Optimizing MPI Collectives for X1

Howard Pritchard
(howardp@cray.com)

Outline

- X1 MPI – previous work
- X1 and MPI
- New and not so new algorithms for collective operations
- Results

CUG 2004

X1 MPI History

- Started from CRAY/SGI MPI
  - coded for RISC microprocessor
  - designed for cluster of SMPs
  - some cluster aware collective optimizations
- Simplified code by removing cluster awareness
- Replaced point-to-point MPI in collectives with AMO's and pointer exchanges

CUG 2004

X1 MPI History(2)

- **These changes helped, especially for small system sizes**
  - Eliminate scalar overhead of point-to-point
  - In some cases auto-tasking like approaches used (MPI_Bcast)

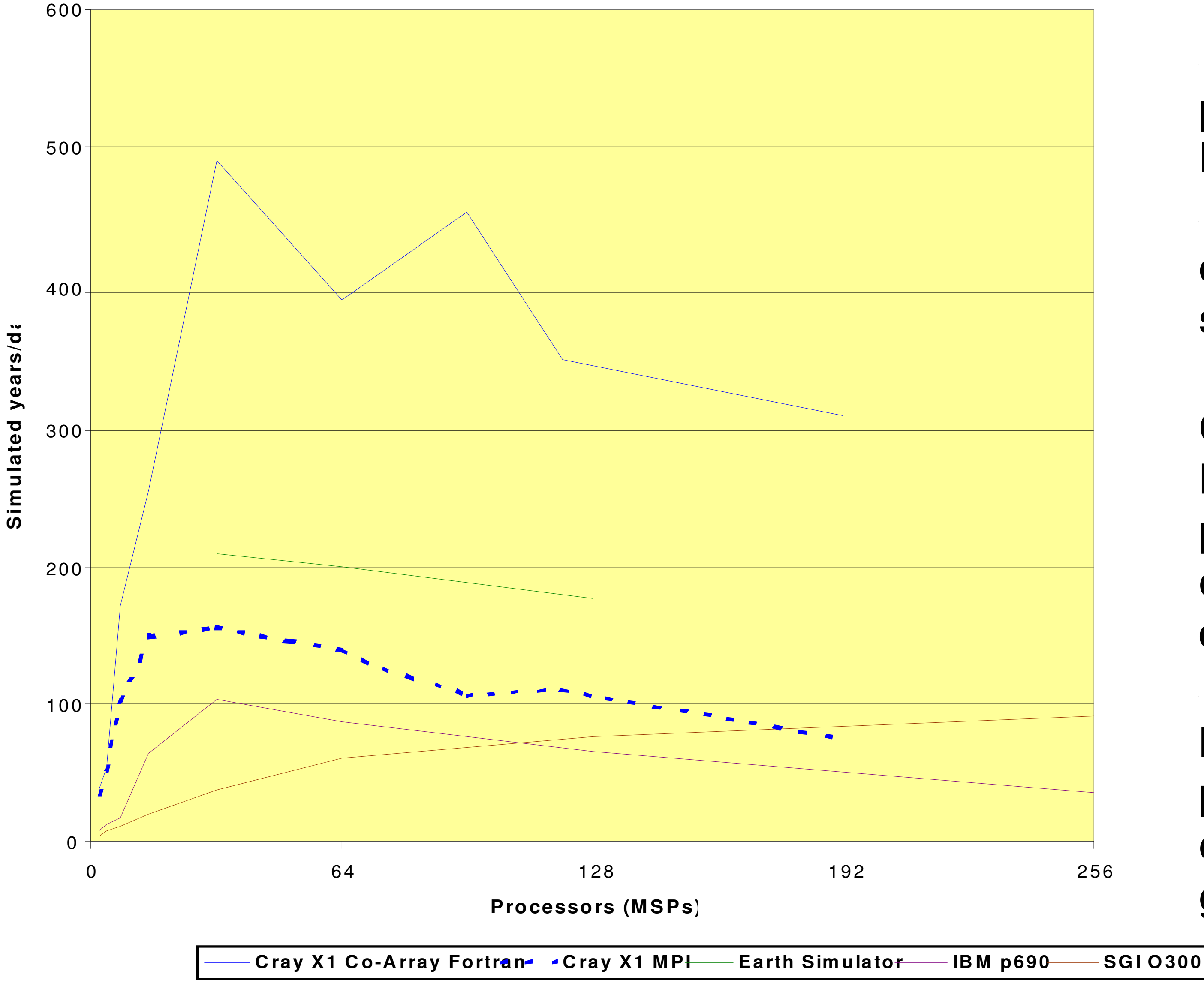
- **But some scalability problems showed up for larger system sizes**

CUG 2004

X1 MPI History(3) - Problems

- Memory hot spots owing to inefficient use of memory banks
- AMOs sometimes caused memory hot spots
- Low aggregate bandwidth in some cases

X1 MPI History(4)



- Barotropic portion of POP
- Generally does not scale well
- Cray X1 Co-Array Fortran performance is excellent
- Cray X1 MPI performance not as good

CUG 2004

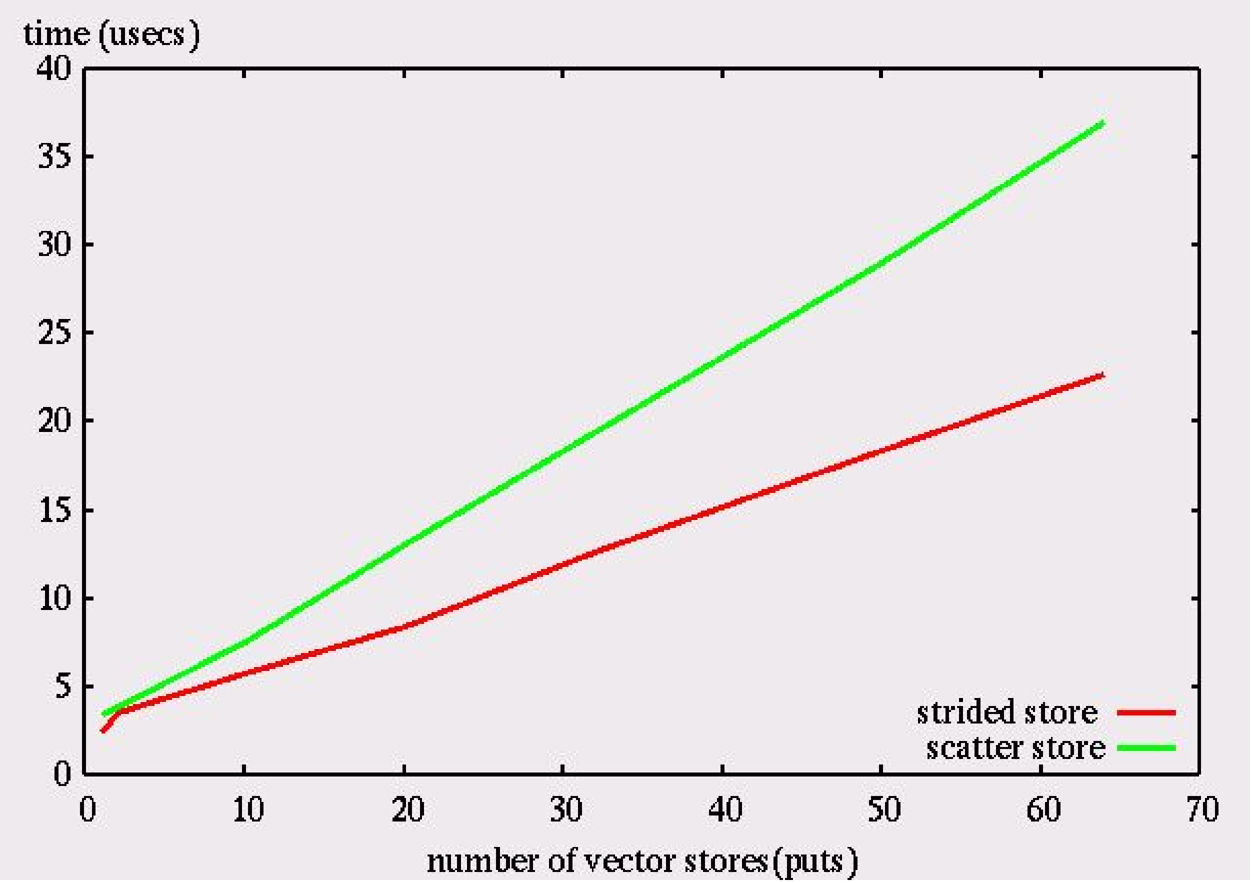
X1 and MPI – Good Things

- Distributed Memory Program Model makes things easy
  - Don't have to learn another RDMA protocol
  - No need to use buffers unless you want to, just exchange pointers
- Hundreds of outstanding load/stores. Order of magnitude(s) greater than scalar processors.
  - Able to poll vectors of values directly from memory
  - Fast vector stores across many nodes
  - Efficient cache coherency protocol

CUG 2004

X1 and MPI – Good Things(2)

Vectors let you do everything at once(almost)



Results from a kernel in which one process in an application team of 128 processes issues one or more vector stores striding across other processes followed by a *gsync* and a succeeding vector store.

X1 vector hardware lets one deliver 80 bytes to 128 process memory in less than 5 µsecs!

CUG 2004

X1 and MPI – Gotchas

- Need to pay attention to memory bank conflicts (network latency)

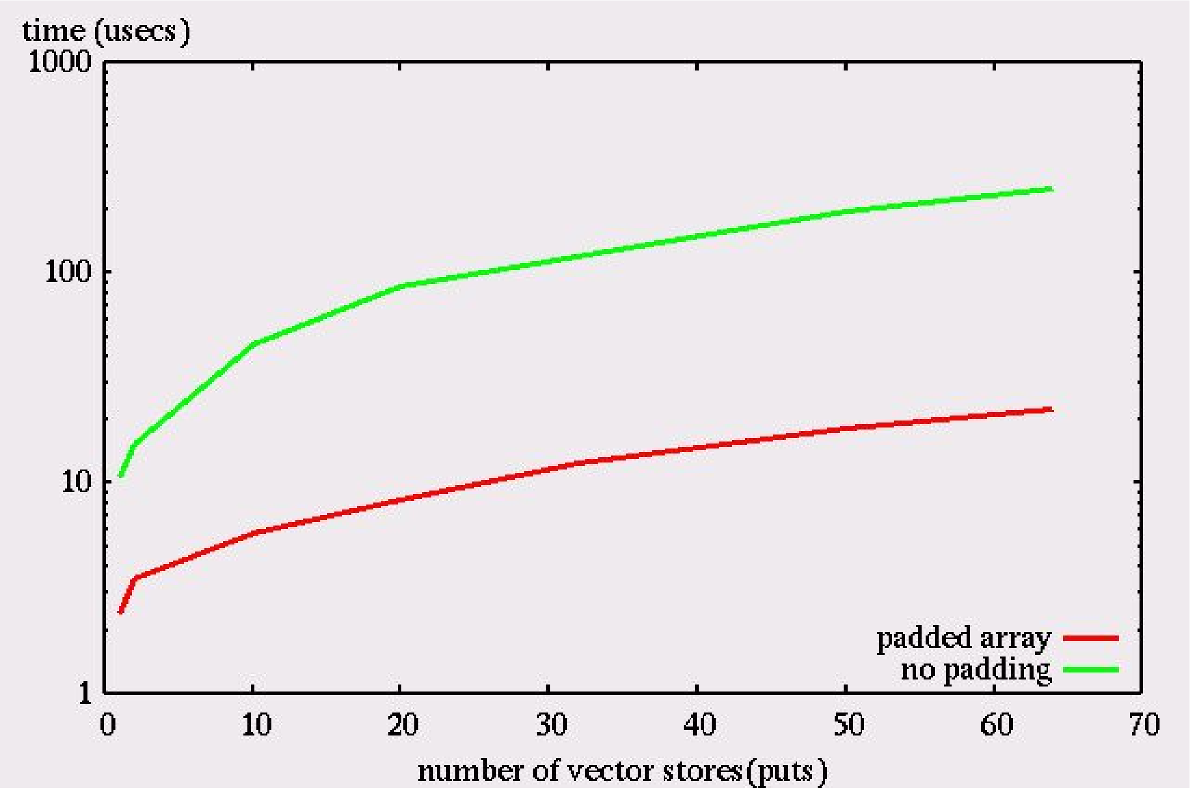
- Don't use AMO's when lots of processes are involved – no AMO cache on X1

- Only use *gsyncs* when necessary, they are expensive when lots of outstanding stores to remote memory have been issued

CUG 2004

X1 and MPI – Gotchas(2)

Memory-bank conflicts



Results from a kernel in which one process in an application team of 128 processes issues one or more vector stores striding across other processes followed by a *gsync* and a succeeding vector store.

This time the experiment is run with and without padding of the target array.

See A Guidebook to Fortran on Supercomputers by Levesque and Williamson, Academic Press (1989) section 4.8 for more info.

CUG 2004

X1 and MPI – Bad Things

- Slow scalar processor with poor branch prediction
- High function call overhead
- Particular issues with SSP mode

Strategy for Optimizations

- Optimize for short messages first
  - Avoid use of AMO's for synchronization
  - Rely on vector polling and strided or scatter puts
- Better traditional algorithms for longer messages for some operations

Optimizing Short Messages

- Use insights from applications analysts' and benchmarkers' CoArray (CAF) workarounds for MPI problems

- Double buffering with padding to reduce synchronization and avoid memory bank conflicts. Data structures associated with MPI internal communicator structure

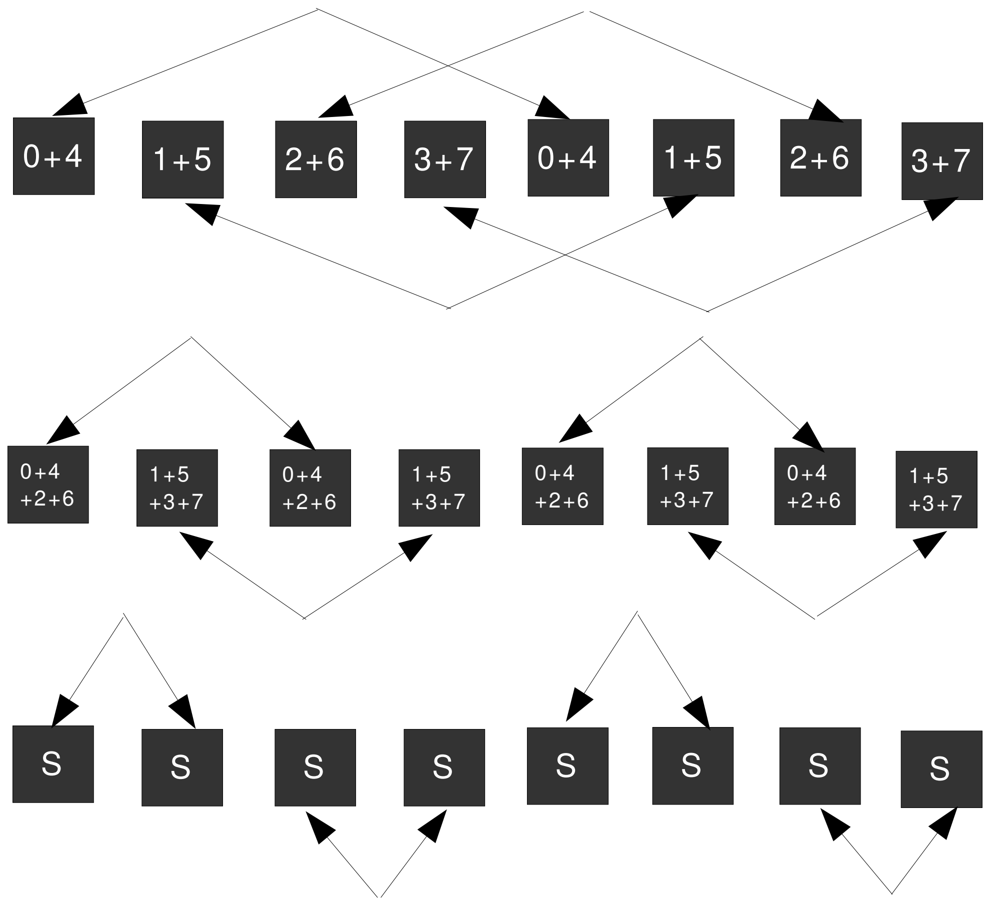
- Use arrays of pointer functions to cut down on branches

CUG 2004

Optimizing Short Messages(2)

Most MPI implementations focus on minimum startup cost for collectives involving short messages:

MPI_Allreduce

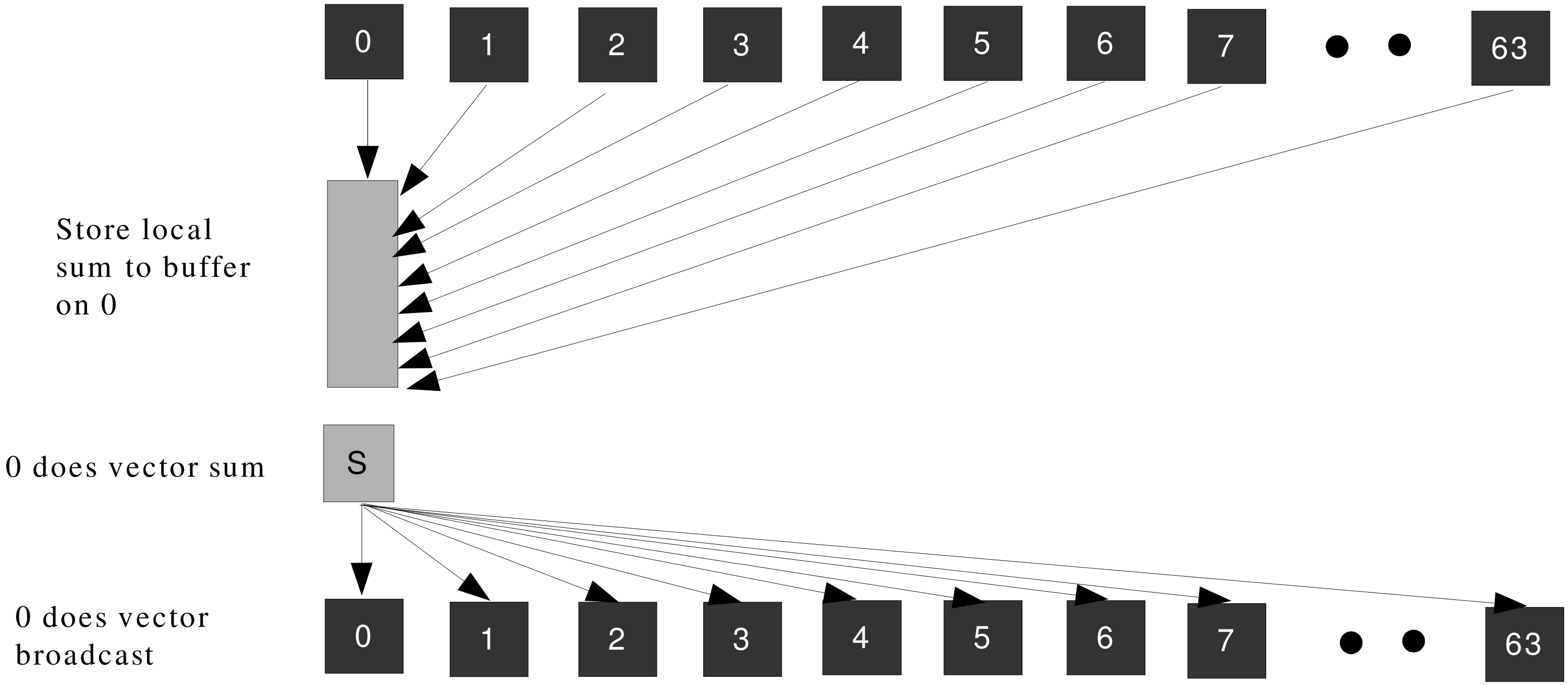


No vector content here, multiple synchronizations.

Optimizing Short Messages(3)

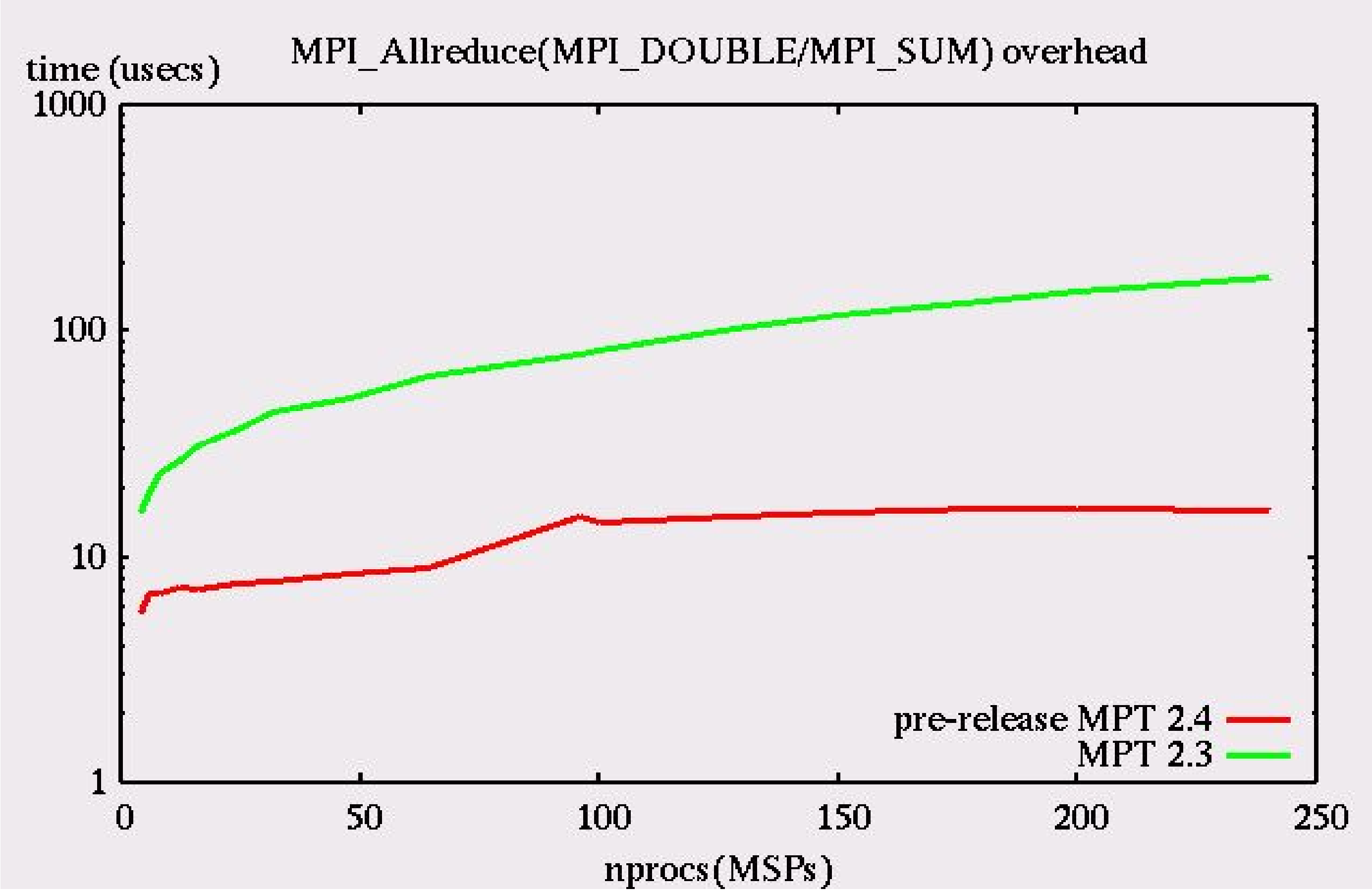
For X1 short message collectives are best treated as a vectorization problem – with vectorization over the process dimension

MPI_Allreduce (up to 64 ranks)



MPI_Allreduce (short message)

Vector algorithm is over an order of magnitude faster at higher process counts



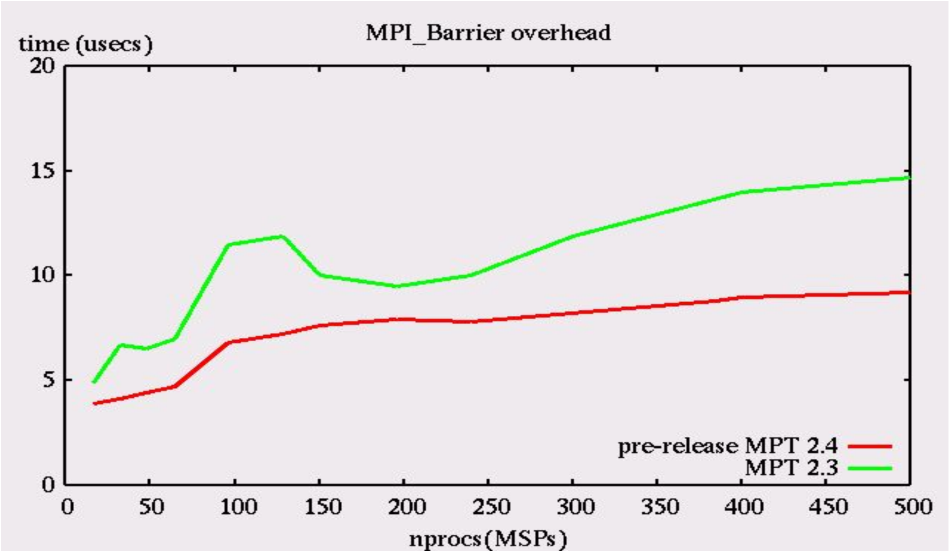
Optimizing Short Messages(4)

- Optimized Mellor-Crummey & Scott(MCS) tree barrier (radix 64) for MPI_Barrier and internal barrier for use in collectives -doesn't use AMOs [MCS]

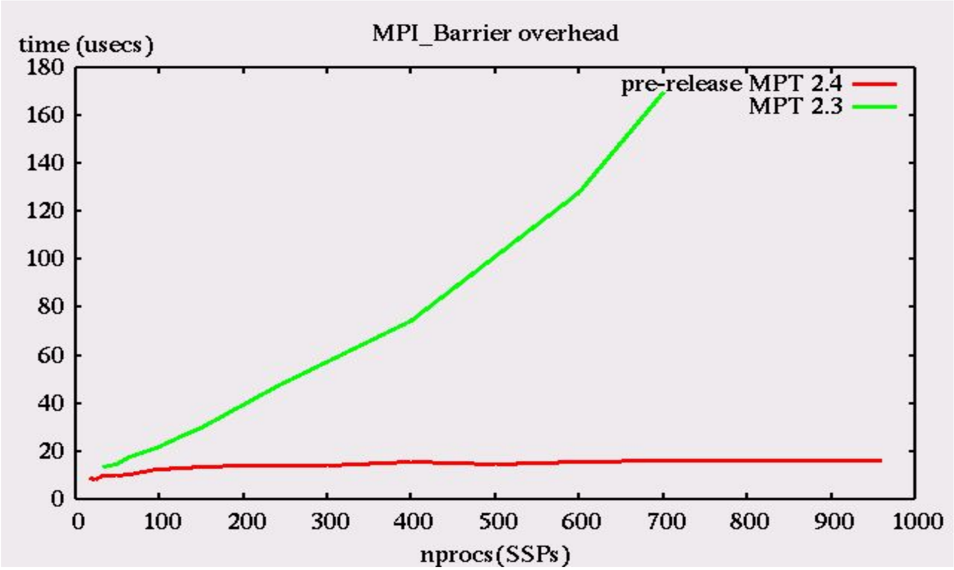
- Optimized internal MPI_Allgather and MPI_Alltoall for short messages to enable efficient gathering of pointers, datatypes, etc. for use in medium and long length operations

CUG 2004

MPI_Barrier

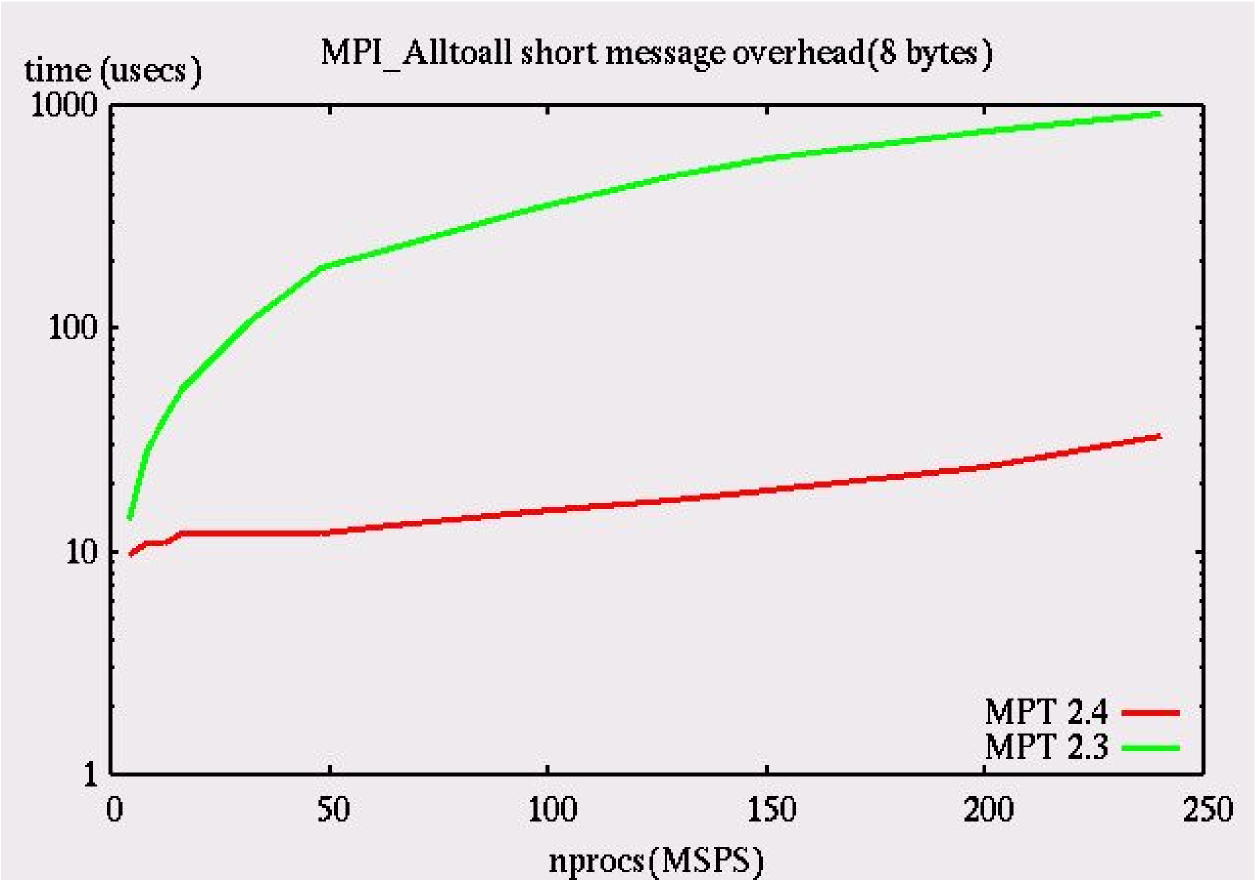


Variants of the MCS tree barrier are under investigation for MPI_Barrier, MPI_Win_fence, and shmem_barrier_all



CUG 2004

MPI_Alltoall (short message)



CUG 2004

Optimizing for longer messages

- Use optimized barrier and short message allgather and alltoall to avoid AMO base synchronization
- Use *put* rather than *get* approach for moving data, gives better bandwidth
- Stream over target rank for medium length messages (MSP mode library)

MPI_Allreduce - longer vectors

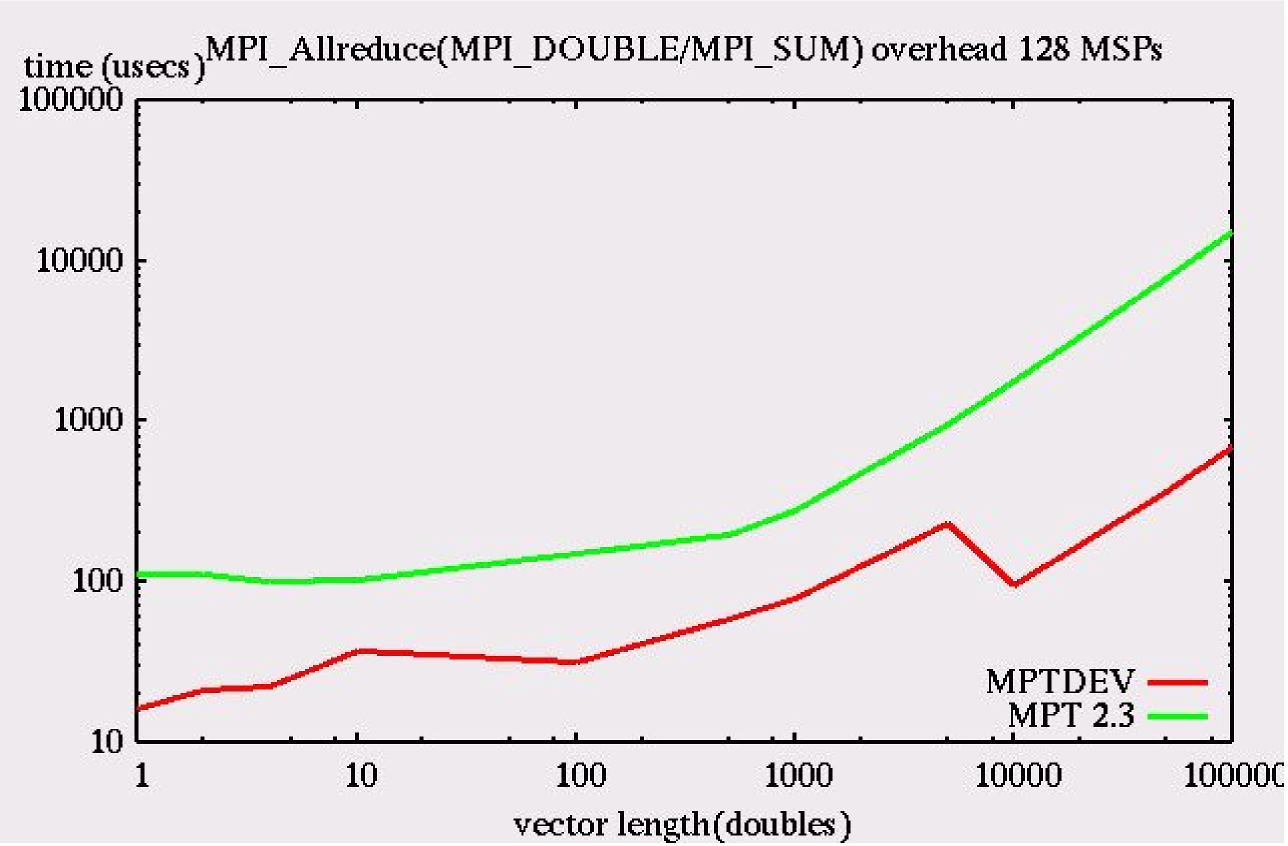
- Vectors greater than 32 bytes but 128 or fewer vector elements use binary tree algorithm with buffers associated with communicator to reduce synchronization requirements

- Vectors for which *nelements/nranks* $\leq$ *64* use binary tree with application buffers

- For vectors with *nelements/nranks* > 64, use a reduce-scatter/gather algorithm [geijn,rab]

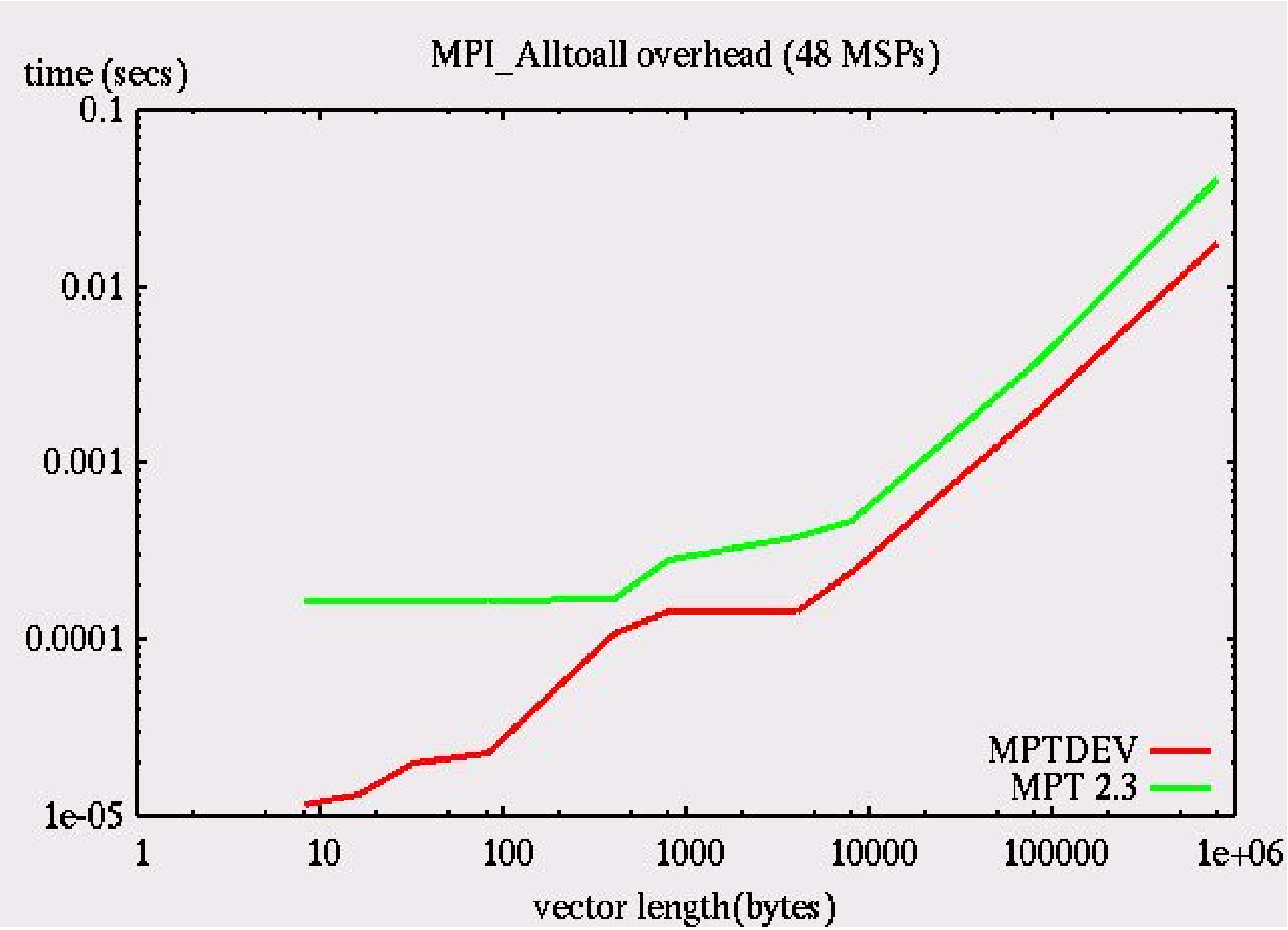
CUG 2004

MPI_Allreduce -longer vectors(2)



Better switchover criteria to reduce-scatter/gather approach are being investigated.

MPI_Alltoall - longer vectors



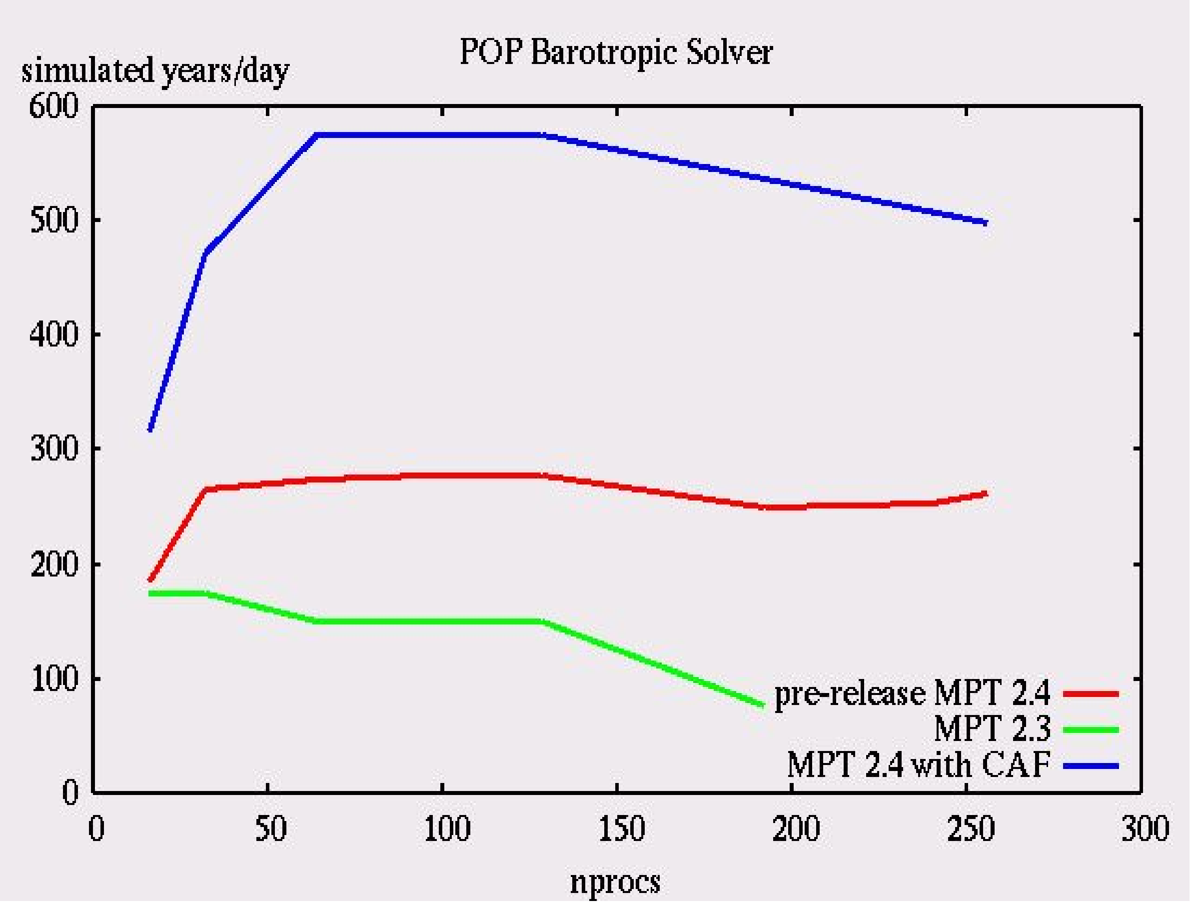
CUG 2004

Application Results

- **Many times a timing profile that shows a lot of time being spent in MPI is NOT an MPI problem – especially for collective calls**

- **Use PAT and a MPI trace library like FMPI to make sure there are no load balance problems – an application which is load balanced on a Beowulf cluster may <u>not</u> be load balanced on X1**

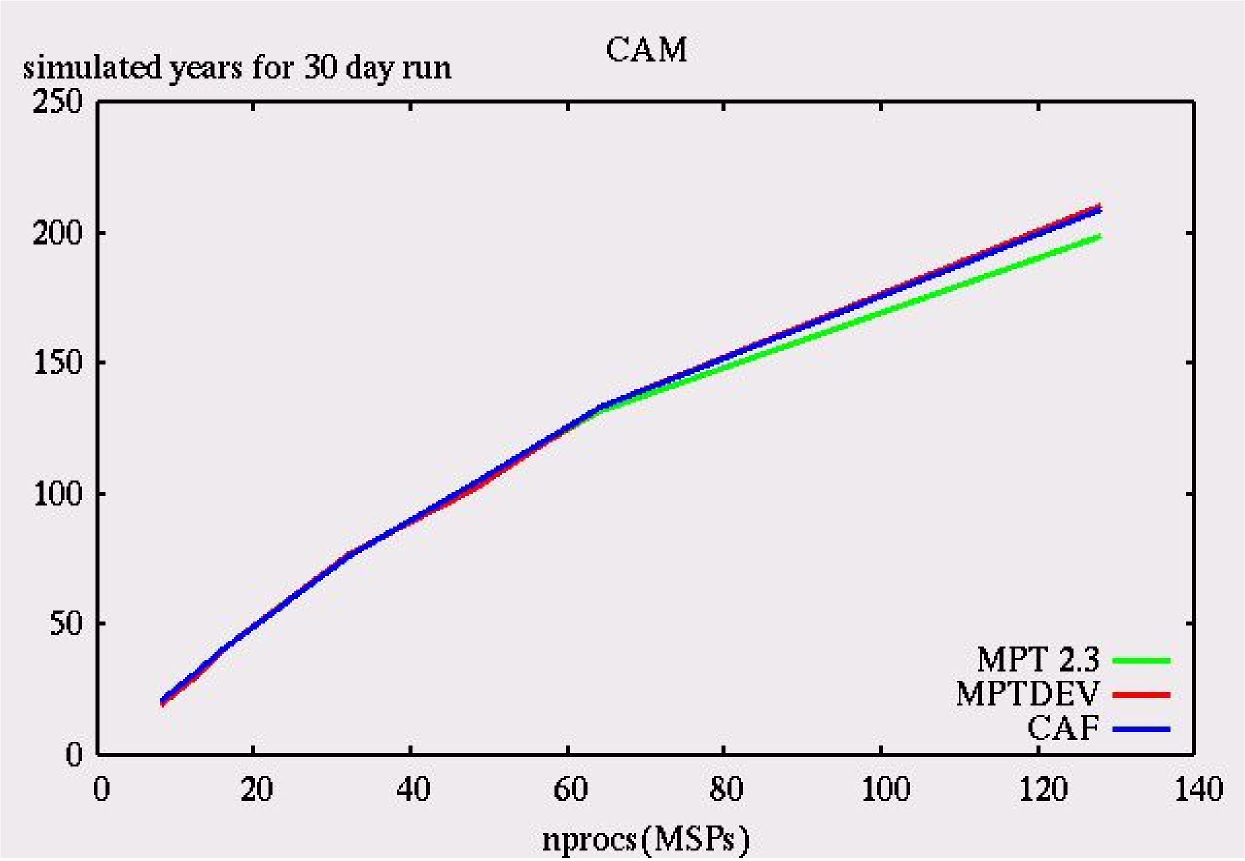
POP Results



The allreduce in CAF and MPT 2.4 are very similar. CAF benefits a lot from inlining of the global sum into the solver.

CUG 2004

CAM Results



New MPI_Allgatherv/MPI_Alltoallv with streaming give similar performance to a CAF version of CAM.

CUG 2004

Release Process

- Use an internal development library to test new algorithms (MPTDEV)
- After testing, algorithms are integrated into the MPT 2.4 pre-release tree
- Selected mods are pushed back into MPT 2.3 release version
- MPT 2.4 planned for release in fall '04 – all collectives optimized in this release

CUG 2004

What's Next?

- Improvements in point-to-point latencies for X1 MPI

- Cray-RS collectives?

References

[mcs] J. H. Mellor-Crummey, and M. L. Scott. Algorithms for scalable synchronization on shared memory multiprocessors. ACM Trans. Computer Systems. Vol 9, No. 1(1991), pp 21-65.

[geijn] M. Barnett, L. Shuler, S. Gupta, D. Payne, R. van de Geijn, and J. Watts. Building a high-performance collective communication library. Proceedings of Supercomputing 1994, pp. 107-116.

[rab] R. Rabenseifner. A new optimized MPI reduce algorithm. High-Performance Computing-Center, Univ. of Stuttgart, Nov. 1997. *http://www.hlrs.de/mpi/myreduce.html.*