IBM Research

BlueGene/L MPI From A User's Point Of View

- or -

A Short Survival Course In How To Annoy Tech Support

George Almási

Feb. 10, 2005

Outline

- This is not a talk about how to invoke mpirun.
  - System software folks will have presented that to you
- This is a talk about what does, and what doesn't work in BlueGene/L MPI.
  - Basic things you should avoid doing
  - Ideas about obtaining good performance
    - point-to-point messaging
      - scaling
      - mapping application into network
    - collective messaging

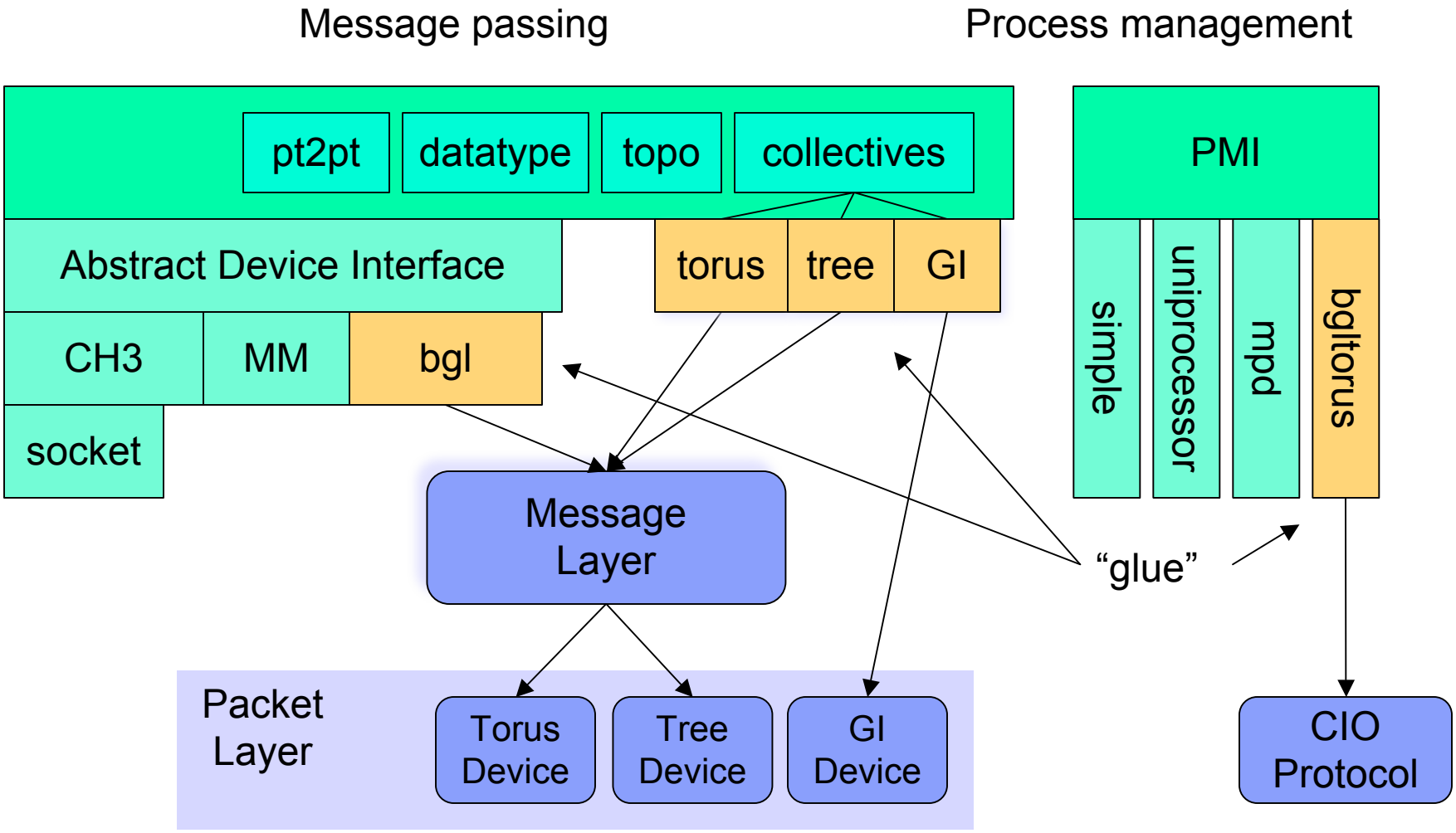
Summary I: will BlueGene like me?

- BlueGene/L MPI is like other implementations of MPI
  - looks like Argonne National Labs' MPICH2
  - because that's what it is.
- As an MPI developer you will have relatively few surprises
  - BlueGene/L MPI is MPI standard 1.2 compliant
    - no one-sided communication
    - no spawning of processes
    - supports thread model MPI_THREAD_SINGLE
    - MPI I/O is still under development
      - packages like HDF5, NETCDF are known to have been ported but presently display relatively poor performance
  - Getting higher performance out of BG/L MPI is inherently harder
    - large scale, fun network

Summary II: Kinds of annoyance you can cause

- **Crashing an application**: an opportunity to bad-mouth IBM
  - somewhat easier than on other platforms, because
    - limited memory on nodes, no virtual memory on nodes
      - memory leaks are going to make their presence felt
    - communication network is in userspace
      - good: high performance
      - bad: user have opportunity to kill process with wild pointers
- **Invoking the Halting Problem:** deadlocking the machine
- **Violating MPI semantics**: laws you didn't know were on the books
- **Causing bad performance:** malice not required
  - “nicely” map the application into the network (hard)
  - avoid load imbalance (hard)
  - avoid network jams (very hard)

BlueGene/L MPI Software Architecture



Write your own communication layer!

- 90% of communication hardware is mapped into user memory
  - Write stuff into high memory areas!
  - Likely to insert malformed packets into the torus.
    - will generate spurious error messages
    - will look somewhat like a system failure
    - may hang your application, and even bring down your partition.
  - Uninitialized pointer dereferences work great.
  - Requires very little effort to hang the machine
    - If you know what you are doing.
  - We have never seen this happen by accident.
    - You cannot accidentally malloc() over network hardware
    - You must set your pointer into a narrow address space

Deadlocking the system

- Talk before you listen.
- Illegal MPI code
  - find it in most MPI books
  - tech support will be very annoyed
- BlueGene/L MPI is designed not to deadlock easily.
  - It will likely survive this code.
- This code will cause MPI to allocate memory to deal with unexpected messages. If MPI runs out of memory, it will stop with an error message

```
CPU1 code:

MPI_Send (cpu2);
MPI_Recv(cpu2);

CPU2 code:

MPI_Send(cpu1);
MPI_Recv(cpu1);
```

Force MPI to allocate too much memory

- Post receives in one order, sends in the opposite order
- This is legal MPI code
- BlueGene/L MPI will choke if the sum of buffers is greater than the amount of physical memory
  - this is an implementation defect that will be fixed in the future

CPU1 code:

MPI_ISend(cpu2, tag_1);

MPI_ISend(cpu2, tag_2);

...

MPI_ISend(cpu2, tag_n);

CPU2 code:

MPI_Recv(cpu1, tag_n);

MPI_Recv(cpu1, tag_{n-1});

...

MPI_Recv(cpu1, tag_1);

Sneaky: violate MPI buffer ownership rules

- write send/receive buffers before completion
  - results in data race on any machine
- touch send buffers before message completion
  - not legal by standard
  - BG/L MPI will survive it today
  - no guarantee about tomorrow
- touch receive buffers before completion
  - BG/L MPI will yield wrong results

```
req = MPI_Isend (buffer);
buffer[0] = something;
MPI_Wait(req);
```

```
req = MPI_Isend (buffer);
z = buffer[0];
MPI_Wait (req);
```

```
req = MPI_Irecv (buffer);
z = buffer[0];
MPI_Wait (req);
```

Causing memory overruns: never wait for MPI_Test

- Have to wait for all requests
  - The standard requires waiting
  - or testing until MPI_Test returns true
- This code works on many other architectures
  - causes tiny memory leaks
- On BG/L this will run the system out of memory very fast
  - MPI_Request requires a lot of memory
  - It's a scaling issue

```
req = MPI_Isend( ... );
MPI_Test (req);
... do something else; forget about
    req ...
```

Straddle collectives with point-to-point messages

- On the ragged edge of legality
- BlueGene/L MPI works
- Multiple networks issue:
  - Isend handled by torus network
  - Barrier handled by GI network

```
CPU 1 code:

req = MPI_Isend (cpu2);
MPI_Barrier();
MPI_Wait(req);
```

```
CPU 2 code:

MPI_Recv (cpu1);
MPI_Barrier();
```

Send flood

- This is legal MPI code
  - also ... stupid MPI code
  - not scalable, even when it works
- BlueGene/L MPI will run out of buffer space
  - This is a bug, and will be fixed
- We have seen this kind of code in the wild
  - Don't write code such as this
  - Even if you think it should work

CPU 1 to n-1 code:

```
MPI_Send(cpu0);
```

CPU 0 code:

```
for (i=1; i<n; i++)
    MPI_Recv(cpu[i]);
```

BlueGene/L ≠ Linux

- You are likely to run into surprises with what you assume runs on the compute nodes
  - Don't try asynchronous File I/O
  - TCP client stuff works:
    - socket(), connect()
  - TCP server stuff doesn't work:
    - bind(), accept()
  - BG/L runs sleep(10000) in 6 seconds!

Virtual Node Mode vs. Coprocessor mode

- Virtual Node Mode:
  - twice the processing power!
  - but not twice the performance
    - half of memory per CPU
    - half of cache per CPU
    - half of network per CPU
    - CPU has to do both computation and communication
- Coprocessor mode:
  - only one CPU available to execute user code
  - but have all memory!
  - other CPU helps with communication
  - currently, only point-to-point communication benefits
    - that is about to change

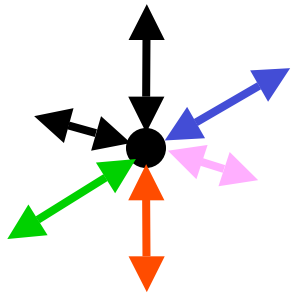
Point-to-point performance (I)

- Two kinds of network routing on BlueGene/L
  - deterministic routing:
    - each packet goes along the same path
    - maintains packet order
    - creates network hotspots
  - adaptive routing
    - packets overtake
    - equalized network load
    - harder on CPUs
    - MPI matching semantics are always correct!
- MPI Short protocol:
  - very short (<250 bytes) messages. Deterministically routed
- MPI Eager protocol:
  - medium size messages
  - “send without asking”
  - deterministically routed
  - latency around 3.3 µs
- MPI Rendezvous protocol:
  - large (> 10KBytes) messages
  - adaptively routed
  - bandwidth optimized

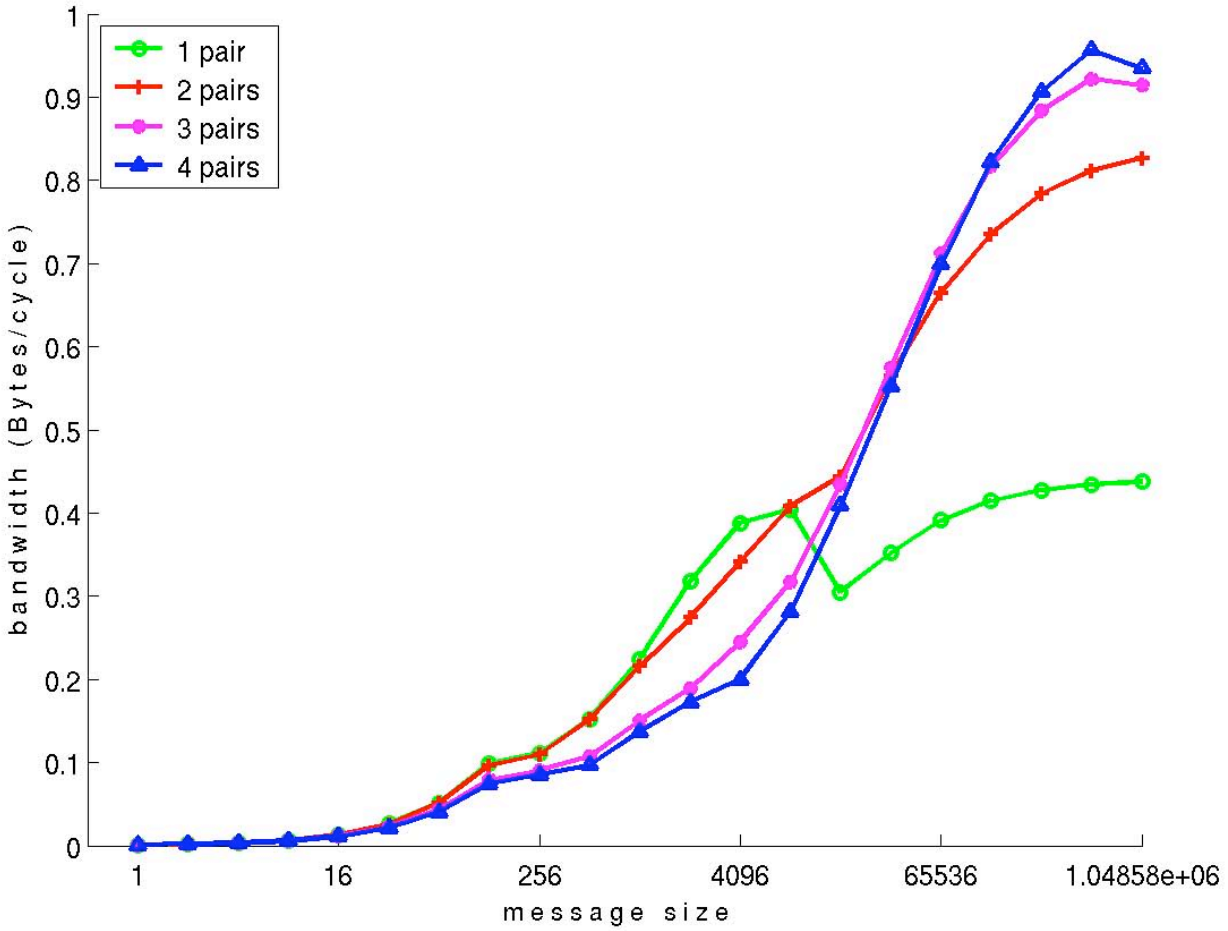
Point to point performance (II)

- The rendezvous treshold (10KBytes) can be changed
  - environment variable: BLMPI_RZV = ...
- Lower the rendezvous treshold if
  - running on a large partition
  - many short messages are overloading the network
  - eager messages are creating artificial hotspots
  - program is not latency sensitive
- Increase the rendezvous treshold if
  - most communication is nearest-neighbor
    - or at least close in Manhattan distance
  - relatively longer messages
  - you need better latency on medium size messages

Bandwidth vs. message size



6-way send+recv



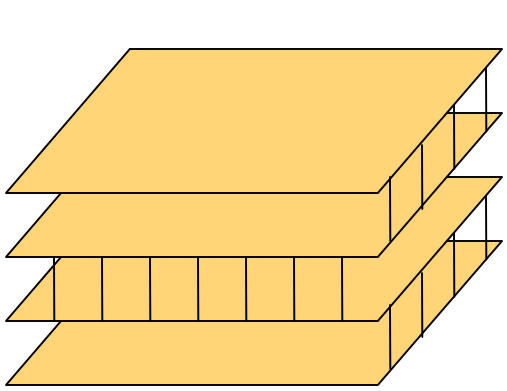
1 byte/cycle = 700MB/s

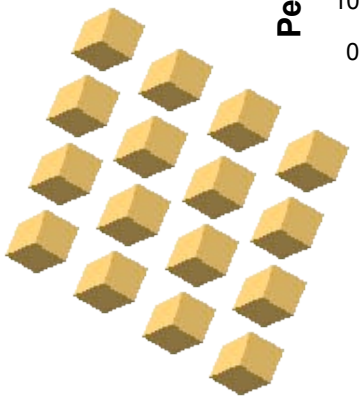
Point-to-point performance (III): Dos and Don'ts

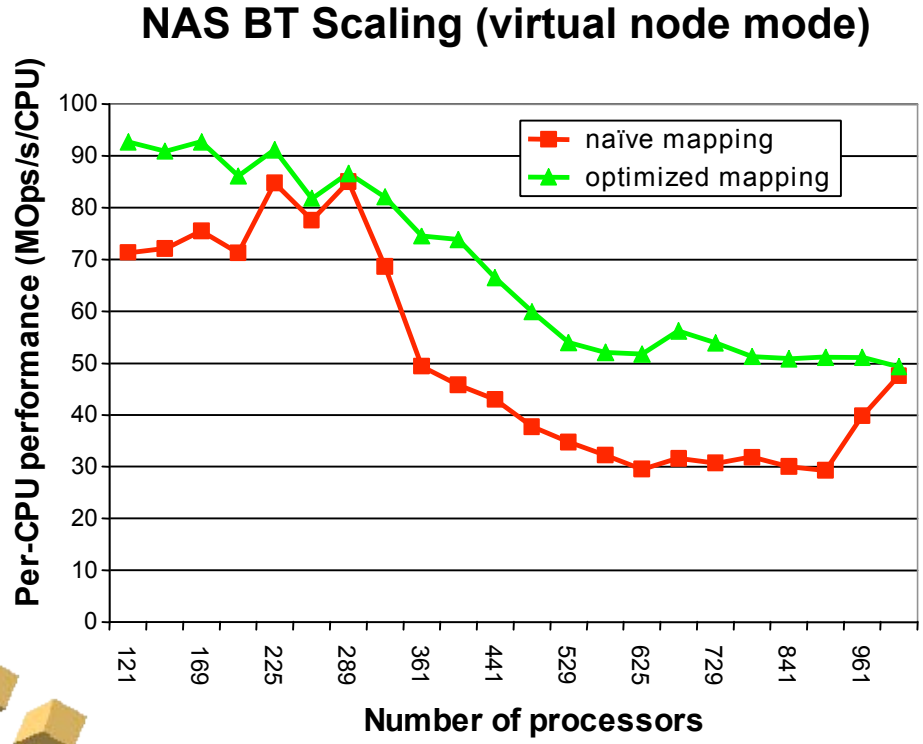
- Overlapping communication and computation:
  - requires care on BlueGene/L
  - keep programs in sync as much as you can
    - alternate computation and communication phases
- Avoid load imbalance
  - bad for scaling
- Shorten Manhattan distance messages have to traverse
  - send to nearest neighbors!
- Avoid synchronous sends
  - increases latency
- Avoid buffered sends
  - memory copies are bad for your health
- Avoid vector data, non-contiguous data types
  - BG/L MPI doesn't have a nice way to deal with them
- Post receives in advance
  - unexpected messages damage performance

The all-important torus mapping

- NAS BT
  - 2D mesh communication pattern
  - Map on 3D mesh/torus?
    - Folding and inverting planes in the 3D mesh
- NAS BT scaling:
  - Computation scales down with n^{-2}
  - Communication scales down with n^{-1}







How to map an application to the torus?

- set up a mapping file

```
0 0 0 0
1 0 0 0
2 0 0 0
3 0 0 0
...
```

- associate torus coordinates to MPI ranks 0 to n-1.
- Yeah, but why quadruplets?
- Use mapping file as argument in mpirun invocation

MPI Collective performance (I)

- Rule 1: Use collectives whenever you can
  - Point-to-point performance has huge overheads
  - “I can do a better job with point-to-point than you miserable jocks can do with collectives”
    - We don't think so.
- Rule 2: Mapping is all-important for good collective performance
  - Most collective implementations prefer certain communicator shapes
- Rule 3: don't do anything crazy, like
  - Use different buffer sizes for a broadcast call (illegal)
  - Use heterogeneous data types for broadcast (legal, but crazy)
  - Use misaligned buffers (legal and not crazy, but we don't like it anyway)
  - Run point-to-point messages across the communicator at the same time that a collective is underway (legal, but not cheap)

Summary of Optimized BG/L MPI Collectives

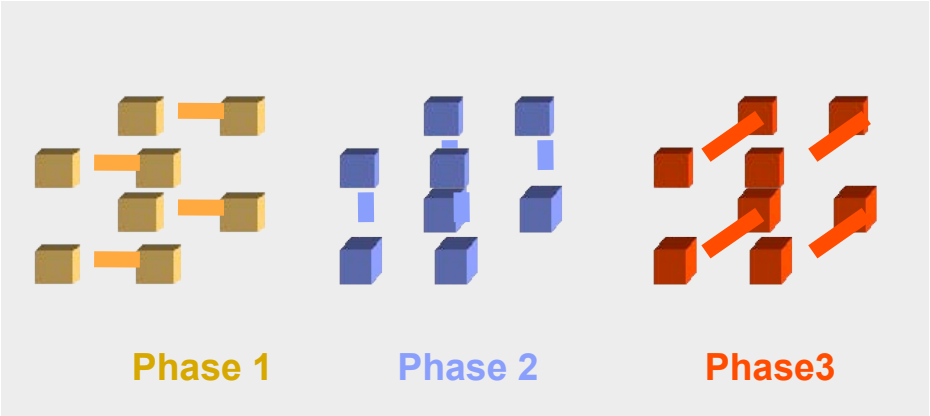
	Condition	Network	Performance
Barrier	COMM_WORLD	GI	1.5us
	COMM_WORLD	Tree	5 us
	Rectangular communicator	Torus	10-15 us
Broadcast	COMM_WORLD	Tree	350 Mbytes/s
	Rectangular communicator	Torus	320 Mbytes/s (0.48 Bytes/cycle)
	Rectangular communicator	Torus	**TBD**: low latency
Allreduce	COMM_WORLD, fixed point	Tree	350 Mbytes/s, low latency
	COMM_WORLD, floating pt	Tree	40 Mbytes/s (0.06Bytes/c)
		Tree	**TBD:** >= 120MB/s, low latency
	Hamilton Path	Torus	120 Mbytes/s
	Rectangular communicator	Torus	80 Mbytes/s
	Rect. comm. + short msg	Torus	10-15 us latency
	TBD: other shapes	Torus	**TBD:** high bandwidth FP
Alltoall[v]	Any communicator	Torus	84-97% of peak
Allgatherv	rectangular	Torus	Same as broadcast

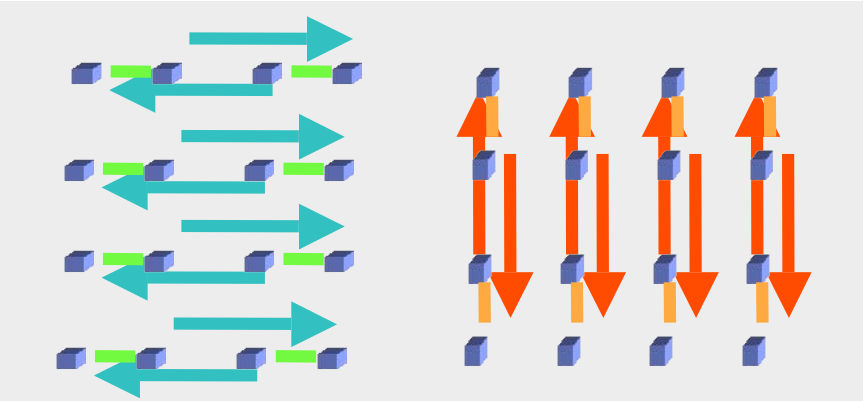
Optimizing collective performance: Barrier and short-message Allreduce

- **Barrier** is implemented as an **allgather** in each dimension
  - BG/L torus hardware can send deposit packets on a line
  - Low latency broadcast
- Since packets are short, likelihood of conflicts is low
- Latency = O(xsize+ysize+zsize)

- **Allreduce** for very short messages is implemented with a similar multi-phase algorithm

impl. by Yili Zheng

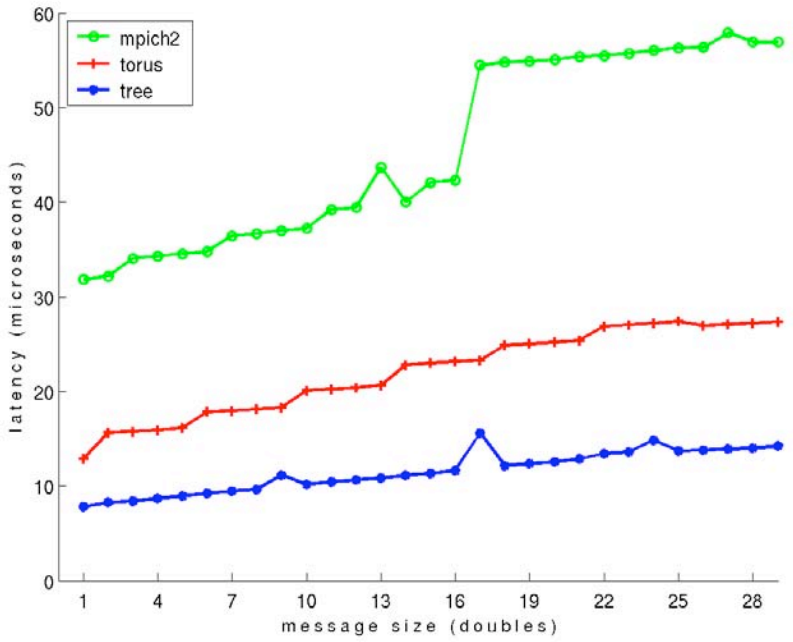




Barrier and short message Allreduce: Latency and Scaling

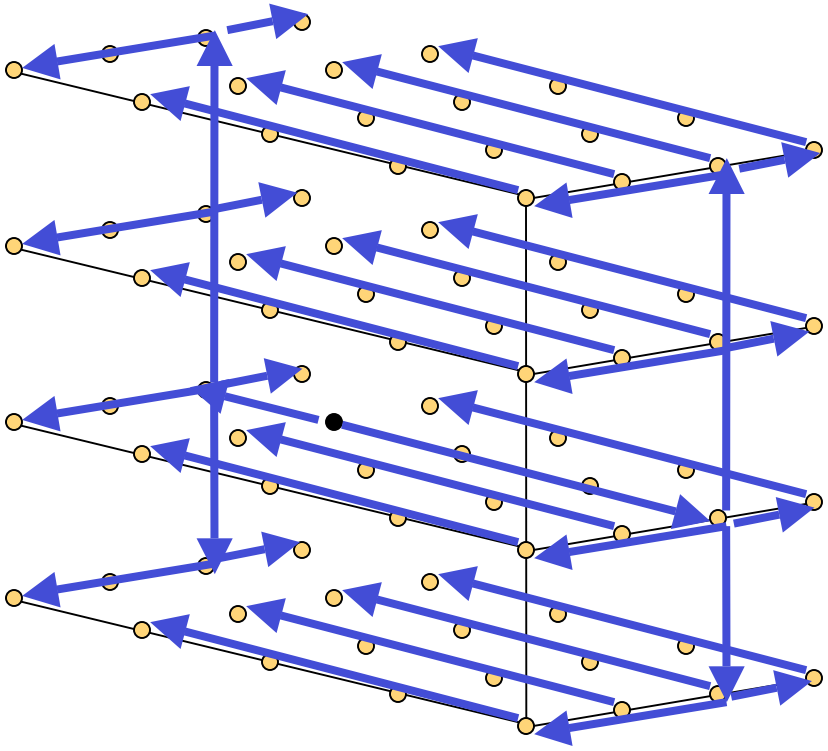
Barrier latency vs. machine size

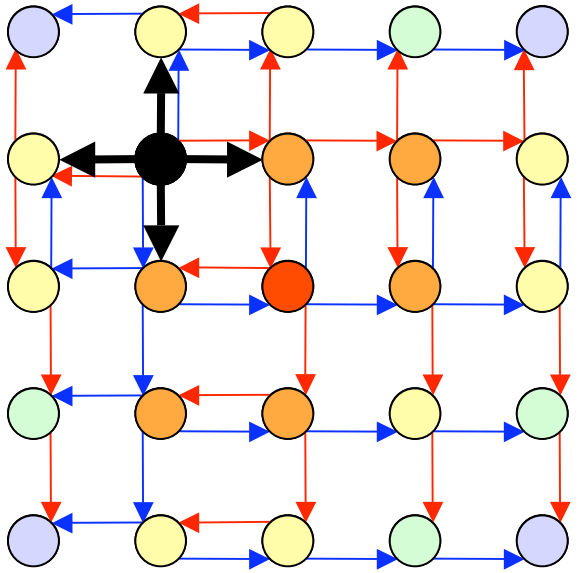
Short-message Allreduce latency vs. message size



MPI_Bcast on a mesh: algorithm details

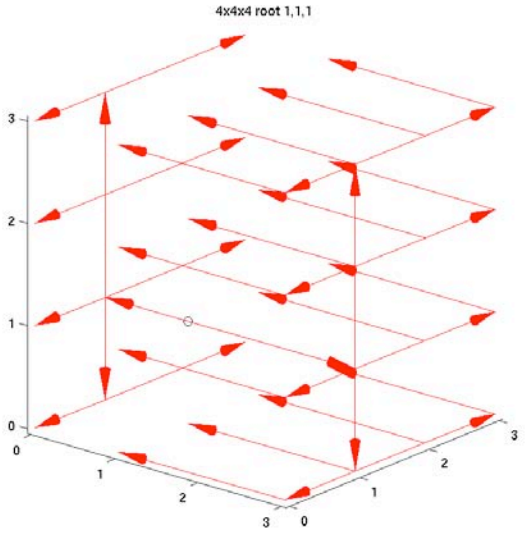
with John Gunnels, Nils Smeds, Vernon Austel, Yili Zheng, Xavier Martorell

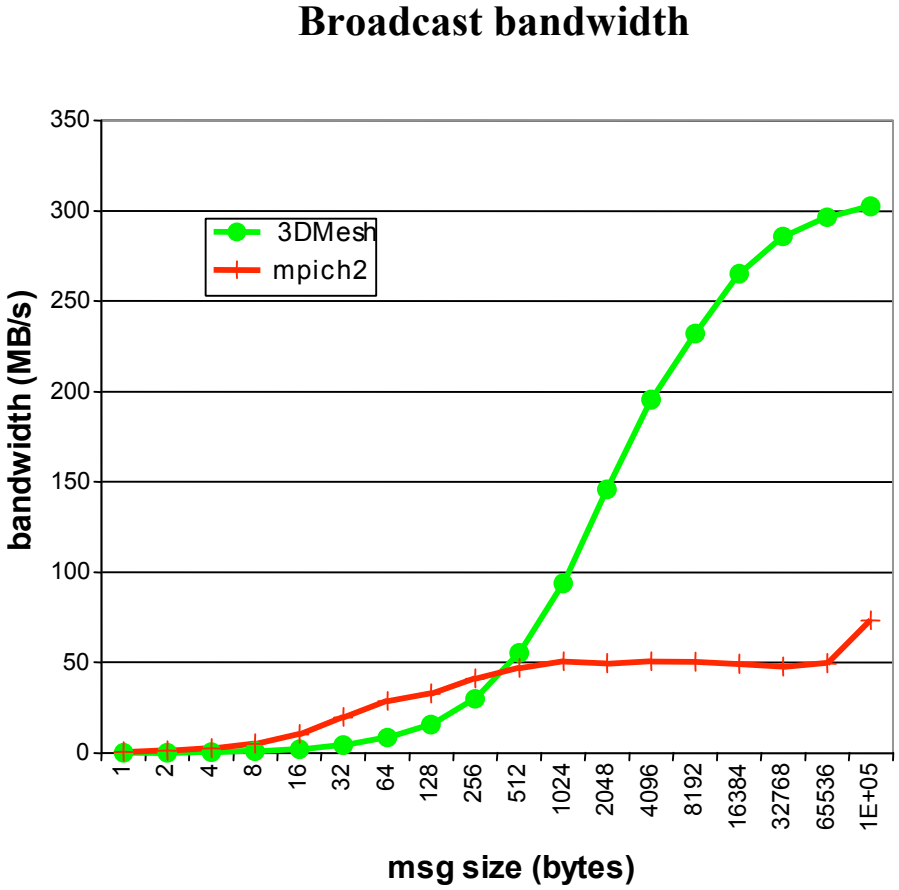




MPI_Bcast performance

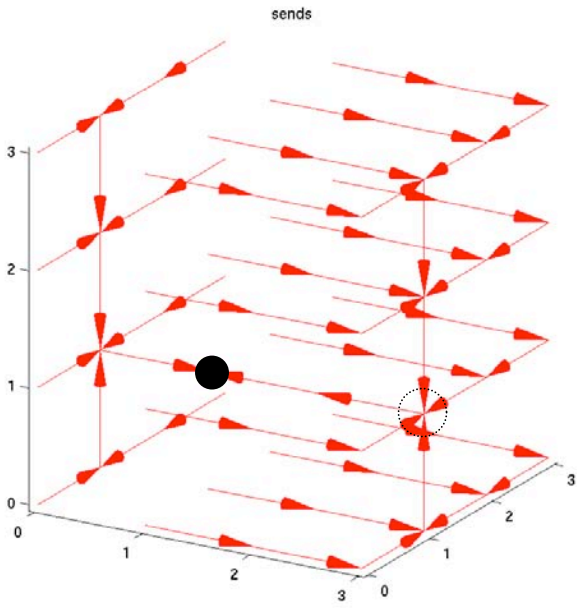
- MPICH2: stable but slow
- Tree broadcast:
  - only for MPI_COMM_WORLD
- Torus broadcast:
  - any rectangular communicator
  - Uses deposit bit
  - "menu" system

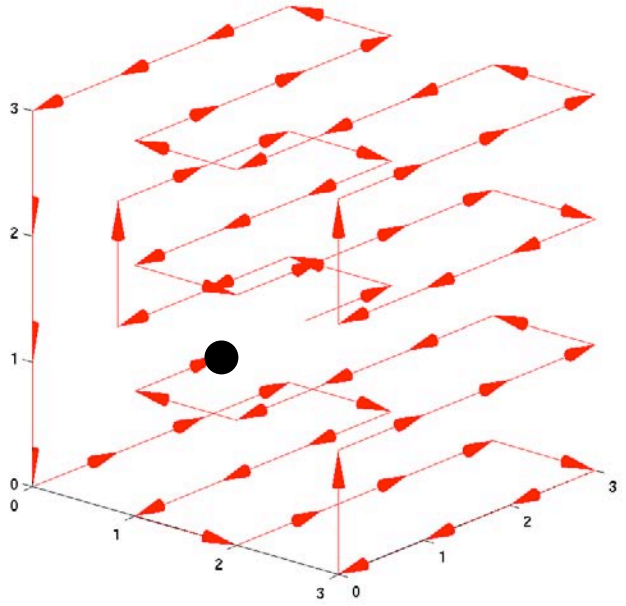




Optimized collectives: Allreduce for long messages

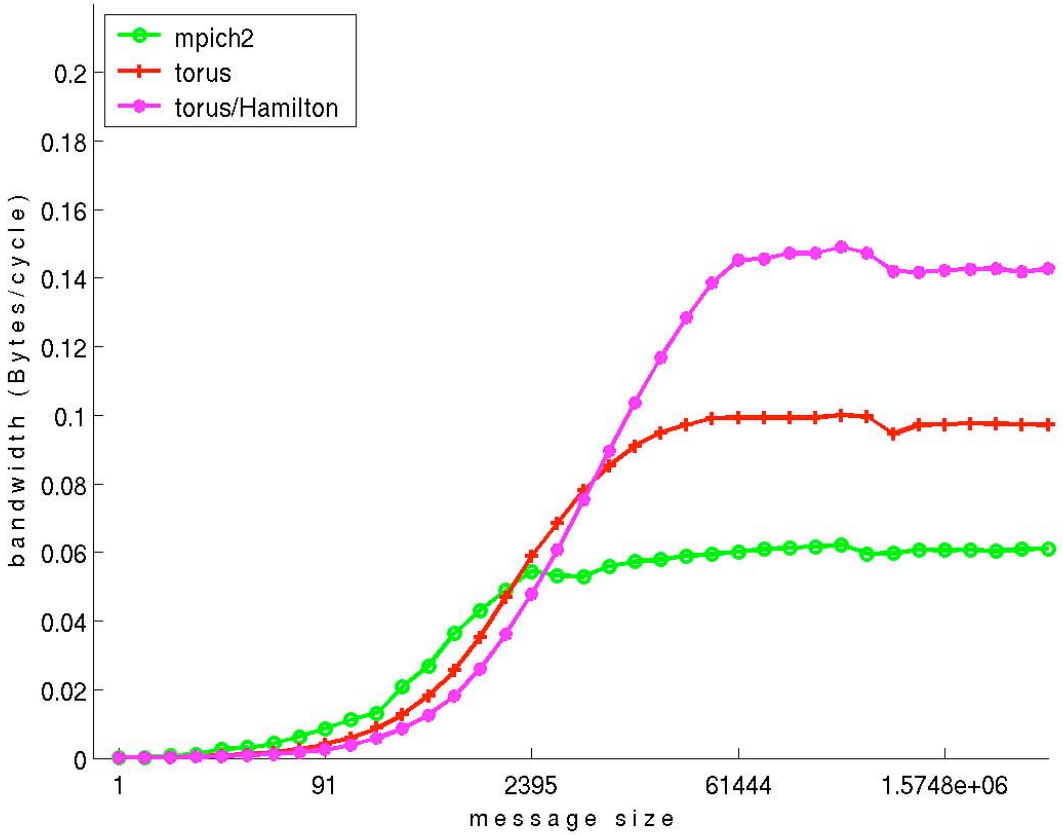
- Allreduce: standard "menu"
  - Similar to broadcast
  - Reasonable latency
  - Strongly CPU limited
- Allreduce: Hamiltonian path "menu"
  - Single line snaking through torus
  - Very high latency
  - Somewhat better bandwidth





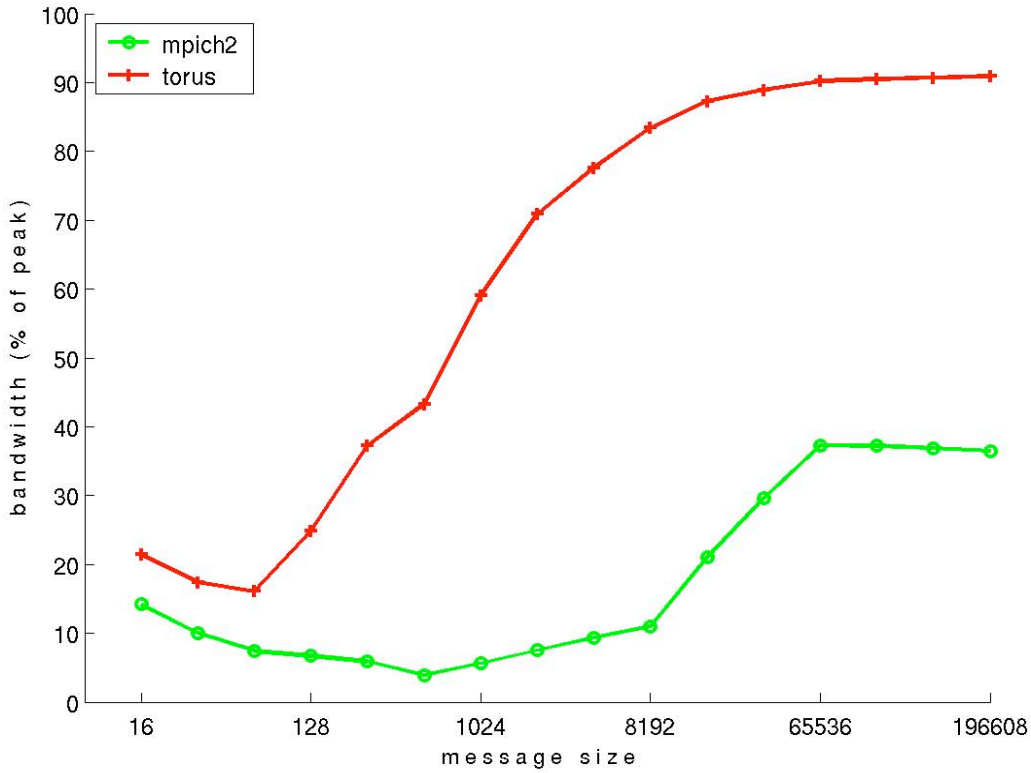
Impl. by Chris Erway

Optimized collectives: Allreduce bandwidth



Optimized collectives: Alltoall[v]

- Performance measured as percentage of peak, which is function of partition “shape”
- MPICH2 implementation not suitable for torus network
- Optimized implementation: 90% of peak
- **Impl. by Charles Archer**
- measured on an 8x8x8 partition



Conclusion

- You have been warned.
  - If you call tech support you will get asked tedious questions about the things I have outlined in this presentation.
- BG/L MPI is a moving target. Some things are going to improve over the next few months
  - flow control to handle send flood issues
  - better optimized collective performance
  - MPI I/O
- We would love to hear about your porting experience.