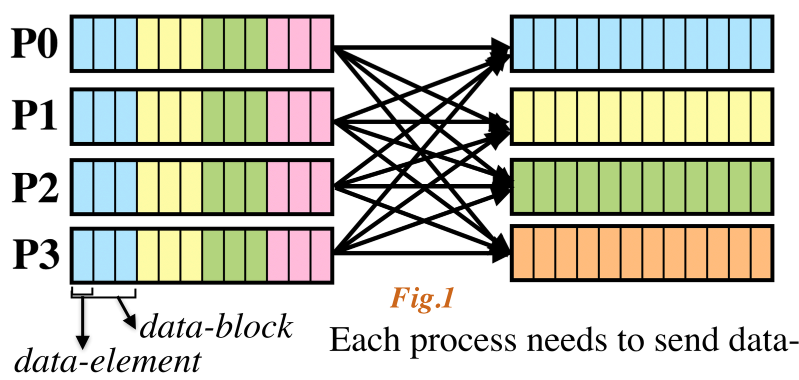
# Parameterized Radix-r Bruck Algorithm for All-to-all Communication

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

## Introduction

**Background:** Uniform all-to-all communication (MPI_Alltoall) is one of the most important and extensively utilized communication patterns in modern HPC applications.




*Each process has P (process count) data-blocks, and each data-block consists of n data-elements.*

Fig.1

Each process needs to send data-block i to process i

The standard MPI_Alltoall implementations include the spread-out algorithm and Bruck algorithm. A selection between them is made at runtime based on N and P.

*P: process count, N: size per data-block*

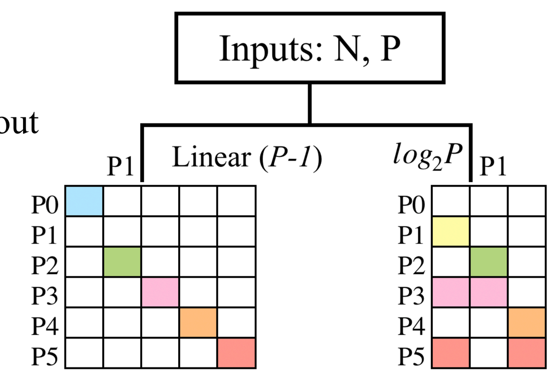



Fig.2

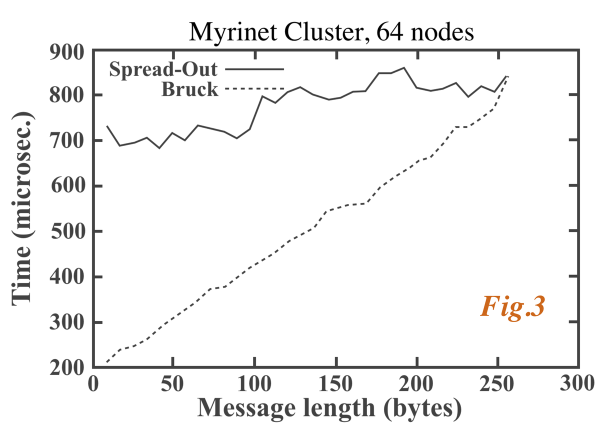



Fig.3

**Hockney performance mode**:

$$\alpha + n\beta$$

Latency cost per communication exchange; Message size (bytes); Transfer time per bytes

The Bruck algorithm works well for short messages (latency-dominated), whereas the spread-out algorithm performs well for larger messages (bandwidth-dominated).

A trade-off between the comm start-up cost (latency) and the data-transfer cost (bandwidth).

*The Bruck algorithm increases the total number of comm steps while decreasing the total sent message size when we increase radix-r.*

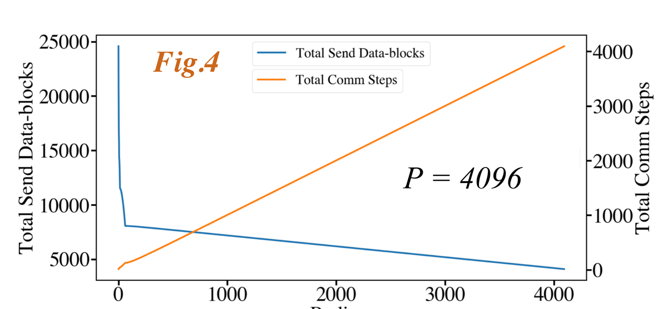



Fig.4

## Motivation

**Motivation:** The current standard MPI library implementations only use two special cases:

- the spread-out algorithm is optimal with respect to the measured data-transfer cost.
- The Bruck algorithm with radix-two is optimal with respect to the measure of the start-up time.

However, these two cases are not the best solutions for some scenarios. Therefore, we conducted experimental investigations of the tuneable Bruck algorithm with varying radix-r. We also figure out how to calculate the number of data-blocks transferred.

## Bruck Algorithm with Radix-r

Bruck's algorithm requires three phases:

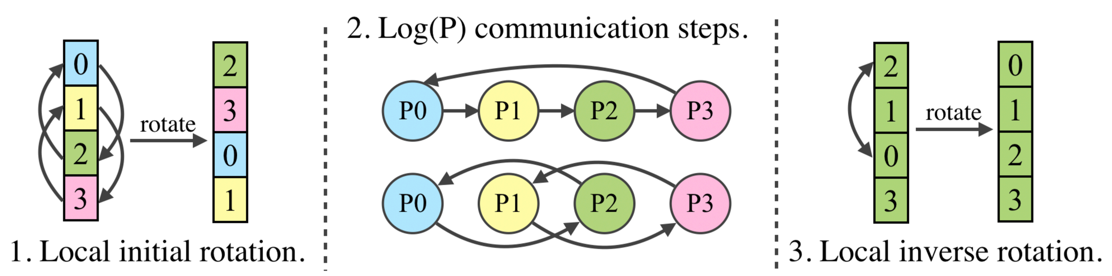



Fig.5

An example of the first communication step ($P = 8$):

*The data-block indexes can be encoded using radix-r representation with $w = \lceil \log_r^P \rceil$ digits, resulting in w rounds. For each round k, there are at most r - 1 steps.*

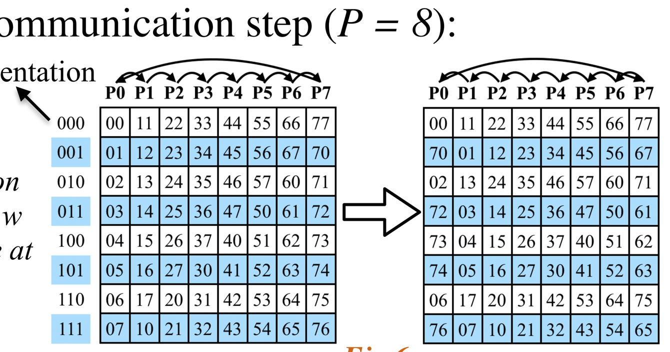



Fig.6

Therefore, the number of communication steps:

$$numC = w \times (r-1)$$

However, if $r^w > P$, the last round has fewer steps than the other rounds. Therefore:

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

For each step $z$ $(0 < z < r)$ in $x$ $(-1 < x < w)$ 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 step is:

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

We can easily calculate *numC* and *numD* for any given *P* using these equations, such as Fig.4.

From Fig.4, we observe that when $r = \lceil \sqrt{P} \rceil$, *numD* roughly doubles the linear value, but *numC* is much less than the linear one. Theoretically, this *r* delivers the optimal overall performance.

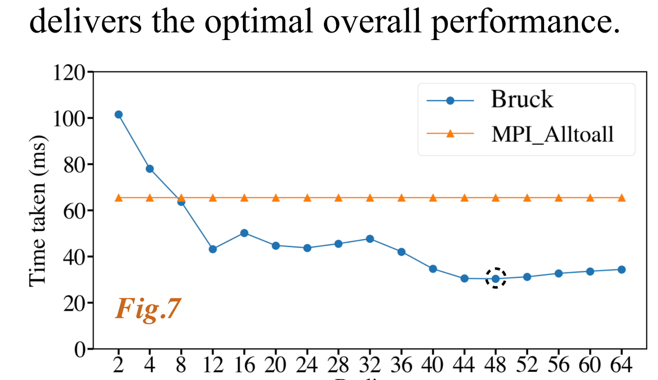



Fig.7

$r = 48$ $(\lceil \sqrt{2048} \rceil = 46)$ delivers the best performance.

Examples of the Bruck algorithm with radix r = 2, 4, and 5 for 6-process:

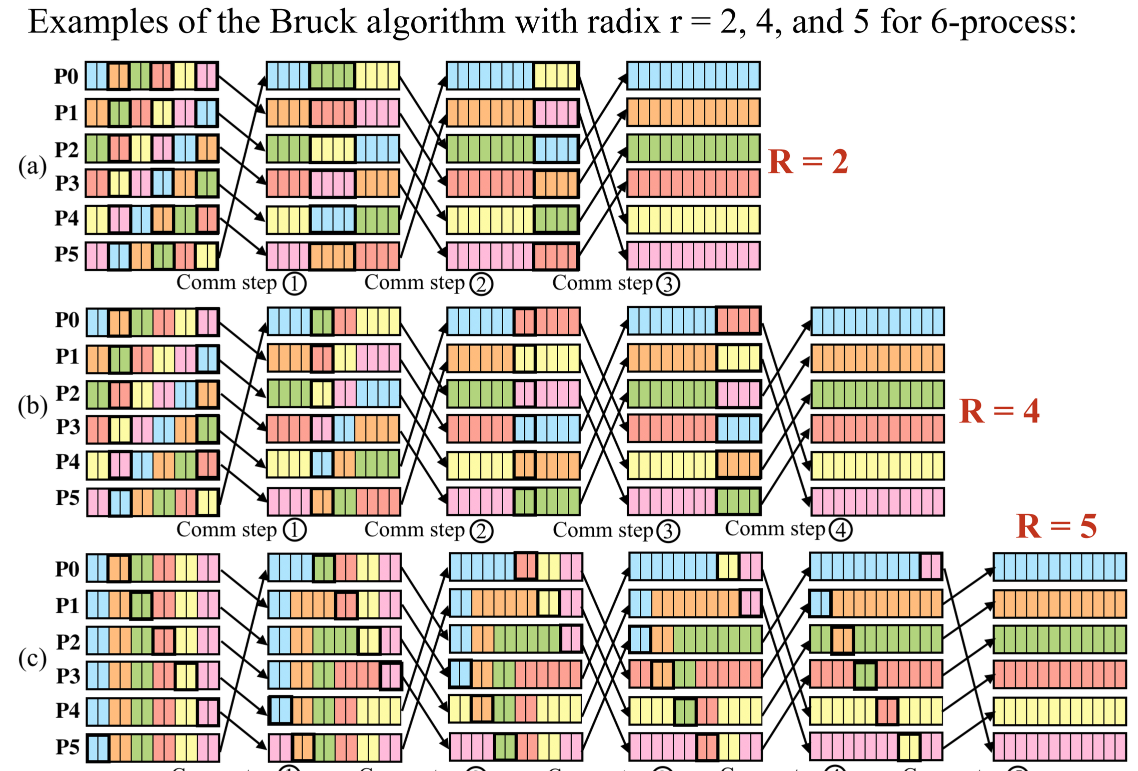
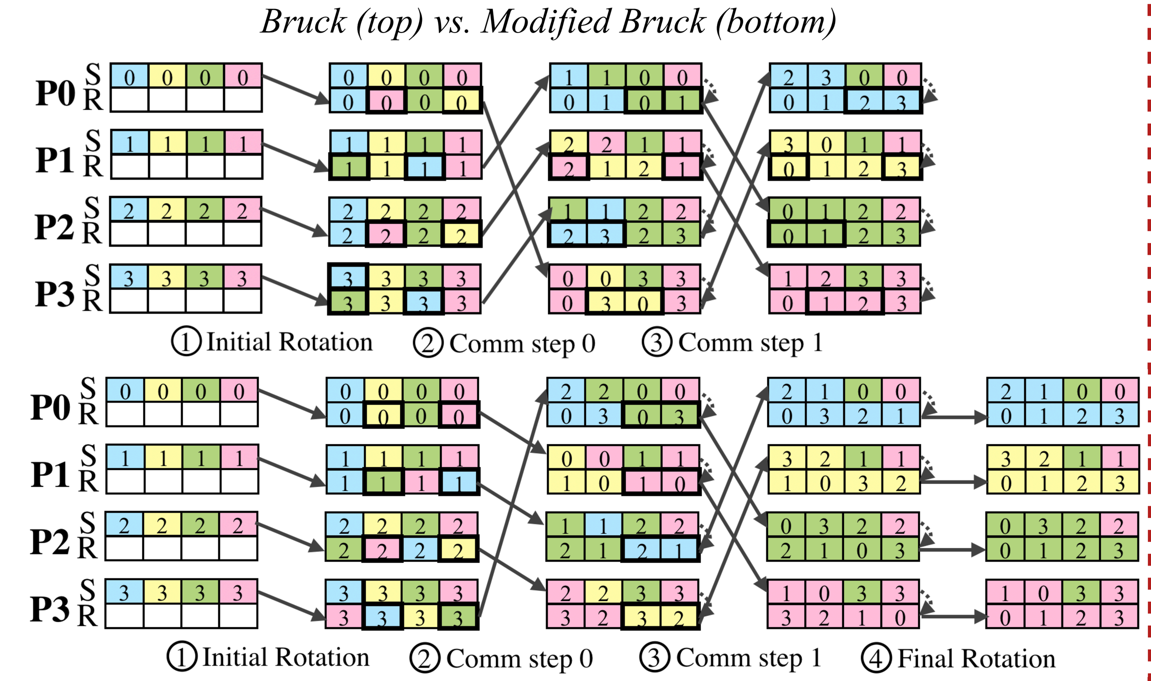



Fig.8

## Modified Bruck

The modified Bruck algorithm improves upon the Bruck algorithm by eliminating the final rotation phase.

*Bruck (top) vs. Modified Bruck (bottom)*

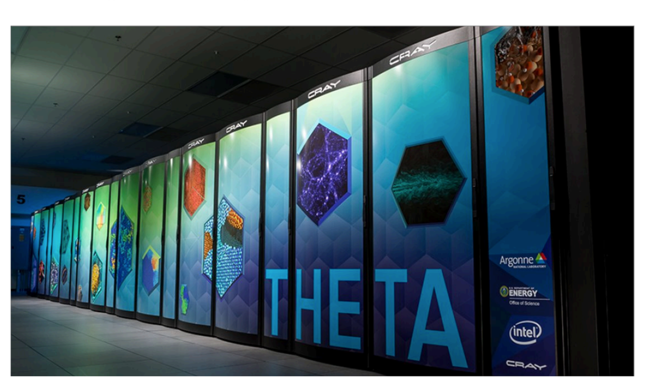



## Evaluation

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

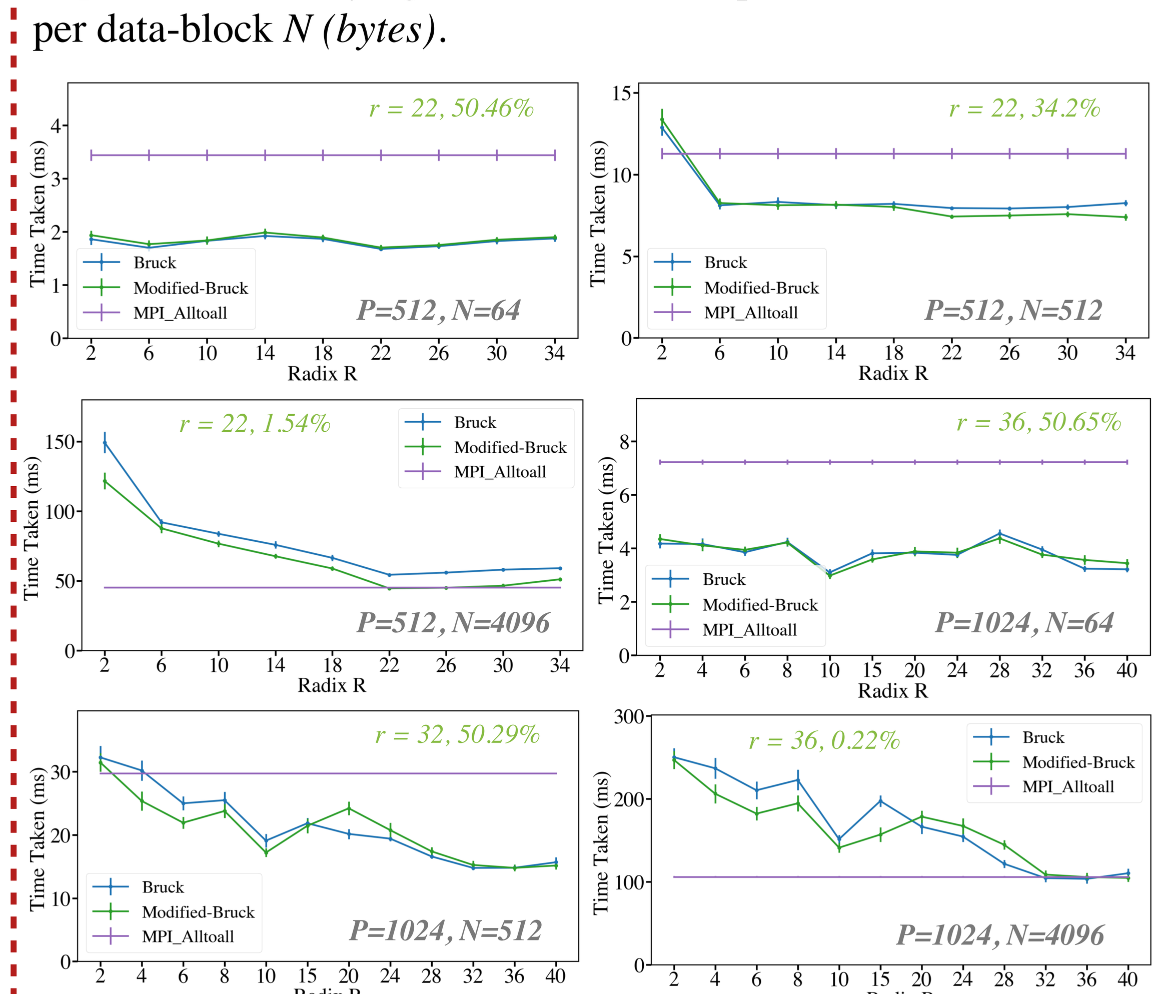
Argonne NATIONAL LABORATORY

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

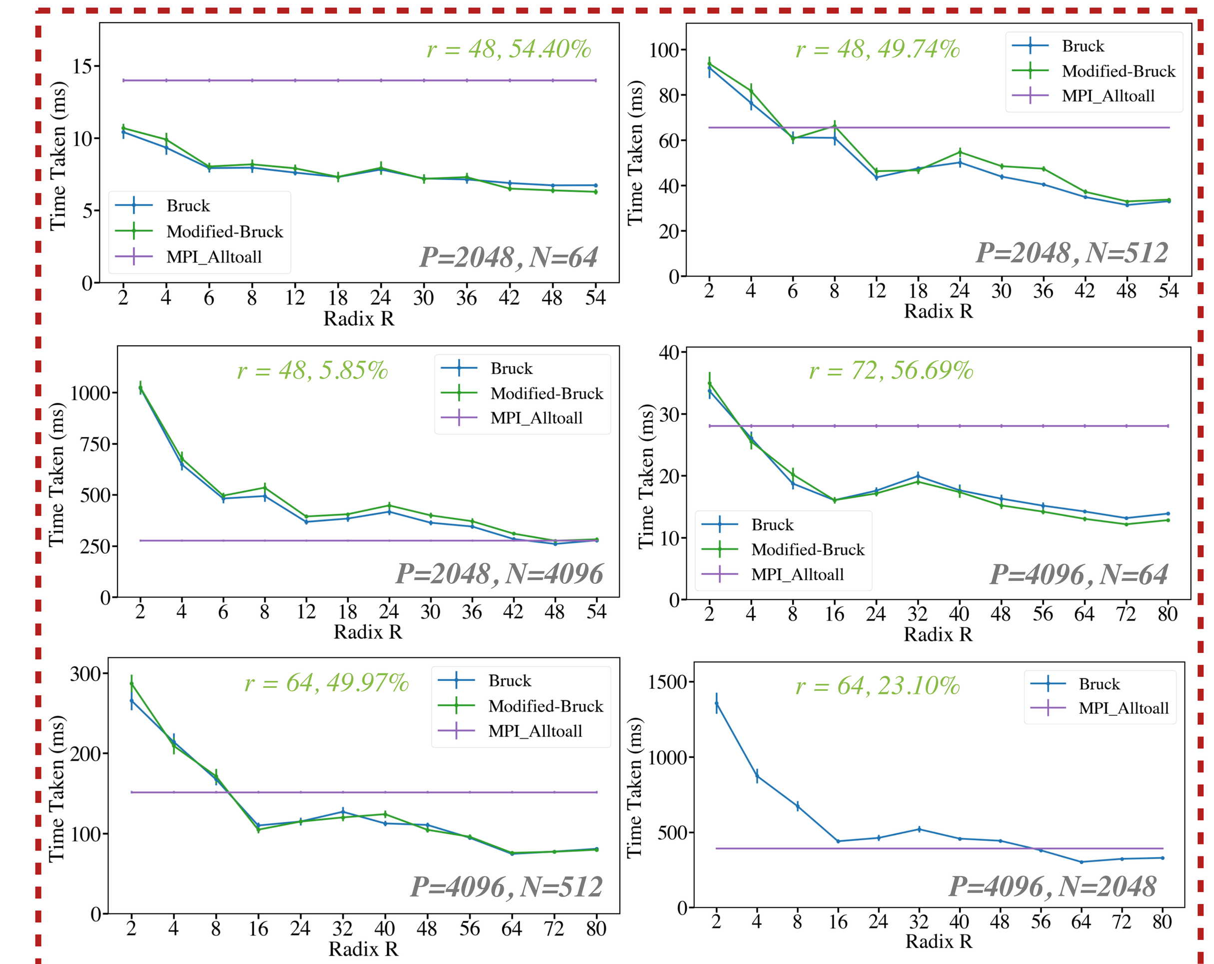



We repeated each experiment 100 times and plotted the mean along with the standard deviations (as error bars).

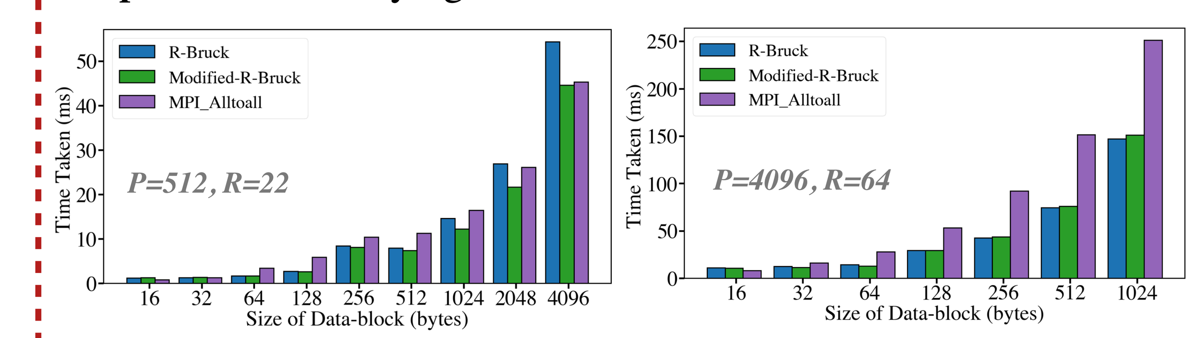
***Experiment A***: varying radix *r* with fixed process count *P* and size per data-block *N (bytes)*.



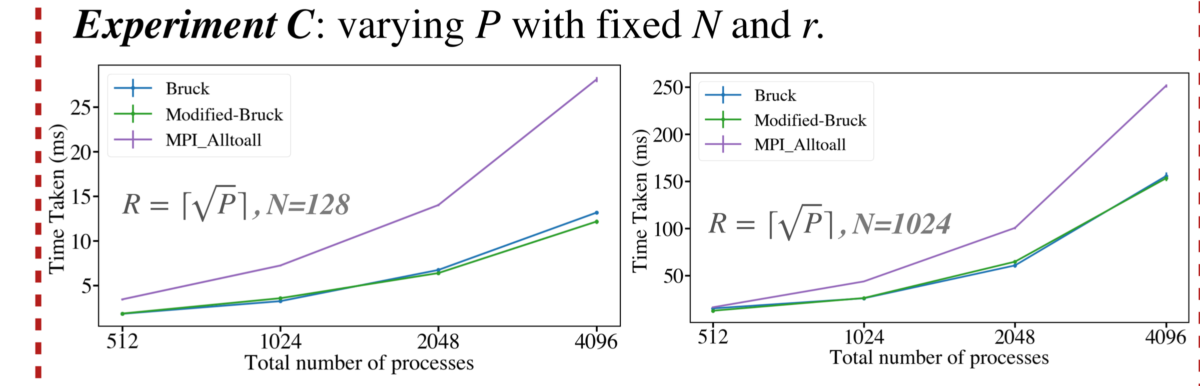

$(\lceil \sqrt{512} \rceil = 23)$, $(\lceil \sqrt{1024} \rceil = 32)$, $(\lceil \sqrt{2048} \rceil = 46)$, $(\lceil \sqrt{4096} \rceil = 64)$




***Experiment B***: varying *N* with fixed *P* and *r*.




***Experiment C***: varying *P* with fixed *N* and *r*.




The Bruck algorithm with r near $\lceil \sqrt{P} \rceil$ works well in most cases.

## Conclusion

**Conclusion:** In this paper, we explored the Bruck algorithm with varying radix r and figured out the mathematics of calculating the number of sent data-blocks per step and the sum of them. We performed scaling studies for a range of message sizes and radixes, and demonstrated that Bruck with optimal radix outperforms vendor-optimized Cray's MPI_Alltoall by as much as *57%* for some workloads and scales.

**Future work:** To optimize the Bruck algorithm with radix-r, we plan to preserve a more local communication pattern. Such as, we split all processes into groups based on their locations on the physical nodes, and the processes within one node perform a Bruck algorithm internally followed by an intra-node Bruck algorithm. Moreover, more work needs to be done to build a decision model that decides the value of r based on P and N automatically.

## Acknowledgements

This work was funded in part by NSF RII Track-4 award 2132013. We thank the ALCF's Directors Discretionary awards for offering us the compute hours on the Theta Supercomputer.