Adaption of the n-way Dissemination Algorithm 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*—This paper presents an adaption of the n-way dissemination algorithm, such that it can be used for an allreduce operation, which is - together with the barrier operation - one of the most time consuming collective communication routines available in most parallel communication interfaces and libraries. Thus, a fast underlying algorithm with few communication rounds is needed. The dissemination algorithm is such an algorithm and already used for a variety of barrier implementations due to its speed. Yet, this algorithm is also interesting for the split-phase allreduce operations, as defined in the Global Address Space Programming Interface (GASPI) specification, due to its small number of communication rounds. Even though it is a butterfly-like algorithm, significant improvements in runtime are seen when comparing this implementation on top of ibverbs to different message-passing interface (MPI) implementations, which are the de facto standard for distributed memory computing.**

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

I. Introduction

A main aspect in distributed application programming is the communication of data. An emerging communication interface designed for high performance computing (HPC) applications is the Global Address Space Programming Interface (GASPI) specification [1]. It 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 [2]. 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 [3] or the tree algorithm of Mellor-Crummey and Scott [4]. These are then used to first reduce and then broadcast the data. Also, more barrier related algorithms like the butterfly barrier of Brooks [5] or the tournament algorithm described by Debra Hensgen et al. in the same paper as the dissemination algorithm [6] 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 only needs $\lceil \log_{n+1}(P) \rceil$ communication rounds and involves all ranks in every round. Also, the butterfly barrier has $k = \lceil \log_2(P) \rceil$ communication rounds to do besides only being fit for 2^k participants.

One of the most heavily used algorithms for barrier operations is the dissemination algorithm presented by Hensgen et al. in 1988 [6]. It is used in different programming APIs and libraries like the MPICH implementation [7] of MPI, the global address space networking API (GASNet) [8] and the second generation of the Global address Programming Interface (GPI-2) v1.1.0 [9] for barrier implementations due to its speed. In 2006 Torsten Hoefler et al. [10] have based the n-way dissemination algorithm on this work to exploit implicit parallelism of the InfiniBand network.

However, until today the dissemination algorithm and the developments have not been used for allreduce operations, partly due to the problems arising when using the dissemination algorithm with a number of nodes $P \neq 2^k$. The same problem arises for the n-way dissemination algorithm if the number of involved nodes does not equal $(n+1)^k$. This will be elaborated in more detail below.

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 [1].

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, the reasons why the n-way dissemination algorithm is not suitable for the allreduce collective operation are described. The cases in which the algorithm may be used as is are identified and also those cases, in which the algorithm must be adapted to receive correct results are described. Based on the data movement within the algorithm *data boundaries* can be identified with which the algorithm can then be adapted. Through this adaption, the amount of data to be transferred in the last round is reduced, in some cases it is even possible to omit the last round. Even though the shown adaption of the algorithm was done for allreduce operations performed with non-idempotent functions, the benefits regarding runtime are also measurable when used for, e.g., a barrier.

Some more related work will be presented in the next section. The n-way dissemination algorithm is introduced in Section III and then, in Section III-A, the reasons necessitating the adaption are described. An adaption to the n-way dissemination algorithm is proposed in Section IV to resolve these problems. First results, achieved with an implementation of the adapted algorithm on top of ibverbs, are presented in Section V. The conclusion and an outlook of future work are given in Section VI.

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 almost the same communication scheme as the n-way dissemination algorithm has [11][12]. These works were found relatively late in the implementation phase, such that an extensive comparison to this work has not been made yet. But considering the results achieved with this work, also Bruck's algorithm is an interesting candidate for the GASPI allreduce.

In the past years, more and more emphasis has been put on RDMA techniques and algorithms [13][14] due to hardware development, e.g., InfiniBand$^{\text{TM}}$[15] or RDMA over Converged Ethernet (RoCE). While Panda et al. [13] exploit the multicast feature of InfiniBand$^{\text{TM}}$, 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$^{\text{TM}}$architecture [15]. Congestion in fat tree configured networks is still a topic in research, where for example Zahavi is an active researcher [16]. 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.

III. The n-way Dissemination Algorithm

The n-way dissemination algorithm, as presented in [10] 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 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.

A. Using the n-way Dissemination Algorithm for Allreduce

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

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

TABLE I. ROUND-WISE COMPUTATION OF PARTIAL RESULTS IN A 2-WAY DISSEMINATION ALGORITHM

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 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

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

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.

IV. Adapting the n-way Dissemination Algorithm

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 1 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

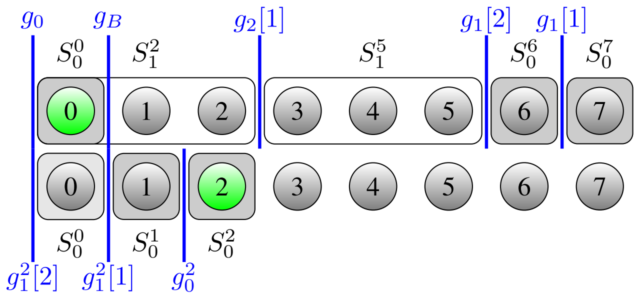


Figure 1. The data boundaries g and received partial results $S_i^{r_{l,j}}$ of ranks 0 and 2.

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:

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

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

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

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 1 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.,

$$g^s_{l_{\mathrm{snd}}}[j_{\mathrm{snd}}] = g_{l_{\mathrm{rcv}}}[j_{\mathrm{rcv}}] \tag{7}$$

or $g^s_{l_{\mathrm{snd}}}[j_{\mathrm{snd}}] = g_B$. 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 $g_{l_{\mathrm{rcv}}}[j_{\mathrm{rcv}}]$. If the operation does not have an inverse, recalculate the final result, hereby omitting the partial result defined through $g_{l_{\mathrm{rcv}}}[j_{\mathrm{rcv}}]$. 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) $l_{\mathrm{rcv}} = 0, l_{\mathrm{snd}} > 1$:

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

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 $l_{\text{snd}} > 1$ 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.

V. Experiments and Evaluation

In the following, it is assumed that an optimal implementation for an allreduce function on modern ccNUMA architectures will always take the form of a hybrid implementation where communication within the node or socket is using shared socket memory and communication across nodes is performed via distributed memory. This not only complies to the GASPI specification but also follows the approach of Panda et al. [13]. The n-way dissemination algorithm was implemented on top of ibverbs and then compared to different MPI implementations, as this is the de facto standard of distributed memory communication. The implementation of the adapted n-way dissemination algorithm was tested on two different systems:

Cluster 1: A system with 96 nodes, each having two sockets with 12 core Ivy Bridge E5-2695 v2 @2.40GHz processors and an InfiniBand ConnectX FDR interconnect in fat tree configuration. On this system, the algorithm was compared to the allreduce and barrier of the Intel MPI 4.1.3 implementation, as this was the fastest MPI implementation available on the system.

Cluster 2: A system with 36 nodes, each having two sockets with 6 core Westmere X5670 @2.93GHz processors and an InfiniBand ConnectX QDR network in fat tree configuration. On this system, the algorithm was compared to the allreduce and barrier routines of MVAPICH 2.2.0 and OpenMPI 1.6.5, because no Intel MPI implementation is available on this system.

In order to obtain the following average runtimes, the algorithms were run 10^3 times on the larger system and 10^5 times on the smaller system to balance single higher runtimes which may be caused through different deterministically irreproducible aspects like jitter, contention in the network or similar. Timings were taken right before the call and again right after the call returned to obtain the average runtimes.

Similar to [10] it was observed, that the choice of n influences the runtime of the routine. The choice of n for the n-way dissemination algorithm will primarily depend on message rate, latency and bandwidth of the underlying network. Ultimately n will have to be determined as a function of these parameters or - much simpler, but less elegant - in the form of static (but network dependent) lookup tables. For the experiments in this section, different $n \in \{1, ..., 7\}$ were tested in the first call of the routine. The routine was internally run 15 times for each n and timed. The n with the lowest average runtime was chosen for the following runs. This overhead is included in the following runtime plots.

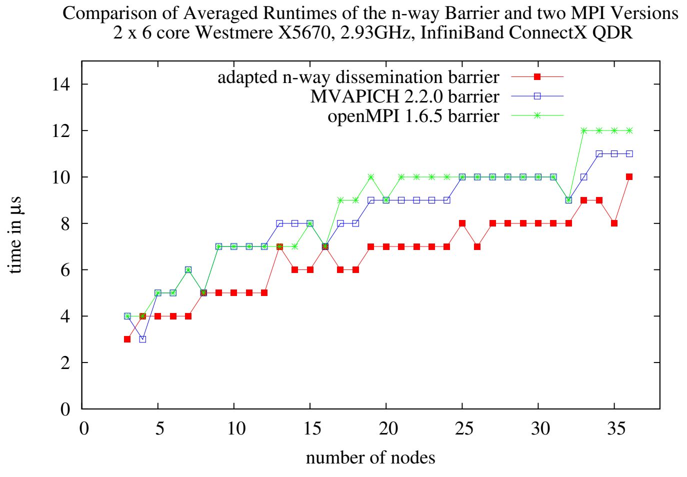


Figure 2. Comparison of the adapted n-way dissemination barrier to available MPI barrier implementations: OpenMPI 1.6.5 and MVAPICH 2.2.0 on Cluster 2.

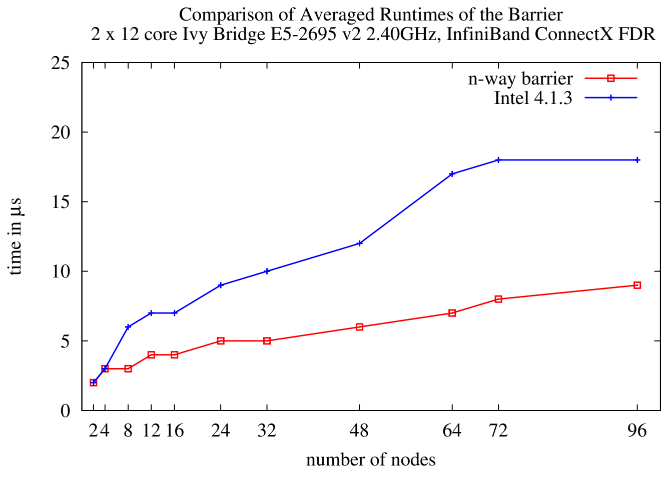


Figure 3. Comparison of the adapted n-way dissemination barrier to the Intel MPI 4.1.3 barrier implementation on Cluster 1.

Even though from the theoretical side it is not necessary to adapt the n-way dissemination algorithm for a correct implementation of the barrier operation or idempotent allreduce functions, the adaption of the last round brings benefits in the runtime for these routines also. In Figure 2, two MPI barrier implementations (MVAPICH 2.2.0 and OpenMPI 1.6.5) are compared to the adapted n-way dissemination barrier on Cluster 2. The same was done on Cluster 1, the results of the averaged runtimes are shown in Figure 3. The runtime of the n-way barrier implementation is not much lower than that of the MPI implementations on the QDR system but on the larger FDR system a great improvement to the Intel MPI implementation can be seen. On the QDR system, the barrier reaches an average runtime of approximately 8 μs when running on 32 nodes. On the larger system this runtime decreases to approximately 5 μs. This significant decrease has to be related to the underlying network and the much higher message rate available on FDR. Here a higher n can be chosen, which correspondingly increases the parallelism in communication and decreases the number of communication rounds needed.

In Figure 4, the first comparisons between MPI allreduces and the adapted n-way dissemination allreduce on the QDR connected system are plotted. There are two different setups: In the first case the allreduce routine was called with one integer as the initial data element to be reduced per process. (upper plot in Figure 4). While there was some improvement compared to the MPI implementations when testing the barrier, in this case no improvement can be seen. This can be explained through the additional time needed for computation in the allreduce.

In the second case the reduce operation works element-wise on an array of 255 elements (lower plot in Figure 4). Here the difference in the runtimes of the OpenMPI allreduce and the MVAPICH allreduce are almost neglectable, while the n-way dissemination allreduce has a much faster averaged runtime. This seems somewhat surprising, as butterfly-like algorithms are known for congesting the network. While it is to be expected that congestion becomes more probable, when increasing the message size, the opposite seems to be the case.

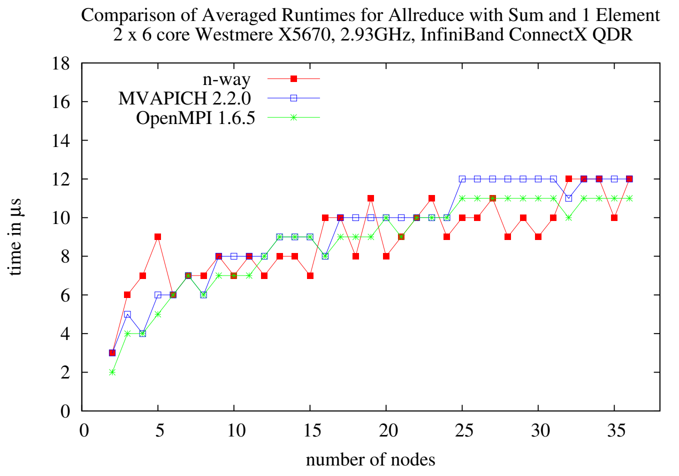
In both test settings the averaged times of the MPI implementations seem more uniform than those of the n-way dissemination algorithm. The main reason for this is that the n-way algorithm provides a higher parallelism per communication step and hence is less robust against jitter.

In Figure 5, a similar picture for the allreduce comparisons on the larger system as for the barrier is seen, i.e., on node level substantially lower runtimes are achieved by the n-way dissemination algorithm than those of Intel MPI 4.1.3. Similar to the tests on the first cluster, the improvements in the barrier runtimes are much higher than those in the allreduce runtimes. Potentially even better results can be achieved for the allreduce operation by internally overlapping the communication and a computation of partial results. We will investigate this as part of future work.

While butterfly-like algorithms often suffer from congestion in the network, this problem is mitigated in the adaption of the n-way dissemination algorithm. The high potential of congestion especially arises through the normally symmetric communication scheme of algorithms like the butterfly itself, the pairwise exchange or the n-way dissemination algorithm with $P = (n+1)^k$. By using the algorithm for $P \neq (n+1)^k$, this symmetry is dissolved. Two further arguments for the use of this algorithm comes straight from the GASPI specification. First of all the message sizes are very limited. According to the GASPI specification the largest messages in an allreduce may be 255 doubles. This amounts to only 2040B, which is less than the limit of 4KB of an InfiniBand packet. In addition, only one allreduce operation may be active per GASPI group at a time. Assuming that applications will not create multiple identical groups, the paths for the packets to be transmitted will not be identical either in multiple allreduces.

VI. Conclusion and Future Work

In this paper we have presented an adaption to the n-way dissemination algorithm, such that it is usable for (user-defined) allreduce operations within GASPI. The main advantage of this modified algorithm is not just an excellent performance, but also the smaller depth of the communication



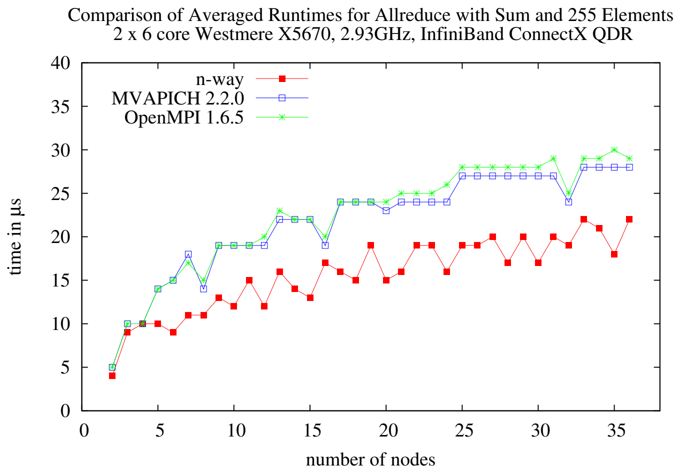


Figure 4. Comparison of the adapted n-way dissemination allreduce to two MPI allreduce implementations with one element per rank to be reduced in the upper plot and 255 elements to be reduce in the lower plot.

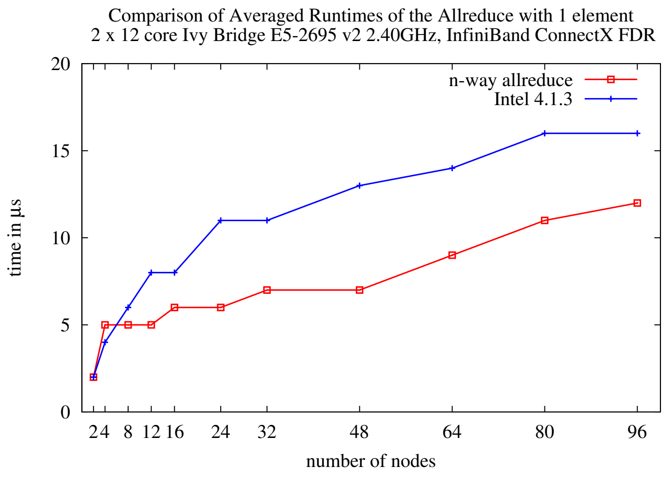


Figure 5. Comparison of the averaged runtimes of the Intel MPI 4.1.3 `MPI_Allreduce` and the n-way dissemination algorithm based on ibverbs.

tree and the fact that this algorithm allows for a participation of all ranks in every step of the communication tree. Both features are key requirements for split-phase collectives and user-defined allreduces of the GASPI PGAS API. Since the n-way dissemination algorithm only requires $\lceil \log_{n+1}(P) \rceil$ communication rounds, a split-phase collective (with progress only within the call) will achieve faster progress than algorithms with deep communication trees mentioned in Section I.

While butterfly-like communication schemes have been abandoned in the past, new hardware with much higher throughput might again make these algorithms with small numbers of communication rounds more attractive in the future. Also, topology developments in high performance interconnects like the Cray Aries interconnect make these communication schemes interesting again [17]. Future work will correspondingly investigate the influence of these new developments on butterfly-like algorithms.

As already mentioned in Section II, future work will also deal with an in-depth analysis and comparison of the adapted n-way dissemination algorithm to other algorithms with similar communication schemes.

References

[1] GASPI Consortium, "GASPI: Global Address Space Programming Interface, Specification of a PGAS API for communication, Version 1.0.1," November 2013. [Online]. Available: http://www.gaspi.de/fileadmin/GASPI/pdf/GASPI-1.0.1.pdf,2015.05.07

[2] Message-Passing Interface Forum, MPI: A Message Passing Interface Standard, Version 3.0. High-Performance Computing Center Stuttgart, 09 2012. [Online]. Available: http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf,2015.05.07

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

[4] 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.

[5] E. D. Brooks, "The butterfly barrier," International Journal of Parallel Programming, vol. 15, no. 4, 1986, pp. 295–307.

[6] 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.

[7] Argonne National Laboratory, "Barrier implementation of MPICH 3.1.2." [Online]. Available: http://www.mpich.org/static/downloads/3.1.2/,2015.05.07

[8] "README File of GASNet." [Online]. Available: http://gasnet.lbl.gov/dist/README,2015.05.07

[9] "Barrier implementation of GPI2-1.1.0." [Online]. Available: https://github.com/cc-hpc-itwm/GPI-2/releases/tag/v1.1.0,2015.05.07

[10] 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.

[11] 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.

[12] J. Bruck, C. tien 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.

[13] 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.

[14] 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–.

[15] InfiniBand Trade Association, InfiniBand architecture specification: release 1.3, R. Dance, Ed. InfiniBand Trade Association, 03 2015, vol. 1. [Online]. Available: http://www.infinibandta.org/content/pages.php?pg=technology_public_specification,2015.05.07

[16] 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.

[17] B. Alverson, E. Froese, L. Kaplan, and D. Roweth, "Cray XC series network," [Online]. Available from: http://www.cray.com/Assets/PDF/products/xc/CrayXC30Networking.pdf, 2015.05.07.