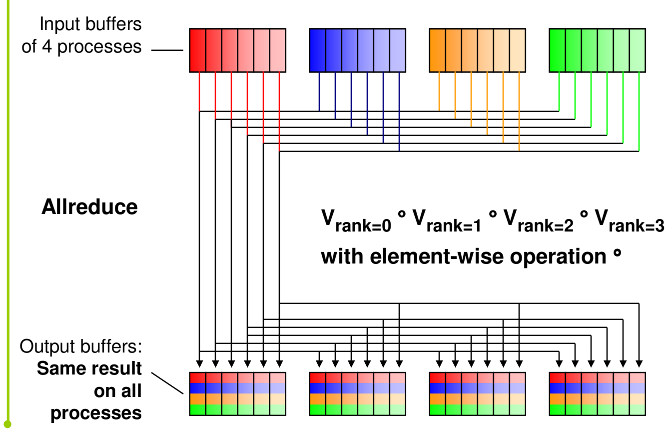
Optimization of MPI_Allreduce and MPI_Reduce

Rolf Rabenseifner, Panos Adamidis
{rabenseifner, adamidis}@hlrs.de

University of Stuttgart
High-Performance Computing-Center Stuttgart (HLRS)
www.hlrs.de

What do we want to do



Basic Principles

Principle I

- Different optimizations for latency and bandwidth
- Latency optimization, e.g.,
  – sending the full input buffers to all processors
  – executing the reduction on all processors
- Bandwidth optimization:
  – splitting the input buffers
  – transferring cross-wise between processes
  – reduction operation only on partial buffers
  – allgather step at the end

(splitting, transferring and reduction steps: i.e., reduce_scattter)

Basic Principles

Principle II

- In case where the number of processors is a power-of-two, then optimization is possible by buffer halving and distance doubling
- In case where the number of processors is non-power-of-two, various algorithms are shown.

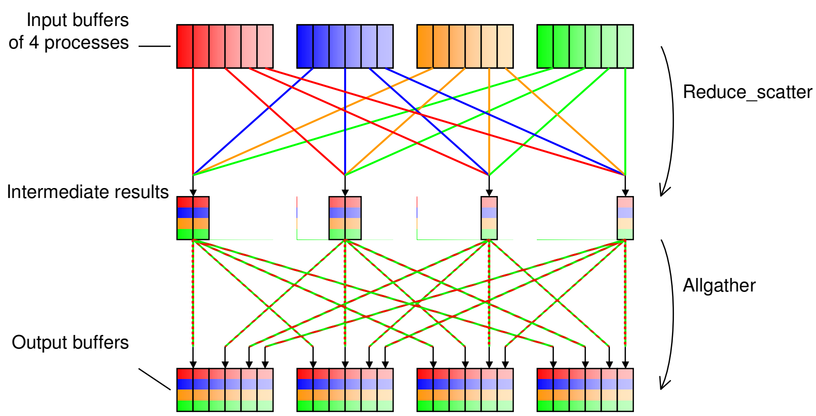
Backround

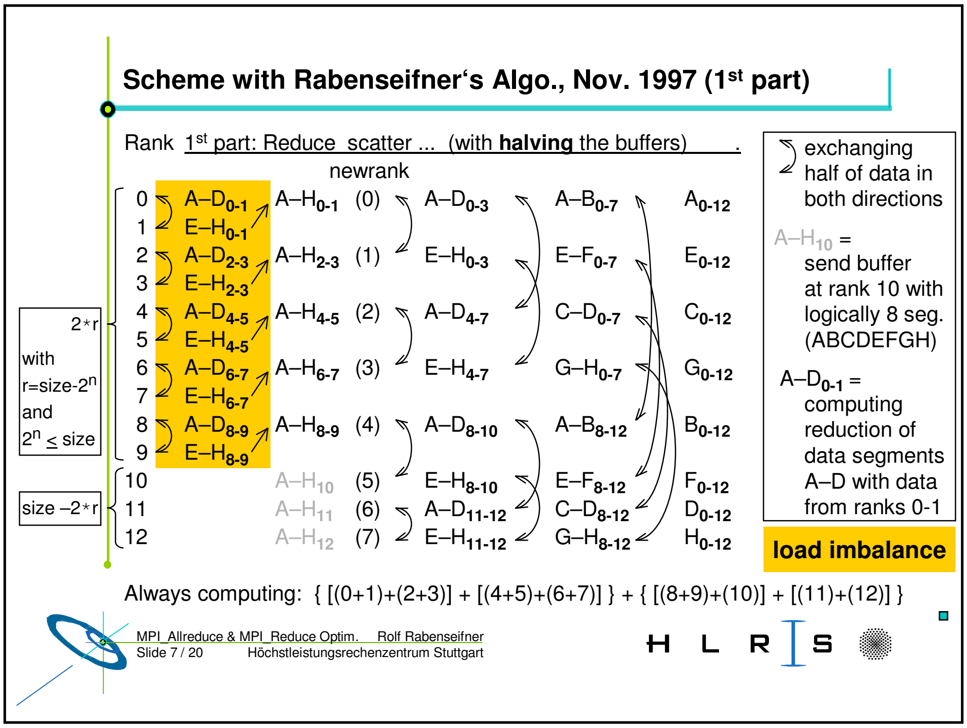
- **37%** of MPI time in **MPI_Allreduce**
- **25%** of user time with **non-power-of-two** number of processes
  – data from automatic profiling of all customers on HLRS CRAY T3E

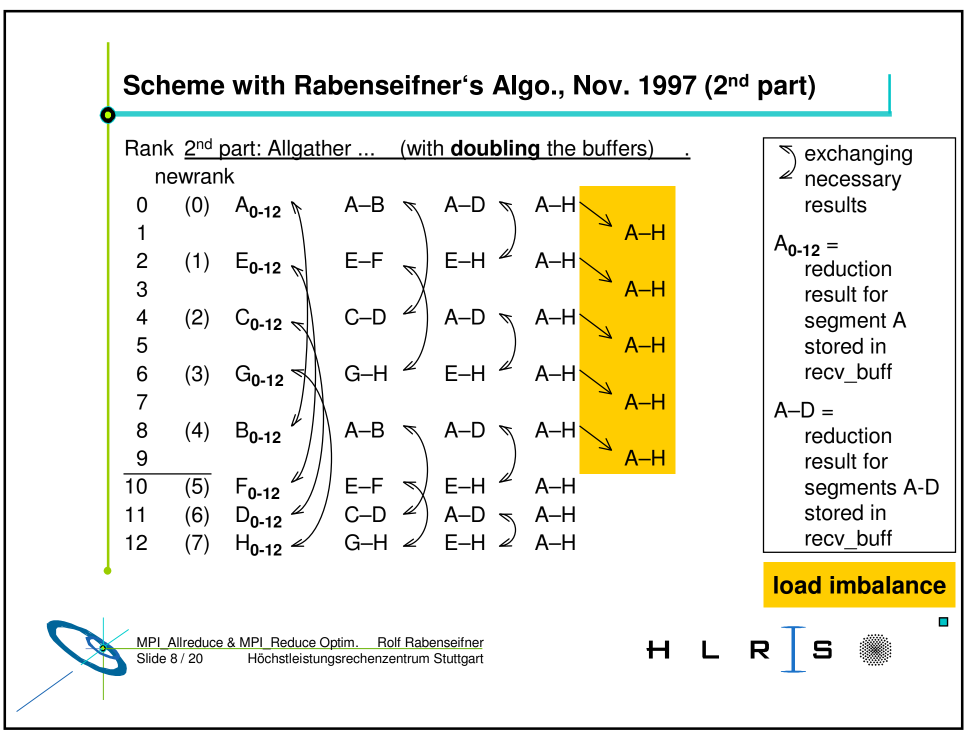
Rabenseifner's Algo., Nov. 1997

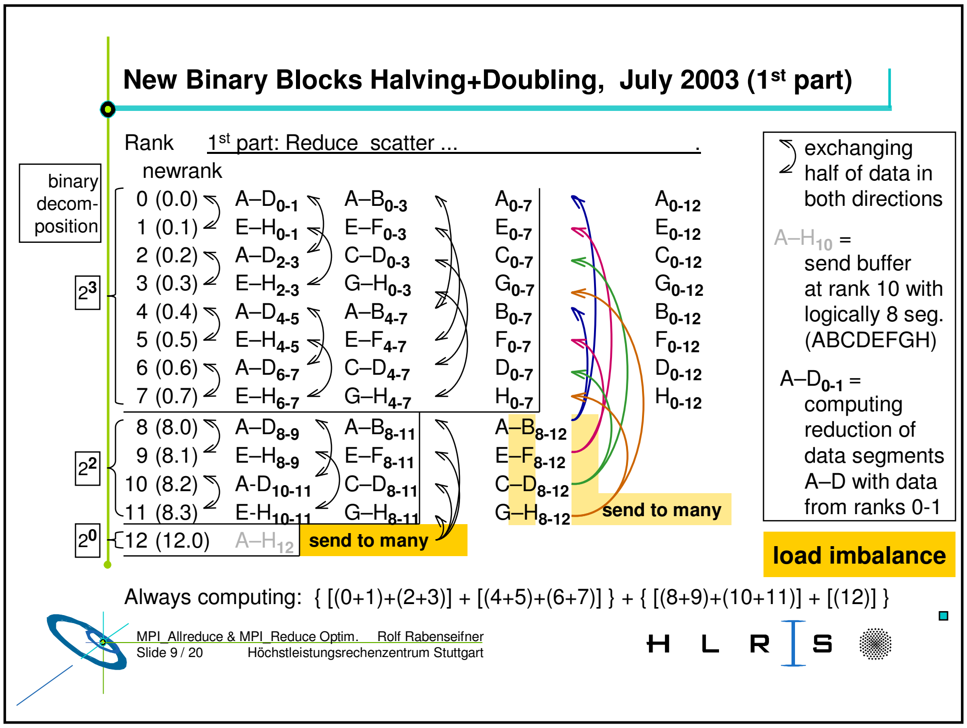
- Standard algorithm used in mpich1:
  - MPI_Reduce = binomial tree
  - MPI_Allreduce = binomial tree + MPI_Bcast
  - Binomial tree is inefficient
    - logarithmic behavior but in each iteration,
      half of the processes **gets inactive** ➔ **bad load balancing**
- Better algorithms (butterfly-algorithms):
  - MPI_Reduce = Reduce_scatter + Gather
  - MPI_Allreduce = Reduce_scatter + Allgather

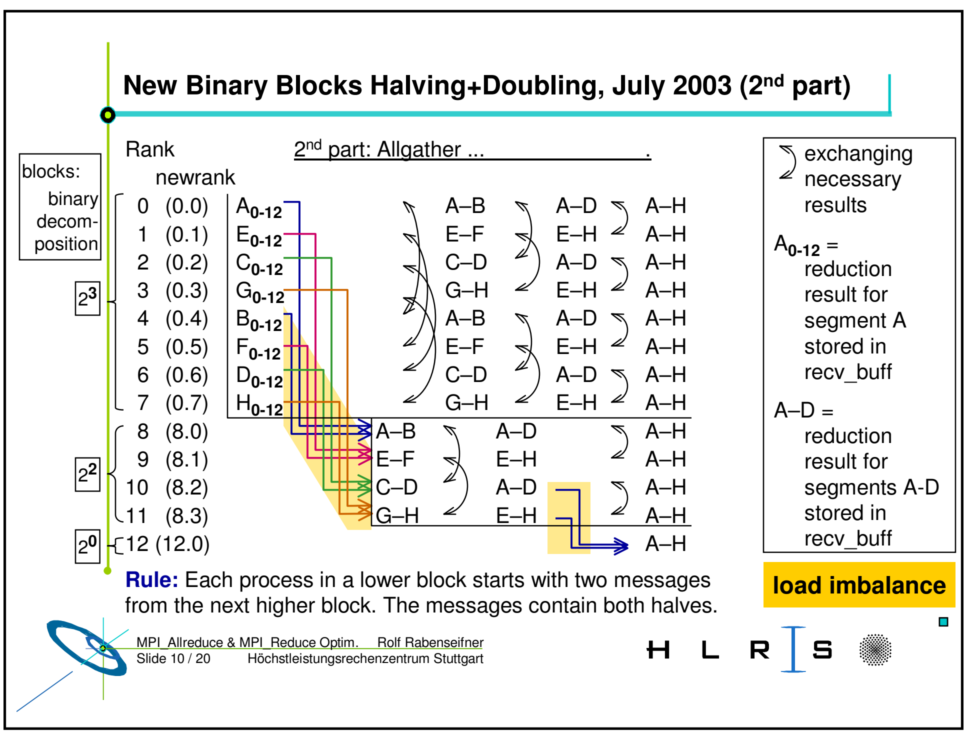
Reduce_scatter and Allgather




Scheme with Rabenseifner's Algo., Nov. 1997 (1st part)
Rank 1st part: Reduce scatter ... (with halving the buffers)
newrank
0 A–D0-1 A–H0-1 (0) A–D0-3 A–B0-7 A0-12
1 E–H0-1
2 A–D2-3 A–H2-3 (1) E–H0-3 E–F0-7 E0-12
3 E–H2-3
4 A–D4-5 A–H4-5 (2) A–D4-7 C–D0-7 C0-12
5 E–H4-5
6 A–D6-7 A–H6-7 (3) E–H4-7 G–H0-7 G0-12
7 E–H6-7
8 A–D8-9 A–H8-9 (4) A–D8-10 A–B8-12 B0-12
9 E–H8-9
10 A–H10 (5) E–H8-10 E–F8-12 F0-12
11 A–H11 (6) A–D11-12 C–D8-12 D0-12
12 A–H12 (7) E–H11-12 G–H8-12 H0-12
2*r
with
r=size-2n
and
2n ≤ size
size –2*r
exchanging half of data in both directions
A–H10 = send buffer at rank 10 with logically 8 seg. (ABCDEFGH)
A–D0-1 = computing reduction of data segments A–D with data from ranks 0-1
load imbalance
Always computing: { [(0+1)+(2+3)] + [(4+5)+(6+7)] } + { [(8+9)+(10)] + [(11)+(12)] }
MPI_Allreduce & MPI_Reduce Optim. Rolf Rabenseifner
Slide 7 / 20 Höchstleistungsrechenzentrum Stuttgart
H L R S


Scheme with Rabenseifner's Algo., Nov. 1997 (2nd part)
Rank 2nd part: Allgather ... (with doubling the buffers)
newrank
0 (0) A0-12 A–B A–D A–H A–H
1
2 (1) E0-12 E–F E–H A–H A–H
3
4 (2) C0-12 C–D A–D A–H A–H
5
6 (3) G0-12 G–H E–H A–H A–H
7
8 (4) B0-12 A–B A–D A–H A–H
9
10 (5) F0-12 E–F E–H A–H
11 (6) D0-12 C–D A–D A–H
12 (7) H0-12 G–H E–H A–H
exchanging necessary results
A0-12 = reduction result for segment A stored in recv_buff
A–D = reduction result for segments A-D stored in recv_buff
load imbalance
MPI_Allreduce & MPI_Reduce Optim. Rolf Rabenseifner
Slide 8 / 20 Höchstleistungsrechenzentrum Stuttgart
H L R S

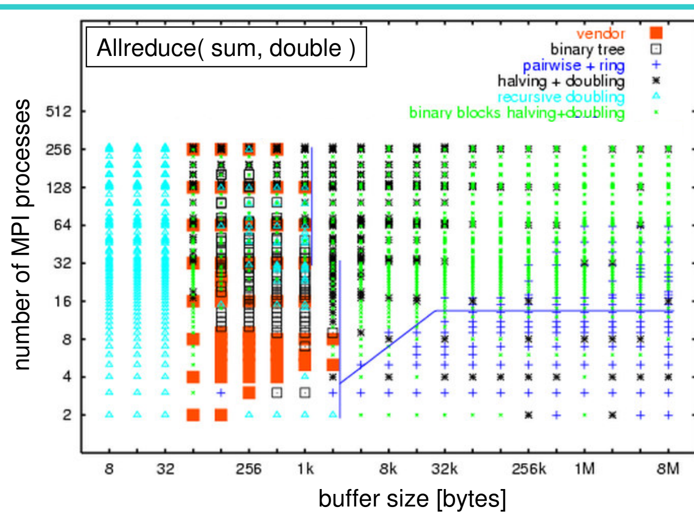

New Binary Blocks Halving+Doubling, July 2003 (1st part)
Rank
1st part: Reduce scatter ...
newrank
binary decom-position
2^3
2^2
2^0
0 (0.0)
1 (0.1)
2 (0.2)
3 (0.3)
4 (0.4)
5 (0.5)
6 (0.6)
7 (0.7)
8 (8.0)
9 (8.1)
10 (8.2)
11 (8.3)
12 (12.0)
A–D$_{0-1}$ E–H$_{0-1}$ A–D$_{2-3}$ E–H$_{2-3}$ A–D$_{4-5}$ E–H$_{4-5}$ A–D$_{6-7}$ E–H$_{6-7}$ A–D$_{8-9}$ E–H$_{8-9}$ A-D$_{10-11}$ E–H$_{10-11}$ A–H$_{12}$
A–B$_{0-3}$ E–F$_{0-3}$ C–D$_{0-3}$ G–H$_{0-3}$ A–B$_{4-7}$ E–F$_{4-7}$ C–D$_{4-7}$ G–H$_{4-7}$ A–B$_{8-11}$ E–F$_{8-11}$ C–D$_{8-11}$ G–H$_{8-11}$
A$_{0-7}$ E$_{0-7}$ C$_{0-7}$ G$_{0-7}$ B$_{0-7}$ F$_{0-7}$ D$_{0-7}$ H$_{0-7}$ A–B$_{8-12}$ E–F$_{8-12}$ C–D$_{8-12}$ G–H$_{8-12}$
A$_{0-12}$ E$_{0-12}$ C$_{0-12}$ G$_{0-12}$ B$_{0-12}$ F$_{0-12}$ D$_{0-12}$ H$_{0-12}$
send to many
send to many
exchanging half of data in both directions
A–H$_{10}$ = send buffer at rank 10 with logically 8 seg. (ABCDEFGH)
A–D$_{0-1}$ = computing reduction of data segments A–D with data from ranks 0-1
load imbalance
Always computing: { [(0+1)+(2+3)] + [(4+5)+(6+7)] } + { [(8+9)+(10+11)] + [(12)] }
MPI_Allreduce & MPI_Reduce Optim. Rolf Rabenseifner
Slide 9 / 20 Höchstleistungsrechenzentrum Stuttgart
H L R S


New Binary Blocks Halving+Doubling, July 2003 (2nd part)
Rank
2nd part: Allgather ...
newrank
blocks: binary decom-position
2^3
2^2
2^0
0 (0.0)
1 (0.1)
2 (0.2)
3 (0.3)
4 (0.4)
5 (0.5)
6 (0.6)
7 (0.7)
8 (8.0)
9 (8.1)
10 (8.2)
11 (8.3)
12 (12.0)
A$_{0-12}$ E$_{0-12}$ C$_{0-12}$ G$_{0-12}$ B$_{0-12}$ F$_{0-12}$ D$_{0-12}$ H$_{0-12}$
A–B E–F C–D G–H A–B E–F C–D G–H A–B E–F C–D G–H
A–D E–H A–D E–H A–D E–H A–D E–H A–D E–H A–D E–H
A–H
exchanging necessary results
A$_{0-12}$ = reduction result for segment A stored in recv_buff
A–D = reduction result for segments A-D stored in recv_buff
load imbalance
Rule: Each process in a lower block starts with two messages from the next higher block. The messages contain both halves.
MPI_Allreduce & MPI_Reduce Optim. Rolf Rabenseifner
Slide 10 / 20 Höchstleistungsrechenzentrum Stuttgart
H L R S

Compared Protocols

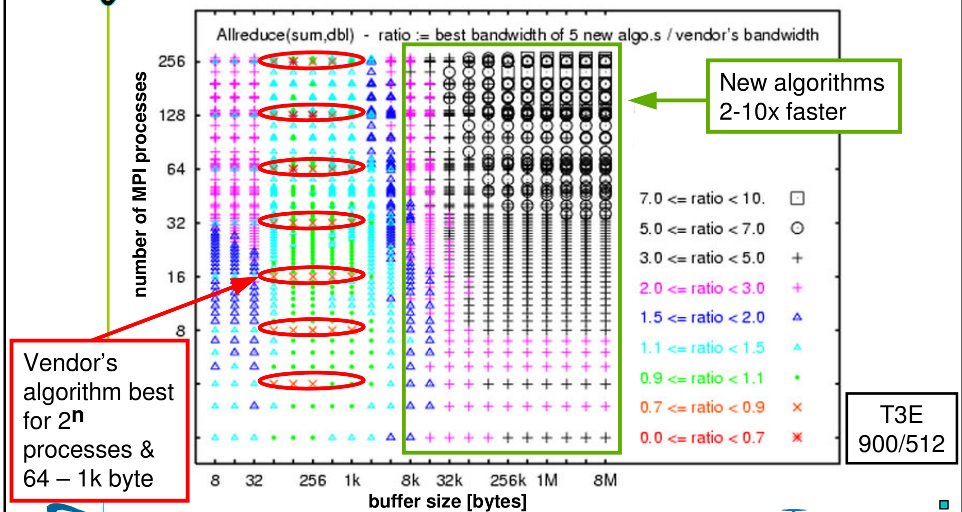
- **Vendor** (MPI_Allreduce and MPI_Reduce of the used MPI library)
- **Binomial tree** + Bcast (i.e., without latency optimization)
- **Recursive doubling** with full buffers (i.e., with latency optimization)
- *Reduce_scatter + Allgather (or Gather)*
  - **Pairwise & Ring**
    - input buffer is devided into (#proc.) pieces of same size
    - optimal load balance but high latency
      O(2x #processes) + O(2x vector size)
  - **Halving & Doubling**
    - O(2x ⌈log(#processes)⌉) + O(4x vector size)
  - **Binary Blocks Based Halving & Doubling**
    - normally better than halving & doubling
    - except for special #processes, e.g. 17, 33, 65,....

Comparison: Fastest Protocol on T3E 900/512

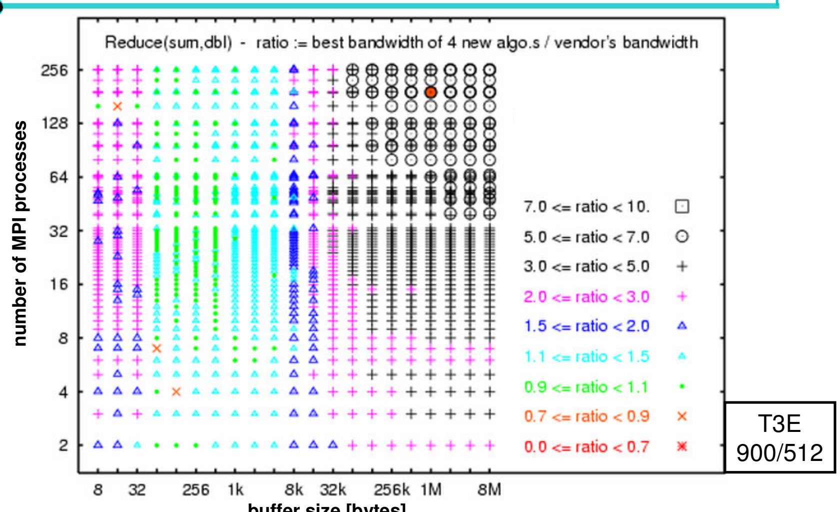


Benchmarks on T3E 900/512, sum of doubles, bandwidth := buffersize / wallclock time

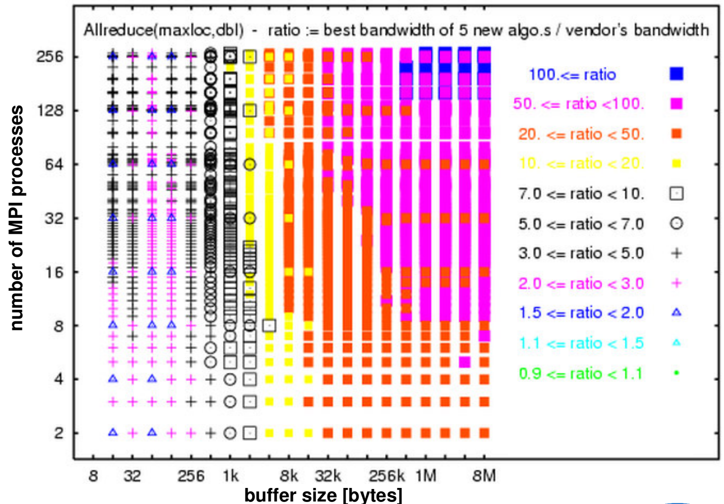
Best algorithm more than x7 faster than vendor's



Reduce: One of the 4 algorithms is always the best

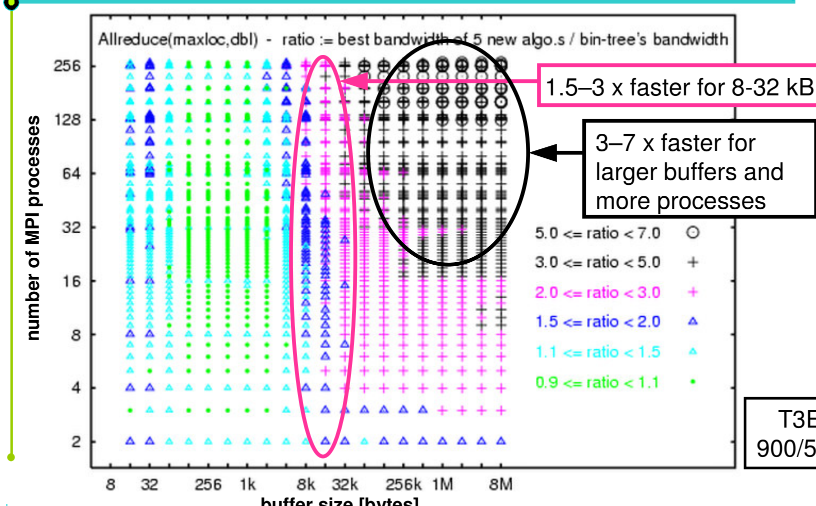


MAXLOC: Vendor's MPI_DOUBLE_INT is extreme slow



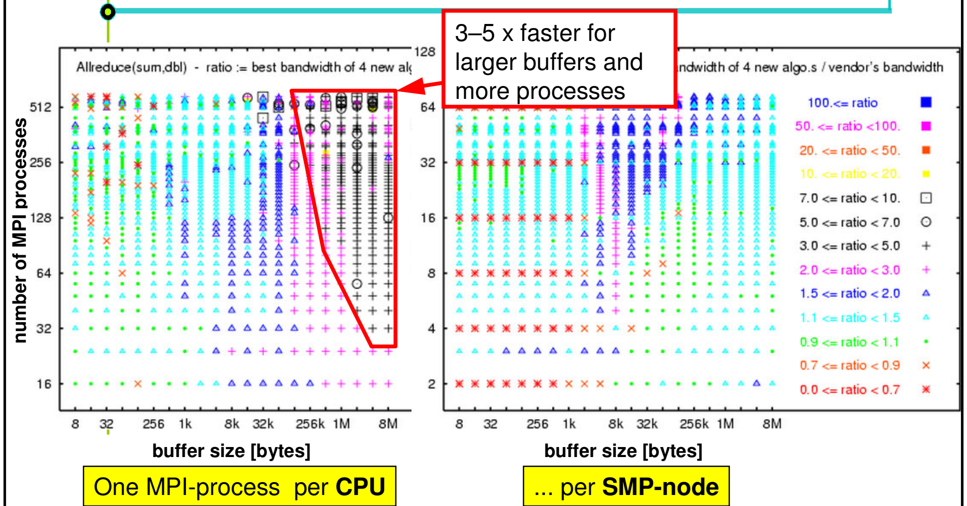
T3E 900/512

MAXLOC: Comparing with optimized binomial-tree



T3E 900/512

MPI_Allreduce on IBM SP at SDSC



Real Benefit

- depends on real usage-pattern
  - # calls
  - # bytes (bandwidth vs. latency)
  - # processes (power-of-two?)
  - operation (MPI_SUM, MPI_MAX, MPI_MAXLOC, user-defined, …)
  - Allreduce / Reduce

- Cray X1
  - code is developed on T3E
  - code is tested on CRAY Opteron clusters, Linux clusters, IA32/IA64, IBM
  - code has a problem on Cray X1, need more time for debugging
  - first result on X1:
    - MAXLOC problem with Cray-MPI: **same as on T3E**

Acknowledgments

- Thanks for helpful discussions
  - Rajeev Thakur (Argonne)
  - Jesper Larsson Träff (NEC)
- Thanks for help with benchmarking
  - Gerhard Wellein (Uni. Erlangen)
  - Thomas Ludwig, Ana Kovatcheva (Uni. Heidelberg)
  - Rajeev Thakur (Argonne)
  - Monika Wierse, Andy Mason (Cray)
  - Patrick H. Worley (ORNL)
  - Terry Hewitt, Mike Pettipher, Adrian Tate (Uni. Manchester)

Conclusion & Future Work

- Latency & Bandwidth optimization of MPI_Allreduce and MPI_Reduce is
  - possible
  - important
  - the '97 algorithm is now part of mpich
- Future work:
  Integrated algorithm under construction
  - smooth optimization for any vector size
  - nearly optimal for any # processes
  - again significantly better bandwidth for non-power-of-two