Proceedings, EuroPVM/MPI 2004, Sep. 19-22, Budapest, Hungary, LNCS, Springer-Verlag, 2004.

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

Rolf Rabenseifner[1] and Jesper Larsson Träff[2]

[1] High-Performance Computing-Center (HLRS), University of Stuttgart, Allmandring 30, D-70550 Stuttgart, Germany,
rabenseifner@hlrs.de,
www.hlrs.de/people/rabenseifner/
[2] C&C Research Laboratories, NEC Europe Ltd., Rathausallee 10, D-53757 Sankt Augustin, Germany,
traff@ccrl-nece.de

Abstract. We present improved algorithms for global reduction operations for message-passing systems. Each of p processors has a vector of m data items, and we want to compute the element-wise "sum" under a given, associative function of the p vectors. The result, which is also a vector of m items, is to be stored at either a given *root* processor (`MPI_Reduce`), or all p processors (`MPI_Allreduce`). A further constraint is that for each data item and each processor the result must be computed in the same order, and with the same bracketing. Both problems can be solved in $O(m + \log_2 p)$ communication and computation time. Such reduction operations are part of MPI (the *Message Passing Interface*), and the algorithms presented here achieve significant improvements over currently implemented algorithms for the important case where p is not a power of 2. Our algorithm requires $\lceil \log_2 p \rceil + 1$ rounds - one round off from optimal - for small vectors. For large vectors twice the number of rounds is needed, but the communication and computation time is less than $3m\beta$ and $3/2m\gamma$, respectively, an improvement from $4m\beta$ and $2m\gamma$ achieved by previous algorithms (with the message transfer time modeled as $\alpha + m\beta$, and reduction-operation execution time as $m\gamma$). For $p = 3 \times 2^n$ and $p = 9 \times 2^n$ and small $m \leq b$ for some threshold b, and $p = q2^n$ with small q, our algorithm achieves the optimal $\lceil \log_2 p \rceil$ number of rounds.

Keywords. Message Passing, MPI, Collective Operations, Reduction.

1 Introduction and Related Work

Global reduction operations in three different flavors are included in MPI, the *Message Passing Interface* [14]. The `MPI_Reduce` collective combines element-wise the input vectors of each of p processes with the result vector stored only at a given *root process*. In `MPI_Allreduce`, all processes receive the result. Finally, in `MPI_Reduce_scatter`, the result vector is subdivided into p parts with given (not necessarily equal) numbers of elements, which are then scattered over the processes. The global reduction operations are among the most used MPI collectives. For instance, a 5-year automatic profiling [12, 13] of all users on a Cray T3E has shown that 37 % of the total MPI time was spent in `MPI_Allreduce` and that 25 % of all execution time was spent in runs that involved a non-power-of-two number of processes. Thus improvements to these collectives are almost always worth the effort.

The p processes are numbered consecutively with ranks $i = 0, \ldots, p-1$ (we use MPI terminology). Each has an input vector m_i of m units. Operations are binary, associative, and possibly commutative. MPI poses other requirements that are non-trivial in the presence of rounding errors:

1. For `MPI_Allreduce` all processes must receive the *same* result vector;
2. reduction must be performed in *canonical order* $m_0 + m_1 + \cdots + m_{p-1}$ (if the operation is not commutative);
3. the same reduction order and bracketing for all elements of the result vector is not strictly required, but should be strived for.

For non-commutative operations $a+b$ may be different from $b+a$. In the presence of rounding errors $a + (b + c)$ may differ from $(a + b) + c$ (two different bracketing's). The requirements ensure consistent results when performing reductions on vectors of floating-point numbers.

We consider 1-ported systems, i. e. each process can send and receive a message at the same time. We assume linear communication and computation costs, i.e. the time for exchanging a message of m units is $t = \alpha + m\beta$ and the time for combining two m-vectors is $t = m\gamma$.

Consider first the `MPI_Allreduce` collective. For $p = 2^n$ (power-of-2), butterfly-like algorithms that for small m are *latency-optimal*, for large m *bandwidth- and work-optimal*, with a smooth transition from latency dominated to bandwidth dominated case as m increases have been known for a long time [5, 16]. For small m the number of communication rounds is $\log_2 p$ (which is optimal [5]; this is what we mean by *latency-optimal*) with $m \log_2 p$ elements exchanged/combined per process. For large m the number of communication rounds doubles because of the required, additional allgather phase, but the number of elements exchanged/combined per process is reduced to $2(m - 1/p)$ (which is what we mean by *bandwidth- and work-optimal*). These algorithms are simple to implement and practical.

When p is not a power-of-two the situation is different. The optimal number of communication rounds for small m is $\lceil \log_2 p \rceil$, which is achieved by the algorithms in [3, 5]. However, these algorithms assume commutative reduction operations, and furthermore the processes receive data in different order, such that requirements 1 and 2 cannot be met. These algorithms are therefore not suited for MPI. Also the bandwidth- and work-optimal algorithm for large m in [5] suffers from this problem. A repair for (very) small p would be to collect (allgather) the (parts of the) vectors to be combined on all processes, using for instance the optimal (and very practical) allgather algorithm in [6], and then perform the reduction sequentially in the same order on each process.

Algorithms suitable for MPI (i.e., with respect to the requirements 1 to 3) are based on the butterfly idea (for large m). The butterfly algorithm is executed on the largest power-of-two $p' < p$ processes, with an extra communication round before and after the reduction to cater for the processes in excess of p'. Thus the number of rounds is no longer optimal, and if done naively an extra $2m$ is added to the amount of data communicated for some of the processes. Less straightforward implementations of these ideas can be found in [13, 15], which perform well in practice.

The contributions of this paper are to the practically important non-powers-of-two case. First, we give algorithms with a smooth transition from latency to bandwidth dominated case based on a message threshold of b items. Second, we show that for the general case the amount of data to be communicated in the extra rounds can be reduced by more than a factor of 2 from $2m$ to less than m (precisely to $m/2^{n+1}$ if p is factorized in $p = q2^n$ with q an odd number). Finally, for certain number of processes $p = q2^n$ with $q = 3$ and $q = 9$ we give latency- and bandwidth optimal algorithms by combining the butterfly idea with a ring-algorithm over small rings of 3 processes; in practice these ideas may also yield good results for $q = 5, 7, \ldots$, but this is system dependent and must be determined experimentally.

The results carry over to `MPI_Reduce` with similar improvements for the non-power-of-two case. In this paper we focus on `MPI_Allreduce`.

Other related work on reduction-to-all can be found in [1–3]. Collective algorithms for wide-area clusters are developed in [7–9], further protocol tuning can be found in [4, 10, 15], especially on shared memory systems in [11]. Compared to [15], the algorithms of this paper furthermore give a smooth transition from latency to bandwidth optimization and higher bandwidth and shorter latency if the number of processes is not a power-of-two.

2 Allreduce for powers-of-two

Our algorithms consist of two phases. In the *reduction phase* reduction is performed with the result scattered over subsets of the p processors. In the *routing phase*, which is only necessary if m is larger than a threshold b, the result vector is computed by gathering the partial results over each subset of processes. Essentially, only a different routing phase is needed to adapt the algorithm to `MPI_Reduce` or `MPI_Reduce_scatter`. For `MPI_Allreduce` and `MPI_Reduce` the routing phase is most easily implemented by reversing the communication pattern of the reduction phase.

It is helpful first to recall briefly the hybrid butterfly algorithm as found in e.g. [16]. For now $p = 2^n$ and a message threshold b is given.

In the *reduction phase* a number of communication and computation *rounds* is performed. Prior to round $z, z = 0, \ldots, n-1$ with $m/2^z > b$ each process i possesses a vector of size $m/2^z$ containing a block of the partial result $((m_{i_0} + m_{i_0+1}) + \cdots + (m_{i_0+2^z-2} + m_{i_0+2^z-1}))$ where i_0 is obtained by setting the least significant z bits of i to 0. In round z process i sends half of its partial result

Process i:

```
// Reduction phase
m' ← m // current data size
d ← 1 // "distance"
while d < p do
        // round
        select r − 1 neighbors of i // Protocol decision
        if m' > b then
           exchange(m'/r) with r − 1 neighbors
           Push neighbors and data sizes on stack
           m' ← m'/r
        else exchange(m') with r − 1 neighbors
        local reduction of r (partial) vectors of size m'
        d ← d × r
end while
// Routing phase
while stack non-empty
        pop neighbors and problem size off stack
        exchange with neighbors
end while
```

Fig. 1. High-level sketch of the (butterfly) reduction algorithm. For p a power of two a butterfly exchange step is used with $r = 2$. For other cases different exchange/elimination steps can be used as explained in Section 3.

to process $i \oplus 2^z$ ($\oplus$ denotes bitwise exclusive-or, so the operation corresponds to flipping the zth bit of i), and receives the other half of this process' partial result. Both processes then performs a local reduction, which establishes the above invariant above for round $z + 1$. If furthermore the processes are careful about the order of the local reduction (informally, either from left to right or from right to left), it can be maintained that the partial results on all processes in a group have been computed in canonical order, with the same bracketing, such that the requirements 1 to 3 are fulfilled. If in round z the size of the result vector $m/2^z \leq b$ halving is not performed, in which case processes i and $i \oplus 2^z$ will end up with the same partial result for the next and all succeeding rounds. For the routing phase, nothing needs to be done for these rounds, whereas for the preceding rounds where halving was done, the blocks must be combined.

A high-level sketch of the algorithm is given in Figure 1. In Figure 2 and Figure 3 the execution is illustrated for $p = 8$. The longer boxes shows the process groups for each round. The input buffer is divided into 8 segments $\mathsf{A}\text{-}\mathsf{H}_i$ on process i. The figure shows the buffer data after each round: $\mathsf{X}\text{-}\mathsf{Y}_{i-j}$ is the result of the reduction of the segments X to Y from processes i to j.

Following this sketch it is easy to see that the reduction phase as claimed takes $n = \log_2 p$ rounds. For $m/p \geq b$ the amount of data sent and received per process is $\sum_{k=0}^{n-1} m/2^{k+1} = m(1 - 1/p)$. For $m \leq b$ the routing phase is empty so the optimal $\log_2 p$ rounds suffice. For $m > b$ some allgather rounds are necessary,

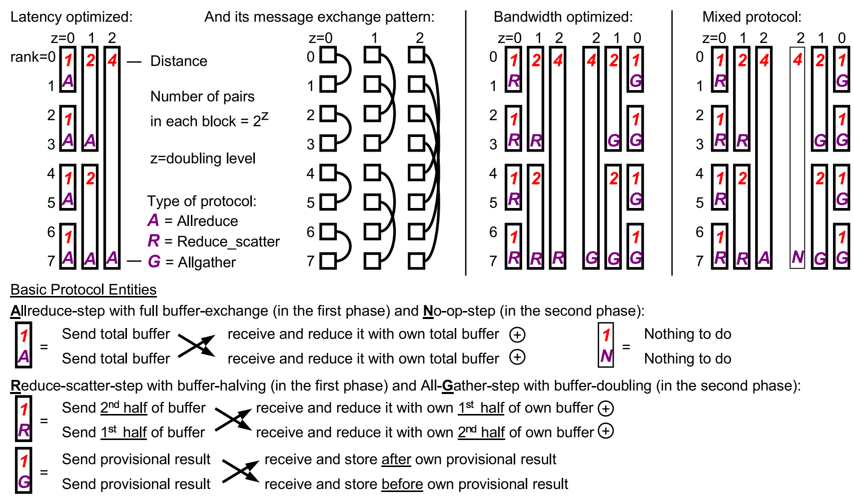

Fig. 2. The butterfly reduction algorithm.

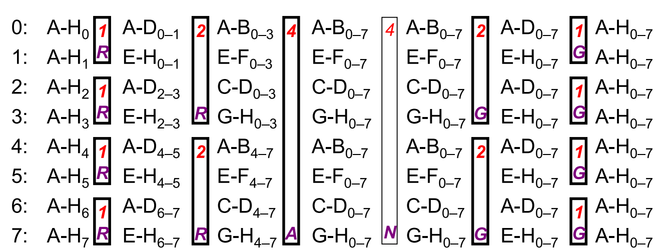

Fig. 3. Intermediate results after each protocol step when the threshold b is reached in round 3.

namely one for each reduction round in which $m/2^k > b$. At most, the number of rounds doubles.

3 The improvements: Odd number of processes

We now present our improvements to the butterfly scheme when p is not a power-of-two. Let for now $p = q2^n$ where 2^n is the largest power of two smaller than p (and q is odd).

For the general case we introduce a more communication efficient way to include data from processes in excess of 2^n into the butterfly algorithm. We call this step *3-2 elimination*. Based on this we give two different algorithms for the general case, both achieving the same bounds. For certain small values of q we show that a ring based algorithm can be used in some rounds of the

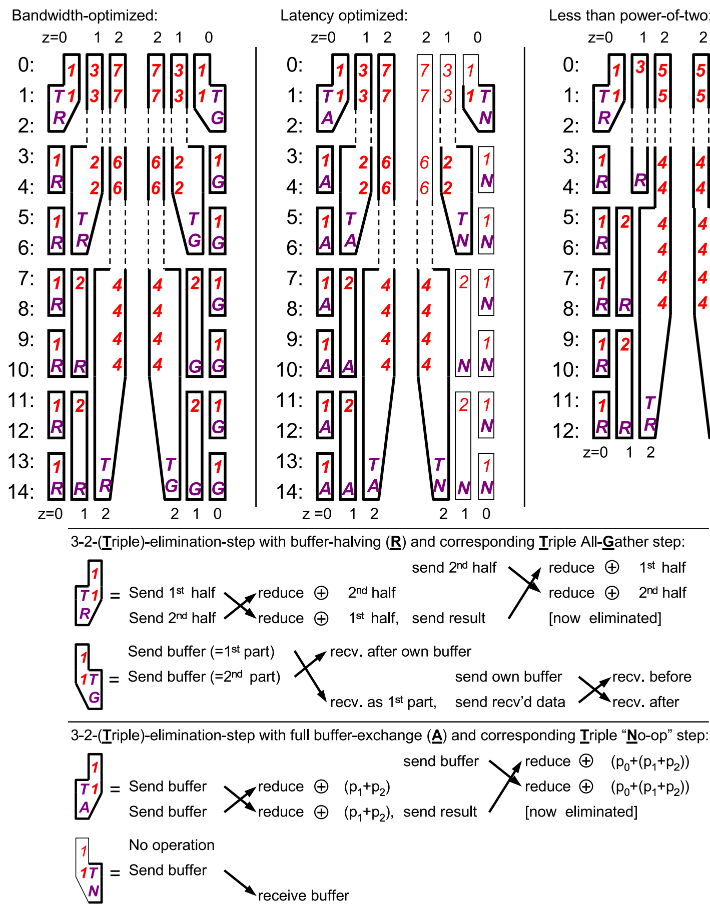

Fig. 4. Overlapping elimination protocol for $p = 15$ and $p = 13$ using 3-2-elimination-steps.

butterfly algorithm, and for certain values of q results in the optimal number of communication rounds.

3.1 The 3-2 elimination step

For $m' > b$ the 3-2 elimination step is used on a group of three processes p_0, p_1, and p_2, to absorb the vector of process p_2 into the partial results of process p_0 and p_1, which will survive for the following rounds. The step is as follows: process p_2 sends $m'/2$ (upper) elements to process p_1, and simultaneously receives $m'/2$ (lower) elements from process p_1. Process p_1 and p_2 can then perform the reduction operation on their respective part of the vector. Next, process p_0 receives the $m'/2$ (lower) elements of the partial result just computed from process p_2,

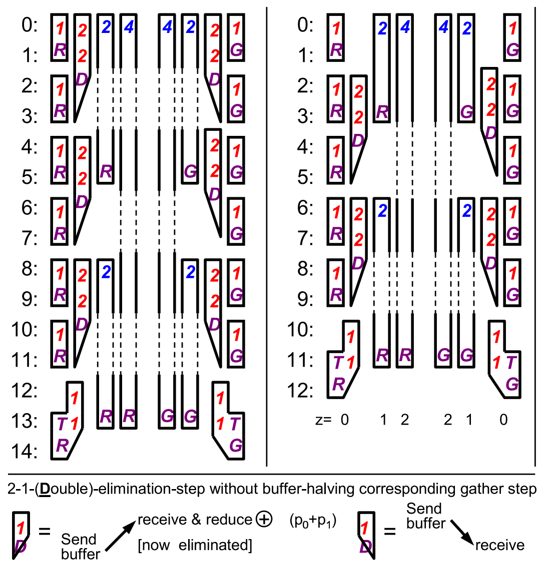

Fig. 5. Non-Overlapping elimination protocol for $p = 15$ and $p = 13$ using 3-2- and 2-1-elimination-steps.

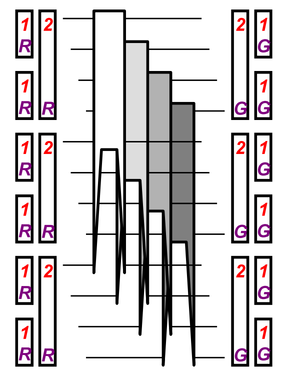

Fig. 6. Plugging in any algorithm for odd number (here 3) of processes after reducing p with the butterfly algorithm (here with two steps) to its odd factor.

and sends $m'/2$ (upper) elements to process p_1. Process p_0 and p_1 compute a new partial result from the $m'/2$ elements received.

As can be seen process p_0 and p_1 can finish after two rounds, both with the half of the elements of the result vector $[m'_0 + (m'_1 + m'_2)]$. The total time for process p_0 and p_1 is $2\alpha + \beta m' + \gamma m'$.

Compare this to the trivial solution based on *2-1 elimination*. First process p_2 sends all its m' elements to process p_1 (2-1 elimination), after which process p_0 and p_1 performs a butterfly exchange of $m'/2$ elements. The time for this solution is $2\alpha + 3/2\beta m' + 3/2\gamma m'$.

For $m' \leq b$, the total buffers are exchanged and reduced, see the protocol entities described in Fig. 4.

The 3-2 elimination step can be plugged into the general algorithm of Figure 1. For $p = 2^n + Q$ with $Q < 2^n$, the total number of elimination steps to be performed is Q. The problem is to schedule these in the butterfly algorithm in such a way that the total number of rounds does not increase by more than 1 for a total of $n + 1 = \lceil \log_2 p \rceil$ rounds. Interestingly we have found two solutions to this problem, which are illustrated in the next subsections.

3.2 Overlapping 3-2 Elimination Protocol

Figure 4 shows the protocol examples with 15 and 13 processes. In general, this protocol schedules 3-2-elimination steps for a group of on $2^z \times 3$ processes in each round z for which the zth bit of p is 1. The 3-2-steps exchange two messages of the same size and are therefore drawn with double width. The first process is not involved in the first message exchange, therefore this part is omitted from the shape in the figure. After each 3-2-step, the third process is eliminated, which is marked with dashed lines in the following rounds. The number of independent pairs or triples in each box is 2^z. As can be seen the protocol does not introduce delays where some processes have to wait for other processes to complete their 3-2 elimination steps of previous rounds, but different groups of processes can simultaneously be at different rounds. Note, that this protocol can be used in general for any number of processes. If p includes a factor 2^n then it starts with n butterfly steps.

3.3 Non-overlapping Elimination Protocol

Figure 5 shows a different protocol that eliminates all excess processes at round $z = 1$. With the combination of one 3-2-elimination-step and pairs of 2-1-elimination-steps any odd number of processes p is thus reduced to its next smaller power-of-two value. Note that for $m > b$ in round $z = 1$ only $m/2$ data are sent in the 2-1-elimination step (instead of m if the 2-1 elimination would have been performed prior to round $z = 0$).

Both the overlapping and the non-overlapping protocol are exchanging the same amount of data and number of messages. For small $m \leq b$ the total time is $t = (1 + \lceil \log_2 p \rceil)\alpha + m(1 + \lceil \log_2 p \rceil)\beta + m\lceil \log_2 p \rceil\gamma$, where the extra round (the α-term) stems from the need to send the final result to the eliminated processes. For large $m > b$ the total time is $t = 2\lceil \log_2 p \rceil\alpha + 2m(1.5 - 1/p')\beta + m(1.5 - 1/p')\gamma$ with $p' = 2^n$ being the largest power of two smaller than p.

This protocol is designed only for odd numbers of processes. For any number of processes it must be combined with the butterfly.

3.4 Small ring

Let now $p = r^q 2^n$. The idea here is to handle the reduction step for the r^q factor by a ring. For $r - 1$ rounds process i receives data from process $(i-1) \bmod r$ and sends data to process $(i+1) \bmod r$. For $m > b$ each process sends/receives only m/r elements per round, whereas for $m \leq b$ each process sends its full input vector along the ring. After the last step each process sequentially reduces the elements received: the requirements 1 and 2 make it necessary to postpone the local reductions until data from all processes have been received. For $m > b$ each process has m/r elements of the result vector $m_0 + m_1 + \ldots + m_{r-1}$. We note that the butterfly exchange step can be viewed as a 2-ring; the ring algorithm is thus a natural generalization of the butterfly algorithm.

For small $m < b$ and if also $r > 3$ the optimal allgather algorithm of [6] would actually be much preferable; however, the sequential reduction remains a bottleneck, and this idea is therefore only attractive for small p (dependent on the ratio of α and β to γ).

Substituting the ring algorithm for the neighbor exchange step in the algorithm of Figure 1, we can implement the complete reduction phase in $(r-1)q+n$

p	m	Ring, latency-opt.	Elimination, lat.-opt.	Ring, bandwidth-opt.	Elimination, bw-opt.
3	S	2 + 0.2 + 0.02 = **2.22** *	3 + 0.3 + 0.02 = 3.32	4 + 0.13 + 0.01 = 4.14	4 + 0.20 + 0.01 = 4.21
	M	2 + 2.0 + 0.20 = **4.20** *	3 + 3.0 + 0.20 = 6.20	4 + 1.33 + 0.07 = 5.40	4 + 2.00 + 0.10 = 6.10
	L	2 + 20. + 2.00 = 24.0	3 + 30. + 2.00 = 35.0	4 + 13.3 + 0.67 = **18.0** *	4 + 20.0 + 1.00 = 25.0
5	S	3 + 0.4 + 0.04 = **3.44** *	4 + 0.4 + 0.03 = 4.43	7 + 0.16 + 0.01 = 7.17	6 + 0.25 + 0.01 = 6.26
	M	3 + 4.0 + 0.40 = **7.40** *	4 + 4.0 + 0.30 = 8.30	7 + 1.60 + 0.08 = 8.68	6 + 2.50 + 0.13 = 8.63
	L	3 + 40. + 4.00 = 47.0	4 + 40. + 3.00 = 47.0	7 + 16.0 + 0.80 = **23.8** *	6 + 25.0 + 1.25 = 32.3
7	S	3 + 0.6 + 0.06 = **3.66** *	4 + 0.4 + 0.03 = 4.43	9 + 0.17 + 0.01 = 9.18	6 + 0.25 + 0.01 = 6.26
	M	3 + 6.0 + 0.60 = 9.60	4 + 4.0 + 0.30 = **8.30** *	9 + 1.71 + 0.09 = 10.8	6 + 2.50 + 0.13 = 8.63
	L	3 + 60. + 6.00 = 69.0	4 + 40. + 3.00 = 47.0	9 + 17.1 + 0.86 = **27.0** *	6 + 25.0 + 1.25 = 32.3
13	S	4 + 1.2 + 0.12 = **5.32** *	5 + 0.5 + 0.04 = 5.54	16 + 0.19 + 0.01 = 16.2	8 + 0.28 + 0.01 = 8.29
	M	4 + 12. + 1.20 = 17.2	5 + 5.0 + 0.40 = **10.4** *	16 + 1.85 + 0.09 = 18.0	8 + 2.75 + 0.14 = 10.9
	L	4 + 120. + 12.0 = 136.	5 + 50. + 4.00 = 59.0	16 + 18.5 + 0.92 = **35.4** *	8 + 27.5 + 1.38 = 36.9
15	S	4 + 1.4 + 0.14 = **5.54** *	5 + 0.5 + 0.04 = **5.54** *	18 + 0.19 + 0.01 = 18.2	8 + 0.28 + 0.01 = 8.29
	M	4 + 14. + 1.40 = 19.4	5 + 5.0 + 0.40 = **10.4** *	18 + 1.87 + 0.09 = 20.0	8 + 2.75 + 0.14 = 10.9
	L	4 + 140. + 14.0 = 158.	5 + 50. + 4.00 = 59.0	18 + 18.7 + 0.93 = 37.6	8 + 27.5 + 1.38 = **36.9** *
23	S	5 + 2.2 + 0.22 = 7.42	6 + 0.6 + 0.05 = **6.65** *	27 + 0.19 + 0.01 = 27.2	10 + 0.29 + 0.01 = 10.3
	M	5 + 22. + 2.20 = 29.2	6 + 6.0 + 0.50 = **12.5** *	27 + 1.91 + 0.10 = 29.0	10 + 2.88 + 0.14 = 13.0
	L	5 + 220. + 22.0 = 247.	6 + 60. + 5.00 = 71.0	27 + 19.1 + 0.96 = 47.1	10 + 28.8 + 1.44 = **40.2** *
	XL	5 + 2200 + 220. = 2425	6 + 600 + 50.0 = 656.	27 + 191. + 9.60 = **228.** *	10 + 288. + 14.4 = 312.
63	S	6 + 6.2 + 0.62 = 12.8	7 + 0.6 + 0.06 = **7.66** *	68 + 0.19 + 0.01 = 68.2	12 + 0.29 + 0.01 = 12.3
	M	6 + 62. + 6.20 = 74.2	7 + 6.0 + 0.60 = **13.6** *	68 + 1.97 + 0.10 = 70.1	12 + 2.94 + 0.15 = 15.1
	L	6 + 620. + 62.0 = 688.	7 + 60. + 6.00 = 73.0	68 + 19.7 + 0.98 = 88.7	12 + 29.4 + 1.47 = 42.9
	XL	6 + 6200 + 620. = 6826	7 + 600 + 60.0 = 667.	68 + 197. + 9.80 = **275.** *	12 + 294. + 14.7 = 321.

message size m: S: $\beta m = 0.1\alpha$, $\gamma m = 0.01\alpha$; L: $\beta m = 10\,\alpha$, $\gamma m = 1.00\alpha$;
M: $\beta m = 1.0\alpha$, $\gamma m = 0.10\alpha$; XL: $\beta m = 100\alpha$, $\gamma m = 10.0\alpha$;

Table 1. Execution time of the four protocols for odd numbers of processes (p) and different message sizes. The time is displayed as multiples of the message transfer latency α. In each line, the fastest protocol is marked (*).

rounds. This gives a theoretical improvement for $r = 3$ and $q = 1, 2$ to the optimal number of $\lceil \log_2 p \rceil$ rounds. The general algorithm would require $\lceil \log_2 p \rceil + 1$ rounds, one more than optimal, whereas the algorithm with ring steps takes 1 round less. Let for example $p = 12 = 3 \times 2^2$. The ring based algorithm needs $2 + 2 = 4$ rounds, whereas the general algorithm would take $\lceil \log_2 12 \rceil + 1 = 4 + 1 = 5$ rounds.

3.5 Comparison

The time needed for latency-optimized (exchange of full buffers) and bandwidth-optimized (recursive buffer halving or exchange of $1/p$ of the buffer) protocols are:

$$\begin{array}{llll} t_{\mathrm{ring,lat-opt.}} & = \alpha\lceil \log_2 p \rceil & +\beta m(p-1) & +\gamma m(p-1) \\ t_{\mathrm{elim.,lat-opt.}} & = \alpha(\lceil \log_2 p \rceil + 1) & +\beta m(\lceil \log_2 p \rceil + 1) & +\gamma m(\lceil \log_2 p \rceil + 1) \\ t_{\mathrm{ring,bw-opt.}} & = \alpha(\lceil \log_2 p \rceil + p - 1) & +\beta m(2(1-1/p)) & +\gamma m(1-1/p) \\ t_{\mathrm{elim.,bw-opt.}} & = \alpha(2\lceil \log_2 p \rceil) & +\beta m(2(1.5-1/p')) & +\gamma m(1.5-1/p') \end{array}$$

with $p' = 2^{\lfloor \log_2 p \rfloor}$. Table 1 compares the 4 algorithms for four cases based on different rations $\beta m/\alpha$ and $\gamma m/\alpha$, and for several numbers of processes p. The fastest protocol is marked in each line. Note, that this table does not necessarily gives the optimal values for the elimination protocols because they may be achieved by using some internal steps with buffer halving and the further steps without buffer halving. One can see that each algorithm has a usage range, where it is significantly faster than the other protocols.

3.6 Putting the pieces together

The 3-2-elimination step and the ring exchange were two alternative exchange patterns that could be plugged into the high-level algorithm of Figure 1 for non-powers-of-two, see also Fig 6. The number of processes $p = 2^n q_1 q_2 ... q_h$ is factorized in a) 2^n for the butterfly protocol, b) small odd numbers q_1, ... q_{h-1} for the ring protocol, and c) finally an odd number q_h for the 3-2-elimination or 2-1-elimination protocol. For given p it is of course essential that each process i at each round z can determine efficiently (i.e., in *constant time*) what protocol is to be used. This amounts to determining a) exchange step (butterfly, 3-2-elimination, 2-1-elimination, ring), b) neighboring process(es), and c) whether the process will be active for the following rounds. We did not give the details; however, for all protocols outlined in the paper this is indeed the case, but as shortcut, Table 1 is now used for the odd factors q_i and vector size reduced by $1/2^n$ if the butterfly protocol uses buffer halving due to long vectors.

4 Conclusion and open problems

We presented an improved algorithm for the `MPI_Allreduce` collective for the important case where the number of participating processes (p) is not a power of two, i.e., $p = 2^n q$ with odd q and $n \geq 0$. For general non-powers-of-two and small vectors, our algorithm requires $\lceil \log_2 p \rceil + 1$ rounds - one round off from optimal. For **large vectors** twice the number of rounds is needed, but the communication and computation time is less than $(1+1/2^{n+1})(2m\beta + m\gamma)$, i.e., an improvement from $2(2m\beta + m\gamma)$ achieved by previous algorithms [15], e.g., with $p = 24$ or 40, the execution time can be reduced by 47%. For **small vectors** and small q our algorithm achieves the optimal $\lceil \log_2 p \rceil$ number of rounds.

The main open problem is whether a latency optimal allreduce algorithm under the MPI constraint 1- 3 with $\lceil \log_2 p \rceil$ rounds is possible for any number of processes. We are not aware of results to the contrary.

References

1. M. Barnett, S. Gupta, D. Payne, L. Shuler, R. van de Gejin, and J. Watts, *Interprocessor collective communication library (InterCom)*, in Proceedings of Supercomputing '94, Nov. 1994.
2. A. Bar-Noy, J. Bruck, C.-T. Ho, S. Kipnis, and B. Schieber. Computing global combine operations in the multiport postal model. *IEEE Transactions on Parallel and Distributed Systems*, 6(8):896–900, 1995.
3. A. Bar-Noy, S. Kipnis, and B. Schieber. An optimal algorithm for computing census functions in message-passing systems. *Parallel Processing Letters*, 3(1):19–23, 1993.
4. E. K. Blum, X. Wang, and P. Leung. Architectures and message-passing algorithms for cluster computing: Design and performance. *Parallel Computing* 26 (2000) 313–332.
5. J. Bruck and C.-T. Ho. Efficient global combine operations in multi-port message-passing systems. *Parallel Processing Letters*, 3(4):335–346, 1993.

6. J. Bruck, C.-T. Ho, S. Kipnis, E. Upfal, and D. Weathersby. Efficient algorithms for all-to-all communications in multiport message-passing systems. *IEEE Transactions on Parallel and Distributed Systems*, 8(11):1143–1156, Nov. 1997.
7. E. Gabriel, M. Resch, and R. Rühle, *Implementing MPI with optimized algorithms for metacomputing*, in Proceedings of the MPIDC'99, Atlanta, USA, pp 31–41.
8. N. Karonis, B. de Supinski, I. Foster, W. Gropp, E. Lusk, and J. Bresnahan, *Exploiting hierarchy in parallel computer networks to optimize collective operation performance*, in Proceedings of the 14th International Parallel and Distributed Processing Symposium (IPDPS '00), 2000, pp 377–384.
9. T. Kielmann, R. F. H. Hofman, H. E. Bal, A. Plaat, R. A. F. Bhoedjang, *MPI's reduction operations in clustered wide area systems*, in Proceedings of the MPIDC'99, Atlanta, USA, March 1999, pp 43–52.
10. A. D. Knies, F. Ray Barriuso, W. J. H., G. B. Adams III, *SLICC: A low latency interface for collective communications*, in Proceedings of the 1994 conference on Supercomputing, Washington, D.C., Nov. 14–18, 1994, pp 89–96.
11. Howard Pritchard, Jeff Nicholson, and Jim Schwarzmeier, *Optimizing MPI Collectives for the Cray X1*, in Proceeding of the CUG 2004 conference, Knoxville, Tennessee, USA, May, 17-21, 2004 (personal communication).
12. R. Rabenseifner, *Automatic MPI counter profiling of all users: First results on a CRAY T3E 900-512*, Proceedings of the Message Passing Interface Developer's and User's Conference 1999 (MPIDC'99), Atlanta, USA, March 1999, pp 77–85.
13. R. Rabenseifner. Optimization of collective reduction operations. In *M. Bubak et al. (Eds.): International Conference on Computational Science (ICCS 2004)*, volume 3036 of *Lecture Notes in Computer Science*, pages 1–9, 2004.
14. M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI – The Complete Reference*, volume 1, The MPI Core. MIT Press, second edition, 1998.
15. R. Thakur and W. D. Gropp. Improving the performance of collective operations in MPICH. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface. 10th European PVM/MPI Users' Group Meeting*, volume 2840 of *Lecture Notes in Computer Science*, pages 257–267, 2003.
16. R. van de Geijn. On global combine operations. *Journal of Parallel and Distributed Computing*, 22:324–328, 1994.