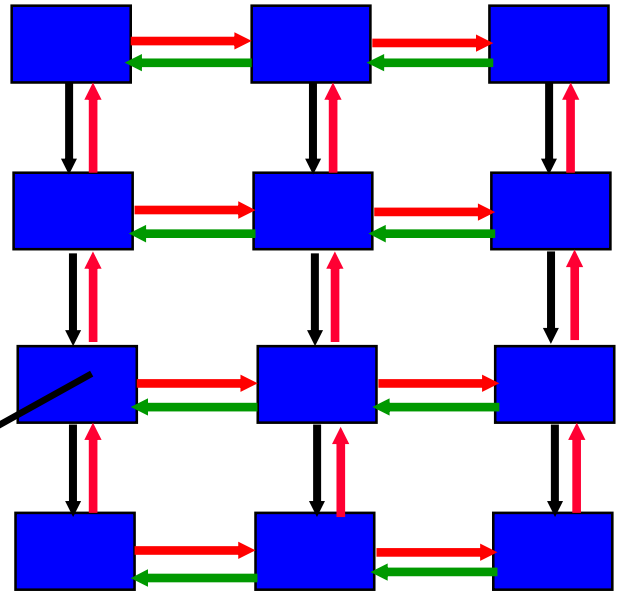


- Three synchronization methods
  - Fence
  - Post-start-complete-wait
  - Lock-unlock
- A natural choice for implementing halo exchanges
  - Multiple communication per synchronization

Halo Exchange

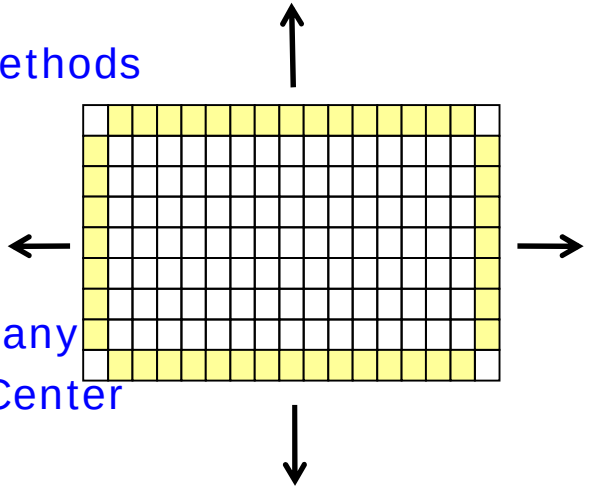
- Decomposition of a mesh into 1 patch per process
  - Update formula typically a(I,j) = f(a(i-1,j),a(i+1,j),a(I,j+1),a(I,j-1),…)
  - Requires access to “neighbors” in adjacent patches



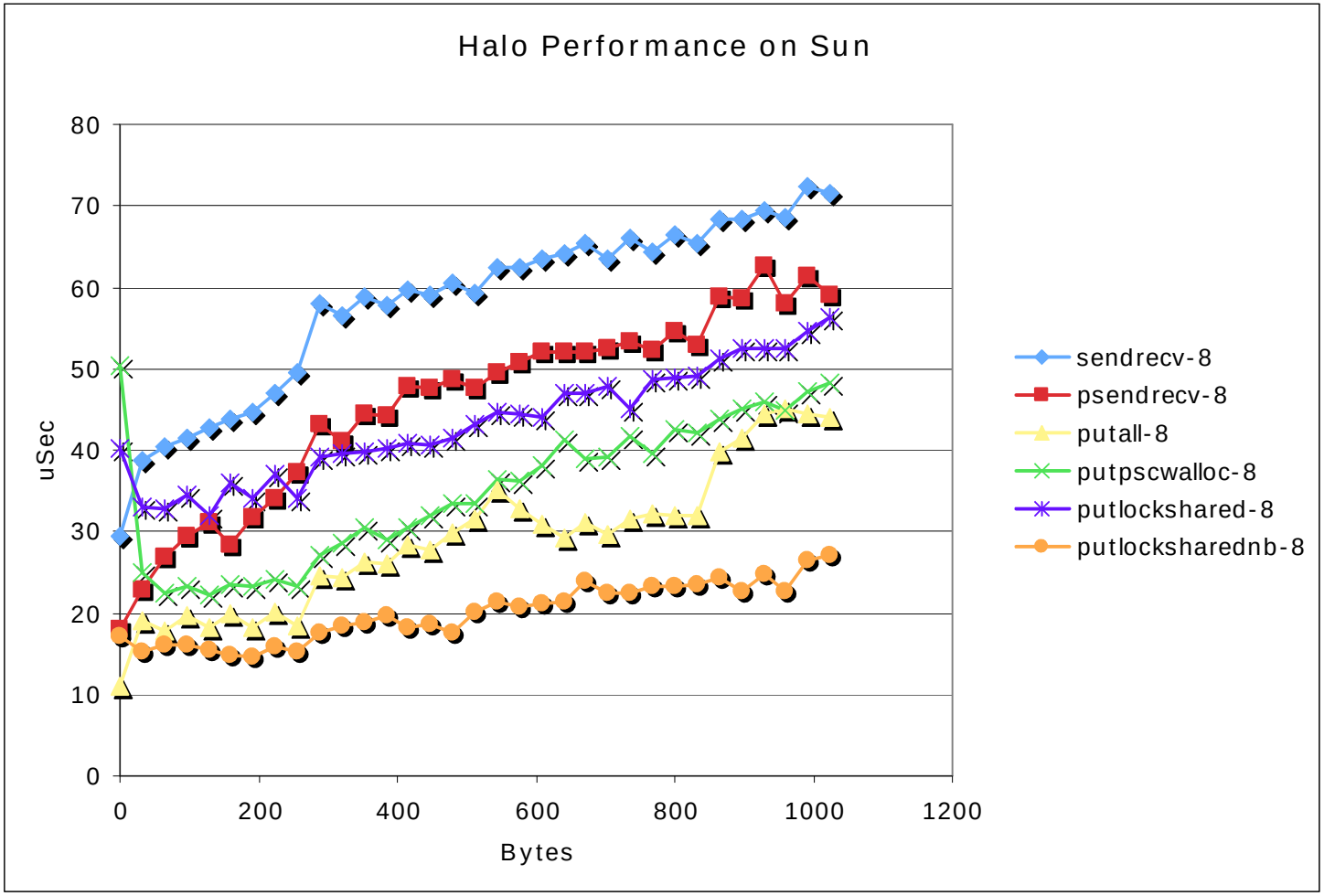


Performance Tests

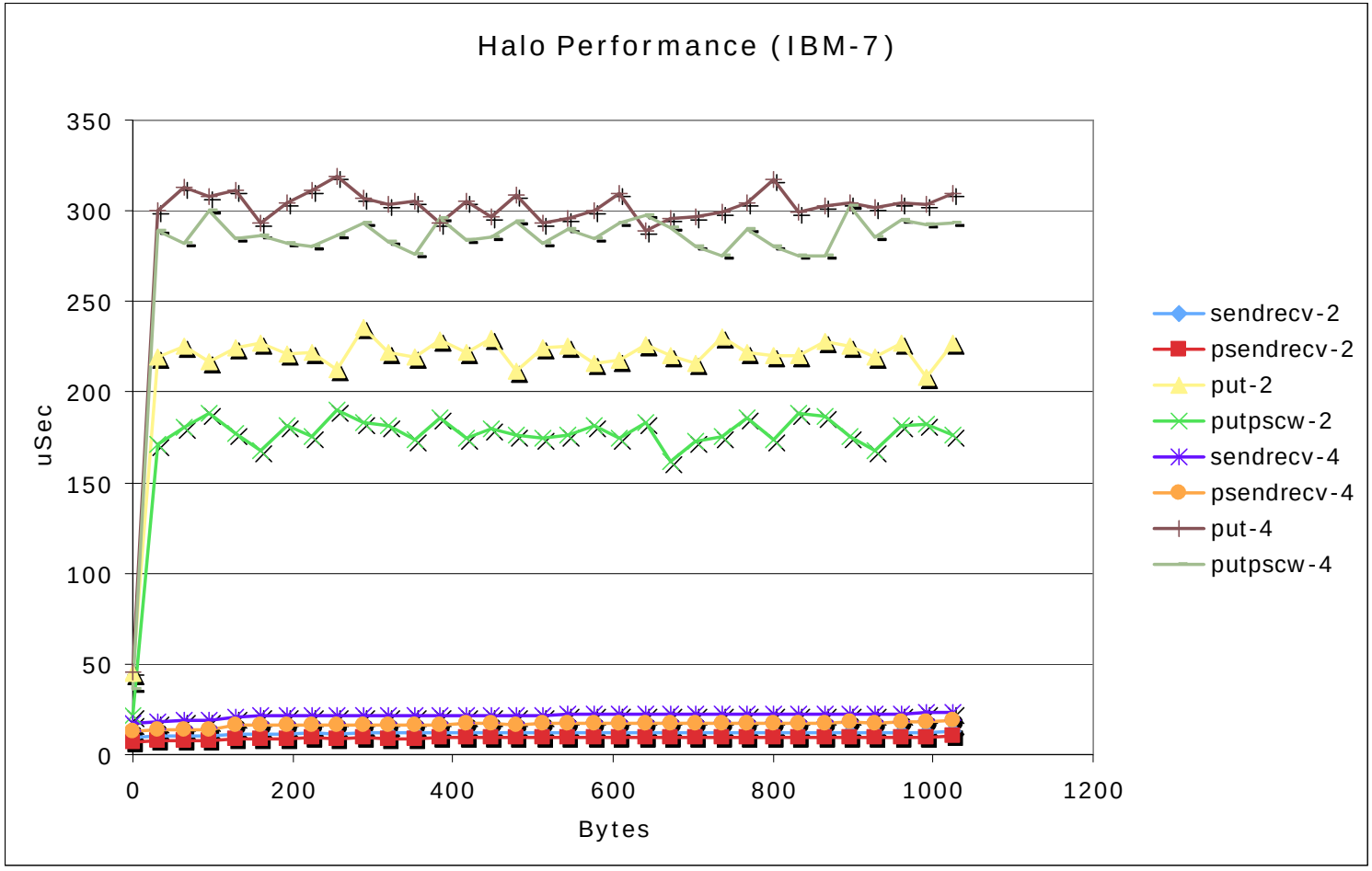
- "Halo" exchange or ghost-cell exchange operation
  - Each process exchanges data with its nearest neighbors
  - Part of the mpptest benchmark; works with any MPI implementation
    - Even handles implementations that only provide a subset of MPI-2 RMA functionality
    - Similar code to that in halocompare, but doesn't use process topologies (yet)
  - One-sided version uses all 3 synchronization methods
- Available from
- http://www.mcs.anl.gov/mpi/mpptest
- Ran on
  - Sun Fire SMP at here are RWTH, Aachen, Germany
  - IBM p655+ SMP at San Diego Supercomputer Center



One-Sided Communication on Sun SMP with Sun MPI



One-Sided Communication on IBM SMP with IBM MPI



Observations on MPI RMA and Halo Exchange

- With a good implementation and appropriate hardware, MPI RMA can provide a performance benefit over MPI point-to-point
- However, there are other effects that impact communication performance in modern machines...

Experiments with Topology and Halo Communication on "Leadership Class" Machines

- The following slides show some results for a simple halo exchange program (halocompare) that tries several MPI-1 approaches and several different communicators:
  - MPI_COMM_WORLD
  - Dup of MPI_COMM_WORLD
    - Is MPI_COMM_WORLD special in terms of performance?
  - Reordered communicator - all even ranks in MPI_COMM_WORLD first, then the odd ranks
    - Is ordering of processes important?
  - Communicator from MPI_Dims_create/MPI_Cart_create
    - Does MPI Implementation support these, and do they help
- Communication choices are
  - Send/Irecv
  - Isend/Irecv
  - "Phased"

Method 1: Use Irecv and Send

- Do i=1,n_neighbors
  Call MPI_Irecv(inedge(1,i), len, MPI_REAL, nbr(i), tag,&
  comm, requests(i), ierr)
  Enddo
  Do i=1,n_neighbors
  Call MPI_Send(edge(1,i), len, MPI_REAL, nbr(i), tag,&
  comm, ierr)
  Enddo
  Call MPI_Waitall(n_neighbors, requests, statuses, ierr)
- Does not perform well in practice (at least on BG, SP).
  - Quiz for the audience: Why?

Method 2: Use Isend and Irecv

- Do i=1,n_neighbors
  Call MPI_Irecv(inedge(1,i),len,MPI_REAL,nbr(i),tag,&
  comm, requests(i),ierr)
  Enddo
  Do i=1,n_neighbors
  Call MPI_Isend(edge(1,i), len, MPI_REAL, nbr(i), tag,&
  comm, requests(n_neighbors+i), ierr)
  Enddo
  Call MPI_Waitall(2*n_neighbors, requests, statuses, ierr)

Halo Exchange on BG/L

	4 Neighbors		8 Neighbors	
	Irecv/Send	Irecv/Isend	Irecv/Send	Irecv/Isend
World	112	199	94	133
Even/Odd	81	114	71	93
Cart_create	107	218	104	194

- 64 processes, co-processor mode, 2048 doubles to each neighbor
- Rate is MB/Sec (for all tables)

Halo Exchange on BG/L

	4 Neighbors		8 Neighbors	
	Irecv/Send	Irecv/Isend	Irecv/Send	Irecv/Isend
World	64	120	63	72
Even/Odd	48	64	41	47
Cart_create	103	201	103	132

- 128 processes, virtual-node mode, 2048 doubles to each neighbor
- Same number of *nodes* as previous table

Halo Exchange on Cray XT4

	4 Neighbors			8 Neighbors	
	Irecv/Send	Irecv/Isend	Phased	Irecv/Send	Irecv/Isend
World	153	153	165	133	136
Even/Odd	128	126	137	114	111
Cart_create	133	137	143	117	117

(Periodic)	4 Neighbors			8 Neighbors	
	Irecv/Send	Irecv/Isend	Phased	Irecv/Send	Irecv/Isend
World	131	131	139	115	114
Even/Odd	113	116	119	104	104
Cart_create	151	151	164	129	128

- 1024 processes, 2000 doubles to each neighbor

Halo Exchange on Cray XT4

	4 Neighbors			8 Neighbors	
	Irecv/Send	Irecv/Isend	Phased	Irecv/Send	Irecv/Isend
World	311	306	331	262	269
Even/Odd	257	247	279	212	206
Cart_create	265	275	266	236	232

(Periodic)	4 Neighbors			8 Neighbors	
	Irecv/Send	Irecv/Isend	Phased	Irecv/Send	Irecv/Isend
World	264	268	262	230	233
Even/Odd	217	217	220	192	197
Cart_create	300	306	319	256	254

- 1024 processes, SN mode, 2000 doubles to each neighbor

Observations on Halo Exchange

- Topology is important (again)
- For these tests, MPI_Cart_create always a good idea for BG/L; often a good idea for periodic meshes on Cray XT3/4
  - Not clear if MPI_Cart_create creates optimal map on Cray
- Cray performance is significantly under what the "ping-pong" performance test would predict
  - The success of the "phased" approach on the Cray suggests that some communication contention may be contributing to the slow-down
    - Either contention along links (which should not happen when MPI_Cart_create is used) or contention at the destination node.
  - To see this, consider the performance of a single process sending to four neighbors

Discovering Performance Opportunities

- Lets look at a single process sending to its neighbors. We *expect* the rate to be roughly twice that for the halo (since this test is only sending, not sending and receiving)

System	4 neighbors		8 Neighbors	
		Periodic		Periodic
BG/L	488	490	389	389
BG/L, VN	294	294	239	239
XT3	1005	1007	1053	1045
XT4	1634	1620	1773	1770
XT4 SN	1701	1701	1811	1808

- BG gives roughly double the halo rate. XTn is much higher
  - It should be possible to improve the halo exchange on the XT by scheduling the communication
  - Or improving the MPI implementation

Discovering Performance Opportunities

- Ratios of a single sender to all processes sending (in rate)
- *Expect* a factor of roughly 2 (since processes must also receive)

System	4 neighbors		8 Neighbors	
		Periodic		Periodic
BG/L	2.24		2.01	
BG/L, VN	1.46		1.81	
XT3	7.5	8.1	9.08	9.41
XT4	10.7	10.7	13.0	13.7
XT4 SN	5.47	5.56	6.73	7.06

- BG gives roughly double the halo rate. XTn is much higher
  - It should be possible to improve the halo exchange on the XT by scheduling the communication
  - Or improving the MPI implementation (But is it topology routines or point-to-point communication? How would you test each hypothesis?)

Efficient Support for MPI_THREAD_MULTIPLE

- MPI-2 allows users to write multithreaded programs and call MPI functions from multiple threads (MPI_THREAD_MULTIPLE)
- Thread safety does not come for free, however
- Implementation must protect certain data structures or parts of code with mutexes or critical sections
- To measure the performance impact, we ran tests to measure communication performance when using multiple threads versus multiple processes
- These results address issues with the *thread* programming model, in the context of a demanding application (an MPI implementation)
- Joint work with Rajeev Thakur (Argonne)

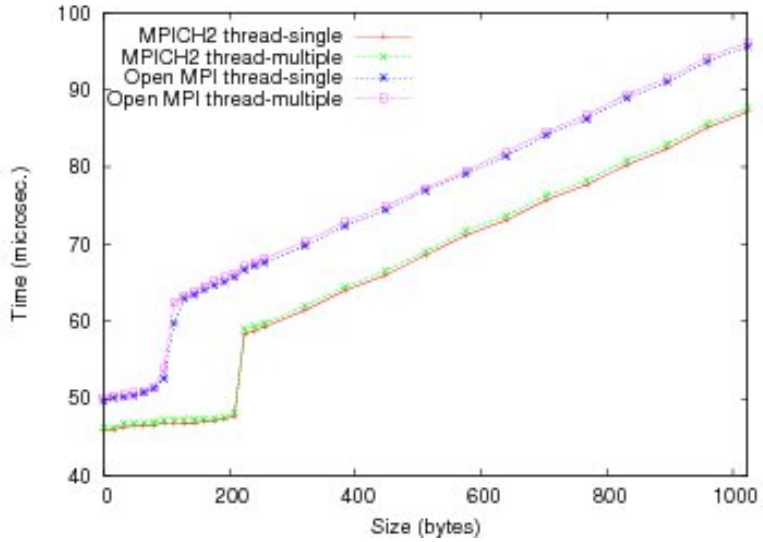
Application Assumptions about Threads

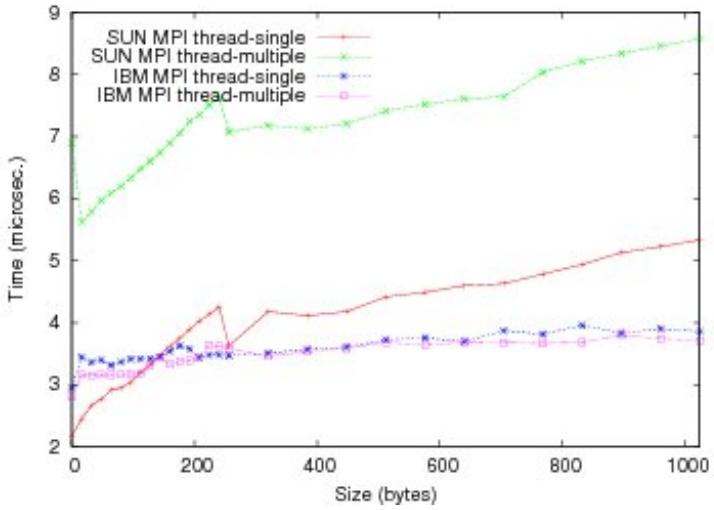
- Thread support cost little, particularly when not used (MPI_Init_thread(MPI_THREAD_FUNNELED)
- Threads and processes have equal communication performance
- Blocked operations in one thread do not slow down other threads

- How true are these assumptions?

Cost of Thread Support

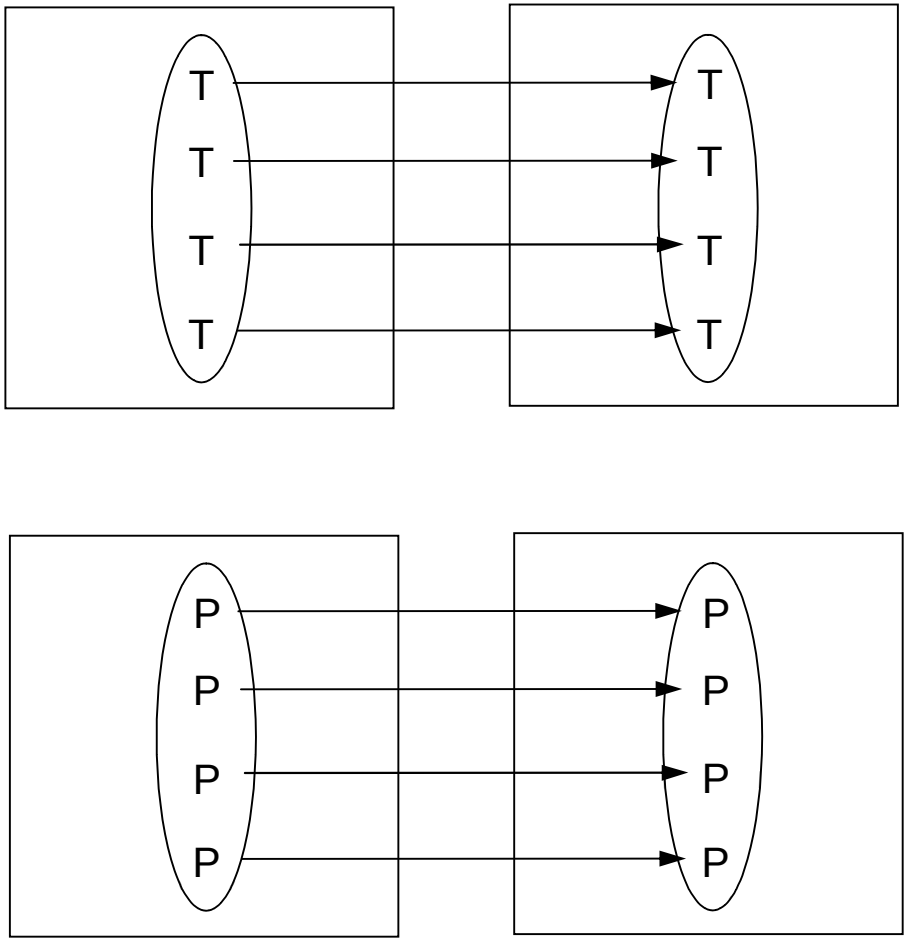
- Can an application use MPI_Init_thread(MPI_THREAD_FUNNELED,...) if it does not need thread support, instead of MPI_Init?
- Requires either a very low cost support for threads *or* runtime selection of no thread locks/atomic operations if THREAD_FUNNELED requested.
- How well do MPI implementations do with this simple test?
  - The IBM SP implementation has very low overhead
  - The Sun implementation has about a 3.5 usec overhead
    - Shows cost of providing thread safety
    - This cost can be lowered, but requires great care



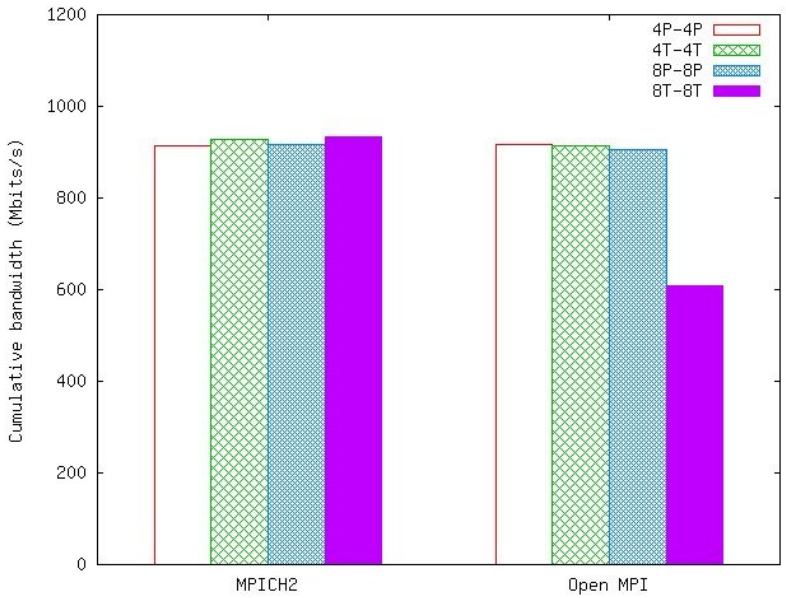


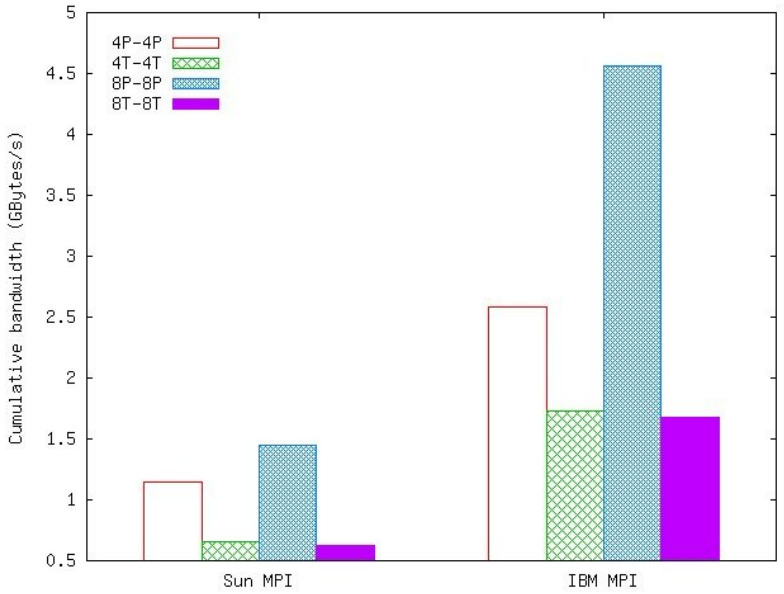
Tests with Multiple Threads versus Processes

- Consider these two cases:
  - Nodes with 4 cores
  - 1 process with four threads aasends to 1 process with four threads, each thread sending, or
  - 4 processes, each with one thread, sending to a corresponding thread
- User expectation is that the performance is the same



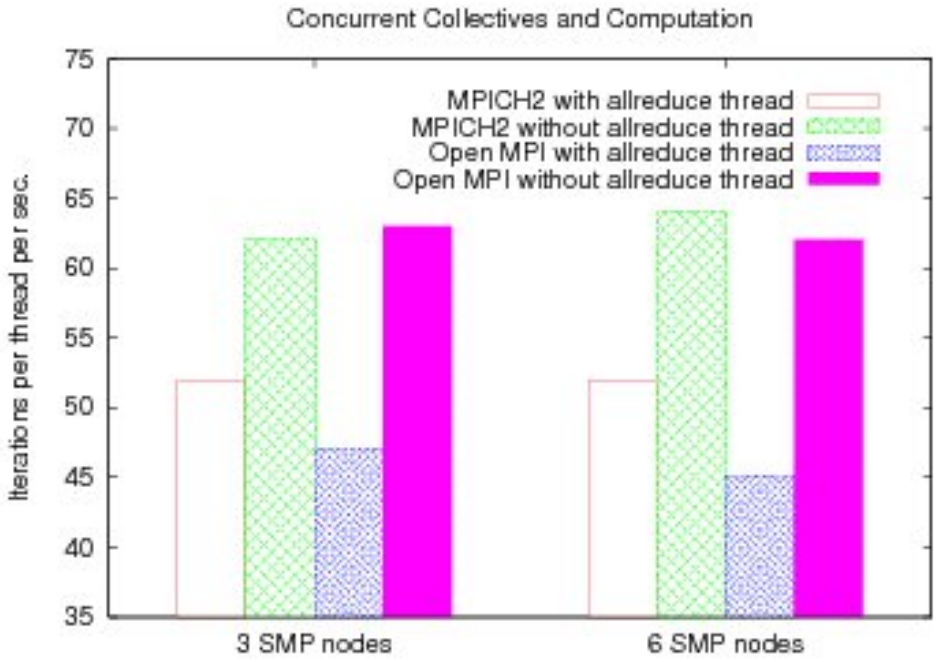
Concurrent Bandwidth Test





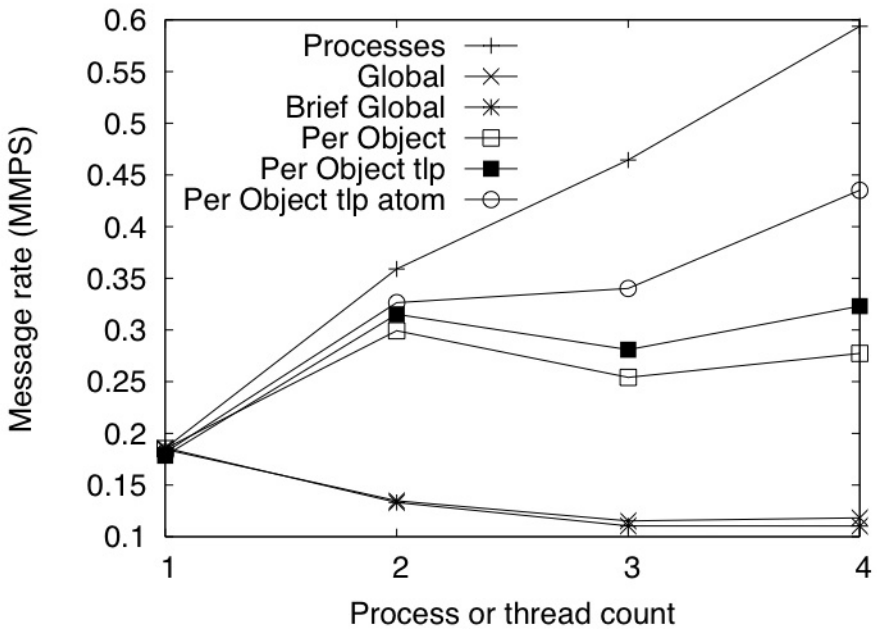
Impact of Blocking Operations

- The MPI Forum rejected separate, non-blocking collective operations (for some good reasons), arguing that these can be implemented by placing a blocking collective in a separate thread.
- Consider such a sample program, where a compute loop (no communication) is in one thread, and an MPI_Allreduce is in the second thread
- Question: How does the presence of the Allreduce thread impact the compute loop?



Challenges in Reducing Thread Overhead

- Communication test involves sending to multiple destination processes
- Using a single synchronization model (global critical section) results in a slowdown
- Using narrower sections permits greater concurrency but still has significant overhead
  - Even when assembly-level processor-atomic, lock-free operations used (atom) and thread-local pools of data structures (tlp)
- Hard to beat processes



Pavan Balaji, Darius Buntinas, David Goodell, William Gropp, and Rajeev Thakur

Notes on Thread Performance

- Providing Thread support with performance is hard
- Applications should not assume fast thread support
- More application-inspired tests are needed

Where Does MPI Need to Change?

- Nowhere
  - There are many MPI legacy applications
  - MPI has added routines to address problems rather than changing them
  - For example, to address problems with the Fortran binding and 64-bit machines, MPI-2 added MPI_Get_address and MPI_Type_create_xxx and deprecated (but did not change or remove) MPI_Address and MPI_Type_xxx.
- Where does MPI need to add routines and deprecate others?
  - One Sided
    - Designed to support non-coherent memory on a node, allow execution in network interfaces, and nonblocking memory motion
    - Put requires ack (to enforce ordering)
    - Lock/put/unlock model very heavy-weight for small updates
    - Generality of access makes passive-target RMA difficult to implement efficiently (exploiting RDMA hardware)
    - Users often believe MPI_Put and MPI_Get are blocking (poor choice of name, should have been MPI_Iput and MPI_Iget).
  - Various routines with “int” arguments for “count”
    - In a world of 64 bit machines and multiGB laptops, 32-bit ints are no longer large enough

Extensions

- What does MPI need that it doesn't have?
- Don't start with that question. Instead ask
  - What tool do I need? Is there something that MPI needs to work well with that tool (that it doesn't already have)?
- Example: Debugging
  - Rather than define an MPI debugger, develop a thin and simple interface to allow any MPI implementation to interact with any debugger
- Candidates for this kind of extension
  - Interactions with process managers
    - Thread co-existance (MPIT discussions)
    - Choice of resources (e.g., placement of processes with Spawn) Interactions with Integrated Development Environments (IDE)
  - Tools to create and manage MPI datatypes
  - Tools to create and manage distributed data structures
    - A feature of the HPCS languages

MPI Forum

- The MPI Forum is the ad hoc group that created the MPI standard
- Made up of vendors, users, and researchers
- Uses a formal process to create and correct the standard
  - Votes, membership rules, etc.
- Anyone with interest may join and attend
  - No fees, other than travel to meetings
- More information
  - http://meetings.mpi-forum.org/

MPI 2.1

- Clarifications to the MPI 2.0 Standard documents, resulting in a single document describing the full MPI 2.1 standard. This includes merging of all previous MPI standards documents into a single document, adding text corrections, and adding clarifying text.
- Status: Combined MPI Standard Document drafted and reviewed
- MPI Forum voted to accept any errata/clarifications
- Thanks to Rolf Rabenseifner, HLRS

MPI 2.2

- Small changes to the MPI 2.1 standard. A small change is defined as one that does not break existing user code, either by interface changes or by semantic changes, and does not require large implementation changes.
- Status:
  - Two nontrival enhancements
    - Reuse of send buffers
    - Consistent use of const
  - Many errata updates
  - William Gropp, UIUC

MPI Forum Efforts: The Next Generation

- MPI 3.0 - Additions to the MPI 2.2 standard that are needed for better platform and application support. These are to be consistent with MPI being a library that provides parallel process management and data exchange capabilities. This includes, but is not limited to, issues associated with scalability (performance and robustness), multi-core support, cluster support, and applications support.

MPI 3 Working Groups

- Application Binary Interface - A common ABI, particularly for commodity hardware/software pairs
- Collective Operations - Additions including nonblocking collectives
- Fault Tolerance - Enhanced support for fault tolerant applications in MPI
- Fortran Bindings - Address the problems and limitations of the Fortran 90 MPI bindings.
- Generalized Requests - Provide alternative progress models that match other parts of the system
- MPI Sub-Setting - Define compatible subsets of MPI
- Point-To-Point Communications - Additions and/or changes
- Remote Memory Access - Re-examine the MPI RMA interface and consider additions and/or changes

What This Means for You

- How can MPI coexist with other programming models?
  - Where does MPI need to change?
  - Where do others need to change?
  - Example: allocation of thread/process resources
- What could MPI provide to better support other tools?
- What can MPI learn from the success of Charm++?
- How should MPI evolve? You can help!