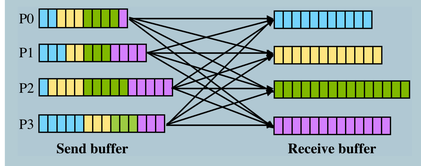
# Padding to Extend the Bruck Algorithm for Non-uniform All-to-all Communication

Ke Fan (kefan@uab.edu), Thomas Gilray (gilray@uab.edu), Sidharth Kumar (sid14@uab.edu)

## Introduction & Motivation ①

**Background:** Non-uniform all-to-all communication enables each process to send varying amounts of data to each other process. Because of its global and non-uniform nature, it is typically difficult to scale and optimize.




Each process contains four data-blocks, each of which contains several data-elements.

**Figure:** Non-uniform all-to-all communication example.

Spread-out implements non-uniform all-to-all communication using non-blocking point-to-point operations, which requires (P-1) communication steps.

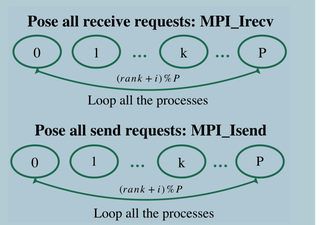



**Figure:** Spread-out algorithm example.

**Motivation:** Linear-complexity performs poorly when applications are deployed on millions of cores, especially for short-message communication, which is dominated by latency. Meanwhile, there is no logarithmic non-uniform all-to-all algorithm that has been adopted by MPI libraries. We are investigating the possibility of expanding the logarithm Bruck algorithm for non-uniform all-to-all communication.

## Bruck's Algorithm ②

Bruck's algorithm is a logarithmic (*logP*) uniform all-to-all algorithm. It is achieved by transmitting an overall larger amount of data but over a smaller number of iterations.

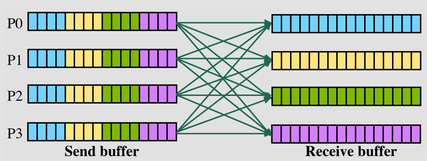



**Figure:** Uniform all-to-all communication example.

Bruck's algorithm, requires three phases: an initial rotation phase, *log(P)* communication steps, and a final rotation phase.

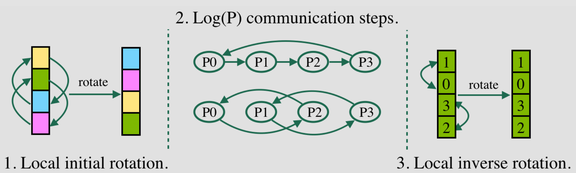



**Figure:** Three phases of the Bruck algorithm.

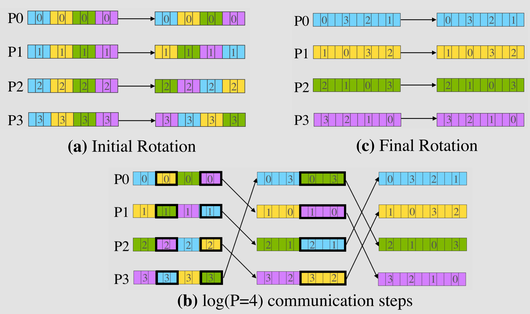



**Figure:** Bruck algorithm example with four processes. Each process has four data-blocks (colored differently), each of which has three data-elements. Each process sequentially sends one data block to each other process.

## Bruck's Algorithm Fails to support Non-uniform All-to-all Communication ③

Bruck's algorithm fails to support messages of varying sizes for two main reasons.

1) Each process is unaware of how much data to expect during each of the intermediate log(P) steps.


P0 sends 3 elements to P1. But P1 doesn't know how many elements it should receive from P0.

2) The received data-elements may be too large to fit into the send buffer segment.


P1 sends one element out. However, it receives 2 elements from P0.

## Padded Bruck Algorithm ④

The Padded Bruck algorithm is a natural extension technique for changing non-uniform all-to-all problems into uniform all-to-all problems using Bruck's algorithm.

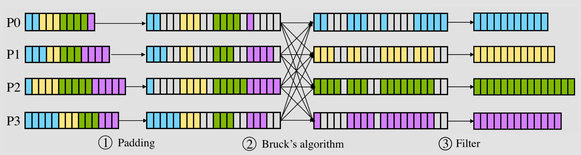



**Figure:** Padded Bruck algorithm example with four processes. Each process has four data-blocks, each of which has various number of data-elements.

This algorithm has three phases:

1. Each data-block was padded with the global maximum size of all (P x P) data-blocks.
2. The Bruck algorithm is used for uniform all-to-all communication.
3. All processes perform a local scan to retrieve the actual data from the padded buffer.

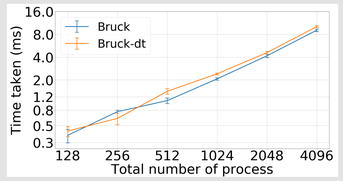
## Evaluation ⑤

All our experiments are performed on the Theta supercomputer at the Argonne Leadership Computing Facility (ALCF).

Argonne NATIONAL LABORATORY

Architecture: Intel-Cray XC40
Cores: 281,088
Speed: 11.7 petaflops
Memory: 843 TB
High-bandwidth Memory: 70TB

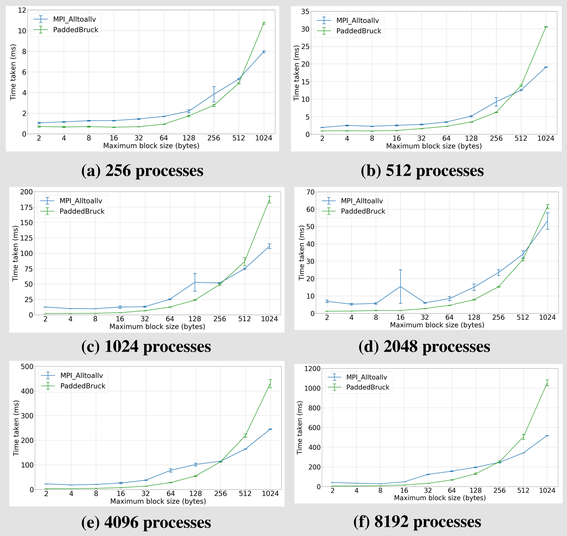

We ran each experiment 25 times and plotted the medians along with the median-absolute deviations (as error bars).




Bruck-dt is implemented with MPI-derived datatypes, whereas Bruck is implemented with memcpy. We find that using an MPI-derived datatype performs poorly for short messages.

**Figure:** Running time of Bruck Implementations (N = 32).

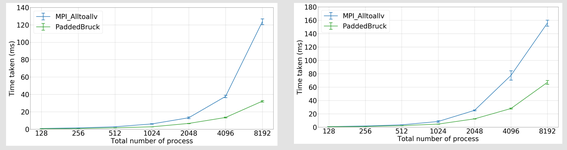
Each process generates data-blocks with a continuous uniform distribution of sizes. This distribution ensures that data-block sizes are sampled uniformly between 0 and the maximum data-block size (N), resulting in an average data-block size of N/2.




(a) 256 processes (b) 512 processes (c) 1024 processes (d) 2048 processes (e) 4096 processes (f) 8192 processes

**Figure:** Data scaling. The maximum data-block size (N) in each experiment ranged from 2 to 1,024 bytes.

In most cases, our Padded Bruck technique outperforms MPI_Alltoallv. At 256 bytes and 512 processes, for example, the Padded Bruck algorithm is 32.52% faster than MPI_Alltoallv.




(a) N = 32 bytes (b) N = 64 bytes

**Figure:** Weak scaling. We fixed the maximum data-block size and varied the number of processes from 128 to 8,192.

With the doubling of processes, the workload increased 4x. Therefore, we saw an increase in execution time with increasing process counts.

Padded Bruck outperforms MPI_Alltoallv by up to 8,192 processes for these data-block sizes. The performance of 8,192 processes, for example, improved by 73.95%.

## Theoretical Performance Model ⑥

Assume the exchange overhead between two processes can be modeled as $\alpha + n\beta$, where $\alpha$ is latency, $\beta$ is bandwidth, and $n$ is the number of bytes transferred.

In padded Bruck, each process sends (*log P x ((P+1)/2) x N)* bytes to others at each round. Comm time of a process is therefore:

$$\alpha \log P + \beta \log P \times ((P+1)/2) \times N$$

The left side of the formula represents the total latency overhead, which is related to the communication steps. Therefore, our padded Bruck algorithm did not increase the latency cost.

## Future Work ⑦

The Two-phase Bruck algorithm is a promising approach for extending the Bruck algorithm to non-uniform all-to-all communication problems.

This approach proposed two solutions to the two challenges:

1. Two-phase transmission at each communication step: the metadata-exchange phase followed by the data-exchange phase at each communication step. The metadata-exchange phase prepares for the data-exchange phase.
2. A temporary buffer that is used alternately with the receive buffer to hold the large received intermediate data-elements.

## Acknowledgements ⑧

We thank the ALCF's Directors Discretionary awards for offering us the compute hours on the Theta Supercomputer.