Configurable Algorithms for All-to-all Collectives

Ke Fan*§, Steve Petruzza†, Thomas Gilray‡, Sidharth Kumar*§
* University of Illinois at Chicago, Chicago, IL, USA
† Utah State University, Logan, UT, USA
‡ University of Alabama at Birmingham, Birmingham, AL, USA
§ Email: {kfan23, sidharth}@uic.edu

***Abstract*—`MPI_Alltoall` is a commonly used collective that allows a fixed-size data block to be exchanged between every pair of processes. The function can be implemented through a logarithmic number of point-to-point communication rounds, where the exact number of rounds and total data exchanged among processes depend on the log base (radix). This paper presents a mathematical foundation for studying all communication patterns for the all-to-all collective by developing parameterized formulas for total communication rounds and data exchanged. The model is used to narrow down a radix, $\sqrt{P}$ (P: process count), that effectively balances latency and bandwidth concerns, yielding optimal performance—as also confirmed via evaluation on the Theta and Polaris supercomputers at ANL. We also present a novel two-layer tunable radix algorithm to take advantage of the shared-memory parallelism offered by modern systems. The algorithm decouples communication rounds into two phases that can be individually optimized to take advantage of the shared memory and high-speed interconnect separately. Our approach demonstrates improvements of up to 3.8× on Theta and 4.2× on Polaris over the vendor-optimized MPICH-based implementation of `MPI_Alltoall` for fast Fourier transform application.**

I. Introduction

`MPI_Alltoall` is a widely used collective that facilitates all-to-all inter-process data exchanges [1], [2]. It is implemented through a logarithmic number of point-to-point communication rounds, with the extreme cases achieved with a log base of 2 and P (P: process count). The $\log_2$ implementation, also known as Bruck, takes the fewest possible number of communication rounds ($\log_2^P$) while transmitting an exaggerated amount of data. The $\log_P$ implementation, on the other hand, takes the largest possible number of rounds (P-1) while transmitting *only* the exact required data (similar to the linear time Spread-out algorithm [3]). Historically, only these two radices have been used by MPI implementations MPICH [4] and OpenMPI [5], [6].

The decision to select between the two radices is made by modeling the communication cost in terms of latency and bandwidth [7]. Latency is the fixed time-cost per communication step, independent of message size, whereas bandwidth is the variable transfer time per byte [8]. All-to-all involving short-sized messages are latency-bound [9] and thus naturally benefit from a lower-radix implementation that minimizes the number of communication rounds. On the other hand, all-to-all involving long messages are bandwidth-bound and, therefore, benefit from a higher-radix implementation that minimizes the total data transmitted.

The past few years have witnessed a renewed interest in implementing collectives [10], particularly all-reduce and all-to-all. This can be attributed to their use to support the communication requirements of parallel data-driven AI/ML applications [11], [12]. A work of specific interest [13] has presented implementations of all-to-all with higher radices. As a result, both K (total communication rounds) and D (total transmitted data-blocks) can be tuned based on the radix. However, it is crucial to correctly set the radix's value correctly to extract performance from such a tunable implementation. While [13] presents a high-radix implementation, it does not provide a heuristic to select the optimal radix.

Generally, as the radix increases from 2 to P, there is an increase in the total number of communication rounds (K) and a corresponding decrease in the overall number of data-blocks exchanged (D). So, as the radix r increases, there is a transition from latency-bound to bandwidth-bound communication. This paper observes that this transition is *not smooth*. We find that both the total number of communication rounds (K) and total data-blocks transmitted (D) are C^0 continuous and C^1 discontinuous (see Figure 4). The point of discontinuity occurs at radix $r = \sqrt{P}$. Up until $\sqrt{P}$, we observe a higher rate of increase for D w.r.t. r, and after $\sqrt{P}$, the rate of increase is distinctly slower. Essentially, radix $r = \sqrt{P}$ strikes a balance between latency and bandwidth by increasing the data transmission bandwidth (compared to radix P) enough to saturate the network without significantly increasing the latency cost (compared to radix 2).

A key novelty of this paper is its derivation of precise parametric formulas for both D and K for varying radices, which form the basis for the derivation of the optimal radix ($\approx \sqrt{P}$). The formulas are then translated into an open-sourced *parameterized* algorithm referred to as tunable-radix all-to-all (*TRA*), in which r can be tuned from 2 to P. We perform a detailed experimental investigation, varying r and P. Our empirical evaluation also confirms that $r \approx \sqrt{P}$ yields near-optimal performance in most cases — matching our theoretical hypothesis.

Another contribution of this paper is the two-layer tunable radix algorithm (referred to as *TRA2*), which takes advantage of the node-level shared memory parallelism available on modern HPC systems. TRA2 decouples its communication rounds into two epochs: *intra-node* data exchanges that benefit from latency-bound exchanges (low radix) and *inter-node* data exchanges that benefit from bandwidth-bound exchanges (high

radix). The two communication rounds can be independently tuned using two distinct radices.

In summary, this paper makes the following contributions:

1) Developed precise mathematical formulas to study the entire range of all-to-all data exchange patterns.
2) From both the formulas and experiments, made a deduction that *TRA* with $r \approx \sqrt{P}$ yields near-optimal performance.
3) Developed *TRA2* that further improves performance by employing a locality-aware communication pattern. *TRA2* decouples the communication rounds into local intra-node data and global inter-node data exchange phases.
4) Performed a detailed evaluation of our techniques using scaling studies (up to $16K$ processes) using microbenchmarks and real applications on Theta and Polaris.

Compared to the vendor's `MPI_Alltoall` implementation (Cray's proprietary MPI, based on MPICH [14]), our approach is up to 5x faster for some micro-benchmarks and up to 4x faster for practical application (Fast Fourier transform [15]).

II. Related Work

All-to-all data shuffle is known to be difficult to scale due to the quadratic nature of its workload. Some studies [16], [17], [18], [19] have been done to optimize uniform all-to-all algorithms. Recent works have looked into optimizing all-to-all for GPU-based clusters [20], [21]. Most relevant to our work is [13], which presented a high-radix implementation of all-to-all. Our paper builds on that work by developing a formal model for the tunable algorithm that helps deduce an optimal radix. Additionally, we created a two-layer algorithm that takes advantage of shared memory available on modern HPC systems.

Bruck algorithm [9] is an efficient implementation of all-to-all collective that reduces K from P to log_2^P by using radix $r = 2$. This algorithm comprises three phases: an initial local rotation phase, a communication phase containing multiple point-to-point data exchange rounds, and a final local rotation phase. The seminal paper [16], studied the performance of all-to-all for varying radices (r). The paper found that neither of the two extreme cases, $r = 2$ or $r = P$ is the best overall choice. Despite this observation, the popular MPI libraries only implement the two extreme ends with $r = 2$ and $r = P$. The paper, however, lacked exposition on three fronts: 1) formulas for calculating K and D when log_r^P is not an integer; 2) derivation of the optimal radix of the algorithm; and 3) detailed experimental evaluation at high scales due to the hardware limitation (they only used 64 processes). We address these three shortcomings in our work.

Optimizations of the Bruck: Träff et al. [19] investigated two incremental enhancements to Bruck. They first presented *modified Bruck*, which eliminates the final rotation phase by rotating the data blocks in a different order at the first step, anticipating the final block order. They then introduced a *zero-copy Bruck* algorithm, which eliminates internal memory copies required by the store-and-forward algorithm by employing a temporary buffer. This optimization, however, increases the overhead for combining the data sent from two buffers and only works well for a narrow range of message sizes. In addition, this paper further reduces explicit copying by using MPI-derived datatypes. However, the results revealed that using derived datatypes is less effective than using explicit buffer management (using memcpy). Cong Xu et al. [22] proposed utilizing a rotation indices array to store the rotated indices during the initial rotation phase rather than rotating the actual data-blocks. This method has shown performance improvement for the Bruck algorithm when D is small. Ke Fan et al. [23] optimized the Bruck algorithm by combining two ideas from [19], [22] for removing the initial and final rotation phases.

MPI collective algorithms with varying radices: Wilkins et al. [24] recent work presents parameterized implementations of popular collectives all reduce, broadcast, all-gather, and all-reduce. Similar to our work, this paper demonstrates that the commonly used radix of 2 and *P* do not necessarily demonstrate optimal performance. This work, however, does not deduce optimal prefixes and does not focus on all-to-all collectives. Jocksch et al. [25] developed a fully connected k-port ($k \geq 1$) algorithm to be used for all-to-all data exchanges in FFT computation. Under this model, each process can exchange (send and receive) k distinct messages with k other processes simultaneously. To maximize the performance of the k-port model, the radix of the algorithm is equal to the number of ports plus one ($k + 1$). In this case, they modified radices when k changed. However, they didn't study the varying radix when k is fixed. Andreas et al. [26] investigated the collective communication operations *allgatherv*, reduce scatter, and *allreduce* with varying radices. The radix is selected based on their measurements of communication times for different message sizes during the installation phase of the library. They claimed that applying high and low radices for short and long messages is efficient, respectively. However, they didn't show any experimental details about selecting the radix in the paper. Gainaru et al. [27] presented a formula for calculating the required number of communication rounds when $w = log_r^P$ is not an integer. However, the authors mainly focused on optimizing the Bruck algorithm with various memory layouts. They did not conduct any experiments to investigate the effects of varying radices.

Locality-aware collectives: Being aware of the physical location of processes (ranks) is essential for optimizing the performance of collectives, as communication costs can vary between pairs of processes; for example, *intra-node* communication is cheaper than *inter-node* communication. Bienz et al. [28] optimized the Bruck algorithm with locality awareness for *allgather* by grouping processes into groups of low-communication overhead regions. Jocksch et al. [25] presented an algorithm to exploit shared memory parallelism by explicitly creating a shared buffer for sending and receiving data within every node. Graham et al. [29] described a new hierarchical collective communication framework and demonstrated performance optimization for MPI barrier and

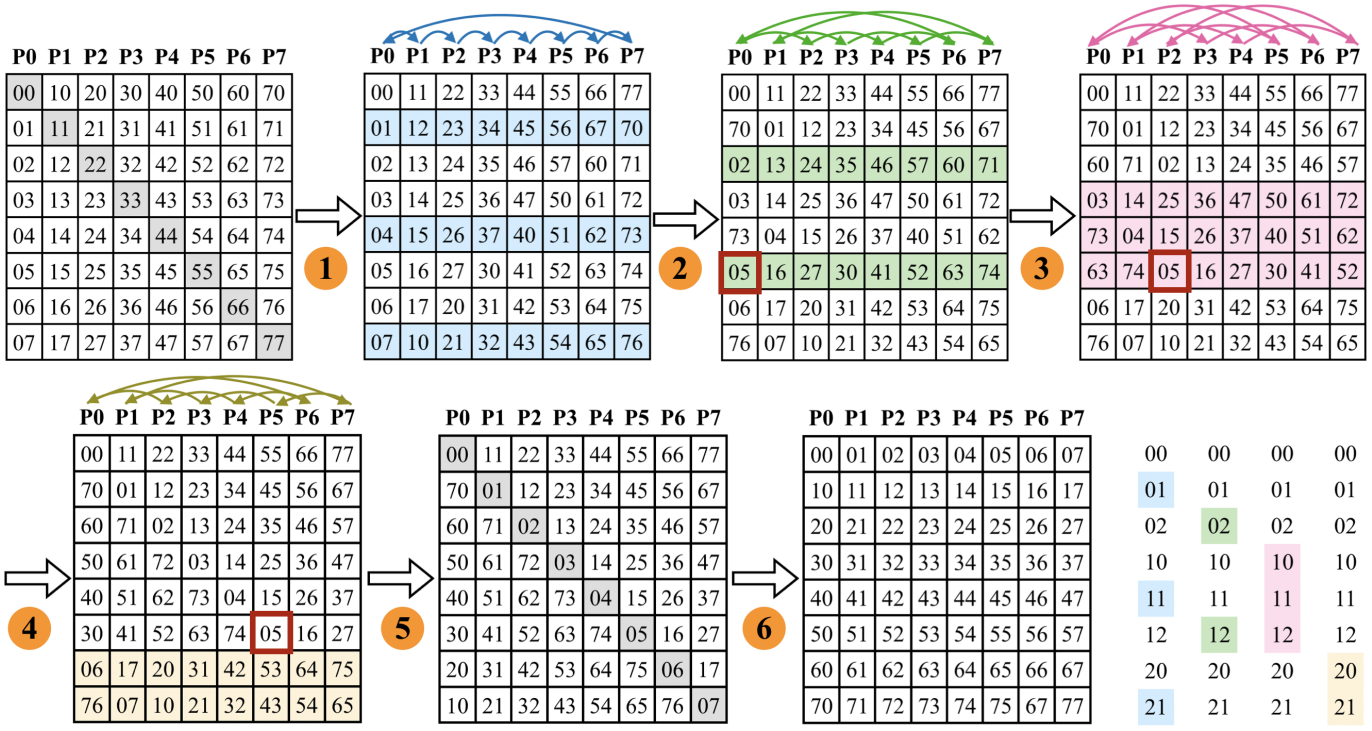

Fig. 1: An example of TRA algorithm with $r = 3$ and $P = 8$ processes: ① is initial rotation phase – moving up *rank* data-blocks; the first data-block for each process after rotation is highlighted in grey. ②, ③, ④, ⑤ are 4 communication rounds; each process exchanges some non-continuous data-blocks per round that are highlighted in a unique color. ⑥ is final rotation phase; the first data-block after rotation is highlighted in grey. The last figure shows the sent data-blocks in 3-representations per round, matching the colors with the previous four communication rounds.

broadcast. Jin Zhang et al. [30] optimized MPI collective communications by proposing a process-matching approach, by formalising the sequence of point-to-point communication operations as a graph mapping task.

III. Tunable-Radix All-to-all (TRA) Algorithm

We have developed the tunable radix all-to-all algorithm (TRA), where the underlying logarithmic radix can be tuned, which in turn tunes the number of communication rounds (K) and the total number of data-blocks transmitted (D). In this section, we derive the parametric formulas for both K and D, and use them to obtain an optimal radix.

Setup: With P processes, `MPI_Alltoall` can be expressed as follows. Every process has a *send buffer* (initialized with data), logically made of P data-blocks ($S[0 \dots P-1]$), each with n b-byte elements. Similarly, processes also have a *receive buffer* (initially empty), logically made out of P data-blocks ($R[0 \dots P-1]$) with n b-byte (data-block size: $(n \times b)$) elements. Both the send buffer and the receive buffer are contiguous 1-D arrays of size $P \times n \times b$ bytes where all data-blocks $S[0 \dots P-1]$ and $R[0 \dots P-1]$ are laid out in increasing block order. During communication, every process with rank p ($0 \leqq p \leqq P-1$) transmits the data-block $S[i]$ ($0 \leqq i \leqq P-1$) to a process with rank i and receives a data-block from rank i into the data-block $R[i]$.

Algorithm steps: In this section, we detail the three key steps of *TRA*: (a) an initial local rotation phase; (b) an inter-process data exchange phase with multiple communication rounds; and (c) a final local rotation phase. Both the initial and final rotation phases are independent of the radix (r). The data exchange phase is parameterized under the radix (r), which can range from *2* to P ($2 \leqq r \leqq P$). In the communication phase, indices of all P data-blocks are first encoded using the *r-base*. The number of digits for r-base encoding for every data-block, w, is therefore $= \lceil log_r^P \rceil$. Each communication round k is identified by two variables: x ($0 \leq x < w$) and z ($1 \leq z < r$). For example, when $w = 2$ and $r = 3$, the first three communication rounds correspond to $x = 0$ and z ranges from 0 to 2 (inclusive) (see Figure 1, matches blue, green, pink rounds respectively). The three steps of *TRA* can be precisely formulated as follows:

1) Local shift of data-blocks from S to R: $R[i] = S[(p + i)\%P]$ (see Figure 1①).
2) Communication steps: In each step k parameterized by x and z (translating to a nested for-loop), process p sends to process $((p + z \times r^x)\%P)$ all the data-blocks $R[i]$ whose x^{th} bit of i in *r-base* is z. Process p receives data from process $((p - z \times r^x + P)\%P)$ into S, and replaces $R[i]$ (just sent) locally (see Figure 1②-⑤).
3) Local inverse shift of data-blocks from R to R: $R[i] =$

$R[(p-i)\%P]$ (see Figure 1⑥).

A. *Parameterizing* K *and* D

The communication step of *TRA* consists of K ($log_2^P \leqq K \leqq P$) point-to-point exchange rounds where a process transmits a total of D data-blocks across all rounds. K and D are negatively correlation – higher K implies smaller D and vice versa. Using the popularly used topology agnostic hockney model [31], the performance of any point-to-point communication can be modeled as a function of its latency (α) and bandwidth (β) costs. Latency (α) is the fixed cost per communication step, which is independent of communication size, whereas bandwidth (β) is the transfer time per byte [8]. Suppose we transfer m-byte message from one process to another; then the communication cost can be modeled as $T = \alpha + m \times \beta$. Under this paradigm, the cost of an all-to-all exchange can be expressed as $\alpha \times K + \beta \times D \times S$, where S is the size in bytes of every data-block. Therefore, it is critical to quantify K and D, with K being a latency-related metric and D being a bandwidth-related metric. To this end, in this section, we present four propositions with formulas for both K and D, where they are parameterized as a function of the radix r. Specifically, the first two propositions are designed for the case in which $P = r^w$, while the last two are designed for a more general case in which $P < r^w$, in which case $w = \lceil log_r^P \rceil$. Recall that w is the number of digits needed for r-base encoding of the P data-blocks.

Proposition 1: When $P = r^w$ ($2 \leq r < P$), total number of communication rounds, $K = w(r-1)$.

Proof 1: Let's focus on one process, $P0$. The all-to-all operation requires $P-1$ point-to-point rounds if $P0$ sends one data-block per round. With r-base, one point-to-point round can be divided into multiple rounds. For example, $P0$ transmitting its 5^{th} data-block to $P5$ (5 is 12 with 3-base encoding: $1 \times 3 + 2$) with communication distance $L = 5$ $(5-0)$ can be broken in two steps: $P0$ transmitting to $P2$ with $L = (2-0) = 2$ (Figure 1② to ③)) and then $P2$ to $P5$ with $L = (5-2) = 3$ (Figure 1③ to ④)) (highlighted by red boxes). If all $P-1$ rounds are decomposed in the same manner, the communication rounds can be rescheduled by grouping all data-blocks based on the values of x and z that designate a new communication round k, assuming x^{th} bit of a data-block index with r-base is z (e.g., for the index 12 in 3-base, $z = 2$ at $x = 0$ and $z = 1$ when $x = 1$). Plus, $z = 0$ for any x is meaningless (The index 0 with r-base is owned by each process itself). Therefore, x ranges from 0 to $w-1$ and z ranges from 1 to $z-1$, resulting in $w(r-1)$ communication rounds in total. For example, as shown in Figure 2, $P0$ groups all data-blocks whose 0^{th} bit is 1 (colored in blue) at the first communication round. The entire all-to-all communication, in this instance, can be finished in four rounds. In general, the all-to-all operation can be done using two nested loops, with the outer loop traversing w digits with r-base and the inner loop traversing each digit's $r-1$ distinct non-zero values. Therefore, $K = w(r-1)$.

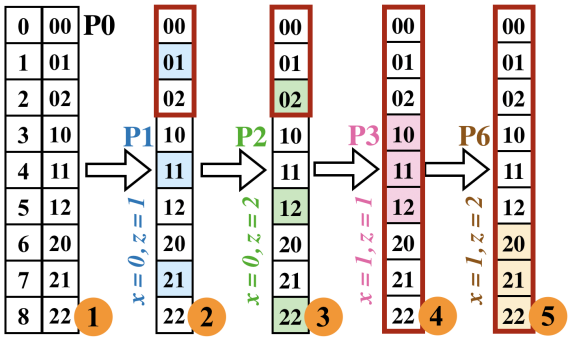

Fig. 2: An example of the *TRA* algorithm with $r = 3$ and $P = 9$ processes, using an illustration of process P to demonstrate how to complete the algorithm in 4 rounds. We initially encode indexes of data-blocks using *r-base* encoding that uses 2 digits (①). Then, $P0$ sends all data-blocks whose x_{th} digit is value z ($1 \leq z < r$) to the process with distance ($z \times r^x$) each round – ② first round to $P1$; ③ second round to $P2$; ④ third round to $P3$; ⑤ first round to $P6$. Each round treats every r^{x+1} data-blocks as a group. The first group is highlighted with a red box. A sent continuous unit per round (starts from r^x and contains r^x data-blocks) in each colored group.

Proposition 2: When $P = r^w$, total number of transmitted data blocks, $D = w(r-1) \times r^{w-1}$.

Proof 2: To finish the method in exactly $w(r-1)$ rounds, each process must transmit all data-blocks whose x^{th} digit is equal to z (e.g., Figure 2②). With r-base, we partition all data-blocks into $g = P/r^{(x+1)} = r^{(w-x-1)}$ groups, with $r^{(x+1)}$ data-blocks in each group (e.g., in Figure 2② ($x = 0$ and $z = 1$), there are 3 groups where each has 3 data-blocks (the first group per round is highlighted with red box). We suppose that the continuous r^x data-blocks with the same x^{th} digit form a unit (e.g., in Figure 2④, 3 pink data-blocks are a unit, but in Figure 2②, each data-block are a unit.). Then each group has r units and each process transmits one unit with value z from each group per round (e.g., in Figure 2②, each process sends one blue unit from each of the 3 groups.). Therefore, each process transfers $g \times r^x = r^{(w-x-1)} \times r^x = r^{(w-1)}$ data-blocks per round. That is each process transfers $w(r-1) \times r^{w-1}$ data-blocks in total.

Proposition 3: When $P < r^w$, total number of communication rounds $K = w(r-1) - \lfloor (r^w - P)/r^{w-1} \rfloor$.

Proof 3: Let $P' = r^w$. In *Proof* 1, we showed that any all-to-all algorithm requires $w(r-1)$ communication rounds with P' processes. Compared to P', P excludes v values of the highest $(w-1)^{th}$ digit with *r-base*. Assuming that $d = P' - P$, then $v = \lfloor d/r^{w-1} \rfloor$, since each value z with x^{th} bit has r^{w-1} data-blocks. This means $(w-1)^{th}$ digit has only $(r-1-v)$ distinct values (e.g., in Figure 3⑥, the 3^{rd} digit only has value 1 ($d = (27-11) = 16$, $v = \lfloor 11/6 \rfloor = 1$), $(r-1)-v = 1$).). Therefore, any all-to-all algorithm requires $w(r-2) + (r-1-v) = w(r-1) - v = w(r-1) - \lfloor (r^w - P)/r^{w-1} \rfloor$

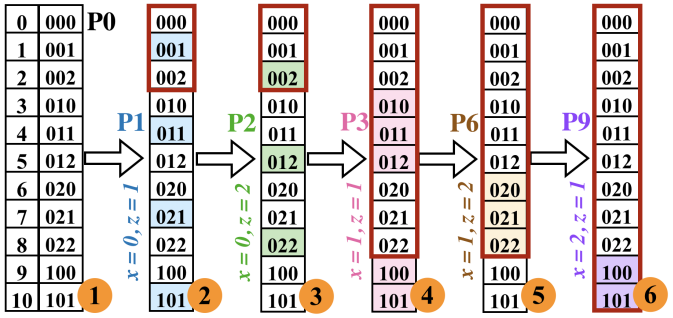

Fig. 3: An example of the *TRA* algorithm with $r = 3$ and $P = 11$ processes, using an illustration of process $P0$. ① encode indexes of data-blocks using *r-base* with 3 digits; ②, ③, ④, ⑤, ⑥ are communication rounds.

communication rounds.

Proposition 4: When $P < r^w$, $lgc = P \ \% \ r^{x+1}$ $(0 \leq x < w)$, $t = lgc - z \times r^x$ $(1 \leq z < r)$, and $lc = \lfloor P/r^{x+1} \rfloor \times r^x$, a communication round characterized by x and z transmits h data-blocks:

1) $h = lc$, if $(t \leq 0)$)
2) $h = lc + r^x$, else if $(\lfloor t/r^x \rfloor > 0)$
3) $h = lc + t \ \% \ r^x$, else

with, $D = \sum_{x=0}^{w-1} \sum_{z=1}^{r-1} h$.

Proof 4: As stated previously, each process transmits one unit from each of g groups per round, where a unit consists of continuous r^x data-blocks with the same x^{th} digit (The first group per round is highlighted with a red box in Figure 3). Due to $P < r^w$, the last group may have fewer data-blocks than others (e.g., in Figure 3③, the last group only has 2 data-blocks (100, 101).). Except for the last group, each process transmits z^{th} unit in each group per round, so that each process sends at least $lc = g \times r^x$ to others (e.g., in Figure 3③, there are 4 groups, but 3 green units are transmitted.). Assuming $g = \lfloor P/r^{(x+1)} \rfloor$, the number of data-block in the last group is $lgc = P \ \% \ r^{x+1}$ (e.g., in Figure 3, when $x = 0$, $lgc = 11 \ \% \ 3 = 2$.). Let $t = lgc - z \times r^x$. If $t \leq 0$, it indicates that no data-blocks will be transmitted in the last group; hence, each process sends lc data-blocks per round (e.g., in Figure 3③, $t = 0$, so no data-blocks are sent in the last group). If $\lfloor t/r^x \rfloor > 0$, then the entire z^{th} unit in the last group is transmitted, so each process sends $(lc + r^x)$ data-blocks (e.g., in Figure 3②, $\lfloor t/r^x \rfloor = 1$, so the last blue unit is sent.). Otherwise, it indicates that part of data-blocks in the last group are transmitted, so each process sends $lc + t \ \% \ r^x$ data-blocks (e.g., in Figure 3⑥, $lgc = 11 \ \% \ 27 = 11$, $t = 2$, $\lfloor t/r^x \rfloor = 0, t\%r^x = 2$, so only 2 data-blocks are sent).

Note, that D is the total number of transmitted data-blocks by a process, and therefore the total amount of data transmitted by a process is $D \times S$ bytes (S: the size of data-block in bytes).

B. *Optimal Radix*

As is evident from the four prepositions, both the total amount of data-blocks transmitted (D) and the total number of communication rounds K depend on the radix (r) and the total number of processes (P). To understand the nature of this dependence, we use the formulas obtained in the prepositions to obtain values of both K and D for all possible combinations of r and P. Due to space constrictions, we show plots for $P = 1{,}024, 2{,}048, 4{,}096$ and $8{,}192$, for each of which r is swept from 0 to P (Figure 4).

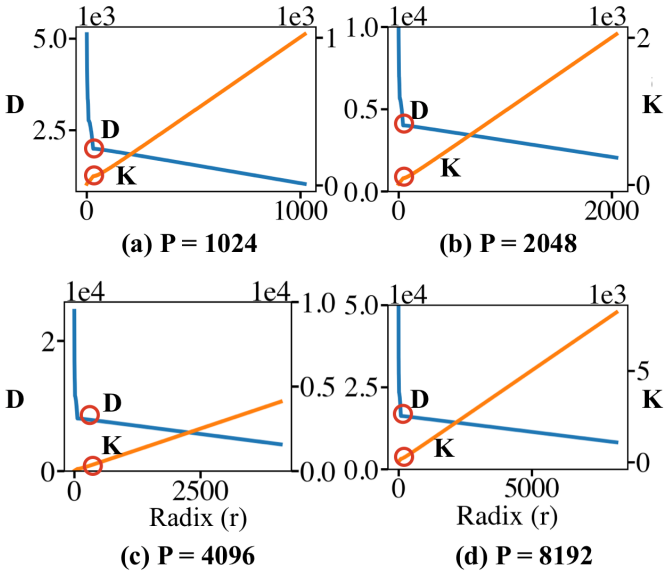

Fig. 4: Mathematical simulation with varying process counts (P): K (orange) increases while D (blue) decreases with increasing radices.

Overall, we observe that when radix increases, K exhibits an increasing trend, and D exhibits a decreasing trend. We note two extreme cases of the communication spectrum. 1) when $r = 2$, the algorithm has the smallest K ($\log_2^P$) but the largest D ($w2^{w-1}$ $(w = \lceil log_2^P \rceil)$). This results in low latency and is thus suited for all-to-all with short messages. 2) When $r = P - 1$, the algorithm has the largest K ($\leq (P-1)$) but the smallest $D\times$ $(P-1)$. This configuration is suited for long message sizes when communication is bound by bandwidth. Popular MPI libraries implement the all-to-all collective using these two extreme cases. The message size and process count determine which case to choose at runtime.

While exploring the space between these two extreme cases, interestingly, we observe a point of C^1 discontinuity in both the K and D curves, as depicted by the red circles in Figure 4, with the D curve being much more noticeable. Before these points, D decreases, and K increases dramatically with increasing values of r. After this discontinuous point, the rate of decrease/increase of D/K becomes slow, decreasing at max by 1 with every increasing value of r. We found this inflection point occurs when $r = \sqrt{P}$, referred to as the optimal radix in the paper.

Using our formulas, if $r = \sqrt{P}$, then the required number of communication rounds $K = w(r-1) = 2\sqrt{P} - 2$, which is significantly less than $(P-1)$ rounds when $r = P$.

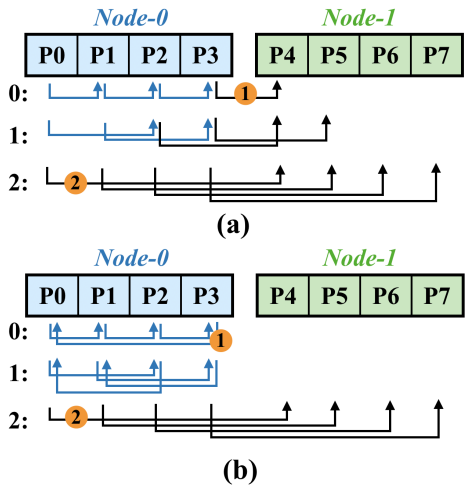

Fig. 5: Communication pattern for (a) *TRA* and (b) *TRA2* with $r = 2$ and $P = 8$, focusing on the first node only. There are 2 nodes (differentiated by colors) with 4 cores (processes) per node. The colored arrows represent *local* inter-node comm, and the black arrows represent *inter-node* global comm.

We demonstrate this difference by computing K for the two radices at $P = 16{,}384$. K at $r = \sqrt{P}$ is 254 which is $\approxeq 64\times$ less than the K ($= 16{,}384$) at $r = P$. On the other hand, D at $r = \sqrt{P}$ is 32,512 which is $\approxeq 3.5\times$ smaller than the D ($= 114{,}688$) at $r = 2$. In essence, the optimal radix strikes a balance between the total number of communication rounds (K) and total data transmitted (D)—significantly decreasing K (e.g., $64\times$) while only reasonably increasing D (e.g., $3.5\times$). The optimal radix finds a balance between the log base 2 and P and can potentially result in optimal performance. We validate this empirically in Section V-A.

We note that the optimal radix is deduced from the trends of K and D, which are parameterized via P and r. Naturally, the performance of all-to-all must also consider the size of a data-block (S), which our formulas do not consider– we discuss this limitation in Section VII.

IV. Two-layer Tunable-Radix All-to-all (TRA2)

The computing environment of the modern HPC system has multiple CPU cores per node with shared memory [25]. Data exchanges between cores on the same nodes translate to a direct memory copy. Therefore, they are faster than data exchanges between cores on different nodes, which require data to move via the network [28], [20]. To exploit this extra locality offered by the shared memory, we develop a new algorithm called the two-layer (**2**) **T**unable **R**adix **A**ll-to-all algorithm (TRA2), that improves upon *TRA*.

TRA2 can be explained by three of its key characteristics: (c1) reduced number of global *inter-node* data exchange rounds; (c2) decoupled communication, with an initial set of *intra-node* data exchange rounds that only require message transfers that access the shared memory (and are thus much faster), and a final set of *inter-node* data exchange rounds that have message transfers between nodes; and (c3) an optimal data exchange pattern where data packets with the same source and destination ranks are bundled together. c1, c2, and c3 are all closely linked, as the global *inter-node* data exchanges of the otherwise initial rounds get pushed to the later (inter-node) rounds, where they are bundled with data packets with matching destination ranks, therefore reducing the total number of global data exchanges and also decoupling the communication rounds into *intra-node* and *inter-node* rounds. Note that the initial *intra-node* rounds route the global *inter-node* data packets to intermediate processes (within the node), preparing them to be transferred during the *inter-node* rounds.

Example: We demonstrate c1, c2 and c3 in Figure 5. It can be seen that *TRA* has 7 global *inter-node* data exchanges (black arrows) compared to *TRA2*'s four. And, only the last communication round of *TRA2* involves inter-node data exchange compared to all three rounds of *TRA*. Looking more closely, it can be seen, that the P3 $\rightarrow$ P4 (Figure 5(a)①) *inter-node* data exchange in the first communication round of *TRA* gets translated to $P3 \rightarrow P0$ (Figure 5(b)①) *intra-node* communication followed by P0 $\rightarrow$ P4 (Figure 5(b)②) *inter-node* communication in *TRA2*. This makes it possible to decouple *intra-node* from *inter-node* communication and also reduces the total number of global *inter-node* exchanges. Finally, we observe that the total number of data exchanges remains unchanged, as the $P0 \rightarrow P4$ (Figure 5(b)①) data exchange of the intermediate data packet is grouped with the other $P0 \rightarrow P4$ (Figure 5(a)②) data exchange.

Intra-node and inter-node radices: With *TRA2*, we have the additional flexibility to *independently* tune the radices of both the *intra-node* and *inter-node* phases. The required number of *intra-node* and *inter-node* rounds (K_1 and K_2) with radices r_1 and r_2, respectively are (from Section III-A):

$$w_1 = \lceil log_{r_1}^{Q} \rceil, \;\; K_1 = w_1(r_1 - 1) - \lfloor r_1 - P/r_1^{w_1 - 1} \rfloor$$
$$w_2 = \lceil log_{r_2}^{N} \rceil, \;\; K_2 = w_2(r_2 - 1) - \lfloor r_2 - P/r_2^{w_2 - 1} \rfloor$$

As *intra-node* communication involves all-to-all data exchanges only within a node, the communication round i ($0 \leq i < K_1$) eliminates all global (inter-node) communications for those rounds. Additionally, if $r = r_1 = r_2$ and $log_{r_1}^{Q}$, $log_{r_2}^{N}$, and log_{r}^{P} are all integers (N: number of nodes, Q: number of processes per node), then the *TRA2* algorithm requires the same number of communication rounds as *TRA* i.e. ($K = K1 + K2$). For example, in Figure 5, there are $N = 2$ nodes with $Q = 4$ cores each. Using a radix of 2, both of these patterns require $log_2^8 = 3$ rounds.

A. Implementation

Decoupling the communication rounds into intra-node and inter-node data exchange phases is only possible when the total number of processes (P) an application runs on is an integral multiple of Q (number of cores per node). This step ensures that the intra-node communication perfectly aligns with the inter-node communication by performing an exact

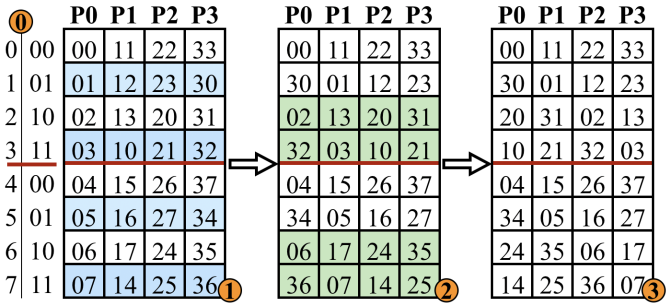

Fig. 6: Example of intra-node communication with $r = 2$, $P = 8$, $N = 2$, and $Q = 4$ focusing on the first node. ⓪ shows 2-representations of $(i \ \% \ Q)$ (i is index of a data-block). In the initial rotation phase, each process moves up $(rank \ \% \ Q)$ data blocks in every Q data block. ① represents the rotation result. ① and ② are two communication rounds in which the sent data blocks are shown in color. In ③, every process receives the needed data blocks but requires another rotation phase to arrange them in ascending order.

fraction $(1/Q)$ of the total all-to-all workload. While this is a limitation of *TRA2*, one should note that users typically map their applications to use all cores available on a node and, hence, would typically not run into this limitation.

Intra-node communication phase is achieved by performing N (node count) concurrent all-to-all exchanges. Each of these N groups involves Q processes. The total number of processes is $P = Q \times N$. The intra-node communication is effectively achieved by performing initial and final rotations independently for every group of Q data-blocks, followed by data exchanges. To perform the local rotations, we re-index the P logical data-blocks into groups of Q indices starting from 0 to $(Q-1)$ (see Figure 6, second column of indices, ranging from 00 to 11). The two intra-node groups can be seen as separated by the horizontal red line. The rotation phase is followed by K_1 communication rounds, transmitting all data-blocks to their appropriate destinations, the (*node-local* data-blocks to the final target process and *global inter-node* data-blocks to an intermediate process). As an optimization, all data-blocks in different logical groups are merged and transmitted simultaneously (e.g., $P0$ in Figure 6, transmits all colored data-blocks *all at once* to P1 in the first round). The formulaic description of the intra-node communication is shown below:

1) Local shift of data-blocks $(0 \leq j \leq Q)$ in each group $(0 \leq i < N)$: $R[i \times Q + j] = S[i \times Q + (gp + j) \ \% \ Q]$, where $gp = p \ \% \ Q$ (p: rank id of process).
2) Communication steps in all groups: In each step k comprising x $(0 \leq x < w_1)$ and z $(1 \leq z < r_1)$, process p sends to process $(ni \times Q + (gp - z \times r^x + Q)\%Q)$ all the data-blocks $R[i]$ whose x^{th} bit of i in *r-base* is z, where $ni = \lfloor p/Q \rfloor$ is the id of each node. Process p receives data from process $(ni \times Q + (gp + z \times r^x)\%Q)$ into S, and replaces $R[i]$ (just sent) locally.

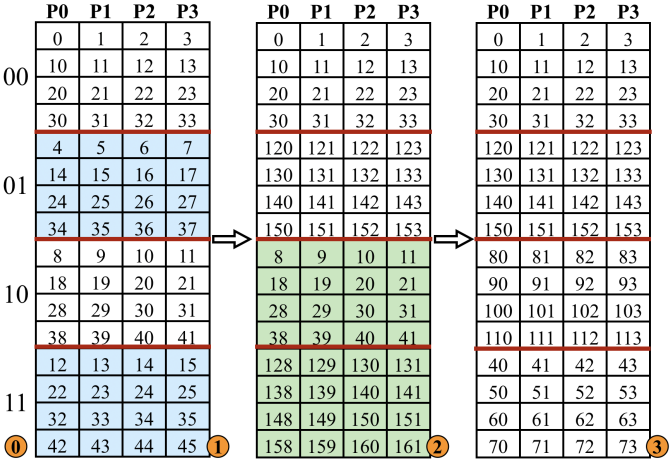

Fig. 7: An example of inter-node communication ($r = 2$, $P = 16$, $Q = 4$), focusing on the first node. In this communication, we treat all processes in each node as a *super-process* that involves all-to-all communication. ⓪ shows 2-representations of indices of nodes. ① and ② are two communication rounds, in which sent *super-data-blocks* are colored. ③ shows the state after communication but requires another rotation phase.

3) Local inverse shift of data-blocks from R to S: $S[i \times Q + j] = R[i \times Q + (gp - j + Q) \ \% \ Q]$.

In the ***inter-node communication*** phase, groups of Q processes operate as a virtual process, called a *super-process*. This communication is analogous to *intra-node* communication (see Figure 6), with the exception that the communication occurs between *super-processes* rather than processes. For example, in Figure 7, every *super-process* comprises four processes. The number of *super-processes* is $N = P/Q$. Typically, Q is the actual number of physical cores on a node. Similarly, a *super-block* consists of Q^2 data-blocks that are always transferred to the same destination *super-process*. Therefore, the number of *super-blocks* is the same as that of *super-process*. For example, in Figure 7, every 4×4 data-blocks constitute a *super-block* (e.g., the first 16 blue data-blocks that need to be sent to the next *super-process*) (differentiated by red lines). To perform the two rotation phases, the indices of *super-blocks* are encoded into r_2 representation instead of indices of data-blocks. For example, in Figure 7, the four *super-blocks* are encoded as 00, 01, 10, 11. The communication phase follows the Q-port model in which every Q point-to-point data exchange is delivered simultaneously. Each process in a *super-process* serves as a communication port, which transmits a $1/Q$ message to the corresponding process in another super-process. In Figure 7, for instance, *super-process*-0 (containing $P0$, $P1$, $P2$, and $P3$) must send data to *super-process*-1 (containing $P4$, $P5$, $P6$, and $P7$) in the first communication round (transferring the two blue colored *super-blocks*). In this case, four processes in *super-process*-0 can simultaneously send four direct messages to four processes in *super-process*-1 ($P0 \rightarrow P4$, $P1 \rightarrow P5$, $P2 \rightarrow P6$, $P3 \rightarrow P7$). The formulaic

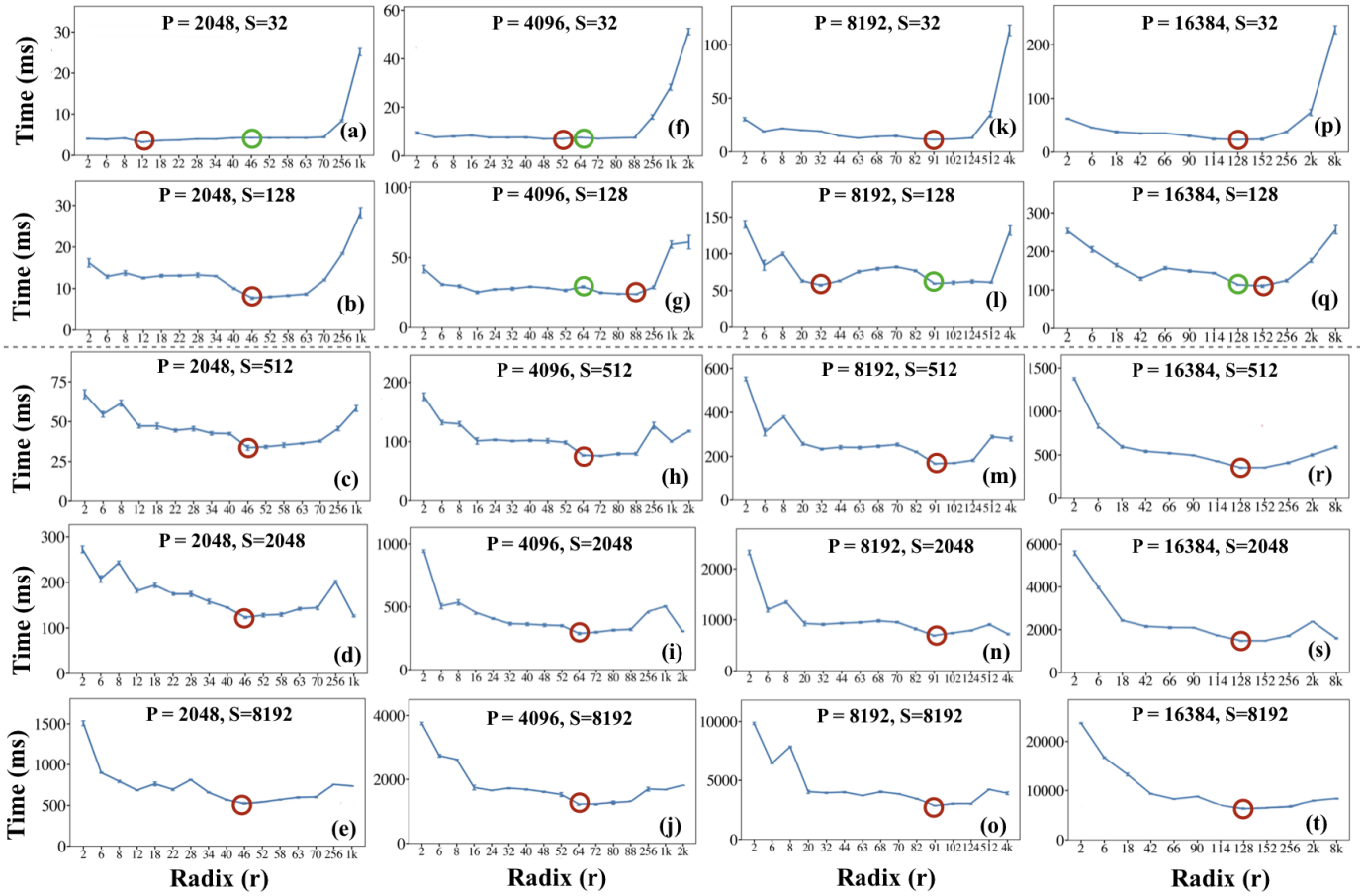

Fig. 8: Performance analysis of TRA on Theta with varying process count (P), size per data-block (S), and radix (r). A red circle highlights the best r with optimal performance in each figure, while a green circle indicates $r = \sqrt{P}$ when it is not equal to the best radix. ($\lceil\sqrt{2048}\rceil = 46$, $\sqrt{4096} = 64$, $\lceil\sqrt{8192}\rceil = 91$, $\sqrt{16384} = 128$).

description of inter-node communication is as follows:

1) Local shift of data-blocks for all groups ($0 \leq i < N$): *memcpy*($R[(ni + i)\%N \times Q]$, $S[i \times Q]$, Q), where $ni = \lfloor p/Q \rfloor$ is the id of each node.
2) Communication steps across groups: In each step k comprising x ($0 \leq x < w_2$) and z ($1 \leq z < r_2$), process p sends to process $((ni \times Q + (gp - z \times r^x + N))\ \%\ N)$ all the data-blocks $R[i]$ whose x^{th} bit of i in *r-base* is z, where $gp = p\ \%\ Q$ (p: rank id of process). Process p receives data from process $(ni \times Q + (gp + z \times r^x)\ \%\ N)$ into S, and replaces $R[i]$ (just sent) locally.
3) Local inverse shift of data-blocks: *memcpy* ($R[(ni - i + N)\%N \times Q]$, $R[i \times Q]$, Q) ($0 \leq i < N$).

V. Evaluation

We conduct a thorough evaluation of our algorithms using micro-benchmarks on the Theta supercomputer of Argonne National Laboratory (ANL) [32]. Theta is a Cray machine with a peak performance of 11.69 petaflops, 4,392 compute nodes with 64 cores per node. To show the generalizability of our algorithm, we also conducted a handful of experiments on the Polaris supercomputer at ALCF (Cray MPICH 8.1.16) [33]. Polaris is a 44 petaflop HPC system with 560 nodes each with 32 AMD CPU cores and 4 A100 GPUs. Both systems use an Aries interconnect with Dragonfly topology.

To demonstrate the efficacy of *TRA* and *TRA2*, we compare their performance against vendor-optimized `MPI_Alltoall` in Cray MPI, which is a proprietary, closed-source implementation from Cray based on the MPICH distribution [14]. In Section III-A, we demonstrated that the performance of all-to-all is influenced by two parameters: process count (P) and radix (r). We, therefore, perform a set of experiments in which we vary these two parameters for both *TRA* (Section V-A) and *TRA2* (Section V-B). In addition to these two parameters, we also explore the impact of the size of data-block (S). We deduced the optimal radix for both *TRA* and *TRA2* from those experiments. Finally, we conducted comparative experiments of *TRA* and *TRA2* (using their optimal radices) against `MPI_Alltoall` (Section V-C). All our experiments were performed for a minimum of 100 iterations, and we reported the meantime and the standard deviations (using error bars). We used the MPI-everywhere programming model that maps one rank per core (e.g., we have 64/32 processes per node on Theta/Polaris).

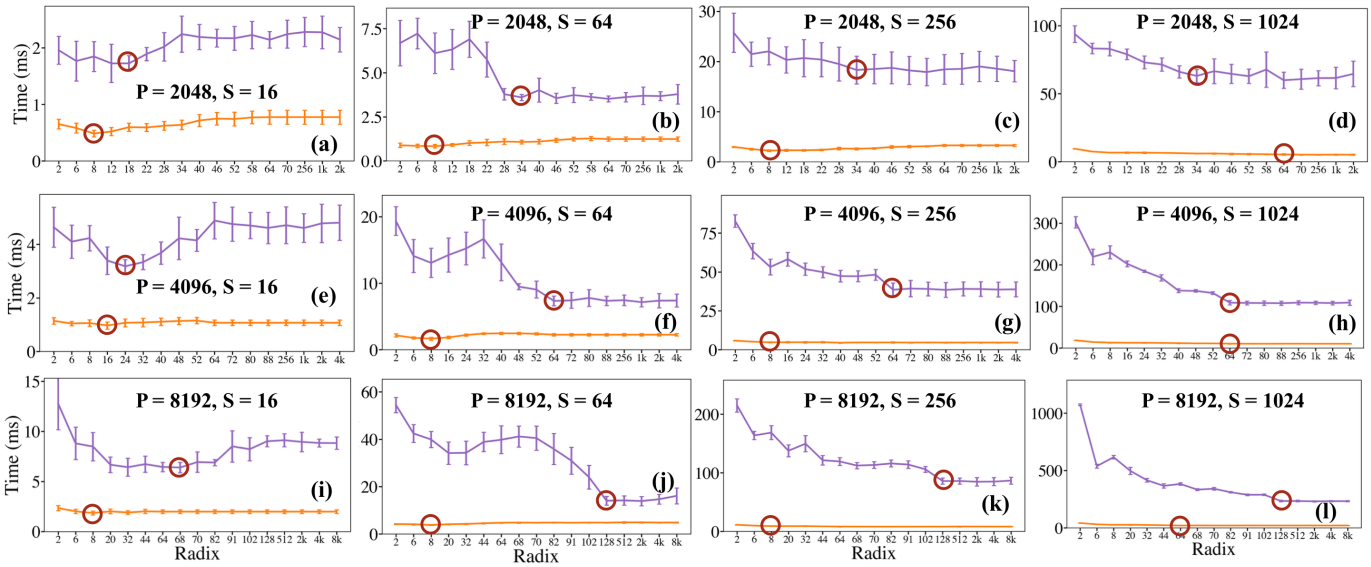

Fig. 9: Performance analysis of TRA2 on Theta (orange line: intra-node comm., purple line: inter-node comm.) with varying process count (P), size per data-block (S), and radix (r). The best $r1$ and $r2$ for intra-node comm and inter-node comm with optimal performance are highlighted by red circles.

A. *Performance analysis of TRA*

To understand the performance of *TRA*, we conducted experiments that varied P from 512 to 16,384, S from 16 to 8,192 bytes, and r from 2 to $P/2$. Due to space constraints, we only present a subset of the results in Figure 8. For each experiment (graph), we fixed P and S while varying r. In each graph, we highlight the most optimal performance with a red circle while highlighting the performance at $r = \sqrt{P}$ with a green circle. From the results shown in Figure 8, we make the key observation that $r \approx \sqrt{P}$ consistently outperforms other radices for all process counts (P) for large-sized data-blocks ($S \geq 512$) (matching the formulaic deduction of Section III-B). This can be seen for all graphs below the dashed line in Figure 8, where the optimal radix (red circle) overlaps with $\sqrt{P}$ (green circle). For example, with $P = 2{,}048$ and $S = 128$, $r = 2$ and $r = 46$ take 16.17 and 7.699 milliseconds, respectively; $r = 46$ is more than $2\times$ faster than $r = 2$ (Figure 8(b)).

We also make the observation that for relatively small-sized data blocks, the optimal radix is many times smaller than $\sqrt{P}$. For example, see the graphs above the dashed line. One important thing to note is that the commonly used radix of 2 does not yield peak performance in these cases, and the optimal radix is typically close to $\sqrt{P}$. Overall, this trend can be explained by the fact that small-sized data blocks benefit more from the latency benefits (achieved with smaller radices) than large data blocks.

B. *Performance analysis of TRA2*

Similar to Section V-A, we fixed P and S while varying radices r_1 (intra-node comm) and r_2 (inter-node comm) in each experiment, as shown in Figure 9. From the results, we observe that (1) the intra-node communication follows the rule of *TRA* that works well with $r_1 = \sqrt{Q}$ (Q: process count/node) (latency-bound); (2) the inter-node communication, on the other hand, follows the spread-out like pattern that works well with $r_2 = N$ (N: the number of nodes) (bandwidth-bound).

For most runs, the most optimal r_1 is 8 (=$\sqrt{64}$). However, we also observe that the overall performance of *TRA2* is dominated by inter-node communication This relatively lower cost of the intra-node communication further demonstrates the efficacy of *TRA2*, as these intra-node exchange rounds would otherwise be part of expensive global communication rounds with *TRA*.

Interestingly, the inter-node communication (purple lines) does not yield optimal performance with $r_2 = \sqrt{P}$. Our experiments show that $r_2 = N(P/Q)$ outperforms other radices in most cases. For example, the optimal radix for $P = 2{,}048$ and $S = 256$ is 34 ($2048/64 = 32$) (Figure 9(c)), and it is 128 ($8192/64 = 128$) for $P = 8{,}192$ and $S = 1{,}024$ (Figure 9(l)). This is expected since the inter-node rounds of *TRA2* perform aggregated data exchanges only at the node level – with all processes within a node exchanging data with all processes on some other node. Since processes on a node share bandwidth, it is easier to saturate the network– making it bandwidth-bound. For inter-node communication, $r_2 = P/Q$, which corresponds to the bandwidth-bound, linear time spread-out communication pattern, guarantees a minimum amount of data transmitted and yields the most optimal performance.

C. *Comparison with vendor-optimized all-to-all*

In this section, we compared the performance of *TRA* and *TRA2* against the vendor-optimized `MPI_Alltoall`. We vary S from 16 to 4,096 bytes and P from 512 to 16,384. We use the optimal radix of $r = \sqrt{P}$ for *TRA* and $r_1 = \sqrt{Q}$ (Q: number of processes per node) and $r_2 = N$ (N: number

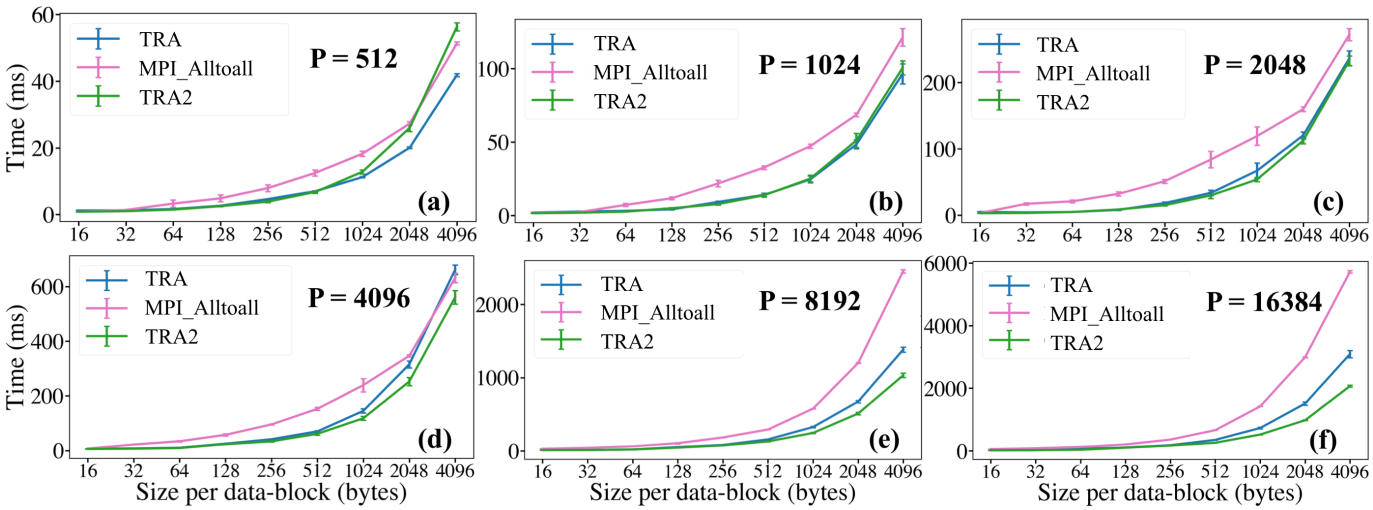

Fig. 10: Performance comparison of TRA with optimal radix ($\sqrt{P}$), TRA2 with optimal radices ($r1 = Q$ (intra), $r2 = N$ (inter)), and `MPI_Alltoall` with varying process count (P), size per data-block (S) (on Theta).

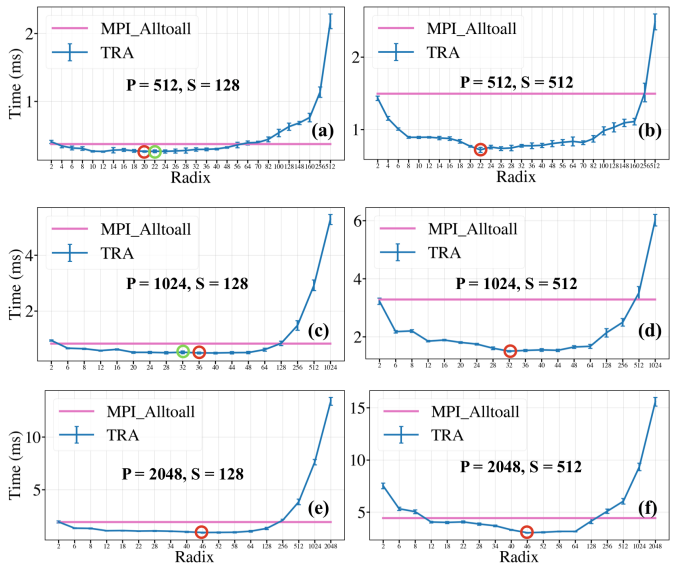

Fig. 11: Performance analysis of *TRA* on Polaris with varying process count (P), size per data-block (S), and radix (r). A red circle highlights the best r with optimal performance in each figure, while a green circle indicates $r = \sqrt{P}$ when it is not equal to the best radix. ($\lceil\sqrt{512}\rceil = 22$, $\sqrt{1024} = 32$, $\lceil\sqrt{2048}\rceil = 46$).

of nodes) for *TRA2*. The results are shown in Figure 10. We can observe that both *TRA* and *TRA2* consistently outperform `MPI_Alltoall` (pink lines) for almost all P and S. For example, when $P = 2{,}048$, *TRA* and *TRA2* are 3.98x and 5.02x faster than `MPI_Alltoall` at $S = 32$ bytes (Figure 10(c)), respectively. We also observe that *TRA2* (green lines) beats *TRA* (blue lines) for almost all Ps and Ss. These experiments demonstrate our algorithms' efficacy and strategy to pick the optimal radices for both *TRA* and *TRA2*. Any application that relies on all-to-all communication is bound to benefit from our approach.

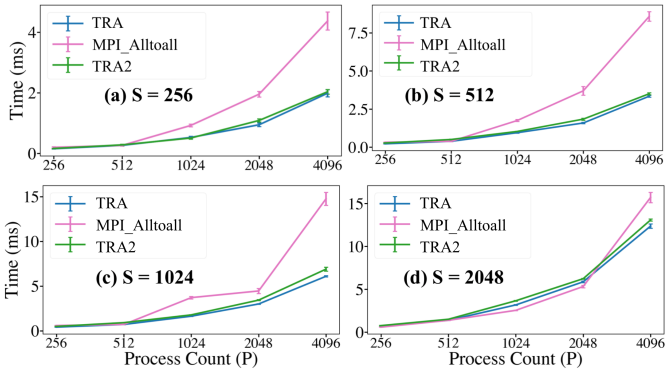

Fig. 12: Performance comparison of *TRA* with optimal radix ($r = \sqrt{P}$), *TRA2* with optimal radices ($r1 = \sqrt{Q}$ (intra), $r2 = N$ (inter)), and `MPI_Alltoall` on Polaris.

D. *Evaluation on Polaris*

To show the overall generalizability of our approach, we also ran all experiments on the Polaris supercomputer at ANL. Due to space limitations, we report the performance of a handful of experiments that demonstrated key trends. In Figure 11, we show the performance of *TRA* as a function of the radix. As expected, we observe an optimal performance at $\sqrt{P}$ for most cases and also see improved performance compared to the vendor-optimized implementation of `MPI_Alltoall`. In Figure 12, we compare the performance of *TRA* and *TRA2* against `MPI_Alltoall`, in a weak scaling mode. Here, also both *TRA* and *TRA2* outperform `MPI_Alltoall`. However, unlike Theta, the performance difference between *TRA* and *TRA2* is relatively less pronounced.

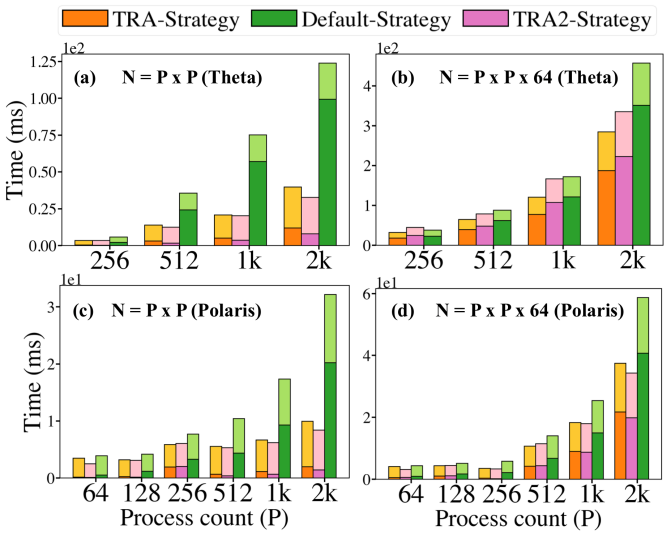

Fig. 13: Comparison results with different sizes of complex DFT (N). The darker part (at the bottom) is the all-to-all time and the lighter part (at the top) is the compute time.

VI. Application: Fast Fourier Transform (FFT)

In this section, we apply and evaluate both the *TRA* and *TRA2* algorithms using a one-dimensional parallel FFT (Fast Fourier transform) application built on top of the FFTW-3 [34] library. Three-dimensional fast Fourier transform computation (FFT) is an important component of many scientific applications [35] ranging from fluid dynamics to astrophysics and molecular dynamics. FFTW [15] is a widely used open-source library that computes the discrete Fourier transform (DFT) in parallel using MPI. The main FFT algorithm used in FFTW is the Cooley-Tukey algorithm [36], which is a classic DFT computing algorithm that reduces the time complexity from $O(N^2)$ to $O(NlogN)$ for N-sized data.

FFTW relies extensively on all-to-all exchanges to perform matrix transpose operations. The data type of a complex DFT in FFTW-3 is `fftw_complex`, which contains two doubles, corresponding to the real and imaginary parts of a complex number. This application performs three matrix transposes, which account for a significant portion of the runtime. The FFTW-3 library's default strategy is to use `MPI_Alltoall` to perform the transpose tasks – referred to as *Default-Strategy* (green bars in Figure 13). To compare against the *Default-Strategy*, we implemented all-to-all using *TRA* and *TRA2* with their optimal radices – referred to as *TRA-Strategy* and *TRA2-Strategy* respectively (orange and pink bars in Figure 13).

Figure 13 illustrates the comparison results of the three strategies with varying process count (P) and size of complex DFT (N) on both Theta and Polaris. In each sub-figure, we plot both the overhead of the sum of three transpose operations (bottom solid parts of stacked bars) and the entire application (whole bars). In Figure 13(a)(c), with $N = P^2$, each process is assigned a P complex DFT, with each process exchanging one complex DFT with every other process (2 doubles = 16 bytes). Similarly, each process exchanges 1,024 bytes with every other process in Figure 13(b)(d), where $N = P^2 \times 64$. We make the observation that, on both Polaris and Theta, both *TRA* and *TRA2* outperform the *Default-Strategy* at all scales. Furthermore, on Polaris, *TRA2* outperforms *TRA* in 10 out of 12 process counts, and on Theta in 4 out of the 8 runs.

VII. Limitations

This paper makes contributions to better understanding (and reason about) the entire range of (configurable) all-to-all communication patterns and correspondingly demonstrates a span of achievable performances. This work is also the first step in developing an automated method to estimate the optimal radix for both *TRA* and *TRA2*. However, the work has one shortcoming, while we can deduce an optimal radix ($r = \sqrt{P}$ for *TRA* and $r_1 = \sqrt{Q}$ and $r_2 = P/Q$ for *TRA2*), based on the formulas derived in Section III-A, for K (total communication rounds) and D (total number of transmitted blocks) parameterized via P (process count) and r (radix); the performance of all-to-all also naturally depends on the workload which must also factor in the size of a data-block (S) – which currently does not figure in the formulas.

While we can create a rough bound for performance based on S for *TRA*. For example, from Figure 8, we observe that for larger block sizes (≥ 128 bytes) optimal radix is $\approx \sqrt{P}$, and for smaller block sizes the optimal radix is typically $\leq \sqrt{P}$. Such bounds are difficult to obtain for *TRA2*. Moreover, even for *TRA*, this bound may shift on other HPC systems (as it does for Polaris). It is therefore essential to construct a data-driven performance model [37], [38] that is guided by both the theoretical underpinnings of this work along with a wide range of empirical exploration involving multiple HPC systems and the entire parameter space including S, P, r, and Q. Machine learning could be one of the potential techniques for building this model. The performance model could potentially also present a more robust way of choosing between *TRA* and *TRA2*, as automatically selecting the correct algorithm continues to be a challenging task (as seen in the application section). Therefore, to obtain a more robust performance model, this high-dimensional parametric space needs to be rigorously explored.

VIII. Conclusions

MPI collective operations are commonly used in large distributed applications and are a principal cause of scalability bottlenecks [11]. This paper focuses on improving the performance of uniform all-to-all communication (`MPI_Alltoall`). In the process, we developed parameterized formulas to compute the total number of communication rounds (K) and data blocks transferred (D) as a function of the radix (r) and process count (P) and used the formulas to derive the optimal radix for both *TRA* and *TRA2*. We developed formulas to calculate the total number of communication rounds (K) and data-blocks transferred (D) for various configurations, which can be used to pre-analyze communication performance for any given P, S, and r. We

demonstrated a range of configurations where our algorithms *TRA* and *TRA2* outperformed the vendor-optimized implementations on two HPC systems. We found that radix $r = \lceil \sqrt{P} \rceil$ is the most effective configuration in most cases for the *TRA* algorithm. Our techniques can potentially improve a range of applications that rely on uniform all-to-all communication. Our open-source implementations can be easily adopted by applications and vendors, providing consistent parameters as `MPI_Alltoall`. Our improvements should also be directly applicable, with some caveats, to `MPI_Alltoallv` where message sizes are permitted to vary.

IX. Acknowledgement

This work was partly funded by NSF Collaborative Research Awards 2401274 and 2221812, and NSF PPoSS Planning and Large awards 2217036 and 2316157. We thank the ALCF's Director's Discretionary (DD) program for providing us with compute hours to run our experiments on the Polaris and Theta supercomputers located at the Argonne National Laboratory.

References

[1] N. Netterville, K. Fan, S. Kumar, and T. Gilray, "A visual guide to mpi all-to-all," in *2022 IEEE 29th International Conference on High Performance Computing, Data and Analytics Workshop (HiPCW)*. IEEE, 2022, pp. 20–27.

[2] A. Sewell, K. Fan, A. R. Shovon, L. Dyken, S. Kumar, and S. Petruzza, "Bruck algorithm performance analysis for multi-gpu all-to-all communication," in *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region*, 2024, pp. 127–133.

[3] Q. Kang, R. Ross, R. Latham, S. Lee, A. Agrawal, A. Choudhary, and W.-k. Liao, "Improving all-to-many personalized communication in two-phase i/o," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–13.

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

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

[6] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine *et al.*, "Open mpi: Goals, concept, and design of a next generation mpi implementation," in *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*. Springer, 2004, pp. 97–104.

[7] T. Kielmann, H. E. Bal, and K. Verstoep, "Fast measurement of logp parameters for message passing platforms," in *International Parallel and Distributed Processing Symposium*. Springer, 2000, pp. 1176–1183.

[8] T. Worsch, R. Reussner, and W. Augustin, "On benchmarking collective mpi operations," in *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*. Springer, 2002, pp. 271–279.

[9] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of collective communication operations in mpich," *The International Journal of High Performance Computing Applications*, vol. 19, no. 1, pp. 49–66, 2005.

[10] D. E. Bernholdt, S. Boehm, G. Bosilca, M. Gorentla Venkata, R. E. Grant, T. Naughton, H. P. Pritchard, M. Schulz, and G. R. Vallee, "A survey of mpi usage in the us exascale computing project," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 3, p. e4851, 2020.

[11] C. Renggli, S. Ashkboos, M. Aghagolzadeh, D. Alistarh, and T. Hoefler, "Sparcml: High-performance sparse communication for machine learning," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019.

[12] S. Kumar and T. Gilray, "Distributed relational algebra at scale," in *International Conference on High Performance Computing, Data, and Analytics (HiPC)*. IEEE, 2019.

[13] D. Taru, N. Islam, G. Zheng, R. Kalidas, A. Langer, and M. Garzaran, "High radix collective algorithms," *Proceedings of EuroMPI 2021*, 2021.

[14] "Mpi on Theta," https://www.alcf.anl.gov/support-center/theta/mpi-theta.

[15] http://fftw.org/, FFTW Home Page.

[16] J. Bruck, C.-T. Ho, S. Kipnis, E. Upfal, and D. Weathersby, "Efficient algorithms for all-to-all communications in multiport message-passing systems," *IEEE Transactions on parallel and distributed systems*, vol. 8, no. 11, pp. 1143–1156, 1997.

[17] A. Faraj and X. Yuan, "Automatic generation and tuning of mpi collective communication routines," in *Proceedings of the 19th annual international conference on Supercomputing*, 2005, pp. 393–402.

[18] M. G. Venkata, R. L. Graham, J. Ladd, and P. Shamis, "Exploring the all-to-all collective optimization space with connectx core-direct," in *2012 41st International Conference on Parallel Processing*. IEEE, 2012.

[19] J. L. Träff, A. Rougier, and S. Hunold, "Implementing a classic: Zero-copy all-to-all communication with mpi datatypes," in *Proceedings of the 28th ACM international conference on Supercomputing*, 2014.

[20] K. S. Khorassani, C.-H. Chu, Q. G. Anthony, H. Subramoni, and D. K. Panda, "Adaptive and hierarchical large message all-to-all communication algorithms for large-scale dense gpu systems," in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 2021, pp. 113–122.

[21] Q. Zhou, P. Kousha, Q. Anthony, K. Shafie Khorassani, A. Shafi, H. Subramoni, and D. K. Panda, "Accelerating mpi all-to-all communication with online compression on modern gpu clusters," in *International Conference on High Performance Computing*. Springer, 2022, pp. 3–25.

[22] C. Xu, M. G. Venkata, R. L. Graham, Y. Wang, Z. Liu, and W. Yu, "Sloavx: Scalable logarithmic alltoallv algorithm for hierarchical multicore systems," in *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*. IEEE, 2013, pp. 369–376.

[23] K. Fan, T. Gilray, V. Pascucci, X. Huang, K. Micinski, and S. Kumar, "Optimizing the bruck algorithm for non-uniform all-to-all communication," in *Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing*, 2022, pp. 172–184.

[24] M. Wilkins, H. Wang, P. Liu, B. Pham, Y. Guo, R. Thakur, P. Dinda, and N. Hardavellas, "Generalized collective algorithms for the exascale era," in *2023 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2023, pp. 60–71.

[25] A. Jocksch, M. Kraushaar, and D. Daverio, "Optimized all-to-all communication on multicore architectures applied to ffts with pencil decomposition," *Concurrency and Computation: Practice and Experience*, vol. 31, no. 16, p. e4964, 2019.

[26] A. Jocksch, N. Ohana, E. Lanti, V. Karakasis, and L. Villard, "Optimised allgatherv, reduce_scatter and allreduce communication in message-passing systems," *arXiv preprint arXiv:2006.13112*, 2020.

[27] A. Gainaru, R. L. Graham, A. Polyakov, and G. Shainer, "Using infiniband hardware gather-scatter capabilities to optimize mpi all-to-all," in *Proceedings of the 23rd European MPI Users' Meeting*, 2016.

[28] A. Bienz, S. Gautam, and A. Kharel, "A locality-aware bruck allgather," *arXiv preprint arXiv:2206.03564*, 2022.

[29] R. Graham, M. G. Venkata, J. Ladd, P. Shamis, I. Rabinovitz, V. Filipov, and G. Shainer, "Cheetah: A framework for scalable hierarchical collective operations," in *2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 2011, pp. 73–83.

[30] J. Zhang, J. Zhai, W. Chen, and W. Zheng, "Process mapping for mpi collective communications," in *European Conference on Parallel Processing*. Springer, 2009, pp. 81–92.

[31] J. Pješivac-Grbović, T. Angskun, G. Bosilca, G. E. Fagg, E. Gabriel, and J. J. Dongarra, "Performance analysis of mpi collective operations," *Cluster Computing*, vol. 10, no. 2, pp. 127–143, 2007.

[32] "Theta," https://docs.alcf.anl.gov/theta/.

[33] "Polaris machine overview," https://www.alcf.anl.gov/polaris.

[34] M. Frigo and S. G. Johnson, "The design and implementation of fftw3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005.

[35] L. Dalcin, M. Mortensen, and D. E. Keyes, "Fast parallel multidimensional fft using advanced mpi," *Journal of Parallel and Distributed Computing*, vol. 128, pp. 137–150, 2019.

[36] J. Cooley and J. Tukey, "An algorithm for the machine computation of the complex fourier series, in mathematics of computation." *April*, 1965.

[37] M. Wilkins, Y. Guo, R. Thakur, N. Hardavellas, P. Dinda, and M. Si, "A fact-based approach: Making machine learning collective autotuning feasible on exascale systems," in *2021 Workshop on Exascale MPI (ExaMPI)*. IEEE, 2021, pp. 36–45.

[38] S. Kumar, A. Saha, V. Vishwanath, P. Carns, J. A. Schmidt, G. Scorzelli, H. Kolla, R. Grout, R. Latham, R. Ross *et al.*, "Characterization and modeling of pidx parallel i/o for performance optimization," in *Proceedings of the international conference on high performance computing, networking, storage and analysis*, 2013, pp. 1–12.