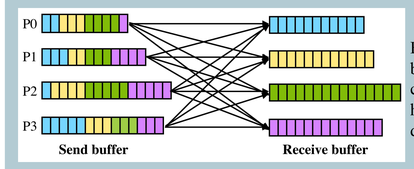
Padding to Extend the Bruck Algorithm for Non-uniform All-to-all Communication

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 allows each process to send different amounts of data to all other processes. This routine is one of the most important and extensively utilized operations in MPI. However, due to its global and non-uniform nature, it is typically difficult to scale and optimize.



Each process has 4 data-blocks with different colors. Each data-block has a different number of data-elements.

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

The Spread-out algorithm is a general implementation of non-uniform all-to-all communication based on non-blocking point-to-point communication. It takes *(P-1)* (linear in number of processes) communication steps.

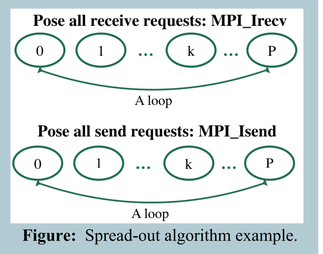


Figure: Spread-out algorithm example.

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 classic uniform all-to-all algorithm that reduces the total number of communication steps from *P* to *log(P)*. It is achieved by transmitting an overall larger amount of data but over a smaller number of iterations.

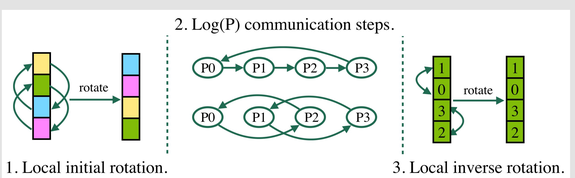


Figure: Three phases of the Bruck algorithm.

Bruck's algorithm, in its original form, requires three phases: a local initial rotation phase, *log(P)* communication steps, and a local final rotation phase.

Assume each process has a send buffer *S* (filled with data), which is logically comprised of *P* data-blocks. Each data block is made up of *n* data elements that are always transferred together. Similarly, processes also have a receive buffer *R* (initially empty).

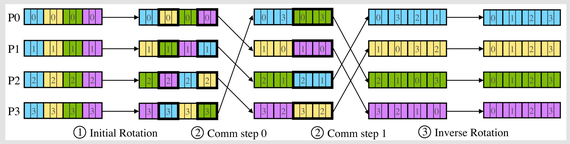


Figure: Bruck algorithm example.

1. A local initial rotation phase: *R[i] = S[(p + i) % P]*
2. The *log(P)* communication steps: in each step k, process p sends to process $(p + 2^k)\%P$ all the data-blocks *R[i]* whose k_{th} bit of i is 1, and receives data from process $(p - 2^k + P)\%P$ into *S[i]*, and replace *R[i]* with *S[i]* locally.
3. A local final rotation phase: *R[i] = R[(P - i + P) % P]*

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 could be too large to fit into the segment in the send buffer.

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

Padded Bruck Algorithm ④

The Padded Bruck algorithm is a natural extension strategy for applying Bruck's algorithm to non-uniform all-to-all problems by transforming them into uniform all-to-all problems.

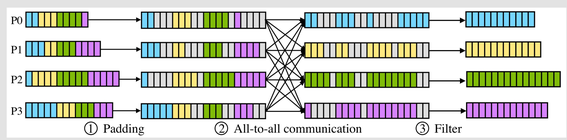


Figure: Padded Bruck algorithm example.

This algorithm has three phases:

1. Padded each data-blocks with the global maximum size of all *(P x P)* data-blocks.
2. Apply the Bruck algorithm to conduct uniform all-to-all communication among processes.
3. All processes perform a local scan to retrieve the actual data from the padded buffer.

This algorithm does not increase the latency cost.

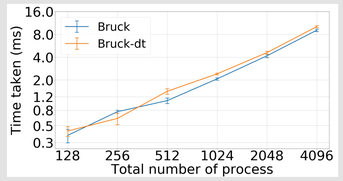
Evaluation ⑤

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

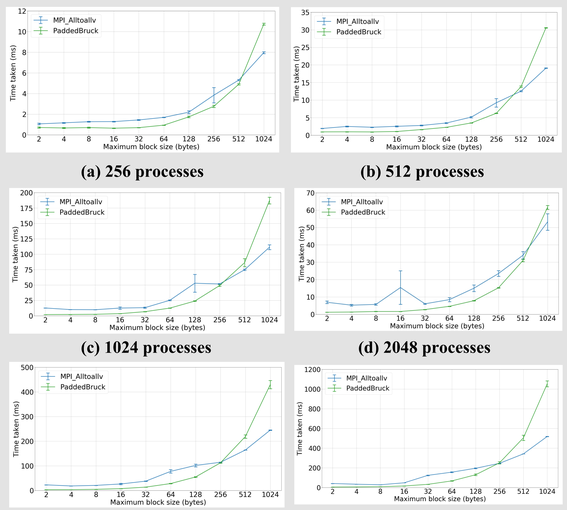
We ran each experiment 25 times and plotted the medians along with the median-absolute deviations (as error bars).



The Bruck-dt is implemented by using MPI-derived datatypes, while Bruck is done with memcpy. We observe that implementation with an MPI-derived datatype performs poorly for short messages.

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

In our experiments, every process generates data-blocks whose sizes follow a continuous uniform distribution. This distribution ensures that data-block sizes are uniformly sampled between 0 and the maximum data-block size *(N)*, thus yielding an average data-block of size *N/2*.



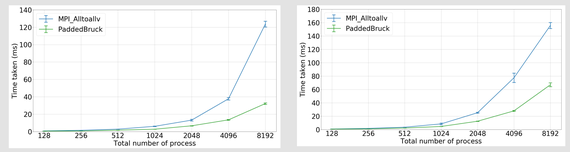
(a) 256 processes **(b) 512 processes**

(c) 1024 processes **(d) 2048 processes**

(e) 4096 processes **(f) 8192 processes**

Figure: Data scaling

In each experiment, the maximum data-block size (N) in this experiment ranged from 2 bytes to 1, 024 bytes. Our Padded Bruck technique outperforms MPI_Alltoallv in most cases. For example, the Padded Bruck algorithm is 32.52% faster than MPI_Alltoallv at 256 bytes with 512 processes.



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

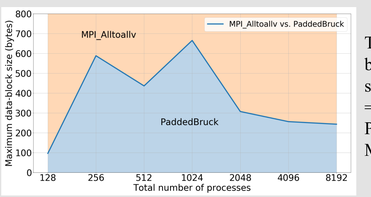
Figure: Weak scaling

In a weak-scaling experiment on all-to-all communication, the workload increased by 4 times with the doubling of processes. Therefore, we saw an increase in execution time with increasing process counts.

For each experiment, we fixed the maximum data-block size, and varied the number of processes from 128 to 8,192.

We observe that for these data-block sizes, Padded Bruck outperforms MPI_Alltoallv by up to 8,192 processes. For example, we observed a 73.95% improvement in the performance of 8,192 processes.

Performance Model ⑥



The performance model can be used to answer questions such as, with P = 300 and N = 400, should we use the Padded Bruck, or Cray's MPI_Alltoallv?

Figure: Performance Model

We attempt to answer this problem by identifying a bounded space of N (maximum size of data-block), P (number of processes) pairs for which the scheme is expected.

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.

This algorithm offers us a considerable improvement in rooms.

Acknowledgements ⑧

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