More Efficient Reduction Algorithms for Non-power-of-two Number of Processors in Message-Passing Parallel Systems

Rolf Rabenseifner (HLRS), Jesper Larsson Träff (NEC)

rabenseifner@hlrs.de traff@ccrl-nece.de

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

C&C Research Laboratories NEC Europe Ltd. St. Augustin (www.ccrl-nece.de)

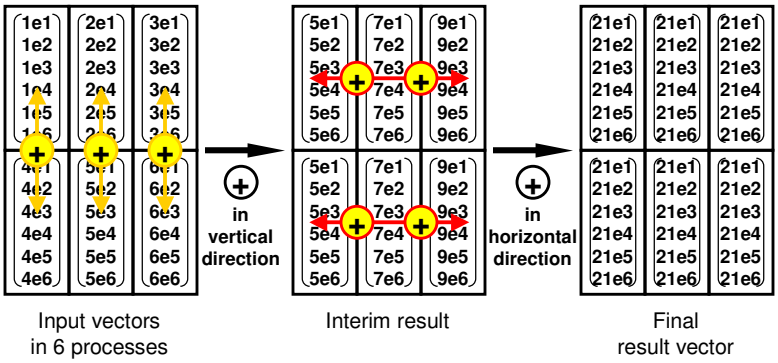
EuroPVM/MPI 2004, Budapest, Sep. 19-22, 2004

Methods

- Factorizing #processes = $2^n \cdot q_1 \cdot q_2 \cdot q_3 \cdots q_m$ ($n \geq 0$, $m \geq 0$, odd q_i)
  - Processes logically in a hypercube with dim=n+m
- Latency-optimization → with full buffer exchange
- Bandwidth-optimization → with buffer halving (or splitting) [**butterfly alg.**]
- Mixing both methods for medium vector sizes
- Factorization implemented with recursive distance doubling (×2) or $\times q_i$
- Factor q=2: optimal butterfly algorithm
- **Odd factors q_i**: several **new** algorithms
  - 2 elimination algorithms:
    - **only with 3-to-2 elimination steps**
    - **with 3-to-2 and 2-1 elimination steps**
  - Ring algorithm
- Buffer handling
- Further methods

Processes ordered as hypercube

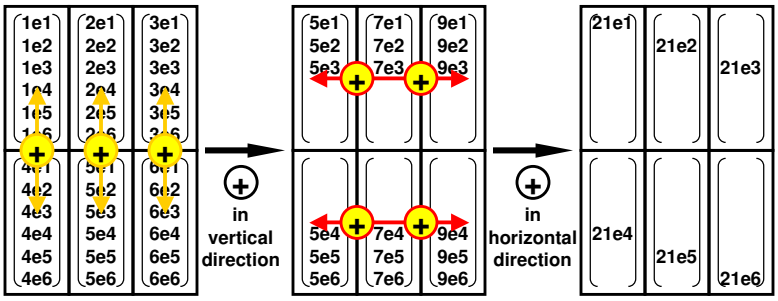
- Reduction separately in each direction



Input vectors in 6 processes | Interim result | Final result vector

- **Pro: Minimal number of messages – Goal: O(log #proc)**
- **Con: Full size vector transfer in each direction**

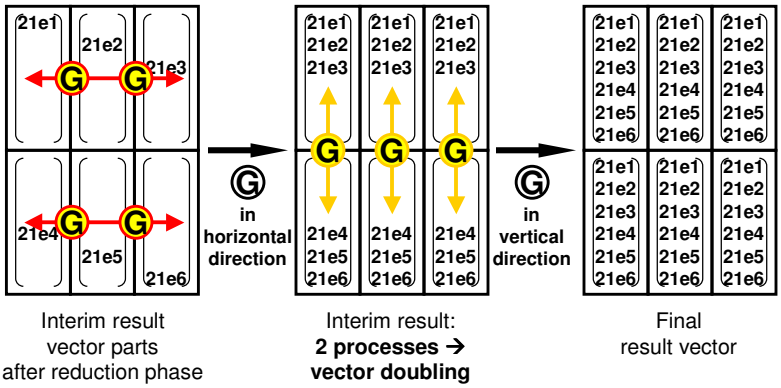
Splitting the vectors



Input vectors in 6 processes | Interim result: **2 processes → vector halving** | Next interim result: **3 processes → scattered into 3 parts**

- **Reduce_scatter phase**
- **Additional allgather phase → next slide**

In reverse sequence, with MPI_Allgather: Reduction results gathered to all processes



Interim result vector parts after reduction phase

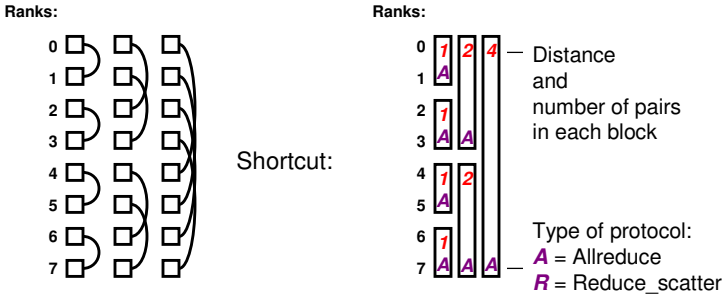
Interim result: **2 processes → vector doubling**

Final result vector

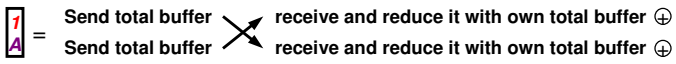
- **Pro: Sum of all message sizes < 2 * vector size**
- **Con: Doubling the number of messages**

Allreduce / Reduce_scatter phase for power-of-two processes

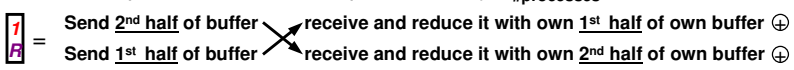
- Communication and reduction protocol with **distance doubling**



- Latency optimized – total reduction result on all processes

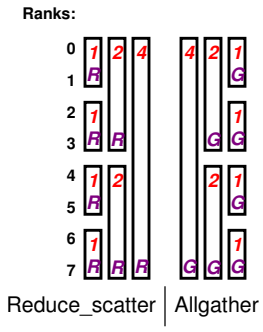


- Bandwidth optimized – on each process: only $^{1}/_{\#processes}$ of reduction result

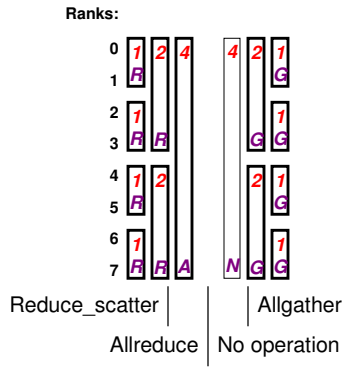


Allreduce = Reduce_scatter + Allgather (Butterfly)

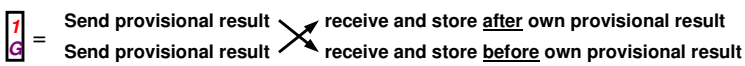
- Pure Protocol:



- Mixed Protocol:



- Allgather protocol:

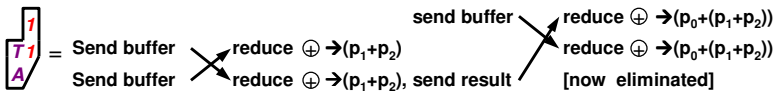


Methods

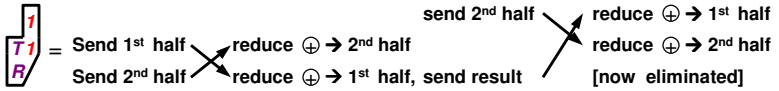
- Factorizing #processes = $2^n \cdot q_1 \cdot q_2 \cdot q_3 \cdot \cdot \cdot q_m$ ($n \geq 0$, $m \geq 0$, odd q_i)
  - Processes logically in a hypercube with dim=n+m
- Full buffer exchange for best latency
- Buffer halving (or splitting) for best bandwidth (butterfly algorithm)
- Mixing both methods for medium vector sizes
- Factorization implemented with recursive distance doubling (×2) or $\times q_i$
- Factor q=2: optimal butterfly algorithm
- **Handling odd factors q_i: several new algorithms**
  - **2 elimination algorithms with doubling&halving:**
    - **only with 3-to-2 elimination steps**
    - **with 3-to-2 and 2-1 elimination steps**
  - **Ring algorithm**
- Buffer handling
- Further methods

Methods for non-power-of-two number of processes

- 3-to-2-process (**T**riple) and 2-to-1-process (**D**ouble) reduction eliminates processes in the allreduce/reduce_scatter epoch, i.e., those processes are not used in further reduction steps
- 3-to-2-process reduction (**T**riple):
  - Allreduce/reduce_scatter phase:
    - **Latency optimized**



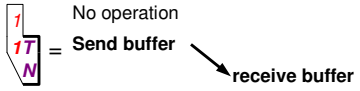
    - **Bandwidth optimized**



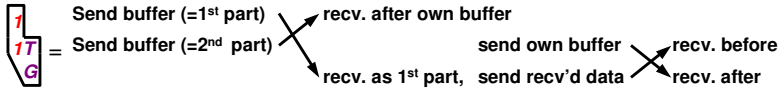
This work was done together with Jesper Larrson Träff, NEC.

Methods for non-power-of-two number of processes

- 3-to-2-process reduction (**T**riple):
  - Allgather phase:
    - **Latency optimized**



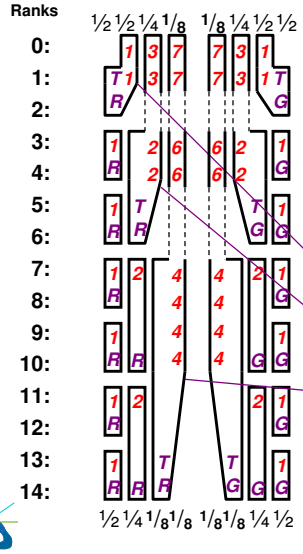
    - **Bandwidth optimized**



This work was done together with Jesper Larrson Träff, NEC.

skipped

Example with 2^z–1 processes, bandwidth optimized



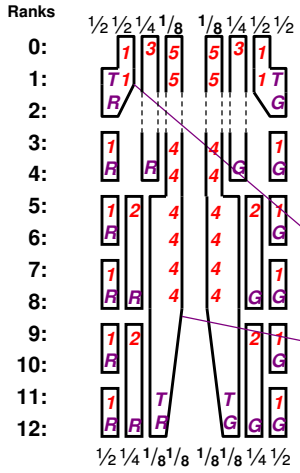
- Overlapping protocol blocks:
  - Total message size < 3 * buffer size
  - Total computation size < 1.5 * buffer size at each process that is never eliminated
- Same rules for any odd number of processes

8 + 4 + 2 + 1 = 15 processes

to be eliminated in Triple-steps

This work was done together with Jesper Larrson Träff, NEC.

Example with odd number of processes, bandwidth optimized



- Overlapping protocol blocks:
  - Total message size < 3 * buffer size
  - Total computation size < 1.5 * buffer size at each process that is never eliminated
- Same rules for any odd number of processes

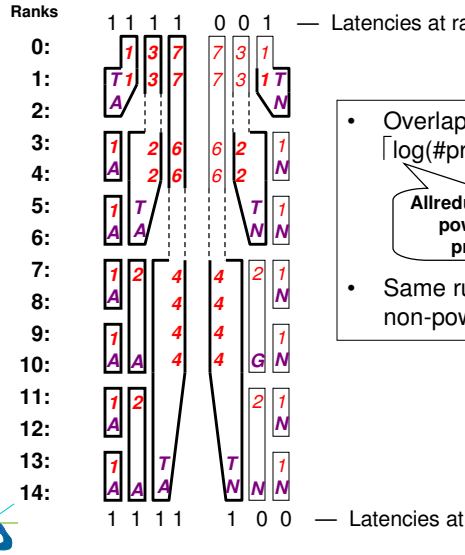
8 + 4 + 1 = 13 processes

to be eliminated in Triple-steps

This work was done together with Jesper Larrson Träff, NEC.

skipped

Example with 2^z–1 processes, latency optimized



- Overlapping protocol blocks: $\lceil \log(\#\text{processes}) \rceil + 1$ latencies
  - Allreduce, result on power-of-two processes
  - One step: sending to the other processes
- Same rules for any non-power-of-two number of processes

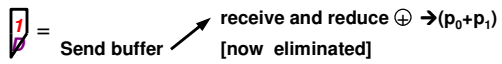
This work was done together with Jesper Larrson Träff, NEC.

Methods

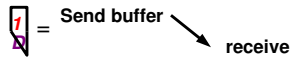
- Factorizing #processes = $2^n \cdot q_1 \cdot q_2 \cdot q_3 \cdots q_m$ ($n \geq 0$, $m \geq 0$, odd q_i)
  - Processes logically in a hypercube with dim=n+m
- Full buffer exchange for best latency
- Buffer halving (or splitting) for best bandwidth (butterfly algorithm)
- Mixing both methods for medium vector sizes
- Factorization implemented with recursive distance doubling (×2) or $\times q_i$
- Factor q=2: optimal butterfly algorithm
- **Handling odd factors q_i: several new algorithms**
  - **2 elimination algorithms with doubling&halving:**
    - **only with 3-to-2 elimination steps**
    - **<u>with 3-to-2 and 2-1 elimination steps</u>**
  - **ring algorithm**
- Buffer handling
- Further methods

2. Algorithm with Triple and Double Reduction steps

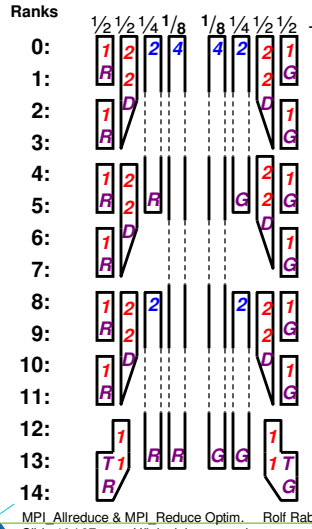
- 2-to-1-process reduction (**D**ouble):
  - Same protocol for “Latency optimized” and “Bandwidth optimized”
  - Allreduce/reduce_scatter epoch:



  - Allgather epoch:



Example with 2^z–1 processes, bandwidth optimized

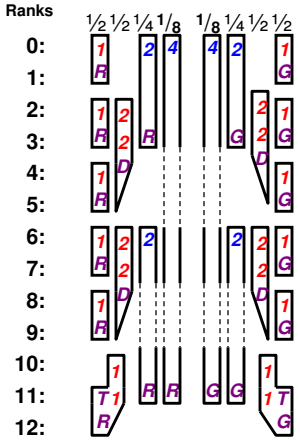


— message size at all active ranks

- Non-overlapping protocol blocks:
  - Total message size < 3 * buffer size
  - Total computation size < 1.5 * buffer size at each process that is never eliminated
- Same costs as with previous algorithm
- Same rules for any odd number of processes

skipped

Example with odd number of processes, bandwidth optimized



- Splitting the number of processes into
  - L × (2-process-symmetric-reduction [*R*])
  - M × (2 × 2-process-symmetric-reduction [*R*] + 2 × 2-1-reduction [*D*])
  - 1 × (3-2-reduction [*T*])

  with $2(L+M+1) = 2^{\lfloor \log \#\text{processes} \rfloor}$

Methods

- Factorizing #processes = $2^n \cdot q_1 \cdot q_2 \cdot q_3 \cdots q_m$ ($n \geq 0$, $m \geq 0$, odd q_i)
  - Processes logically in a hypercube with dim=n+m
- Full buffer exchange for best latency
- Buffer halving (or splitting) for best bandwidth (butterfly algorithm)
- Mixing both methods for medium vector sizes
- Factorization implemented with recursive distance doubling (×2) or $\times q_i$
- Factor q=2: optimal butterfly algorithm
- **Handling odd factors q_i: several new algorithms**
  - **2 elimination algorithms with doubling&halving :**
    - **only with 3-to-2 elimination steps**
    - **with 3-to-2 and 2-1 elimination steps**
  - **Ring algorithm**
- Buffer handling
- Further methods

Further Optimization of odd #processes with ring-algorithms

- **Latency optimization**:
  - Algorithm:
    - Use Bruck's "Concatenation" = Allgather algorithm (optimal, i.e., $O(\lceil \log \#p \rceil)$ for any #p)
    - Then reduce all data in each process
- **Bandwidth optimization**:
  - Algorithm:
    - Divide input vector logically into (#processes) parts
    - Alltoall: send j^{th} parts from all input vectors to process j
    - Then local reduction of all j^{th} vector-parts in each process j
    - Then Bruck's Allgather

skipped

Further Optimization of odd #processes with ring-algorithms

The inner part for odd #processes can be additionally improved:

- **Latency optimization**:
  - Algorithm:
    - Use Bruck's "Concatenation" = Allgather algorithm
    - Then reduce all data in each process
  - Criterion — Ring-algorithm — Elimination-Protocol — **[p = #processes]**
    - **Latency**
      - Ring-algorithm: **$O(\lceil \log(p) \rceil)$ always faster**
      - Elimination-Protocol: **$O(\lceil \log(p) \rceil + 1)$**
    - **Transfer**
      - Ring-algorithm: **$O(p - 1)$ faster if p=3,5**
      - Elimination-Protocol: **$O(\lceil \log(p) \rceil + 1)$ faster if $p \geq 7$**
    - **Operations**
      - Ring-algorithm: **$O(p - 1)$ faster if p=3**
      - Elimination-Protocol: **$O(\lceil \log(p) \rceil)$ faster if $p \geq 5$**

#processes = p	3	5	7	9	11	13	15
Latencies	2 / 3	3 / 4	3 / 4	4 / 5	4 / 5	4 / 5	4 / 5
Transfer / input_buffer_size	2 / 3	4 / 4	6 / 4	8 / 5	10 / 5	12 / 5	14 / 5
Operations / input_buffer_elements	2 / 2	4 / 3	6 / 3	8 / 4	10 / 4	12 / 4	14 / 4

skipped

Further Optimization of odd #processes with ring-algorithms

- **Bandwidth optimization**:
  - Algorithm:
    - Divide input vector logically into (#processes) parts
    - Alltoall: send j^{th} parts from all input vectors to process j
    - Then local reduction of all j^{th} vector-parts in each process j
    - Then Bruck's Allgather
  - Criterion

Criterion	Ring-algorithm	Elimination-protocol	
• Latency	• $O(p-1+\lceil \log(p) \rceil)$	• $O(2\lceil \log(p) \rceil)$	[p = #processes]
• Transfer	• $O(2(p-1)/p)$	• $O(2(1.5-1/p'))$	[p' = $2^{\lfloor \log p \rfloor}$]
• Operations	• $O((p-1)/p)$	• $O(1.5-1/p')$	i.e., O(half of transfer)

better

#processes = p	3	5	7	9	11	13	15
Latencies	4 / 4	7 / 6	9 / 6	12 / 8	14 / 8	16 / 8	18 / 8
Transfer per input_buffer_...	1.33 / 2.00	1.60 / 2.50	1.71 / 2.50	1.78 / 2.75	1.82 / 2.75	1.85 / 2.75	1.87 / 2.75

Best algorithm for odd factor q_i

Algorithm	Latency	+ Transfer	+ Operations
Lat.-opt. Ring	$O(\lceil \log(p) \rceil)$	$+ s(O(p-1)$	$+ O(p-1))$
Lat.-opt. Elimi.	$O(\lceil \log(p) \rceil+1)$	$+ s(O(\lceil \log(p) \rceil+1)$	$+ O(\lceil \log(p) \rceil))$
Bw.-opt. Elimi.	$O(2\lceil \log(p) \rceil)$	$+ s(O(3-2/p')$	$+ O(1.5-1/p'))$
Bw.-opt. Ring	$O(p-1+\lceil \log(p) \rceil+1^{*)})$	$+ s(O(2-2/p)$	$+ O(1-1/p))$

p=number of processes, s=vector size

*)some additional messages to prohibit local buffer copy in Bruck's allgather algorithm

Example for p=125

Algorithm	Latency	+ Transfer	+ Operations	→ Optimization for large p:
Lat.-opt. Ring	O(7)	+ s(O(124)	+ O(124))	→ **very small vectors**
Lat.-opt. Elimi.	O(8)	+ s(O(8)	+ O(7))	→ **small vectors**
Bw.-opt. Elimi.	O(14)	+ s(O(3)	+ O(1.5))	→ **med.–long vectors**
Bw.-opt. Ring	O(132)**)	+ s(O(2)	+ O(1))	→ **very long vectors**

**)may be reduced to O(24) with ring(5)*ring(5)*ring(5) instead of ring(125)

skipped

Buffer handling

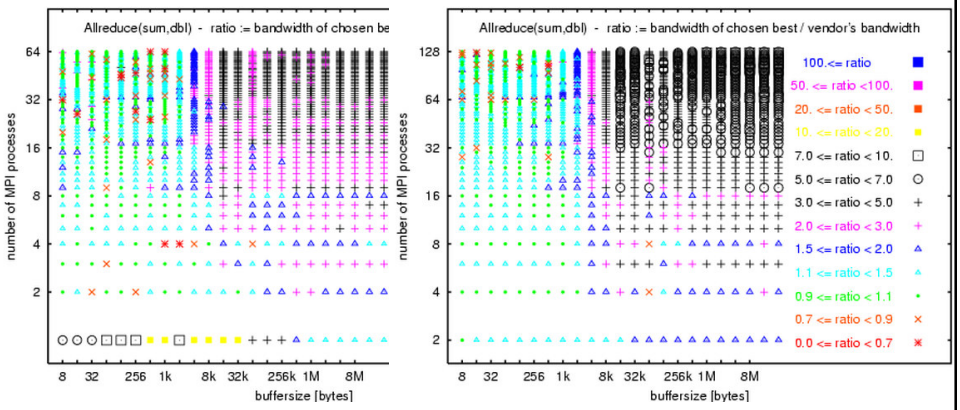
- If operation is commute
  - No local buffer copying,
  - Except in Bruck's allgather, used in latency optimized ring (i.e., with small vectors)
- If operation is not commute (only with user defined operations)
  - Minimizing buffer copying be receiving data to optimized location

Further optimization methods

- Pipelining:
  - Splitting vectors into chunks
  - Overlaying chunk message transfer with reduction already received chunk
- Segmentation
  - For algorithms that do not use all processors at the same time
    - **E.g., MPI_Reduce with binomial tree**
  - Executing the operation for several segments of the input vector
  - (Recursive halving and doubling should be faster)
  - Smaller scratch buffers

Measurements on 128 x Dual-Opteron with Myrinet 2000

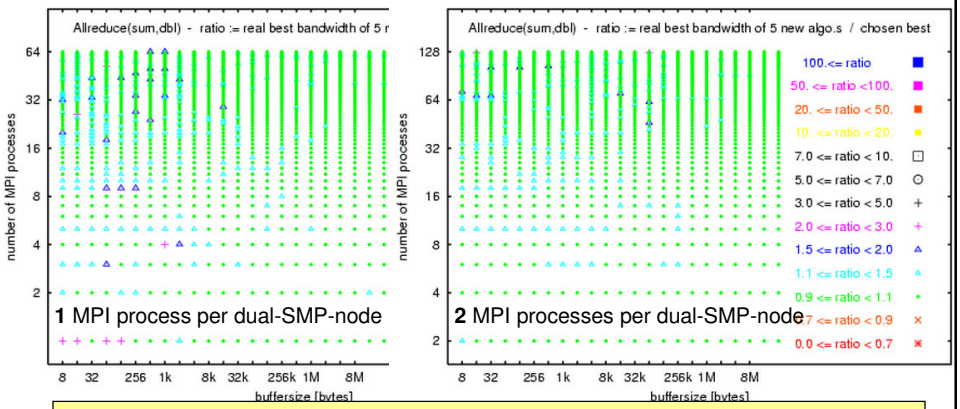
MPIAllreduce(sum, double) bandwidth: Ratio “new routine” / “mpich1.2.5..10-gm-pgi”



1 MPI process per dual-SMP-node **2** MPI processes per dual-SMP-node

Is automatic protocol-choosing real best?

Bandwidth-ratio “best of some choices” / “automatic choosing”



1 MPI process per dual-SMP-node **2** MPI processes per dual-SMP-node

In few cases, manual choices may be up to 1.5 better than automatic

Summary

- Optimized MPI_Allreduce
  - any number of processes #p
  - especially non-power-of-two #p
  - any vector length
- Optimal latency for $\#p = 2^n$ or $\#p = 2^n * 3$ → $O(\lceil \log_2 p \rceil)$
  - one additional round for other #p → $O(\lceil \log_2 p \rceil + 1)$
- Bandwidth optimized: if $\#p = 2^n * q$ then → $O(\underbrace{2(1+1/2^{n+1})}_{\text{between 2 and 3}}s)$
  - Instead of previous O(4s) **[mpich ≥1.2.6]**
- Smooth transition
  - from latency to bandwidth optimized
- Implementations
  - By NEC: (non-disclosed, 3-2-elimination)
  - By HLRS: freely available on request (2-1-elimination and ring) (e-mail to rabenseifner@hlrs.de)