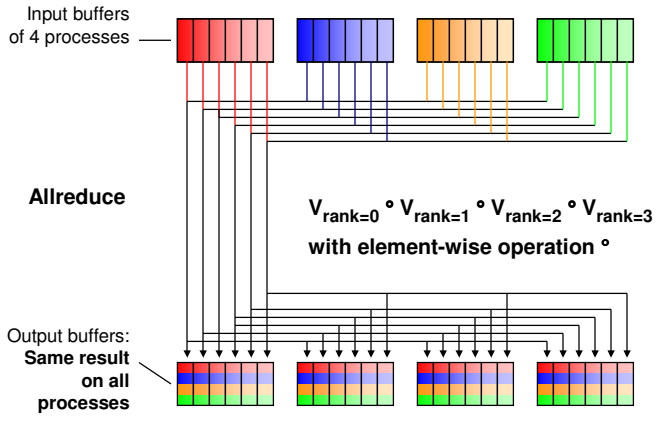
# Optimization of Collective Reduction Operations

Rolf Rabenseifner
**rabenseifner@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

} 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

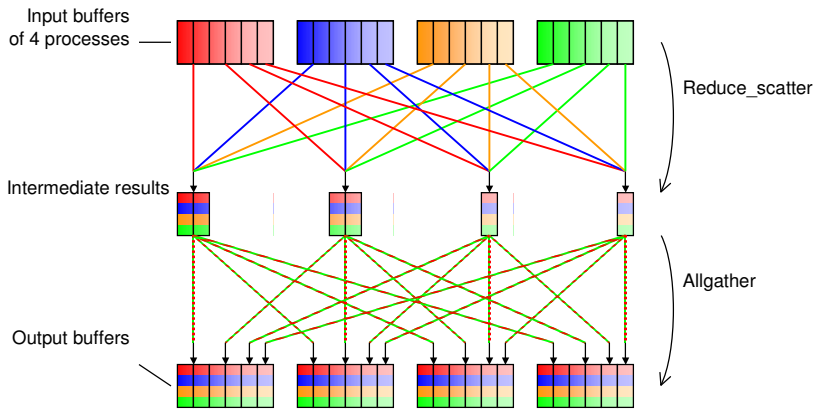
- **37%** of MPI time in **MPI_Allreduce**
- **25%** of user time with **non-power-of-two** number of processes
  - data from automatic profiling of all customers on HLRS CRAY T3E

## 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 | | $A–D_{0-1}$ | $A–H_{0-1}$ | (0) | $A–D_{0-3}$ | $A–B_{0-7}$ | $A_{0-12}$ |
| 1 | | $E–H_{0-1}$ | | | | | |
| 2 | | $A–D_{2-3}$ | $A–H_{2-3}$ | (1) | $E–H_{0-3}$ | $E–F_{0-7}$ | $E_{0-12}$ |
| 3 | | $E–H_{2-3}$ | | | | | |
| 4 | | $A–D_{4-5}$ | $A–H_{4-5}$ | (2) | $A–D_{4-7}$ | $C–D_{0-7}$ | $C_{0-12}$ |
| 5 | | $E–H_{4-5}$ | | | | | |
| 6 | | $A–D_{6-7}$ | $A–H_{6-7}$ | (3) | $E–H_{4-7}$ | $G–H_{0-7}$ | $G_{0-12}$ |
| 7 | | $E–H_{6-7}$ | | | | | |
| 8 | | $A–D_{8-9}$ | $A–H_{8-9}$ | (4) | $A–D_{8-10}$ | $A–B_{8-12}$ | $B_{0-12}$ |
| 9 | | $E–H_{8-9}$ | | | | | |
| 10 | | | $A–H_{10}$ | (5) | $E–H_{8-10}$ | $E–F_{8-12}$ | $F_{0-12}$ |
| 11 | | | $A–H_{11}$ | (6) | $A–D_{11-12}$ | $C–D_{8-12}$ | $D_{0-12}$ |
| 12 | | | $A–H_{12}$ | (7) | $E–H_{11-12}$ | $G–H_{8-12}$ | $H_{0-12}$ |

$2*r$ with $r=size-2^n$ and $2^n \leq size$

$size-2*r$

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)] }



## Scheme with Rabenseifner's Algo., Nov. 1997 (2nd part)

Rank 2nd part: Allgather ... (with **doubling** the buffers) .

| newrank | | | | | | |
|---|---|---|---|---|---|---|
| 0 | (0) | $A_{0-12}$ | A–B | A–D | A–H | A–H |
| 1 | | | | | | |
| 2 | (1) | $E_{0-12}$ | E–F | E–H | A–H | A–H |
| 3 | | | | | | |
| 4 | (2) | $C_{0-12}$ | C–D | A–D | A–H | A–H |
| 5 | | | | | | |
| 6 | (3) | $G_{0-12}$ | G–H | E–H | A–H | A–H |
| 7 | | | | | | |
| 8 | (4) | $B_{0-12}$ | A–B | A–D | A–H | A–H |
| 9 | | | | | | |
| 10 | (5) | $F_{0-12}$ | E–F | E–H | A–H | |
| 11 | (6) | $D_{0-12}$ | C–D | A–D | A–H | |
| 12 | (7) | $H_{0-12}$ | G–H | E–H | A–H | |

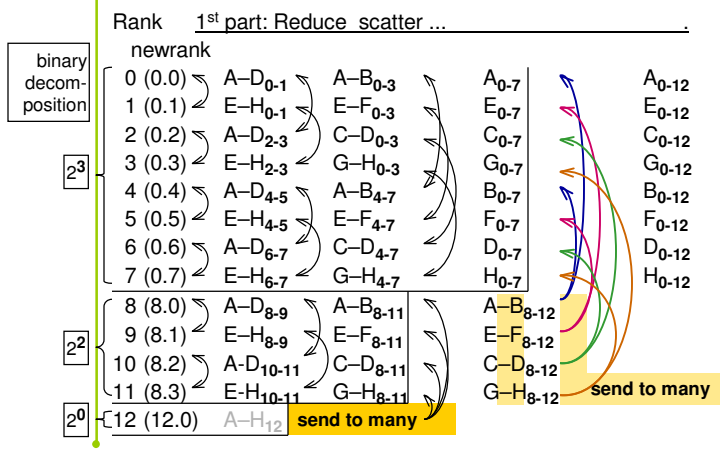
exchanging necessary results

$A_{0-12}$ = reduction result for segment A stored in recv_buff

A–D = reduction result for segments A-D stored in recv_buff

**load imbalance**

## New Binary Blocks Halving+Doubling, July 2003 (1st part)




exchanging half of data in both directions

$A–H_{10}$ = send buffer at rank 10 with logically 8 seg. (ABCDEFGH)

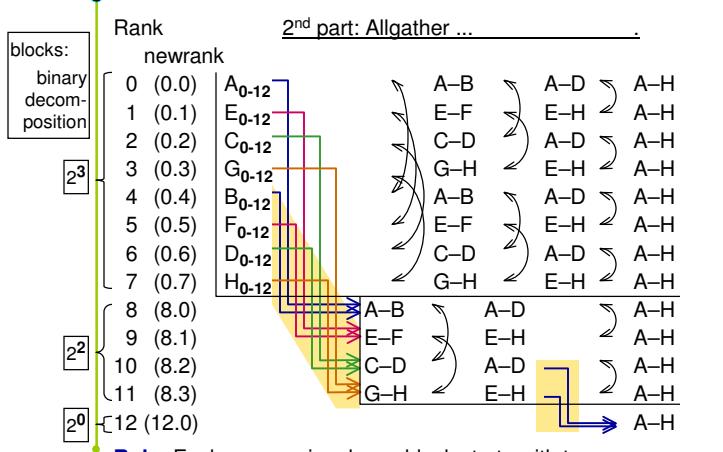
$A–D_{0-1}$ = computing reduction of data segments A–D with data from ranks 0-1

**load imbalance**

Always computing: { [(0+1)+(2+3)] + [(4+5)+(6+7)] } + { [(8+9)+(10+11)] + [(12)] }



## New Binary Blocks Halving+Doubling, July 2003 (2nd part)




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**

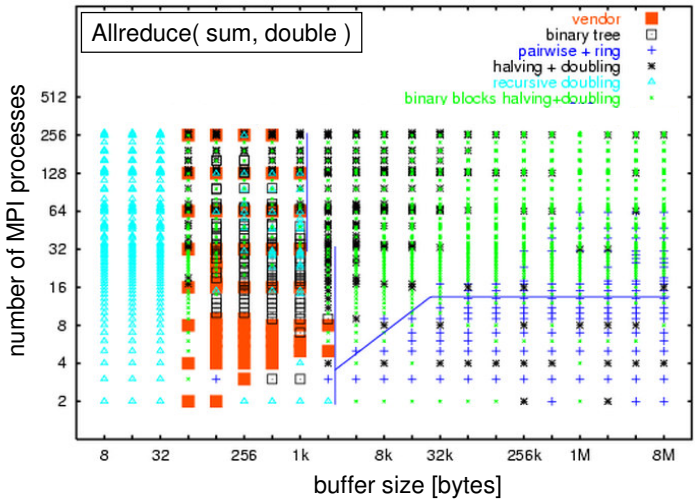
**Rule:** Each process in a lower block starts with two messages from the next higher block. The messages contain both halves.

## 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,....



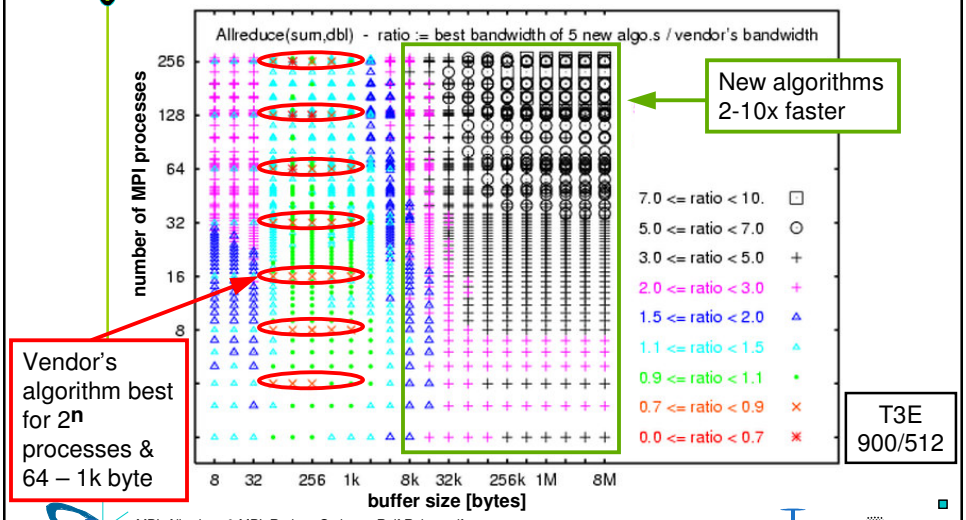
## Comparison: Fastest Protocol on T3E 900/512






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

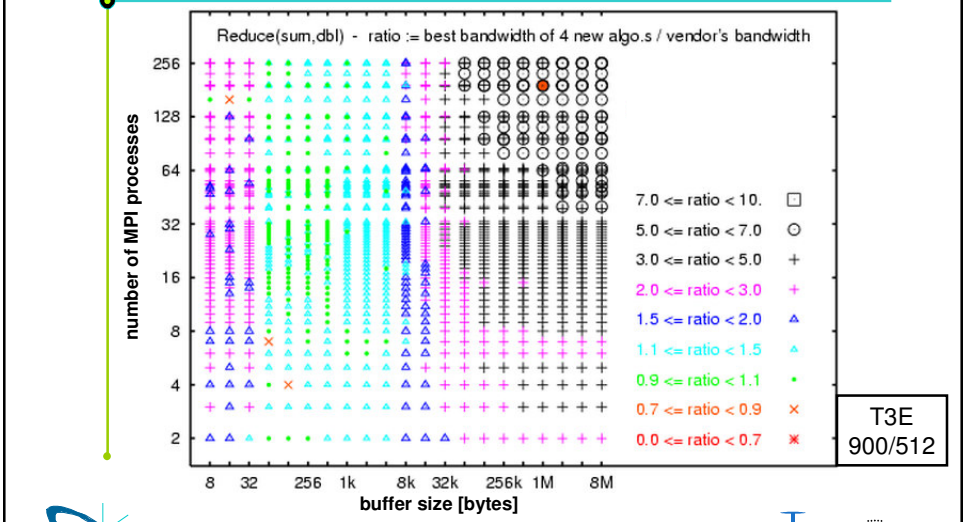
## Best algorithm more than x7 faster than vendor's




MPI_Allreduce & MPI_Reduce Optim. Rolf Rabenseifner
Slide 13 / 19 Höchstleistungsrechenzentrum Stuttgart

H L R S

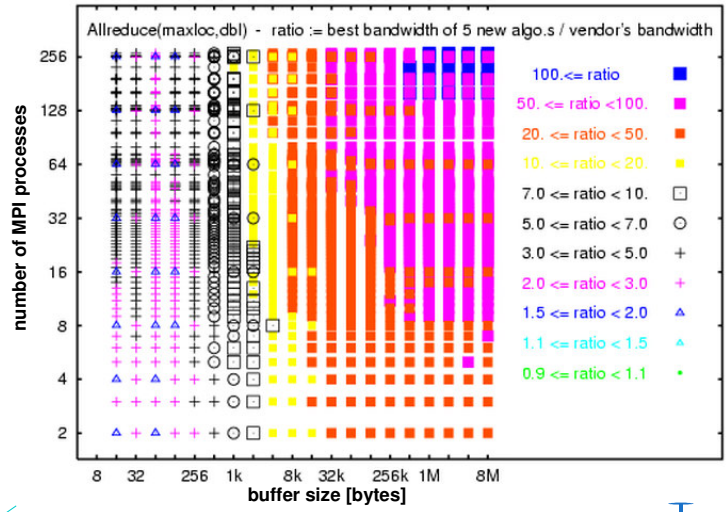
## Reduce: One of the 4 algorithms is always the best




MPI_Allreduce & MPI_Reduce Optim. Rolf Rabenseifner
Slide 14 / 19 Höchstleistungsrechenzentrum Stuttgart

H L R S

## MAXLOC: Vendor's MPI_DOUBLE_INT is extreme slow

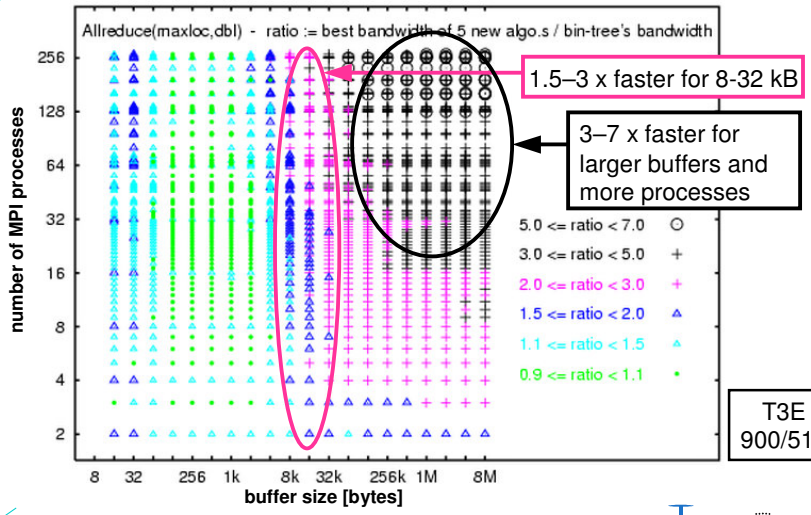



T3E 900/512



H L R S

## MAXLOC: Comparing with optimized binomial-tree

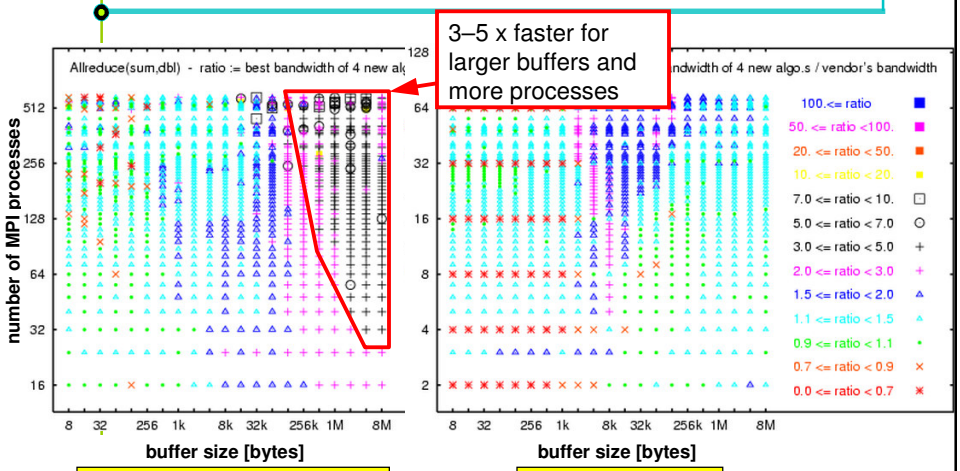



T3E 900/512



H L R S

## MPI_Allreduce on IBM SP at SDSC




One MPI-process per **CPU**

... per **SMP-node**



## 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