Butterfly-like Algorithms for GASPI Split Phase Allreduce

Vanessa End
and Ramin Yahyapour

Gesellschaft für wissenschaftliche
Datenverarbeitung mbH Göttingen
Göttingen, Germany
Email: <vanessa.end>,
<ramin.yahyapour>@gwdg.de

Christian Simmendinger
and Thomas Alrutz

T-Systems Solutions for Research GmbH
Stuttgart/Göttingen, Germany
Email: <christian.simmendinger>,
<thomas.alrutz>@t-systems-sfr.com

***Abstract*—Collective communication routines pose a significant bottleneck of highly parallel programs. Research on different algorithms for disseminating information among all participating processes in a collective communication has brought forth many different algorithms, some of which have a butterfly-like communication scheme. While these algorithms have been abandoned from usage in collective communication routines with larger messages, due to the congestion that arises from their use, these algorithms have ideal properties for split-phase allreduce routines: all processes are involved in the computation of the result in each communication round and they have few communication rounds. This article will present several different algorithms with a butterfly-like communication scheme and examine their usability for a GASPI allreduce library routine. The library routines will be compared to state-of-the-art MPI implementations and also to a tree-based allreduce algorithm.**

Keywords–GASPI; Allreduce; Partitioned Global Address Space (PGAS); Collective Communication; Algorithms.

I. INTRODUCTION

In high performance computing (HPC), one of the main bottlenecks is always communication. As we are looking into the exascale age, this bottleneck becomes even more important than before: with more processes participating in the computation of a problem, also more communication between these processes is necessary. But already in the past, this bottleneck has been observed - especially when using collective communication routines, e.g., barrier or allreduce routines, where all processes (of a given group) are active in the communication. Therefore, many different algorithms have been developed in the course of time to reduce the runtime of collective routines and thus, the overall communication overhead. Key to this reduction of runtime is the underlying communication algorithm.

In this paper, we extend our work from [1], where we have introduced an adaption of the n-way dissemination algorithm, such that it is usable for split-phase allreduce operations, as they are defined in, e.g., the Global Address Space Programming Interface (GASPI) specification [2]. GASPI is based on one-sided communication semantics, distinguishing it from message-passing paradigms, libraries and application programming interfaces (API) like the Message-Passing Interface (MPI) standard [3]. In the spirit of hybrid programming (e.g., combined MPI and OpenMP communication) for improved performance, GASPI's communication routines are designed for inter-node communication and leaves it to the programmer to include another communication interface for intra-node, i.e., shared-memory communication. Thus, one GASPI process is started per node or cache coherent non-uniform memory access (ccNUMA) socket.

To enable the programmer to design a fault-tolerant application and to achieve perfect overlap of communication and computation, GASPI's non-local operations are equipped with a timeout mechanism. By either using one of the predefined constants `GASPI_BLOCK` or `GASPI_TEST` or by giving a user-defined timeout value, non-local routines can either be called in a blocking or a non-blocking manner. In the same way, GASPI also defines split-phase collective communication routines, namely `gaspi_barrier`, `gaspi_allreduce` and `gaspi_allreduce_user`, for which the user can define a personal reduce routine. The goal of our research is to find a fast algorithm for the allreduce operation, which has a small number of communication rounds and, whenever possible, uses all available resources for the computation of the partial results computed in each communication round.

Collective communication is an important issue in high performance computing and thus, research on algorithms for the different collective communication routines has been pursued in the last decades. In the area of the allreduce operation, influences from all other communication algorithms can be used, e.g., tree algorithms like the binomial spanning tree (BST) [4] or the tree algorithm of Mellor-Crummey and Scott [5]. These are then used to first reduce and then broadcast the data. Also, more barrier related algorithms like the butterfly barrier of Brooks [6] or the tournament algorithm described by Debra Hensgen et al. in the same paper as the dissemination algorithm [7] influence allreduce algorithms.

Yet, none of these algorithms seems fit for the challenges of split-phase remote direct memory access (RDMA) allreduce, with potentially computation-intense user-defined reduce operations over an InfiniBand network. The tree algorithms have a tree depth of $\lceil \log_2(P) \rceil$ and have to be run through twice, leading to a total of $2\lceil \log_2(P) \rceil$ communication rounds.

In each of these rounds, a large part of the participating ranks remain idling, while the n-way dissemination algorithm and Bruck's algorithm only need $\lceil \log_{n+1}(P) \rceil$ communication rounds and involve all ranks in every round. Also, the butterfly barrier has $k = \lceil \log_2(P) \rceil$ communication rounds to traverse, but it is also only fit for 2^k participants.

There are two key features which make (n-way) dissemination based allreduce operations very interesting for both split-phase implementations as well as user-defined reductions, like they are both defined in the GASPI specification [2].

1) Split-phase collectives either require an external active progress component or, alternatively, progress has to be achieved through suitable calls from the calling processes. Since the underlying algorithm for the split-phase collectives is unknown to the enduser, all participating processes have to repeatedly call the collective several times. Algorithms for split-phase collectives hence ideally both involve all processes in every communication step and moreover ideally require a minimum number of steps (and thus a minimum number of calls). The n-way dissemination algorithm exactly matches these requirements. It requires a very small number of communication rounds of order $\lceil \log_{n+1}(P) \rceil$ and additionally involves every process in all communication rounds.
2) User-defined collectives share some of the above requirements in the sense that CPU-expensive local reductions ideally should leverage every calling CPU in each round and ideally would require a minimum number of communication rounds (and hence a minimum number of expensive local reductions).

In the following section, we will describe related work. In Section III, we will shortly introduce the algorithms chosen for the experiments, elaborating on the adaption of the n-way dissemination algorithm. In addition to the adapted n-way dissemination algorithm, this paper will also present Bruck's algorithm [8] and the butterfly algorithm [6] with two adaptions for $P \neq 2^k$ in more detail. While we have only shown experimental results of the allreduce function with the sum operation in the former paper, we will now also show results using the minimum and the maximum operation in allreduce. The experimental setup and experimental results are presented in Section IV, where we also evaluate the results of the experiments. Section V will then give a conclusion of the work and an outlook on future work.

II. Related Work

Some related work, especially in terms of developed algorithms, has already been presented in the introduction. Still to mention is the group around Jehoshua Bruck, which has done much research on multi-port algorithms, hereby developing a k-port algorithm with a very similar communication scheme as that of the n-way dissemination algorithm [8], [9]. These works were found relatively late in the implementation phase of the adapted n-way dissemination algorithm, why an extensive comparison of the two has been postponed to this paper.

In the past years, more and more emphasis has been put on RDMA techniques and algorithms [10][11] due to hardware development, e.g., InfiniBand™[12] or RDMA over Converged Ethernet (RoCE) [13]. While Panda et al. [10] exploit the multicast feature of InfiniBand™, this is not an option for us because the multicast is a so called unreliable operation and in addition an optional feature of the InfiniBand™architecture [12]. Congestion in fat tree configured networks is still a topic in research, where for example Zahavi is an active researcher [14]. While a change of the routing tables or routing algorithm is often not an option for application programmers, the adaption of node orders within the API is a possible option.

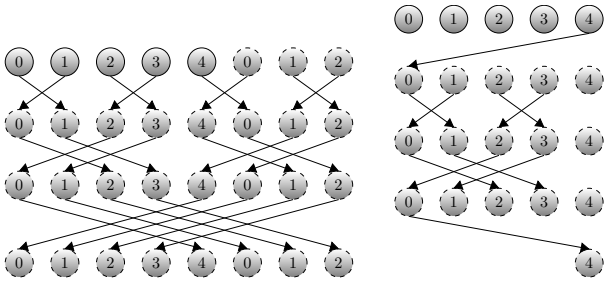

Figure 1. Comparison of the Butterfly Algorithm for $P = 5$ with virtual processes (left) to the Pairwise Exchange Algorithm for the same number of processes (right).

III. Algorithms

Since communication is one of the most important bottlenecks in parallel computing, many different algorithms have been developed for the numerous different collective communication routines. In this section, several algorithms, usable for collective communication routines, will be presented. Our focus lies on algorithms with butterfly-like communication schemes, as these are at the moment not used for communication with large messages, but our initial research shows that in modern architecture, the congestion does not arise in the way expected. In addition to this, the algorithms are not used for allreduce operations at all, because they potentially deliver wrong results for some numbers of participating processes, if implemented in their original design. With some adaptions, this is no longer true for these algorithms. We start the presentation of algorithms with the name-giving algorithm, the butterfly algorithm.

A. Butterfly Algorithm and Pairwise Exchange Algorithm

Eugene D. Brooks introduced the butterfly algorithm in the *Butterfly Barrier* in 1986 [6]. It has been designed for operations with $P = 2^k$ participants. It then has $k = \lceil \log_2 P \rceil$ communication rounds, where in each round l, rank p communicates with $p \pm 2^{l-1}$. Since this algorithm was not intended for the use with $P = 2^{k-1} + q < 2^k$ processes, a first adaption was made: *virtual processes* were introduced to virtually have $P' = 2^k$ processes to use the algorithm on. Existing processes adopted the role of these virtual processes as depicted in Figure 1. Processes 0 to 2 act as if they were additional processes 5, 6 and 7 to comply to the communication scheme for $P = 8$. This introduces unnecessary additional communication and overhead. While this is not too dramatic in the case of a barrier, this becomes very interesting when the message sizes increase. Even when $P = 2^k$, the symmetric communication scheme of the butterfly algorithm quickly leads to congestion in network topologies where there is exactly one link from one processor to another, as this link will be used in both directions

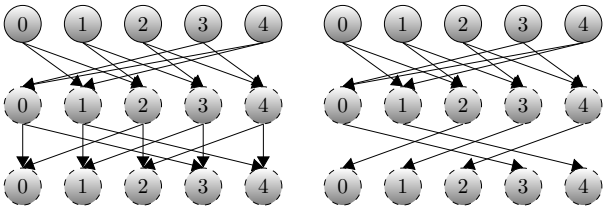

Figure 2. Comparison of the 2-way dissemination algorithm (left) to the adapted 2-way dissemination algorithm (right) for 5 ranks.

at the same time. In addition, the adaption makes the algorithm unusable for non-idempotent allreduce operations, as data is transferred and processed more than once.

A different adaption to the original algorithm leads to the *Pairwise Exchange Algorithm* (PE). This algorithm is identical to the previous algorithm if $P = 2^k$ and is for example described in [15]. If the number of processes is not a power of two, but rather $2^k + q$, then the q "leftover" processes first communicate with the first q processes and then wait until the 2^k remaining processes have finished the algorithm, as shown in Figure 1.

The first adaption of the butterfly algorithm would make it possible to use this algorithm for $P \neq 2^k$, but it would lead to a repeated inclusion of initial data from the virtual processes, if used for an allreduce. The pairwise exchange algorithm also solves this problem, why this will be the only adaption used in the experiments shown below.

B. (n-way) Dissemination Algorithm

The basis of the n-way dissemination algorithm is the dissemination algorithm developed by Hensgen et al. in 1988 [7]. To be exact, this algorithm is equivalent to a 1-way dissemination algorithm as it is defined by Hoefler et al. in 2006 [16]. In Hoefler's n-way dissemination algorithm, each participating process sends and receives n messages per communication round - instead of just one as presented by Hensgen et al.

Similar to the butterfly algorithm for $P \neq 2^k$, the n-way dissemination algorithm transfers certain data elements more than once to the participating ranks if $P \neq (n+1)^k$. This is exemplarily shown for a 2-way dissemination algorithm with 5 ranks in Figure 2 on the left. Nevertheless, the algorithm shows excellent performance in barrier operations, where it does not matter, whether a flag is communicated once or twice. It does not seriously impact the runtime and especially it does not alter the result of the routine. Using this algorithm for allreduce is not practicable in these cases though - the result will be wrong and different on all participating nodes. To still use this algorithm for allreduce operations, we have presented an adaption to the n-way dissemination algorithm, which overcomes these problems in [1]. The below described adaption of the communication scheme is depicted in Figure 2 in direct comparison to the original communication scheme.

The n-way dissemination algorithm, as presented in [16] has been developed for spreading data among the participants, where n is the number of messages transferred in each communication round. As the algorithm is not exclusive to nodes, cores, processes or threads, the term *ranks* will be used

TABLE I. ROUND-WISE COMPUTATION OF PARTIAL RESULTS IN A 2-WAY DISSEMINATION ALGORITHM (FROM [1])

rank	round 0	round 1	round 2
0	x_0	$S_1^0 = x_0 \circ x_8 \circ x_7$	$S_2^0 = S_1^0 \circ S_1^6 \circ S_1^3$
1	x_1	$S_1^1 = x_1 \circ x_0 \circ x_8$	$S_2^1 = S_1^1 \circ S_1^7 \circ S_1^4$
2	x_2	$S_1^2 = x_2 \circ x_1 \circ x_0$	$S_2^2 = S_1^2 \circ S_1^8 \circ S_1^5$
3	x_3	$S_1^3 = x_3 \circ x_2 \circ x_1$	$S_2^3 = S_1^3 \circ S_1^0 \circ S_1^6$
4	x_4	$S_1^4 = x_4 \circ x_3 \circ x_2$	$S_2^4 = S_1^4 \circ S_1^1 \circ S_1^7$
5	x_5	$S_1^5 = x_5 \circ x_4 \circ x_3$	$S_2^5 = S_1^5 \circ S_1^2 \circ S_1^8$
6	x_6	$S_1^6 = x_6 \circ x_5 \circ x_4$	$S_2^6 = S_1^6 \circ S_1^3 \circ S_1^0$
7	x_7	$S_1^7 = x_7 \circ x_6 \circ x_5$	$S_2^7 = S_1^7 \circ S_1^4 \circ S_1^1$
8	x_8	$S_1^8 = x_8 \circ x_7 \circ x_6$	$S_2^8 = S_1^8 \circ S_1^5 \circ S_1^2$

rank	round 0	round 1	round 2
0	x_0	$S_1^0 = x_0 \circ x_7 \circ x_6$	$S_2^0 = S_1^0 \circ S_1^5 \circ S_1^2$
1	x_1	$S_1^1 = x_1 \circ x_0 \circ x_7$	$S_2^1 = S_1^1 \circ S_1^6 \circ S_1^3$
2	x_2	$S_1^2 = x_2 \circ x_1 \circ x_0$	$S_2^2 = S_1^2 \circ S_1^7 \circ S_1^4$
3	x_3	$S_1^3 = x_3 \circ x_2 \circ x_1$	$S_2^3 = S_1^3 \circ S_1^0 \circ S_1^5$
4	x_4	$S_1^4 = x_4 \circ x_3 \circ x_2$	$S_2^4 = S_1^4 \circ S_1^1 \circ S_1^6$
5	x_5	$S_1^5 = x_5 \circ x_4 \circ x_3$	$S_2^5 = S_1^5 \circ S_1^2 \circ S_1^7$
6	x_6	$S_1^6 = x_6 \circ x_5 \circ x_4$	$S_2^6 = S_1^6 \circ S_1^3 \circ S_1^0$
7	x_7	$S_1^7 = x_7 \circ x_6 \circ x_5$	$S_2^7 = S_1^7 \circ S_1^4 \circ S_1^1$

in the following. The P participants in the collective operation are numbered consecutively from $0, \ldots, P-1$ and this number is their rank. With respect to rank p, the ranks $p+1$ and $p-1$ are called p's *neighbors*, where $p-1$ will be the *left-hand neighbor*.

Let P be the number of ranks involved in the collective communication. Then $k = \lceil \log_{n+1}(P) \rceil$ is the number of communication rounds the n-way dissemination algorithm needs to traverse, before all ranks have all information. In every communication round $l \in \{1, \ldots, k\}$, every process p has n peers $s_{l,i}$, to which it transfers data and also n peers $r_{l,j}$, from which it receives data:

$$\begin{aligned} s_{l,i} &= p + i \cdot (n+1)^{l-1} \mod P \\ r_{l,j} &= p - j \cdot (n+1)^{l-1} \mod P, \end{aligned} \tag{1}$$

with $i, j \in \{1, \ldots, n\}$. Thus, in every round p gets (additional) information from $n(n+1)^{l-1}$ participating ranks - either directly or through the information obtained by the sending ranks in the preceding rounds.

When using the dissemination algorithm for an allreduce, the information received in every round is the partial result the sending rank has computed in the round before. The receiving rank then computes a new local partial result from the received data and the local partial result already at hand.

Let S_l^p be the partial result of rank p in round l, $\circ$ be the reduction operation used and x_p be the rank's initial data. Then rank p receives n partial results $S_{l-1}^{r_{l,i}}$ in round l and computes

$$S_l^p = S_{l-1}^p \circ S_{l-1}^{r_{l,1}} \circ S_{l-1}^{r_{l,2}} \circ \cdots \circ S_{l-1}^{r_{l,n}}, \tag{2}$$

which it transfers to its peers $s_{l+1,i}$ in the next round. This data movement is shown in Table I for an allreduce based on a 2-way dissemination algorithm. First for 9 ranks, then for 8 participating ranks. By expanding the result of rank 0 in round

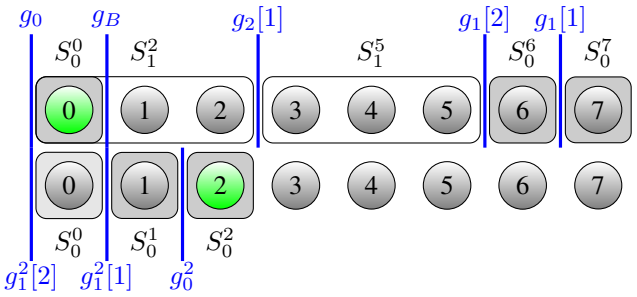

Figure 3. The data boundaries g and received partial results $S_i^{r_l,j}$ of ranks 0 and 2 (from [1]).

2 from the second table, it becomes visible, that the reduction operation has been applied twice to x_0:

$$S_2^0 = (x_0 \circ x_7 \circ x_6) \circ (x_5 \circ x_4 \circ x_3) \circ (x_2 \circ x_1 \circ x_0). \quad (3)$$

In general, if $P \neq (n+1)^k$, the final result will include data of at least one rank twice: In every communication round l, each rank receives n partial results each of which is the reduction of the initial data of its $(n+1)^{l-1}$ left-hand neighbors. Thus, the number of included initial data elements is described through

$$\sum_{i=1}^{l} n(n+1)^{i-1} + 1 = (n+1)^l \quad (4)$$

for every round l.

In the cases of the maximum or minimum operation to be performed in the allreduce, this does not matter. In the case of a summation though, this dilemma will result into different final sums on the participating ranks. In general, the adaption is needed for all operations, where the repeated application of the function to the same element changes the final result, so called *non-idempotent* functions.

The adaption of the n-way dissemination algorithm is mainly based on these two properties: (1) in every round l, p receives n new partial results. (2) These partial results are the result of the combination of the data of the next $\sum_{i=0}^{l-1} n(n+1)^{i-1} + 1$ left-hand neighbors of the sender. This is depicted in Figure 3 through boxes. Highlighted in green are those ranks, whose data view is represented, that is rank 0's in the first row and rank 2's in the second row. Each box encloses those ranks, whose initial data is included in the partial result the right most rank in the box has transferred in a given round. This means for rank 0, it has its own data, received S_0^6 and S_0^7 in the first round (gray boxes) and will receive S_1^5 and S_1^2 from ranks 2 and 5 in round 2 (white boxes).

As each of the boxes describes one of the partial results received, the included initial data items can not be retrieved by the destination rank. The change from one box to the next is thus defined as a *data boundary*. The main idea of the adaption is to find data boundaries in the data of the source ranks in the last round, which coincide with data boundaries in the destination rank's data. When such a correspondence is found, the data sent in the last round is reduced accordingly. To be able to do so, it is necessary to describe these boundaries in a mathematical manner. Considering the data elements included in each partial result received, the data boundaries of the receiver p can be described as:

$$g_{l_{\text{rcv}}}[j_{\text{rcv}}] = p - n \sum_{i=0}^{l_{\text{rcv}}-2} (n+1)^i - j_{\text{rcv}}(n+1)^{l_{\text{rcv}}-1} \bmod P, \quad (5)$$

where $j_{\text{rcv}}(n+1)^{l_{\text{rcv}}-1}$ describes the boundary created through the data transferred by rank $r_{l_{\text{rcv}},j_{\text{rcv}}}$ in round l_{rcv}.

Also, the sending ranks have received partial results in the preceding rounds, which are marked through corresponding boundaries. From the view of rank p in the last round k, these boundaries are then described through

$$g^s_{l_{\text{snd}}}[j_{\text{snd}}] = p - s(n+1)^{k-1} - n \sum_{i=0}^{l_{\text{snd}}-2} (n+1)^i - j_{\text{snd}}(n+1)^{l_{\text{snd}}-1} \bmod P, \quad (6)$$

with $s \in \{1, \ldots, n\}$ distinguishing the n senders and j_{snd}, l_{snd} corresponding to the above j_{rcv}, l_{rcv} for the sending rank. To also consider those cases, where only the initial data of the sending or the receiving rank is included more than once in the final result, we let l_{snd}, $l_{\text{rcv}} \in \{0, \ldots, k-1\}$ and introduce an additional *base border* g_B in the destination rank's data.

These boundaries are also depicted in Figure 3 for the previously given example of a 2-way dissemination algorithm with 8 ranks. The figure depicts the data present on ranks 0 and 2 after the first communication round in the gray boxes with according boundaries g_B, g_0, $g_1[1]$ and $g_1[2]$ on rank 0 and g_0^2, $g_1^2[1]$ and $g_1^2[2]$ on rank 2. Since the boundaries g_B and $g_1^2[1]$ coincide, the first sender in the last round, that is rank 5, transfers its partial result but rank 2 only transfers a reduction $S' = x_2 \circ x_1$ instead of $x_2 \circ x_1 \circ x_0$.

More generally speaking, the algorithm is adaptable, if there are boundaries on the source rank that coincide with boundaries on the destination rank, i.e.,

$$g^s_{l_{\text{snd}}}[j_{\text{snd}}] = g_{l_{\text{rcv}}}[j_{\text{rcv}}] \quad (7)$$

or $g^s_{l_{\text{snd}}}[j_{\text{snd}}] = g_B$. Then the last source rank, defined through s, transfers only the data up to the given boundary and the receiving rank takes the partial result up to its given boundary out of the final result. Taking out the partial result in this context means: if the given operation has an inverse $\circ^{-1}$, apply this to the final result and the partial result defined through $g_{l_{\text{rcv}}}[j_{\text{rcv}}]$. If the operation does not have an inverse, recalculate the final result, hereby omitting the partial result defined through $g_{l_{\text{rcv}}}[j_{\text{rcv}}]$. Since this boundary is known from the very beginning, it is possible to store this partial result in the round it is created, thus saving additional computation time at the end.

From this, one can directly deduce the number of participating ranks P, for which the n-way dissemination algorithm

is adaptable in this manner:

$$\begin{aligned} P &= g^s_{l_{\mathrm{snd}}}[j_{\mathrm{snd}}] - g_{l_{\mathrm{rcv}}}[j_{\mathrm{rcv}}] \\ &= s(n+1)^{k-1} + n\sum_{i=0}^{l_{\mathrm{snd}}-2}(n+1)^i + j_{\mathrm{snd}}(n+1)^{l_{\mathrm{snd}}-1} \\ &\quad - n\sum_{i=0}^{l_{\mathrm{rcv}}-2}(n+1)^i - j_{\mathrm{rcv}}(n+1)^{l_{\mathrm{rcv}}-1}\ . \end{aligned} \tag{8}$$

For given P, a 5-tuple $(s, l_{\mathrm{snd}}, l_{\mathrm{rcv}}, j_{\mathrm{snd}}, j_{\mathrm{rcv}})$ can be precalculated for different n. Then this 5-tuple also describes the adaption of the algorithm:

Theorem 1: Given the 5-tuple $(s, l_{\mathrm{snd}}, l_{\mathrm{rcv}}, j_{\mathrm{snd}}, j_{\mathrm{rcv}})$, the last round of the n-way dissemination algorithm is adapted through one of the following cases:

1) $l_{\mathrm{rcv}}, l_{\mathrm{snd}} > 0$
The sender $p - s(n+1)^{k-1}$ sends its partial result up to $g^s_{l_{\mathrm{snd}}}[j_{\mathrm{snd}}]$ and the receiver takes out its partial result up to the boundary $g_{l_{\mathrm{rcv}}}[j_{\mathrm{rcv}}]$.
2) $l_{\mathrm{rcv}} > 0, l_{\mathrm{snd}} = 0$
The sender $p - s(n+1)^{k-1}$ sends its own data and the receiver takes out its partial result up to the boundary $g_{l_{\mathrm{rcv}}}[j_{\mathrm{rcv}}]$.
3) $l_{\mathrm{rcv}} = 0, l_{\mathrm{snd}} = 0$
The sender $p - (s-1)(n+1)^{k-1}$ sends its last calculated partial result. If $s = 1$ the algorithm ends after $k-1$ rounds.
4) $l_{\mathrm{rcv}} = 0, l_{\mathrm{snd}} = 1$
The sender $p - s(n+1)^{k-1}$ sends its partial result up to $g^s_{l_{\mathrm{snd}}}[j_{\mathrm{snd}} - 1]$. If $j_{\mathrm{snd}} = 1$, the sender only sends its initial data.
5) $l_{\mathrm{rcv}} = 0, l_{\mathrm{snd}} > 1$
The sender $p - s(n+1)^{k-1}$ sends its partial result up to $g^s_{l_{\mathrm{snd}}}[j_{\mathrm{snd}}]$ and the receiver takes out its initial data from the final result.

Proof: We show the correctness of the above theorem by using that at the end each process will have to calculate the final result from P different data elements. We therefore look at (8) and how the given 5-tuple changes the terms of relevance. We will again need the fact, that the received partial results are always a composition of the initial data of neighboring elements.

1) $l_{\mathrm{rcv}}, l_{\mathrm{snd}} > 0$:

$$\begin{aligned} P &= s\,(n+1)^{k-1} + n\sum_{i=0}^{l_{\mathrm{snd}}-2}(n+1)^i + j_{\mathrm{snd}}\,(n+1)^{l_{\mathrm{snd}}-1} \\ &\quad - n\sum_{i=0}^{l_{\mathrm{rcv}}-2}(n+1)^i - j_{\mathrm{rcv}}\,(n+1)^{l_{\mathrm{rcv}}-1} \\ &= g^s_{l_{\mathrm{snd}}}[j_{\mathrm{snd}}] - g_{l_{\mathrm{rcv}}}[j_{\mathrm{rcv}}]\ . \end{aligned} \tag{9}$$

In order to have the result of P elements, the sender must thus transfer the partial result including the data up to $g^s_{l_{\mathrm{snd}}}[j_{\mathrm{snd}}]$ and the receiver takes out the elements up to $g_{l_{\mathrm{rcv}}}[j_{\mathrm{rcv}}]$.

2) $l_{\mathrm{rcv}} > 0, l_{\mathrm{snd}} = 0$:

$$\begin{aligned} P &= s\,(n+1)^{k-1} - n\sum_{i=0}^{l_{\mathrm{rcv}}-2}(n+1)^i - j_{\mathrm{rcv}}\,(n+1)^{l_{\mathrm{rcv}}-1} \\ &= s\,(n+1)^{k-1} - g_{l_{\mathrm{rcv}}}[j_{\mathrm{rcv}}] \end{aligned} \tag{10}$$

and thus we see that the sender must send only its own data, while the receiver takes out data up to $g_{l_{\mathrm{rcv}}}[j_{\mathrm{rcv}}]$.

3) $l_{\mathrm{rcv}} = 0, l_{\mathrm{snd}} = 0$:

$$P \;=\; s\,(n+1)^{k-1}\ . \tag{11}$$

In the first $k-1$ rounds, the receiving rank will already have the partial result of $n\sum_{i=1}^{k-1}(n+1)^i = (n+1)^{k-1} - 1$ elements. In the last round it then receives the partial sums of $(s-1)\,(n+1)^{k-1}$ further elements by the first $s-1$ senders and can thus compute the partial result from a total of $(s-1)\,(n+1)^{k-1} + (n+1)^{k-1} = s\,(n+1)^{k-1} - 1$ elements. Including its own data makes the final result of $s\,(n+1)^{k-1} = P$ elements. If $s = 1$ the algorithm is done after $k-1$ rounds.

4) $l_{\mathrm{rcv}} = 0, l_{\mathrm{snd}} = 1$:

$$P \;=\; s\,(n+1)^{k-1} + j_{\mathrm{snd}} \tag{12}$$

Following the same argumentation as above, the receiving rank will have the partial result of $s\,(n+1)^{k-1} - 1$ elements. It thus still needs

$$\begin{aligned} & P - \left(s\,(n+1)^{k-1} - 1\right) \\ =\ & s\,(n+1)^{k-1} + j_{\mathrm{snd}} - s\,(n+1)^{k-1} + 1 \\ =\ & j_{\mathrm{snd}} + 1 \end{aligned} \tag{13}$$

elements. Now, taking into account its own data it still needs j_{snd} data elements. The data boundary $g_1\,[j_{\mathrm{snd}}]$ of the sender includes j_{snd} elements plus its own data, i.e., $j_{\mathrm{snd}}+1$ elements. The $j^{\mathrm{th}}_{\mathrm{snd}}$ element will then be the receiving rank's data, thus it suffices to send up to $g_1\,[j_{\mathrm{snd}} - 1]$.

5) $l_{\mathrm{rcv}} = 0, l_{\mathrm{snd}} > 1$:

$$\begin{aligned} P \;&=\; s\,(n+1)^{k-1} \\ &\quad +\; n\sum_{i=0}^{l_{\mathrm{snd}}-2}(n+1)^i + j_{\mathrm{snd}}\,(n+1)^{l_{\mathrm{snd}}-1} \end{aligned} \tag{14}$$

In this case, the sender sends a partial result which necessarily includes the initial data of the receiving rank. This means that the receiving rank has to take out its own initial data from the final result. Due to $l_{\mathrm{snd}} > 1$ the sender will not be able to take a single initial data element out of the partial result to be transferred. ■

Note that the case where a data boundary on the sending side corresponds to the base border on the receiving side, i.e., $g^s_{l_{snd}}[j_{snd}] = g_B$, has not been covered above. In this case, there is no 5-tuple like above, but rather $P - 1 = g^s_{l_{snd}}[j_{snd}]$ and the adaption and reasoning complies to case 4 in the above theorem.

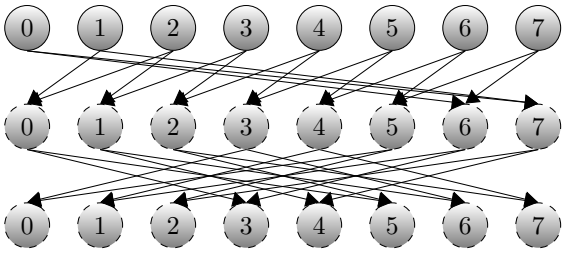

Figure 4. Communication scheme of Bruck's global combine algorithm for $P = 8$.

C. Bruck's Algorithm

In [8], Jehoshua Bruck and Ching-Tien Ho present two algorithms for global combine operations in n-port message-passing systems[1] The first of the two shows many similarities to the n-way dissemination algorithm presented above. While the dissemination algorithm and the n-way dissemination algorithm were both designed for barrier operations, Bruck's algorithm is explicitly designed for global combine operations, i.e., allreduces.

In $\lceil \log_{n+1}(P) \rceil$ communication rounds, every participating process transfers and receives n partial reduction results from other processes. Let $\circ$ be the reduction operation used and x_p be the initial data of process p. The partial results transferred by rank p in round l are computed in two versions: $S_l^p[0]$ is the reduction of all previously received results without the initial data of the computing process and $S_l^p[1] = x_p \circ S_l^p[0]$. In each round, the group of destination ranks is split up into two groups, one of which will receive S_p^0, and the other will receive S_p^1. For determining these groups, two things are necessary: the base $(n+1)$ representation of $P-1$ and the counter c, which counts the number of elements on which the reduction has already been performed.

For ease of readability, the algorithm will here be described with the help of an example for $P = 8$ and $n = 2$ from the view of rank 0. The complete communication scheme for this example is depicted in Figure 4. The general description and the proof can be found in [8]. The algorithm will need $k = \lceil \log_3(8) \rceil = 2$ communication rounds. For each of these rounds l, an α_{l-1} is needed to split the destination ranks in two groups: one receiving $S_l^p[0]$ and the other $S_l^p[1]$. These α_i are computed through the representation of $P - 1 = 7$ in a base 3 notation:

$$7 = (21)_3 = (\alpha_1 \alpha_0)_3. \tag{15}$$

In the first round, only the partial result $S_1^0[1] = x_0$ is transferred to $\alpha_{k-1} = \alpha_1 = 2$ process. The destination processes are

$$s_{1,1} \equiv p-1 \mod (P) \equiv -1 \mod 8 \equiv 7 \tag{16}$$
$$s_{1,2} \equiv p-2 \mod (P) \equiv -2 \mod 8 \equiv 6. \tag{17}$$

At the same time, rank 0 will receive partial results from its peers $r_{1,1} \equiv p+1 \mod (P) \equiv 1$ and $r_{1,2} = 2$, namely $S_1^1[1] = x_1$ and $S_1^2[1] = x_2$. Rank 0 can then calculate new partial results to be transferred in the following round:

$$S_2^0[0] = S_1^0[0] \circ S_1^1[1] \circ S_1^2[1] = x_1 \circ x_2 \tag{18}$$
$$S_2^0[1] = S_1^0[1] \circ S_1^1[1] \circ S_1^2[1] = x_0 \circ x_1 \circ x_2. \tag{19}$$

At the same time, c is increased to $c = \alpha_1 = 2$, which will be needed for the computation of the communication peers in the next round. Rank 0 will now transfer $S_2^0[1]$ to $\alpha_0 = 1$ rank:

$$s_{2,1} \equiv p - \alpha_0 \cdot (c+1) \mod (P) \equiv -3 \mod 8 \equiv 5, \tag{20}$$

and $S_2^0[0]$ to the remaining $n - \alpha_0 = 2 - 1 = 1$ rank:

$$s_{2,2} \equiv p - c - \alpha_0 \cdot (c+1) \mod (P) \equiv -5 \mod 8 \equiv 3. \tag{21}$$

At the same time, rank 0 will receive partial results from ranks

$$r_{2,1} \equiv p + (c+1) \mod (P) \equiv 3 \mod 8 \equiv 3 \tag{22}$$
$$r_{2,2} \equiv p + c + \alpha_0(c+1) \mod (P) \equiv 5 \mod 8 \equiv 5. \tag{23}$$

Then, rank 0 can compute the final result

$$S_3^0[1] = S_2^0[1] \circ S_2^3[1] \circ S_2^5[0] = x_0 \circ x_1 \circ x_2 \circ (x_3 \circ x_4 \circ x_5) \circ (x_6 \circ x_7). \tag{24}$$

Bruck's algorithm was the last to be presented in this paper, and a comparison of the different algorithms will be given in the next subsection.

D. Comparison

The algorithms with a butterfly-like communication scheme presented in this paper have some significant differences, starting with the number of communication rounds needed to complete the algorithm. The pairwise exchange algorithm needs $\lfloor \log_2(P) \rfloor + 2$ communication rounds, while the adapted n-way dissemination algorithm and Bruck's algorithm only need $\lceil \log_{n++1}(P) \rceil$ communication rounds. In a split-phase allreduce, this will lead to a significant difference in the number of repeated calls to the allreduce routine. In addition to that, q ranks will be idling in the PE algorithm, while the other $P - q$ ranks need to do some computation between the communication steps. To still exploit the full potential of a split-phase allreduce, an application will have to distribute the workload accordingly.

Even though Bruck's algorithm and the adapted n-way dissemination algorithm need the same number of communication rounds to complete an allreduce, an important difference is the applicability to different group sizes P. While Bruck's algorithm works for all pairs (n, P), the n-way dissemination algorithm can not be adapted for all pairs. In those cases, where the algorithm is not adaptable, alternative solutions need to be found for the n-way dissemination algorithm. One possibility could be, to transfer larger messages in the communication rounds, carrying not only a given partial result but maybe some additional initial data items to complete the allreduce properly. Nevertheless, the adaption to the n-way dissemination algorithm can be an important addition to the repertoire of allreduce algorithms in a communication library, because it makes sense to have different algorithms for different combinations of message sizes, number of participating ranks and reduction routines, as described, e.g., in [17].

[1]The notation has been heavily changed from the original paper to fit the notation throughout the rest of the paper.

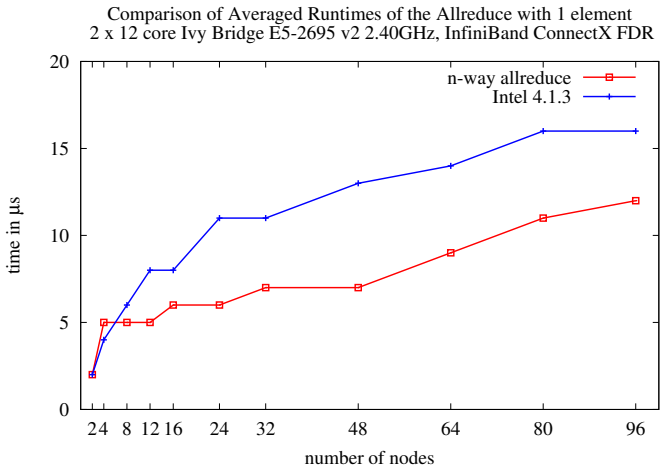

Figure 5. GASPI allreduce with 1 integer and SUM implemented on top of ibverbs in comparison to MPI allreduce (from [1]).

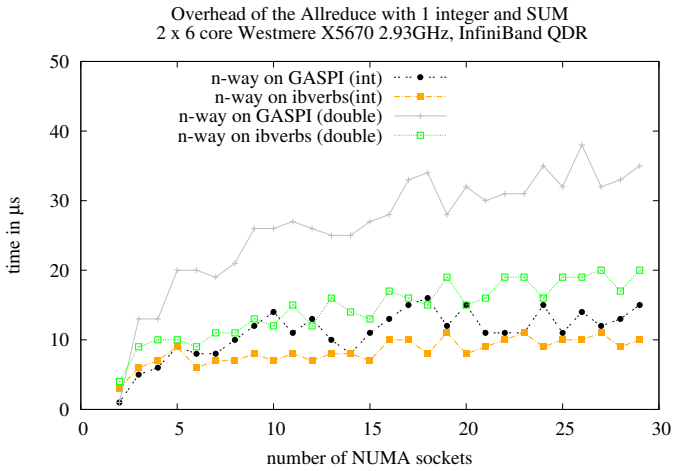

Figure 6. Comparison of allreduce with sum, implemented with ibverbs and implemented as GASPI library routine Cluster 1. One GASPI process per socket.

A possibly very important advantage of Bruck's algorithm and the n-way dissemination algorithm in comparison to the PE is the choice of communication peers. While the PE algorithm has a true butterfly communication scheme, the other two algorithms do not. Depending on the underlying network and routing, two messages will be transferred in opposite directions on the same path in a true butterfly scheme. This will not happen in the butterfly-like schemes of Bruck's algorithm and the n-way dissemination algorithm.

In this paper, we cannot show experiments and results for all different use-case scenarios, but will show a comparison of the three algorithms, implemented as allreduce library routines on top of GASPI.

IV. EXPERIMENTS AND RESULTS

We have implemented the described algorithms as allreduce library functions, using only GASPI routines, in the scope of a GASPI collective library and tested the routines on two different systems:

Cluster 1: A system with 14 nodes, each having two sockets with 6-core Westmere X5670 @2.93GHz processors and an InfiniBand QDR network in fat tree configuration. On this system, the algorithm was compared to the allreduce routines of MVAPICH 2.2.0 and OpenMPI 1.6.5, because no Intel MPI implementation is available on this system. In the following plots, only the OpenMPI runtime is shown, because the MVAPICH implementation is much slower. Thus, a user would not use this implementation for allreduce-heavy jobs and for a better readability, we do not plot these runtimes.

Cluster 2: A system with two sockets nodes of with 8-core Sandy Bridge E5-2670/1600 @2.6GHz processors and an InfiniBand FDR10 network in fat tree configuration. On this system, the algorithm was compared to the allreduce routines of Intel MPI 4.1.3.049 and OpenMPI 1.8.1. In the following plots, only the Intel MPI runtime is shown, because it was always the faster implementation.

The cluster that was used for the tests in the previous paper was no longer available for experiments. In the previous paper [1], we had implemented only the n-way dissemination algorithm directly on top of ibverbs, which showed a significant performance improvement when compared to an Intel MPI, as seen in Figure 5. To enable portability to different GASPI implementations, the option of implementing library routines was now chosen. The GASPI implementation used, is the GPI2-1.1.1 by the Fraunhofer ITWM [18].

We will show runtime comparisons for the smallest possible message size (one integer) and the largest possible message size (255 doubles) in GASPI allreduce routines. In the second case, the reduction operation is applied element-wise to an array of 255 doubles. The runtimes shown are average times from 10^4 runs to balance single higher runtimes which may be caused through different deterministically irreproducible aspects like jitter, contention in the network and similar. Timings were taken right before the call and then again immediately after the call returned. Between two calls of an allreduce, a barrier was called to eliminate caching effects.

We have started one GASPI process per NUMA socket, which is the maximum number of GASPI processes that can be started per node. In addition to a comparison with the fastest MPI implementation on each cluster, we have also implemented a binomial spanning tree as a GASPI allreduce library routine, to show the difference between a good performing tree implementation and an implementation with butterfly-like algorithms. For better readability of the plots, we have omitted the graphical representation of the runtimes of the GPI2 allreduce, because it was, as to be expected, faster than the library routines in most cases.

To convey an idea of the overhead induced through the implementation of the allreduce as a GASPI library routine instead of implementing the allreduce directly with ibverbs, this overhead is depicted in Figure 6. The runtime for the allreduce with one integer increases by a factor of up to 1.84 and with 255 doubles, it even increases by a factor of up to 2.18. This will have to be kept in mind, when regarding the following results.

While the BST and the PE transfer a fixed number of messages per communication round, the n-way dissemination

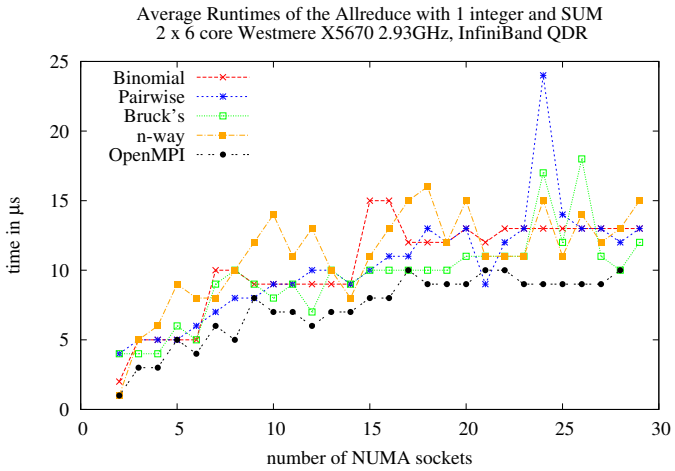

Figure 7. Comparison of Allreduce implementations with 1 integer and sum as reduction operation on Cluster 1. One GASPI process per socket.

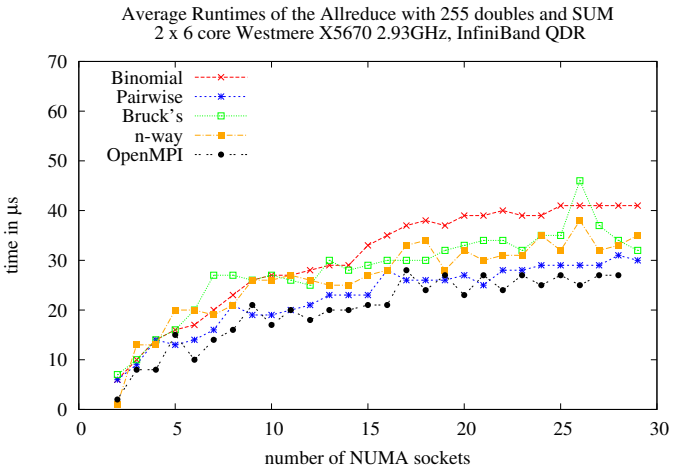

Figure 9. Comparison of Allreduce implementations with 255 doubles and sum as reduction operation on Cluster 1. One GASPI process per socket.

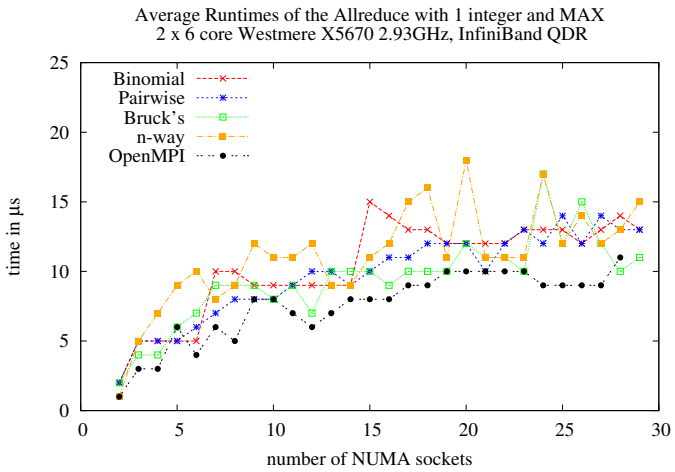

Figure 8. Comparison of Allreduce implementations with 1 integer and maximum as reduction operation on Cluster 1. One GASPI process per socket.

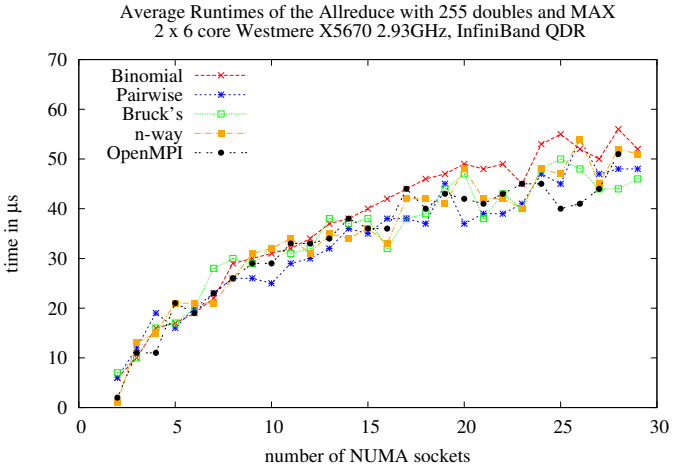

Figure 10. Comparison of Allreduce implementations with 255 doubles and maximum as reduction operation on Cluster 1. One GASPI process per socket.

algorithm and Bruck's algorithm may transfer different numbers of messages per communication round. Since Bruck's algorithm works for all combinations of (n, P), we have fixed $n = 5$ for these experiments. For the n-way dissemination algorithm the n is chosen in the first call of the allreduce routine and the smallest n possible is chosen. This procedure differs from the procedure in the former paper, where a number of allreduces was started in the first call and the fastest n was chosen. Further research has shown, that the overhead induced by calling a sufficiently high number of allreduces to chose a n in this first call is not necessarily compensated through the potentially faster following allreduces. In the future, static but network dependent lookup tables need to be developed or further research on the choice of n depending on the bandwidth, latency and message rate of the underlying network needs to be done.

In Figures 7 to 10 the runtime results on Cluster 1 are shown with one GASPI process started per NUMA socket. Figures 7 and 8 show the runtime results for the allreduce with one integer and sum, respectively maximum reduction operation. Figures 9 and 10 show the same for 255 doubles. In all cases except the maximum operation with 255 doubles, none of the library routines are faster than the OpenMPI implementation. This comes as no surprise, as the allreduce library routines are implemented on top of GASPI routines, while the OpenMPI allreduce may make direct use of ibverbs routines. The runtimes of the GASPI library allreduce are steadier when using the allreduce on 255 doubles than on one integer. When increasing the message size, the butterfly-like algorithms have faster runtimes than the BST. Even though it has been suggested, that the symmetric communication scheme of the PE algorithm will lead to a high congestion in the network, this is not confirmed by the results of the experiments: The pairwise exchange algorithm has a runtime close to the OpenMPI implementation. When using the maximum as reduction operation, all butterfly-like algorithms have similar runtimes and are even faster than the OpenMPI allreduce

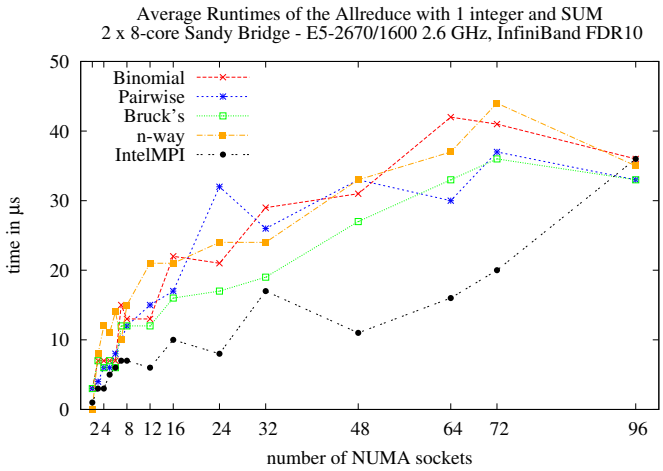

Figure 11. Comparison of Allreduce implementations with one integer and sum as reduction operation on Cluster 2. One GASPI process per socket.

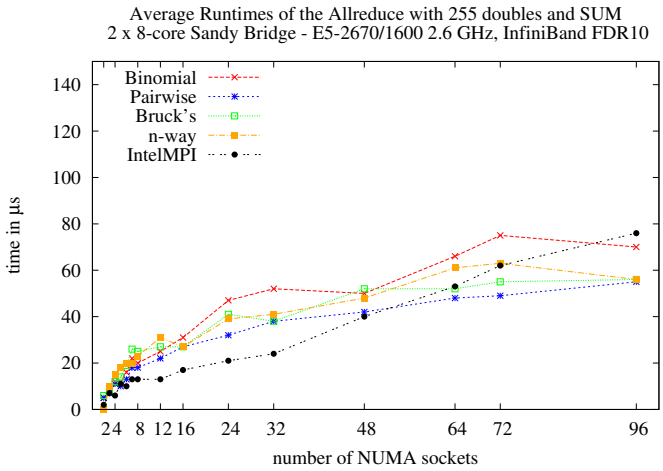

Figure 13. Comparison of Allreduce implementations with 255 doubles and sum as reduction operation on Cluster 2. One GASPI process per socket.

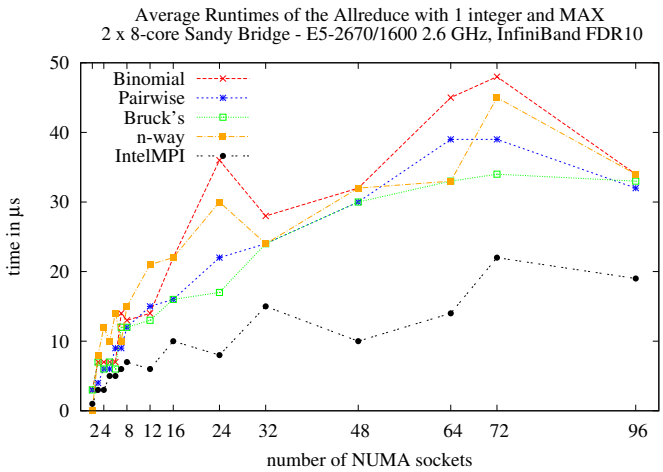

Figure 12. Comparison of Allreduce implementations with one integer and max as reduction operation on Cluster 2. One GASPI process per socket.

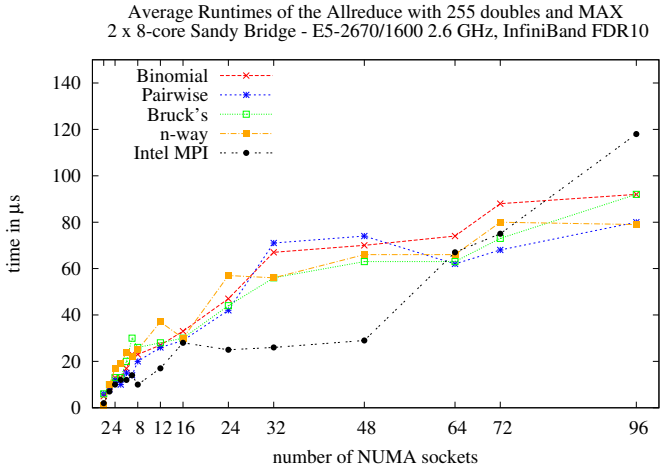

Figure 14. Comparison of Allreduce implementations with 255 doubles and max as reduction operation on Cluster 2. One GASPI process per socket.

implementation for several process numbers. Overall, Bruck's algorithm shows best results for small messages, i.e., one integer, and the PE algorithm shows the best results for large messages, i.e., allreduces on 255 doubles. The adapted n-way dissemination algorithm runtime plot is very volatile, especially for small messages. At least for larger messages, it shows consistently faster runtimes than the BST. For all algorithms, the allreduce with large messages and the maximum operation is significantly slower than the equivalent allreduce with summation as reduction routine.

Figures 11 to 14 show the averaged runtime results on Cluster 2. Here, the difference in runtime between the Intel MPI allreduce and the GASPI allreduce routines is significant for small messages, as can be seen in Figures 11 and 12. Only for larger numbers of involved processes the runtimes of the GASPI routines and those of the Intel MPI implementation converge (Figure 11). While the plots are not as erratic as on Cluster 1, this might be due to the fact, that on this system we could not test every process count. Especially Bruck's algorithm outperforms the other butterfly-like algorithms and the BST for allreduces with small messages. For large messages, i.e., 255 doubles, the GASPI library implementations of the allreduce and the Intel MPI implementation have similar runtimes. Even though the Intel MPI implementation is still faster for a process count up to 48, the gap to the runtimes of the GASPI library has closed to a great extent. With higher number of processes, the GASPI library routines are even faster than the Intel MPI allreduce. Again, the PE shows surprisingly good results, especially for large messages and the sum operation, while the BST's runtimes are at the upper limit of the library runtimes.

V. CONCLUSION AND FUTURE WORK

We have examined different algorithms with a butterfly-like communication scheme for the suitability in a GASPI allreduce library function. In [1] we had presented an adaption to the n-way dissemination algorithm, which was here compared to other algorithms with a similar communication structure. Two

important properties of these algorithms are their low number of communication rounds while at the same time involving all processes in each computation step of the algorithm. This makes them ideal candidates for a split-phase allreduce routine as defined in the GASPI specification.

We have seen in the experiments, that algorithms with a butterfly-like communication scheme are often significantly faster than, e.g., the BST and sometimes even reach the performance of existing MPI implementations. This is especially important to note, because the results presented in this article are results obtained from library implementations, i.e., not directly implemented on ibverbs but rather with GASPI routines. As shown in Figure 6, the overhead induced through this additional layer of indirection can slow a routine down by a factor of 2. Considering this, an implementation of the allreduce routine with ibverbs should accelerate the routine to approximately the level of the MPI implementations shown for small messages and even faster in the case of large messages. This is a relevant starting point for future research.

Another important comparison to make is the influence of the different network interconnects on the algorithm. While in the former paper, the FDR network had an immense influence on the runtime of the n-way dissemination algorithm. In this case, we are comparing a QDR network to a FDR-10 network and do not see the same performance increase. Instead, we partially even see a decrease in speed. While Bruck's algorithm does not need more than 10 μs for small messages Cluster 1, it needs 16 μs on Cluster 2. For large messages we see a speedup from 32 μs to 30 μs for the global maximum and from 30 μs to 27 μs for the global sum. This again highlights the importance of adjusting the used algorithms to the underlying network and will be investigated further in the scope of a library with collective routines for GASPI implementations.

All in all, algorithms with a butterfly-like communication scheme should not be ignored for new communication routines and libraries. The increasing message rates and network topology developments might make the use of these algorithms very feasible again.

REFERENCES

[1] V. End, R. Yahyapour, C. Simmendinger, and T. Alrutz, "Adapting the n-way Dissemination Algorithm for GASPI Split-Phase Allreduce," in INFOCOMP 2015, The Fifth International Conference on Advanced Communications and Computation, June 2015, pp. 13 – 19.

[2] GASPI Consortium, "GASPI: Global Address Space Programming Interface, Specification of a PGAS API for communication Version 16.1," https://raw.githubusercontent.com/GASPI-Forum/GASPI-Forum.github.io/master/standards/GASPI-16.1.pdf, February 2016, retrieved 2016.11.29 at 13:07.

[3] Message-Passing Interface Forum, MPI: A Message Passing Interface Standard, Version 3.0. High-Performance Computing Center Stuttgart, 09 2012.

[4] N.-F. Tzeng and H.-L. Chen, "Fast compaction in hypercubes," IEEE Transactions on Parallel and Distributed Systems, vol. 9, 1998, pp. 50–55.

[5] J. M. Mellor-Crummey and M. L. Scott, "Algorithms for scalable synchronization on shared-memory multiprocessors," ACM Transactions on Computer Systems, vol. 9, no. 1, Feb. 1991, pp. 21–65.

[6] E. D. Brooks, "The Butterfly Barrier," International Journal of Parallel Programming, vol. 15, no. 4, 1986, pp. 295–307.

[7] D. Hensgen, R. Finkel, and U. Manber, "Two algorithms for barrier synchronization," International Journal of Parallel Programming, vol. 17, no. 1, Feb. 1988, pp. 1–17.

[8] J. Bruck and C.-T. Ho, "Efficient global combine operations in multi-port message-passing systems," Parallel Processing Letters, vol. 3, no. 04, 1993, pp. 335–346.

[9] J. Bruck, C.-T. Ho, S. Kipnis, E. Upfal, and D. Weathersby, "Efficient algorithms for all-to-all communications in multi-port message-passing systems," in IEEE Transactions on Parallel and Distributed Systems, 1997, pp. 298–309.

[10] S. P. Kini, J. Liu, J. Wu, P. Wyckoff, and D. K. Panda, "Fast and Scalable Barrier using RDMA and Multicast Mechanisms for InfiniBand-based Clusters," in Recent Advances in Parallel Virtual Machine and Message Passing Interface. Springer, 2003, pp. 369–378.

[11] V. Tipparaju, J. Nieplocha, and D. Panda, "Fast collective operations using shared and remote memory access protocols on clusters," in Proceedings of the 17th International Symposium on Parallel and Distributed Processing, ser. IPDPS '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 84.1–.

[12] InfiniBand Trade Association, "Infiniband architecture specification volume 1, release 1.3," https://cw.infinibandta.org/document/dl/7859, March 2015, retrieved 2016.11.29 at 13:09.

[13] ——, "Infiniband architecture specification volume 1, release 1.2.1, annex a16," https://cw.infinibandta.org/document/dl/7148, 2010, retrieved 2016.11.29 at 13:08.

[14] E. Zahavi, "Fat-tree Routing and Node Ordering Providing Contention Free Traffic for MPI Global Collectives," J. Parallel Distrib. Comput., vol. 72, no. 11, Nov. 2012, pp. 1423–1432.

[15] R. Gupta, V. Tipparaju, J. Nieplocha, and D. Panda, "Efficient Barrier using Remote Memory Operations on VIA-Based Clusters," in IEEE Cluster Computing. IEEE Computer Society, 2002, p. 83ff.

[16] T. Hoefler, T. Mehlan, F. Mietke, and W. Rehm, "Fast Barrier Synchronization for InfiniBand," in Proceedings of the 20th International Conference on Parallel and Distributed Processing, ser. IPDPS'06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 272–272.

[17] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of Collective Communication Operations in MPICH," International Journal of High Performance Computing Applications, vol. 19, 2005, pp. 49–66.

[18] Fraunhofer ITWM, "GPI2 homepage," www.gpi-site.com/gpi2, retrieved 2016.11.29 at 13:11.