# Parameterized Radix-r Bruck Algorithm for All-to-all Communication

Ke Fan
*University of Alabama at Birmingham*
Birmingham, USA
kefan@uab.edu

Sidharth Kumar
*University of Alabama at Birmingham*
Birmingham, USA
sid14@uab.edu

## I. Introduction

MPI collectives perform an important set of global communication tasks such as broadcast, gather, and reduction, and are therefore essential for HPC applications [7]. `MPI_Alltoall` is one of the most widely used collective routines that facilitates uniform data exchange between each pair of processes. The standard implementation of `MPI_Alltoall` in MPI libraries (e.g., MPICH [2], OpenMPI [3]) uses a combination of techniques, such as the spread-out algorithm [2] and the Bruck algorithm with radix-two [1]. In terms of process count $P$, the spreadout algorithm has a linear complexity, but the Bruck algorithm with radix-two has a logarithmic complexity. A selection between them is made at runtime based on the message size and the number of processes. The Hockney performance model [4] evaluates the minimum communication cost of collective operations in terms of latency (the required number of communication steps) and bandwidth (the actual data transfer time) [5]. Compared with the spread-out algorithm, the Bruck algorithm with radix-two transfers more data with fewer communication steps. As a result, it works well for short messages (latency-dominated).

In fact, the radix of the Bruck algorithm can be tuned from 2 to $P-1$. This ability makes it possible to tune the total number of communication steps along with the total amount of data transmitted (Fig. 1), which allows performance tuning. Currently, the Bruck algorithm with radix-two and the spread-out algorithm are at opposite ends of the communication spectrum, with a great deal of *unexplored* parameter space in between. Therefore, in this work, we formalized a *generalized* formula and implementation of the Bruck algorithm and performed an experimental investigation of the tunable Bruck algorithm with varying radix-r. We demonstrated that the Bruck algorithm with radix $r = \lceil\sqrt{P}\rceil$ ($P$: total number of processes) is the most effective in most cases. Furthermore, we derive a formula to calculate the number of data-blocks transferred at each communication step. The formula can be used to deterministically precalculate the total workload per communication step for all possible radix-r. In addition, we optimized the Bruck algorithm by using a modified Bruck algorithm [6] that eliminates the final rotation phase that is required by Bruck. We performed scaling studies for a range of message sizes, and radices, and demonstrated that Bruck with optimal radix outperforms vendor-optimized Cray's `MPI_Alltoall` by as much as $57\%$ for some workloads and scales.

## II. Generalized Radix-r Bruck

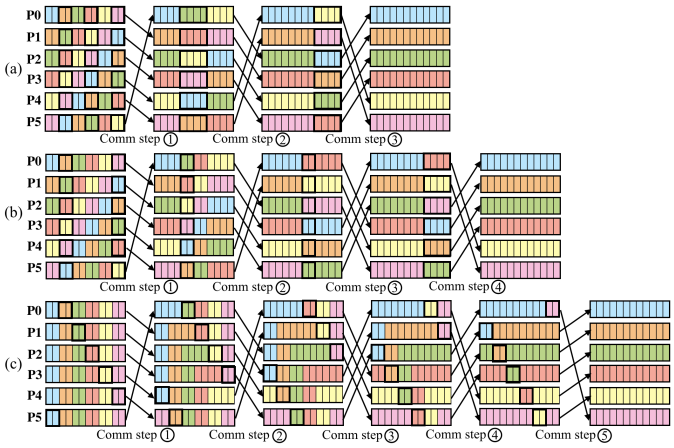



Fig. 1: Examples of the Bruck algorithm with 6 processes and varying radix $r$: (a) $r = 2$ (3 comm steps), (b) $r = 4$ (5 comm steps), and (c) $r = 5$ (5 ($P - 1$) comm steps).

In all-to-all data exchange, each process has a *send* data buffer, which is logically comprised of $P$ data-blocks. Each data-block is made up of some number of data-elements that are always transferred together. For `MPI_Alltoall`, the number of elements is the same for all data-blocks. Similarly, processes also have a *receive* buffer (initially empty). Both the send buffer and the receive buffer are contiguous 1-D arrays where all data-blocks are laid out in increasing block order. During all-to-all communication, each process $i$ needs to send a data-block $j$ ($0 \leqq j \leq P$) to and receive a data-block $i$ from the corresponding process $j$.

In its original form, the Bruck algorithm has three phases: an initial rotation phase that moves its data up by $p$ (the process rank) data-blocks; a communication phase with multiple steps that perform the actual data exchange (seen Fig. 1); and a final rotation phase that places the received data blocks in ascending order. We present the decomposition of the communication phase into a sequence of point-to-point communication rounds, assuming a 1-port fully connected message-passing model in which each process can communicate with only one other process simultaneously. The id of the $i_{th}$ data-block per process is $j$ after the first rotation phase, which can be encoded

using radix-$r$ $(2 \leqq r < P)$ representation with $w = \lceil log_r^P \rceil$ digits. We then go through the digits from 0 to $w-1$ in sequence. For each digit $k$, the Bruck algorithm also includes at most $r-1$ substeps, corresponding to the $r-1$ different non-zero values of $k$. Therefore, the number of communication steps $numC$ is as shown in (1).

$$numC = w \times (r-1) \tag{1}$$

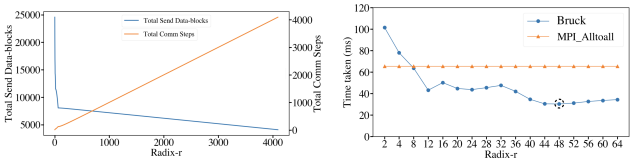
However, if $r^w > P$, the $k = w-1$ round has fewer substeps than the other rounds. Therefore, the actual number of communication steps $numC$ is calculated by (2).

$$numC = w \times (r-1) - \lfloor (r^w - P)/r^{w-1} \rfloor \tag{2}$$

In addition, for each substep $z$ $(0 < z < r)$ in $x$ $(0 \leqq x < w)$ round, each process sends at least $lc = P/r^{x+1} \times r^x$ data-blocks to its destination, and the remaining number of data-blocks is $re = P \ \% \ r^{x+1}$. The number of actual exchanged data-blocks per substep $numD$ is then as below:

$$t = re - z \times r^x, \ numD = \begin{cases} lc, \ if \ (t \leqq 0) \\ lc + r^x, \ else \ if \ (t/r^x > 0) \\ lc + t \ \% \ r^x, \ else \end{cases} \tag{3}$$

We can calculate $numC$ and $numD$ per process for any given $P$ using these equations, as demonstrated in Fig. 2a. This figure shows that $numC$ increases near linearly, but $numD$ decreases dramatically when $r$ is no more than $\lceil \sqrt{P} \rceil$. When $r = \lceil \sqrt{P} \rceil$, $numD$ roughly doubles the linear value, but $numC$ is much less than the linear one. Fig. 2b demonstrated an example of comparing Bruck with `MPI_Alltoall` on theta supercomputer. We double $r$ from 2 to 64, and we observe that $r = 48(\lceil \sqrt{2048} \rceil = 46)$ delivers the best performance ($N$: size in bytes per data-block).




(a) (b)

Fig. 2: (a) $numC$ (orange) increases while $numD$ (blue) decreases when radix-$r$ $(2 \leqq r < P)$ is increasing $(P = 4096)$. (b) Comparing Bruck with varying radix $r$ with MPI_Alltoall on the Theta supercomputer ($P = 2048$, $N = 512$).

## III. Evaluation

All our experiments are performed on the Theta supercomputer at the Argonne Leadership Computing Facility (ALCF). We repeated each experiment 100 times and plotted the mean along with the standard deviations (as error bars).

We implemented both the basic Bruck and modified Bruck algorithms with tuneable parameter radix r. With these implementations, we then report three sets of experiments to compare them with vendor's optimized `MPI_Alltoall`: 1) varying radix r with fixed process count $P$ and size per data-block $N$ (seen Fig. 3); 2) varying $N$ with fixed $P$ and $r$ (seen Fig. 4a); and 3) varying $P$ with fixed $N$ and $r$ (seen Fig. 4b).

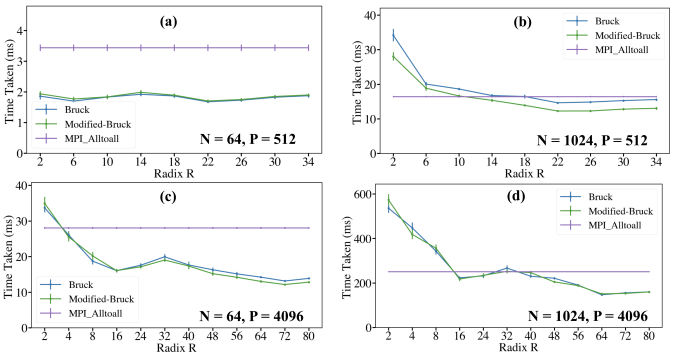



Fig. 3: Comparing Bruck with MPI_Alltoall: varying radix r.

Fig. 3 depicts the outcomes of the 512-process (a and b) and 4096-process (c and d). The radix r in the 512-process experiment ranged from 2 to 56, and it ranged from 2 to 80 in the 4096-process experiment, while the data-block size $N$ was fixed at 64 (a and c) and 1024 bytes (b and d) respectively. From the results, we observe three key trends: (a) the Bruck method with radix-2 does not perform well in most cases; (b) modified Bruck outperforms Bruck when $N$ is relatively large which has a higher final rotation cost, but the improvement is extremely limited; and (c) the Bruck algorithm with $r$ near $\lceil \sqrt{P} \rceil$ works well in most cases. For example, for $P = 512$, the modified Bruck algorithm with radix-22 is $50.46\%$ faster than `MPI_Alltoall` at 64 bytes and $25.42\%$ faster at 1024 bytes. For $P = 4096$, the modified Bruck algorithm with radix-64 is $53.58\%$ faster than `MPI_Alltoall` at 64 bytes and $39.85\%$ faster at 1024 bytes.

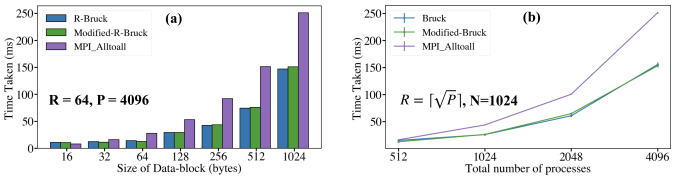



Fig. 4: Comparing Bruck with MPI_Alltoall: (a) varying size per data-block ($N$), (b) varying process counts.

Furthermore, for the 4096-process in Fig. 4a, we fixed $r$ at 64 while varied $N$ from 16 to 1024 bytes. Except for extremely short messages, this result illustrates that Bruck with optimal $r$ works well for all $N$s. In Fig. 4b, we fixed $N = 1024$ bytes and $r$ to be optimal for all $P(512 \leqq P \geqq 4096)$. We can clearly see that Bruck with optimal $r$ performs much better than `MPI_Alltoall` for all $P$s.

## IV. Conclusion

In this paper, we formalized and implemented the open-sourced parameterized Bruck algorithm where the radix can be tuned from 2 to P-1. Then, we conducted an experiment with various Bruck radixes for a wide variety of workloads and process counts. According to the evaluation results, we demonstrated that the Bruck with radix around $r = \lceil \sqrt{P} \rceil$ ($P$: total number of processes) is the most effective in most cases. In comparison to the vendor-optimized `MPI_Alltoall` on the Theta supercomputer, the Bruck with the optimal radix is up to $57\%$ faster for certain workloads and scales.

## REFERENCES


[1] Bruck, Jehoshua, et al. "Efficient algorithms for all-to-all communications in multiport message-passing systems." IEEE Transactions on parallel and distributed systems 8.11 (1997): 1143-1156.

[2] MPICH Home Page. https://www.mpich.org.

[3] OpenMPI Home Page. https://www.open-mpi.org.

[4] Hockney, Roger W. "The communication challenge for MPP: Intel Paragon and Meiko CS-2." Parallel computing 20.3 (1994): 389-398.

[5] Loch, Wilton Jaciel, and Guilherme Piêgas Koslovski. "Sparbit: a new logarithmic-cost and data locality-aware MPI Allgather algorithm." 2021 IEEE 33rd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD). IEEE, 2021.

[6] Träff, Jesper Larsson, Antoine Rougier, and Sascha Hunold. "Implementing a classic: Zero-copy all-to-all communication with MPI datatypes." Proceedings of the 28th ACM international conference on Supercomputing. 2014.

[7] Jocksch, Andreas, Matthias Kraushaar, and David Daverio. "Optimized all-to-all communication on multicore architectures applied to FFTs with pencil decomposition." Concurrency and Computation: Practice and Experience 31.16 (2019): e4964.