A Flexible Forwarding Scheme to Improve Latency-Bound Irregular P2P Communication in MPI

Oguz Selvitopi, Nabil Abubaker, Erkin Aydin, and Cevdet Aykanat, *Member, IEEE*

***Abstract*—We propose an algorithm to efficiently perform latency-bound communication scenarios that consist of many small messages. In these parallel scenarios, processes typically pass around a lot of small-sized messages of a few KBs of size. Performing communication operations with P2P MPI routines or collective MPI routines (including neighborhood collectives) in such scenarios may not always yield the optimal results and may not resolve the latency bottleneck. To this end, we develop a regular structure called virtual process topology (VPT) on which the messages can be communicated in a structured and controlled manner. Using parameters of this topology, one can tune the rate of aggression in tackling the latency costs. We demonstrate that our communication algorithm is preferable to MPI P2P and collective routines for latency-bound communication and it can easily be adapted only by replacing calls to MPI routines in a parallel application. We show how to adapt existing topology-aware mapping heuristics to address the volume overhead due to communicating messages on the VPT. Moreover, we propose a novel swap-based mapping heuristic to address this overhead by optimizing the maximum volume handled by a process. Experiments on synthetic communication graphs as well as real-world applications such as parallel Canonical Polyadic sparse tensor decomposition and parallel sparse matrix-dense matrix multiplication show that our approach is a powerful way of overcoming the bottlenecks posed by sparse and latency-bound irregular communication.**

***Index Terms*—Irregular communication, communication cost, latency, virtual process topology (VPT), communication algorithm, mapping heuristic.**

I. Introduction

THE time a distributed-memory parallel application spends in communication can be detrimental to its scalability especially if the communication operations require comparable or more time than the computations themselves. Among several factors that affect the communication time of a parallel application, the two most commonly used factors for modeling the time aim to encapsulate the latency and bandwidth aspects, where the former relates to the *number* of messages and latter relates to the *sizes* of these messages.

The communication operations may exhibit a certain degree of regularity, which one can take advantage of by realizing them via MPI collectives [2], [3], [4], [5], [6], [7]. For example, in stencil applications, which are utilized in numerical solution of partial differential equations, a process communicates with a well-defined set of a few other processes, or in the case of global collectives each process participates in gathering, scattering, reducing, etc. of data they possess. In the presence of a high variation among the communicating processes, the communication operations become irregular and a certain subset of processes communicate with relatively more processes compared to the others. Consider a scenario where a single process communicates with more than half of the processes in the system via point-to-point (P2P) messages, while the remaining processes communicate with only a few processes, and messages are small, typically no more than a few KBs (Fig. 1 illustrates such an example regarding a sparse-matrix vector multiplication on 256 processes). In such a scenario, this single process has the potential of rendering the whole application unscalable. Moreover, using collectives under similar scenarios may not always prove feasible in terms of efficiency. The goal of this work is to improve the performance of such scenarios, where

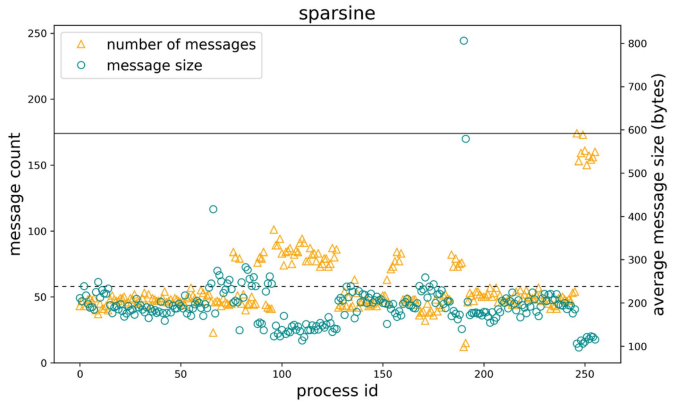


Fig. 1. Message counts and average message size of 256 processes during sparse matrix-vector multiplication (matrix `sparsine`). The triangle markers are for message counts and the circle markers are for average message sizes. The straight horizontal line is the maximum message count and the dashed line is the average message count.

Received 21 February 2025; revised 1 December 2025; accepted 17 February 2026. Date of publication 5 March 2026; date of current version 17 April 2026. The work of Oguz Selvitopi was supported by the Advanced Scientific Computing Research (ASCR) Program of the Department of Energy Office of Science under Grant DE-AC02-05CH11231. The work of Nabil Abubaker and Cevdet Aykanat was supported in part by the Scientific and Technological Research Council of Turkiye (TUBITAK) under Grant 121E391. An earlier version of this paper was presented in part at the SC'19 [DOI: 10.1145/3295500.3356187]. Recommended for acceptance by M. Si. *(Corresponding author: Cevdet Aykanat.)*

Oguz Selvitopi is with Applied Mathematics and Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94703 USA (e-mail: roselvitopi@lbl.gov).

Nabil Abubaker was with the Department of Computer Engineering, Bilkent University, 06800 Ankara, Türkiye. He is now with Forschungszentrum Jülich GmbH, 52428 Jülich, Germany.

Erkin Aydin and Cevdet Aykanat are with the Department of Computer Engineering, Bilkent University, 06800 Ankara, Türkiye (e-mail: aykanat@cs.bilkent.edu.tr).

Digital Object Identifier 10.1109/TPDS.2026.3669506

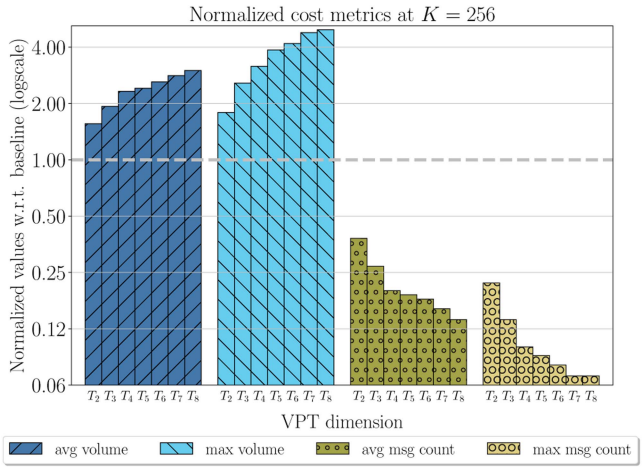


Fig. 2. Four communication metrics of different VPT dimensions normalized with respect to the baseline of communicating messages directly without any VPT on 256 processes. A value $y > 1$ in the figure indicates the baseline is y times better, whereas a value $y < 1$ indicates using VPT is $1/y$ times better.

the communication patterns are sparse and irregular, and there is a high variance among the number of processes each process communicates with.

We propose a structured way of performing sparse and irregular communication operations by organizing processes into a regular structure which we call virtual process topology (VPT). By utilizing this structure, we restrict the processes that can directly communicate with each other and impose a regular communication pattern onto otherwise irregular communication operations. Our ideas for forming the VPT are inspired by the k-ary n-cube networks and there are certain differences such as our VPT is on the software level and oblivious to the underlying networking, and we relax the neighborhood definition slightly to allow more flexibility (for more details about differences, see [1]). We propose a novel store-and-forward (`STFW`) algorithm to realize the communication operations for a given set of processes (along with the data they are to send) and the VPT these processes are organized into. Our methodology can be implemented as an alternative communication pattern in an MPI distribution or the application developers can replace their MPI P2P calls or collective calls (such as `MPI_Alltoallv`) with our communication algorithm. Specifically, our algorithm's function signature is similar to that of MPI collective routine `MPI_Alltoallv`. There are a few additional parameters in ours regarding the structure of the VPT.

The organization of processes into the proposed VPT and performing communications on it enable a trade-off between the maximum message count (which is related to the total latency cost) and the communication volume (which is related to the bandwidth cost). An important parameter in forming a VPT for a given set of processes is its *dimension*. A low-dimensional VPT results in a higher maximum message count and lower communication volume than a high-dimensional VPT. Hence, by varying the VPT dimension, we are able to control the trade-off between the latency and bandwidth costs. We illustrate this in Fig. 2, where we plot the normalized values of four communication cost metrics of the proposed VPT against the baseline of communicating messages through plain P2P messages in the communication phase of sparse-matrix vector multiplication on 256 processes. The utilization of VPT results in smaller average and maximum message counts (the corresponding bars are below 1.0) and larger average and maximum message sizes (the corresponding bars are above 1.0). By varying the VPT dimensions from a low (i.e., T_2) to high (i.e., T_8), we can adjust the rate of trade-off between the latency and bandwidth components to optimize communication. We also describe how to address the increase in the total volume as well as the maximum volume due to the `STFW` scheme proposed to perform the communication on VPT.

Our contributions are summarized as follows:

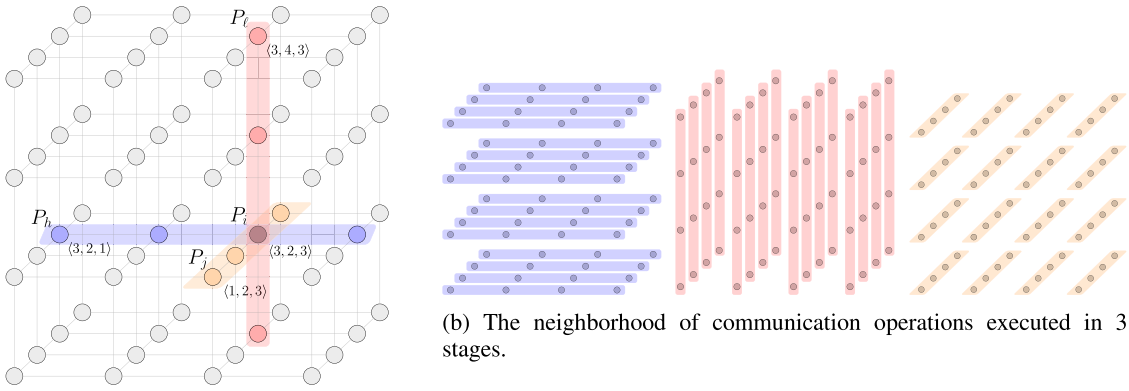
- We propose a novel virtual process topology to regularize sparse and irregular communication operations. We give process neighborhood definitions in this VPT and discuss how to form VPTs of various dimensions.
- We propose a store-and-forward algorithm to realize the communication operations on a given VPT. We describe in detail how messages are communicated in the VPT.
- We investigate various constructive one-to-one mapping algorithms to map the processes onto VPT for the purpose of reducing the increase in the total communication volume due to the `STFW` communication scheme.
- We propose an iterative-improvement heuristic (based on a Kernighan-Lin-based framework [8]) to refine and improve a given one-to-one mapping in terms of maximum communication volume.
- We validate the proposed VPT and communication algorithm against MPI collective communication and P2P routines in a controlled experimental setting where we vary skew in both the communication structure and the message sizes.
- We also validate the performance of the proposed communication algorithms for sparse-kernel operations such as tensor decomposition and sparse matrix–dense matrix multiplication (SpMM) on actual tensor and sparse matrix datasets.
- We provide an efficient and scalable open-source implementation of the proposed VPT and `STFW` scheme in a library called VPTComm.[1]

The rest of this paper is organized as follows. We introduce the virtual process topology and its related definitions in Section II. In Section III, we describe our `STFW` algorithm to perform communication operations on a given VPT. For mapping processes onto VPT, we describe the adaptation of topology-aware constructive mapping heuristics in Section IV-A and the proposed KL-based improvement heuristic in Section IV-B. The proposed methodology is evaluated in Section V. We give our related work in Section VI and conclude in Section VII.

II. Virtual Process Topology

Our focus is on a distributed parallel processing setting where there are K processes, denoted with $\mathcal{P} = \{P_1, P_2, \ldots, P_K\}$,

[1] https://github.com/nfabubaker/VPTComm



(a) A VPT $T_3(4,4,4)$.

(b) The neighborhood of communication operations executed in 3 stages.

Fig. 3. 64 processes organized into a $4 \times 4 \times 4$ virtual process topology and neighborhood information.

TABLE I
SUMMARY OF NOTATIONS

Notation	Description
STFW	Store-and-forward
VPT	Virtual Process Topology
K	Number of processes
$\mathcal{P}$	The set $\{P_1, P_2, \ldots, P_K\}$ of processes
$T_n(k_1, k_2, \ldots, k_n)$	n-dimensional VPT of size $k_1 \times k_2 \times \ldots \times k_n$
k_d	The size of dimension d
$\langle P_i^n, P_i^{n-1}, \ldots, P_i^1\rangle$	Process P_i defined for VPT T_n
P_i^d	The coordinate of P_i in dimension d
$\nu(P_i, d)$	Neighbors of P_i in dimension d
m_{ij}	The original data that needs to be sent from P_i to P_j
$SendSet(P_i)$	The set of processes that P_i needs to send data
M_{ij}	A message from P_i to P_j on the VPT, possibly containing multiple submessages
$(P_\ell, m_{h\ell})$	A submessage on the VPT with P_ℓ being the destination process
fwbuf	A buffer to hold the submessages
$FwSet(m_{ij})$	The set of intermediate processes that forward m_{ij} on the VPT

which communicate with each other via message passing. We consider a communication scenario, in which each process has a set of messages that it needs to send to a subset of processes, denoted with $SendSet(P_i) \subseteq \mathcal{P}$. The message that needs to be sent from P_i to P_j is denoted with m_{ij}. This is a very common scenario prevalent in many parallel applications. We assume K is a power of 2, however, our methodology and algorithms can easily be extended where this is not the case. We use h, i, j, and ℓ for process indices (k is reserved for other following definitions). A summary of notations is presented in Table I.

A VPT organizes K processes into a special structure and is characterized by its dimension, the sizes of these dimensions, and the process neighborhood definition. An n-dimensional VPT is denoted by $T_n(k_1, k_2, \ldots, k_n)$, where dimension $1 \leq d \leq n$ is of size k_d and $K = k_1 \times k_2 \times \ldots \times k_n$. We use T_n and omit the parentheses in the notation when the dimension sizes are irrelevant to the subject discussions. For dimension indices, we use c and d. Each process P_i in the VPT is identified by a vector $\langle P_i^n, P_i^{n-1}, \ldots, P_i^1\rangle$ of n coordinates, where P_i^d is an integer with radix k_d, i.e., $P_i^d \in \{1, 2, \ldots, k_d\}$. P_i and P_j have the same coordinate in dimension d if $P_i^d = P_j^d$ and they are said to be neighbors if they differ only in a single coordinate. The coordinate they differ in is the dimension they are neighbors in. Hence, P_i and P_j are neighbors in dimension d if $P_i^d \neq P_j^d$ and $P_i^c = P_j^c$ for $1 \leq c \neq d \leq n$. In dimension d, there are a total of K/k_d process groups, each of which contains k_d processes. Hence, each process has $k_d - 1$ neighbors in dimension d and the processes in the same group differ only in the dth coordinate. We use the function $\nu(P_i, d)$ to denote the neighbors of P_i in dimension d:

$$\nu(P_i, d) = \{P_j : P_i^c = P_j^c \wedge P_i^d \neq P_j^d \text{ for } 1 \leq c \neq d \leq n\}. \quad (1)$$

Fig. 3(a) illustrates 64 processes organized into a three-dimensional VPT $T_3(4,4,4)$. The neighbors of $P_i = \langle 3,2,3\rangle$ in each dimension are illustrated with a different color. The processes $P_h = \langle 3,2,1\rangle$, $P_j = \langle 1,2,3\rangle$, and $P_\ell = \langle 3,4,3\rangle$ are neighbors of P_i in dimensions three, one, and two, respectively. The figure has links between processes shown by faded lines to make the structure more clear. These links do not indicate the actual neighborhood information. The neighbor processes in each dimension are illustrated in Fig. 3(b).

The way we organize the processes for communication can be considered to be similar to the topology of the k-ary n-cube networks. Our ideas are indeed motivated by the principles governing these networks, however, there are certain differences: (i) the proposed VPT is virtual and on the software level while the networking is on the hardware level, (ii) the neighborhood definition in the VPT is relaxed to include more neighbor processes at each dimension, and (iii) while in k-ary n-cube networks the size of each dimension is the same and equal to k, in the proposed VPT the sizes of the dimensions may differ. For a more detailed comparison, we refer the reader to the earlier work [1].

A. Communication on VPT

A straightforward and common approach is to assume no structure in the process topology and allow each process to communicate with each other, i.e., each P_i simply sends a P2P message to the processes in $SendSet(P_i)$. We address the problem of improving the communication performance of

this scenario by assuming the described VPT. As opposed to the straightforward approach, using a VPT T_n with $n > 1$ disables the direct communication between certain processes and necessitates a methodology based on storing and forwarding messages. In other words, some processes need to communicate *indirectly* with the help of the processes they can directly communicate with. Such an approach reduces the number of messages communicated in the system at the expense of increasing message sizes. We propose an algorithm to perform the communication between processes under the described process topology in the next section.

The end goal of using a VPT is to provide a flexible medium where one can achieve a trade-off between the total latency (i.e., message count) and the bandwidth (i.e., the amount of data communicated) costs by controlling the dimension of the topology. In the proposed `STFW` scheme, for a fixed number of processes, increasing the VPT dimension increases the number of times messages get forwarded, whereas it will result in smaller dimension sizes leading to fewer messages communicated at each stage. Hence, a low-dimensional VPT results in higher total latency and lower bandwidth cost compared to a high-dimensional VPT.

We consider this as a black-box operation called by each process, which simply provides their data to be sent along with the VPT. Instead of using a pair of primitives such as `MPI_Send`/`MPI_Recv` or general collectives such as `MPI_Alltoallv`, each process passes their data and processes they are to communicate to a procedure, which then handles the communication by taking the process topology into account. The described topology actually encompasses the case where each process is allowed to directly communicate with any other process. This is the case with T_1, i.e., there is a single dimension and each process is neighbor to all other processes. In that sense, our algorithm generalizes how processes can communicate in a structured manner and on the extreme end where there is no structure, it corresponds to being able to directly send out P2P messages to any process.

III. A Store-and-Forward Algorithm

In an n-dimensional VPT T_n, the communication between processes is executed in n stages. The communication operations for a process P_i in stage $1 \leq d \leq n$ start after P_i receives all its messages from the previous communication stages. Without loss of generality, we assume that the communications related to the first dimension are executed in the first stage, the ones related to the second dimension are executed in the second stage, etc. We restrict the processes that can communicate with each other in each stage. In stage d, each process P_i is allowed to communicate only with its $k_d - 1$ neighbors in dimension d, i.e., the processes in $\nu(P_i, d)$. P_i may or may not communicate with these neighbors depending on what it has to send in its buffers. Since there may be processes in $SendSet(P_i)$ that are not neighbors of P_i in any dimension, to send data to such processes P_i needs the help of the other processes. This necessitates a *store-and-forward* scheme, in which the messages that need be communicated between non-neighbor processes are stored and forwarded by a well-defined set of intermediate processes.

Before describing the complete algorithm, we first focus on how a single message is communicated between two processes. Consider a message m_{ij} with its source P_i and destination P_j. Depending on where these two processes are in the VPT, m_{ij} may need to visit multiple hops before reaching P_j. To send m_{ij}, P_i checks its coordinate in the first dimension, P_i^1, and if it is different than P_j^1, it forwards this message to one of its neighbors in the first dimension, which is the process with the coordinates $\langle P_i^n, \ldots, P_i^2, P_j^1 \rangle$. Otherwise, i.e., if $P_i^1 = P_j^1$, P_i keeps the message because it does not need to communicate it in this stage. Let P_ℓ be the process that has m_{ij} after the communication in the first stage completes (which is either P_i or $\langle P_i^n, \ldots, P_i^2, P_j^1 \rangle$). P_ℓ then repeats the same process for $d = 2$, and either forwards or stores m_{ij}. This is repeated until m_{ij} reaches its destination. Hence, at stage d, the process that has m_{ij}, P_ℓ, has to decide whether to forward m_{ij} by comparing its dth coordinate with the dth coordinate of the destination process:

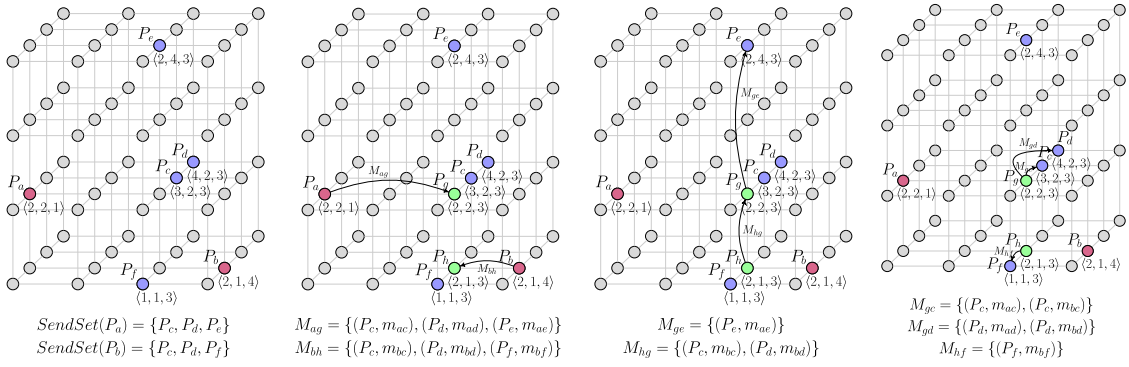
$$\text{forward } m_{ij} \text{ to } \langle P_\ell^n, \ldots, P_\ell^{d+1}, P_j^d, P_\ell^{d-1}, \ldots, P_\ell^1 \rangle \text{ if } P_\ell^d \neq P_j^d,$$

$$\text{store } m_{ij} \text{ if } P_\ell^d = P_j^d.$$

In other words, if P_ℓ and P_j differ in coordinate d, P_ℓ forwards m_{ij} to its neighbor in dimension d whose dth coordinate is the same with that of P_j. Otherwise, it does not communicate the message in this stage. By this logic, it can be easily seen that m_{ij} is forwarded in stage d if $P_i^d \neq P_j^d$, and the number of times it gets forwarded is equal to $|\{d : P_i^d \neq P_j^d\}|$, i.e., the number of coordinates they differ in, which is the Hamming distance. This process of communicating a message in the VPT can be considered to be similar to the e-cube routing for hypercubes [9], and more generally to dimension-ordered deterministic routing.

Consider a process P_i and its $SendSet(P_i)$. Let the coordinates of the processes in $SendSet(P_i)$ differ from those of P_i at and after dth coordinate, i.e., $\{P_\ell \in SendSet(P_i) : P_i^n \neq P_\ell^n, \ldots, P_i^d \neq P_\ell^d, P_i^{d-1} = P_\ell^{d-1}, \ldots, P_i^1 = P_\ell^1\}$. All the relevant messages in this scenario have to be first communicated to some neighbor of P_i in dimension d, say P_j, in order to reach their destination. Hence, the communication between P_i and P_j is actually a message that contains a list of smaller messages, which we refer to as *submessages*. Each submessage is simply a two-tuple that consists of the id of the destination process and the message destined for that process. Note that there is a *single* actual message that is communicated between P_i and P_j, but it contains a number of submessages that will be sorted out by P_j. In fact, P_j may also receive messages from its other neighbors in dimension d, which may inherently contain other submessages, and it may also possess submessages from the messages that are received in previous communication stages. To make a distinction between a message and a submessage, we denote the direct message between P_i and P_j with M_{ij}, and the submessage with source P_i and destination P_j as (P_j, m_{ij}). The submessages are always contained within messages, and these messages are communicated between neighbors in distinct stages of the communication.

Fig. 4 illustrates two processes $P_a = \langle 2, 2, 1 \rangle$ and $P_b = \langle 2, 1, 4 \rangle$ that need to send data to $SendSet(P_a) = \{P_c, P_d, P_e\}$ and $SendSet(P_b) = \{P_c, P_d, P_f\}$, respectively. Both processes need to send their data with the help of their neighbors



(a) initial state (b) stage 1 (c) stage 2 (d) stage 3

Fig. 4. Three stages of communication in a VPT $T_3(4,4,4)$ for $SendSet(P_a) = \{P_c, P_d, P_e\}$ and $SendSet(P_b) = \{P_c, P_d, P_f\}$. The source and destination processes are respectively illustrated in red and blue. Since P_a and P_b cannot directly communicate with the processes in their $SendSet$s, their messages are forwarded via their neighbors, which are illustrated in green.

in the first communication stage. For P_a this neighbor is $P_g = \langle 2,2,3\rangle$ and for P_b it is $P_h = \langle 2,1,3\rangle$. Observe that the message M_{ag} sent from P_a to P_g consists of three submessages $(P_c, m_{ac}), (P_d, m_{ad})$, and (P_e, m_{ae}). The message M_{bh} sent from P_b to P_h also consists of three submessages $(P_c, m_{bc}), (P_d, m_{bd})$, and (P_f, m_{bf}). After these messages are received by P_g and P_h, they process the submessages in them to determine which stage they will be forwarded in. For P_g, the submessage (P_e, m_{ae}) will be forwarded in the second stage while the submessages (P_c, m_{ac}) and (P_d, m_{ad}) will be forwarded in the third stage. For P_h, the submessages (P_c, m_{bc}) and (P_d, m_{bd}) will be forwarded in the second stage while the submessage (P_f, m_{bf}) will be forwarded in the third stage. Observe that a message sent by a process can contain submessages that it received in earlier stages. For instance, the message M_{gc} sent from P_g to P_c in the third stage consists of submessages that P_g received from P_a in the first stage and P_h in the second stage.

Algorithm 1 presents a high-level description of the proposed STFW communication scheme for a given VPT T_n. The algorithm focuses on the operations as performed by $P_i \in \mathcal{P}$. P_i first initializes its buffers in lines 1-3. The buffer fwbuf$[d][x]$ holds the submessages that will be forwarded in stage d to neighbor of P_i whose coordinate d is equal to x, i.e., $P_j^d = x$. Then, P_i traverses the processes in $SendSet(P_i)$ and concatenates each submessage into the respective buffer (lines 4-6). The first stage that m_{ij} will be communicated is given by the index of the smallest coordinate that P_i and P_j differ in. The communication operations are performed in d stages between lines 7-17. In stage d, P_i communicates with a subset of its neighbors in $\nu(P_i, d)$ and examines the submessages in received messages. Using the buffers corresponding to stage d, P_i sends out a message to each P_j if the respective buffer is not empty (lines 9-12). It then examines the received messages from its neighbors in $\nu(P_i, d)$. For each received message M_{ji}, it examines the submessages in them by finding which communication stage they will be forwarded in and putting them in their respective locations in fwbuf (lines 14-17). For a submessage $(P_\ell, m_{h\ell})$ received in stage d, the stage it will be forwarded is given by the smallest coordinate greater than d that P_i and P_ℓ differ in. The data that belongs to P_i is given by the submessages in the locations fwbuf$[d][P_i^d]$ for $1 \le d \le n$ (lines 18-21).

Algorithm 1: Store-and-Forward.

Input: P_i, $SendSet(P_i)$, $T_n(k_1, k_2, \ldots, k_n)$
Output: List L of messages
1: Let fwbuf be a list of size n
2: **for** $d \leftarrow 1$ to n **do**
3: Let fwbuf$[d]$ be a list of size k_d
4: **end for**
5: **for** $P_j \in SendSet(P_i)$ **do**
6: $d \leftarrow \text{argmin}_{c \le n} P_i^c \neq P_j^c$
7: fwbuf$[d][P_j^d] \leftarrow$ fwbuf$[d][P_j^d] \cup (P_j, m_{ij})$
8: **end for**
9: **for** $d \leftarrow 1$ to n **do**
10: Allocate stbuf to receive messages in this stage
11: **for** $P_j \in \nu(P_i, d)$ **do**
12: **if** fwbuf$[d][P_j^d] \neq \emptyset$ **then**
13: Form M_{ij} from the submessages in fwbuf$[d][P_j^d]$
14: Send M_{ij} to P_j
15: **end if**
16: **end for**
17: Wait for messages from my neighbors
18: **for** $M_{ji} \in$ stbuf **do**
19: **for** $(P_\ell, m_{h\ell}) \in M_{ji}$ **do**
20: $c \leftarrow \text{argmin}_{d < c' \le n} P_i^{c'} \neq P_\ell^{c'}$
21: fwbuf$[c][P_\ell^c] \leftarrow$ fwbuf$[c][P_\ell^c] \cup (P_\ell, m_{h\ell})$
22: **end for**
23: **end for**
24: **end for**
25: $L = \emptyset$
26: **for** $d \leftarrow 1$ to n **do**
27: $L \leftarrow L \cup$ fwbuf$[d][P_i^d]$
28: **end for**
29: **return** L

In Algorithm 1, when two submessages $(P_\ell, m_{i\ell})$ and $(P_\ell, m_{j\ell})$ that originate from distinct processes P_i and P_j but

destined for the same process P_ℓ arrive at an intermediate process P_h, they will be put into the same forward buffer and in the rest of the algorithm they will always be communicated within the same messages. Dual of this case, when two submessages (P_j, m_{ij}) and $(P_\ell, m_{i\ell})$ that originate from the same process P_i but destined for the distinct processes P_j and P_ℓ arrive at an intermediate process P_h, they will be put into different forward buffers and in the rest of the algorithm they will always be communicated within distinct messages.

It is interesting to examine certain cases in the proposed VPT. For a given number of processes K, where K is a power of 2, if we use $n = 1$, Algorithm 1 simply corresponds to each process communicating with each other directly. This is no different than P_i sending a direct P2P message to each process in $SendSet(P_i)$. Since $k_d > 1$ for each dimension, the largest T_n we can get is $n = \lg_2 K$, and $k_d = 2$ for all d. In this case, each process can communicate with exactly one process in each stage. In network terminology, this is equivalent to hypercubes, where each node is connected to exactly one node in each dimension. Note that the order of the stages in performing the communication does not matter in terms of the latency bound. The bound on the maximum message count of a VPT T_n is the same regardless of the order of the stages communication is performed.

For a detailed analysis of the proposed STFW algorithm in terms of communication cost metrics such as maximum message count and volume, buffer requirements, as well as the trade-offs between different formations that can be used to construct the VPT, we refer the reader to our earlier work [1].

IV. Mapping Processes to Virtual Process Topology

The STFW communication scheme described in the previous section favors reducing latency costs at the expense of increasing bandwidth costs by increasing the volume of communicated data compared to directly communicating messages. The main reason behind the increased volume is the indirect communication in which messages may visit multiple processes before reaching their destination. Although this choice of trade-off is the main idea of this work, the increased volume has the potential of becoming a bottleneck especially when high-dimensional VPTs are used and/or when there is a big variance in the sizes of the messages. If this increase is overlooked, it has the potential to render the VPT bandwidth-bound. In this section, we propose a solution to this issue.

We illustrate how the bandwidth cost may change with respect to mapping of processes onto VPT in Fig. 5 in a three-dimensional VPT $T_3(3, 3, 3)$. There are three pairs of communicating processes in the figure where each communicating pair is illustrated with a different color. In Fig. 5(a), the messages between pairs (P_a, P_d) and (P_c, P_e) will be forwarded three times while the message between the pair (P_b, P_f) will be forwarded only once. If the messages between P_a and P_d or P_c and P_e are large, this will incur a lot of extra volume compared to directly communicating these messages. Hence, while mapping processes onto a VPT, we should try to reduce the forwarding

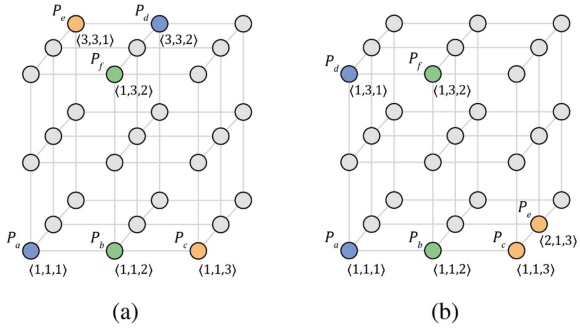


Fig. 5. Two different placement of three pairs of communicating processes on VPT $T_3(3, 3, 3)$. The communicated processes are illustrated with the same color.

overhead especially for large messages by assigning the processes that communicate such messages as close as possible to each other according to the neighborhood definition given in Section II. Fig. 5(b) shows one such mapping where all the communicated messages reach their destination in one hop.

The discussions in the earlier sections assumed each process is associated with a unique set of coordinates in the VPT, and used processes and nodes (i.e., coordinates) in the VPT interchangeably but did not describe how these coordinates are determined. We first define the problem of mapping processes onto VPT nodes (i.e., determining these coordinates) and then describe how to utilize various heuristics to solve this problem. In order to capture the volume of data communicated on a given VPT, we utilize the hop-byte metric [10] as follows:

$$Vol(T_n, \mathcal{M}) = \sum_{m_{ij} \in \mathcal{M}} size(m_{ij}) \cdot dist(P_i, P_j), \qquad (2)$$

where $\mathcal{M} = \{m_{ij} : P_j \in SendSet(P_i)\}$ is the set of original messages to be communicated, $size(m_{ij})$ is the size of message m_{ij} in bytes, and $dist(P_i, P_j)$ is the distance between processes P_i and P_j in the VPT and is given by the number of coordinates they differ in, i.e., $|\{d : P_i^d \neq P_j^d\}|$ for $1 \leq d \leq n$. Hereafter, we make a distinction between the processes and the nodes of a VPT and define a function that maps processes to the nodes of the VPT, and hence assign coordinates to these processes.

We can represent the messages and the VPT with graphs. We represent the messages in $\mathcal{M}$ with directed graph $G_c = (V_c, E_c)$, referred to as the communication graph, where we have a node v_i^c for each P_i and there exists a directed edge from v_i^c to v_j^c with its weight $size(m_{ij})$ for $m_{ij} \in \mathcal{M}$. We use the weight function $w(v_i^c, v_j^c) = size(m_{ij})$ to denote the weight of an edge. We represent a VPT $T_n(k_1, k_2, \ldots, k_n)$ with undirected graph $G_t = (V_t, E_t)$, referred to as the topology graph, where we have a node v_i^t for each node or coordinate in the VPT and there exists an (unweighted) edge between v_i^t and v_j^t if the coordinates corresponding to these two vertices are neighbors in any dimension $1 \leq d \leq n$ (i.e., if they differ in a single coordinate). Then, the problem is to find a mapping $\Pi : V_c \rightarrow V_t$ with the goal of minimizing the volume incurred by communicating the messages according to the STFW algorithm described earlier,

where the volume incurred by a mapping is given by

$$Vol(G_c, G_t, \Pi) = \sum_{(v_i^c, v_j^c) \in E_c} w(v_i^c, v_j^c) \cdot dist(\Pi(v_i^c), \Pi(v_j^c)). \tag{3}$$

Defining these two graphs has the advantage of being able to use any topology-aware mapping heuristic in the literature. Some of the algorithms or tools utilize a specific structured topology, such as mesh, torus, etc. The proposed VPT does not fall under such specific topologies with node-to-node distances different from those of the counterpart topologies. For example, for a fixed number of processes K, $dist(P_i, P_j)$ in a three-dimensional cube mesh (without wrap-around connections) varies between 1 and $3(K^{1/3} - 1)$, whereas it varies between 1 and 3 in three-dimensional VPT $T_3(K^{1/3}, K^{1/3}, K^{1/3})$. We are interested in heuristics that can handle arbitrary topology graphs, which in our case is given by G_t. We next briefly describe the heuristics utilized in this work.

A. Constructive Mapping Strategies

Finding a mapping Π in the above-described problem is an instance of the quadratic assignment problem [11], which is known to be NP-hard [12], [13]. Therefore, heuristics are usually used to find good mappings. The heuristics to obtain a one-to-one mapping can be categorized into two as iterative algorithms and constructive algorithms. Iterative algorithms start with an initial solution and iteratively improve that solution until reaching a stopping criterion. Constructive algorithms, on the other hand, start with an initial one-to-one assignment of two vertices $\Pi(v_1^c) = v_1^t$ and continue for each v_i^c and v_i^t performing the mapping based on a greedy choice that depends on already mapped vertices $\Pi(v_k^c) = v_k^t, 1 \leq k < i$. We consider three existing mapping schemes: *MapSI* [14], *MapMD* [15], and *MapRB* [14]. The first two follow the template described above whereas the last one uses a recursive approach. Constructive algorithms are powerful to be used both alone or to find an initial solution for an iterative algorithm.

MapSI initially chooses the vertex that has the maximum sum of outgoing edge weights, v_1^c, and maps it to a randomly chosen coordinate in VPT, v_1^t. In the ith iteration, this heuristic chooses v_i^c as the vertex with the heaviest edge to already mapped vertices $v_1^c, \ldots, v_{i-1}^c$. v_i^t is chosen independently as the closest vertex to v_{i-1}^t. We abbreviate it as MapSI since v_i^c is chosen considering single (S) heaviest edge and v_i^t is chosen independent (I) of v_i^c. MapSI exists in the associated library LibTopoMap.

MapMD chooses v_i^c in ith iteration as the vertex that has the largest sum of edge weights to already mapped vertices $v_1^c, \ldots, v_{i-1}^c$, i.e., the vertex that maximizes $\sum_{k=1}^{i-1} w(v_k^c, v_i^c)$. In choosing v_i^t, it utilizes the previous mapping information $\Pi(v_1^c), \ldots, \Pi(v_{i-1}^c)$ to select a coordinate that minimizes the increase in volume. Among the available coordinates, MapMD chooses the coordinate that minimizes the incurred volume after selecting v_i^c. In other words, it chooses v_i^t as the coordinate that minimizes $\sum_{k=1}^{i-1}(w(v_k^c, v_i^c) \cdot dist(v_i^t, \Pi(v_k^c) = v_k^t) \mid (v_k^c, v_i^c) \in E_c)$. This is again the hop-byte metric mentioned earlier (3). The letter "M" is used to abbreviate consideration of multiple edges in selecting v_i^c and the letter "D" is used to abbreviate the dependent mapping of v_i^t on v_i^c.

MapRB follows a top-down approach and recursively bipartitions both G_c and G_t with the goal of minimizing the edge cut by assigning heavy edges of G_c to nearby clusters in G_t [14]. MapRB relies on graph partitioning tools to perform the partitioning.

B. Iterative Strategy for Improving a Given Mapping

We propose a swap-based Kernighan-Lin [8] approach for improving a one-to-one mapping possibly produced by a constructive mapping heuristic. The objective of the proposed approach is different than that of the constructive heuristics–which aim to reduce the increase in the total communication volume. To complement the objective of the constructive ones, we select our objective as minimizing the maximum communication volume load, defined as

$$MaxVolLoad = \max_{P_k \in \mathcal{P}} \{VolLoad(P_k)\} = VolLoad(P_b), \tag{4}$$

where P_b is the most heavily-loaded, i.e., the bottleneck, process. The volume load of a process P_k for a given mapping is defined as the sum of its receive and send volume loads:

$$\begin{aligned} VolLoad(P_k) &= SendVol(P_k) + RecvVol(P_k) \\ &= 2 \sum_{\substack{i,j \neq k, m_{ij} \in \mathcal{M}}}^{P_k \in FwSet(m_{ij})} size(m_{ij}) \quad + \sum_{m_{k\ell} \in \mathcal{M}} size(m_{k\ell}) \\ &\quad + \sum_{m_{hk} \in \mathcal{M}} size(m_{hk}), \end{aligned} \tag{5}$$

where $FwSet(m_{ij})$ denotes the set of intermediate processes that forward m_{ij} on the VPT and is given by:

$$\begin{aligned} FwSet(m_{ij}) = \{ &\langle P_i^n, \ldots, P_i^{c+1}, P_j^c, P_j^{c-1}, \ldots, P_j^1 \rangle : \\ & c \in \left(\{d : P_i^d \neq P_j^d\} - \max(\{d : P_i^d \neq P_j^d\}) \right), \\ & \text{for } 1 \leq d \leq n\}. \end{aligned} \tag{6}$$

Then, the messages that will be forwarded by P_k is given by m_{ij} such that $P_k \in FwSet(m_{ij})$. In (5) the first summation is for the send and receive volume load of P_k as an intermediate process, thus the factor of two for this sum. The second and third summation, respectively, is for the send and receive volume load of P_k due to the messages for which it is the source and the destination.

The move strategy in the solution space is selected as swapping the mappings of a pair of neighbor processes. Swap neighborhood is defined to be the pair of processes that are neighbors along any VPT dimension (i.e., the edges in the topology graph G_t). The algorithm proceeds as a sequence of passes until no further improvement is obtained, where each pass consists of a number of swap operations. At each pass, after unlocking all processes, the algorithm iteratively selects the bottleneck process P_b and tries to find the best swap for P_b among its neighbors in $\cup_d \nu(P_b, d)$. Here, the best swap (P_b, P_x) for P_b refers to the swap with a maximum gain, where the gain is the

Algorithm 2: `KL-Bottleneck`.

Input: $\mathcal{P}$
pass $\leftarrow$ True
while pass **do**
  swapList $\leftarrow \emptyset$
  Unlock all processes
  swap $\leftarrow$ True
  while swap **do**
    swap $\leftarrow$ False
    $P_b \leftarrow$ find the current bottleneck process
    (swapGain, P_x) $\leftarrow$ `BestSwap`(P_b)
    if $P_b \neq P_x$ **then**
      Append (swapGain, P_b, P_x) to swapList
      Lock P_b and P_x
      swap $\leftarrow$ True
    end if
  end while
  (maxPassGain, idxMax) $\leftarrow$ highest prefix sum on swapList
  Realize the first idxMax swaps while retracting the remaining swaps
  if maxPassGain ≤ 0 **then**
    pass $\leftarrow$ False
  end if
end while

Algorithm 3: `BestSwap`.

Input: P_b
maxGain $\leftarrow$ -∞
$P_x \leftarrow P_b$
for each neighbor P_i of P_b **do**
  if P_i is unlocked **then**
    $P_{b'} \leftarrow$ find the new bottleneck process if we swap P_b and P_i
    gain $\leftarrow VolLoad(P_b) - VolLoad(P_{b'})$
    if gain $>$ maxGain **then**
      maxGain $\leftarrow$ gain
      $P_x \leftarrow P_i$
    end if
  end if
end for
return (maxGain, P_x)

difference in the communication volume loads of the current and new bottleneck process, P_b and $P_{b'}$ ($P_{b'}$ may remain to be P_b). That is, $gain(P_b, P_x) = VolLoad(P_b) - VolLoad(P_{b'})$. Then, the algorithm locks both P_b and P_x for the current pass to disable their swap with any other process during the remaining swaps of the pass. Note that the algorithm may temporarily realize swaps with negative gains to enable hill-climbing capability. At the end of a pass, the algorithm computes the maximum prefix sum of the gains of the sequence of swaps. Then, it confirms the realization of the swaps in this prefix swap sequence by retracting the remaining swaps of the sequence.

We should note that swapping P_b and P_x may change the communication volume loads of all intermediate processes of the original P2P messages, whose destination and/or source processes are P_b and P_x. So after a swap, it may be better to compute the volume loads of all unlocked processes from scratch to find the new bottleneck process $P_{b'}$. Algorithms 2 and 3 present the pseudo-codes for the basic steps of the proposed algorithm.

V. Experiments

We validate the effectiveness of the proposed VPT and the `STFW` communication algorithm with comprehensive evaluations. We group our evaluations under two parts: comparisons against MPI routines (Section V-B) and impact of mapping for VPT (Section V-C). In each, we include analyses on synthetic communication graphs, followed by communication operations in a real-world application. As applications, we showcase in reducing communication time in distributed-memory parallelization of Canonical Polyadic Decomposition (CPD) and 1D row-parallel SpMM. We explain the datasets used for these applications as well as the generation of communication graphs in Section V-A.

For a thorough and comprehensive analysis of the VPT in terms of performance metrics, scalability in a real-world application, communication performance on different networks (such as 3D torus and Dragonfly networks) and large-scale communication analysis up to 16 K processes, we refer the reader to our earlier work [1]. Our VPT library code is available on https://github.com/nfabubaker/VPTComm.

A. Datasets

Synthetic communication graphs. We generate synthetic datasets to achieve a controlled setting where we can vary the skew in latency and volume in the communication. In this regard, we generate synthetic communication graphs (i.e., G_c in Section IV). There are two basic parameters in the generation of these graphs: the graph structure and the distribution of edge weights. Regarding the structure, we generate Erdős-Rényi graphs (GNP) to simulate mild skew in the communication pattern and recursive matrix model graphs [16] (RMAT) to simulate high skew. We vary density in the GNP graphs with the edge creation probability parameters $0.1, 0.2, 0.4$. For the RMAT graphs we use the parameters $a = 0.51, b = 0.17, c = 0.17, d = 0.15$ and for the density we vary the number of edges with a factor $0.1, 0.2, 0.4$ of $|\mathcal{P}|^2$ (these parameters are indicated with "Scale" column in Tables IV and V) Regarding the distribution of edge weights we evaluate two alternatives: uniform distribution and exponential distribution, both of which are drawn from range $[1, 2x]$, where $x \in \{0.1\text{KB}, 0.5\text{KB}, 1\text{KB}, 2\text{KB}\}$ (the former is indicated with uni/x and the latter with exp/x in Tables IV and V). The four main combinations of graph structures and edge weights capture a wide variety of latency-bound communication.

Tensors. The properties of eight sparse tensors used in distributed-memory CPD are given in Table II. `Enron`, `flickr` and `nips` are 4-mode tensors and the remaining five are 3-mode tensors. `Enron` consists of words of emails in the form (sender, receiver, word, date) quadruplets [17]. `flickr` is a binary

TABLE II
PROPERTIES OF SPARSE TENSORS USED IN CPD

Tensor	I_1	I_2	I_3	I_4	nnz	Density
Enron	6.07K	5.70K	244K	1.18K	54.2M	5.46×10^{-9}
NELL-1	25.5M	2.14M	2.90M	–	144M	9.05×10^{-13}
NELL-2	12.1K	9.2K	28.8K	–	76.8M	2.39×10^{-5}
Netflix	480K	2.18K	17.8K	–	100M	5.40×10^{-6}
facebook-rm	42.2K	40.0K	1.5K	–	738.1K	2.92×10^{-7}
flickr	320K	28.2M	1.61M	731	113M	1.07×10^{-14}
movies-amazon	227K	4.40K	87.8K	–	15.0M	1.72×10^{-7}
nips	2.48K	2.86K	14.03K	17	3.1M	1.83×10^{-6}

TABLE III
PROPERTIES OF THE MATRICES USED IN PARALLEL SPMM

Matrix	#rows/columns	#nonzeros	max #nonzeros in rows	max #nonzeros in columns
Bump_2911	2,911,419	130,641,318	196	196
coPapersCiteseer	434,102	32,073,440	1,188	1,188
coPapersDBLP	540,486	30,491,458	3,299	3,299
dielFilterV3real	1,102,824	90,408,844	271	271
Ge99H100	112,985	8,564,380	470	470
rajat31	4,690,002	20,316,253	1,252	1,252
vas_stokes_2M	2,146,677	65,129,037	637	1,284
wikipedia-20060925	2,983,494	37,269,096	5,852	159,378

tensor representing (user, image, tag, date) quadruplets [18]. `movies-amazon` represents (user, movie, word) triplets extracted from the user reviews of movies in Amazon [19]. Both `NELL-1` and `NELL-2` [20] represent (entity, relation, entity) triplets of the Never Ending Language Learner knowledge base. `Netflix` is a rating dataset that contain (user, business, rating) triplets. `nips` is a dataset of papers published in NIPS from 1987 to 2003, containing (paper, author, word, year) quadruplets [21]. `facebook-rm` consists of the wall-posting information in the form of owner-poster-date triplets from the Facebook New Orleans networks [22].

Sparse matrices. We run parallel SpMM experiments on sparse matrices collected from SuiteSparse matrix collection [23]. The properties of these matrices are presented in Table III. The column dimension of the dense matrix in our experiments is 10, which is chosen to make the communication less latency-bound and to be able to demonstrate the effect of reducing volume with mapping.

B. Comparison Against MPI Routines

1) Communication Graphs: We compare the proposed communication scheme `STFW` against four MPI routines. These MPI routines are the collective routine `MPI_Alltoallv` (A2Av), the neighbor collective routine `MPI_Neighbor_alltoallv` (NA2Av), its reorder option enabled (NA2Av-R), and non-blocking P2P routines (P2P). For the neighbor collective, we create MPI distributed graph topology according to the communication graphs described in Section V-A (by using `MPI_Dist_graph_create_adjacent` routine of MPI). For the `STFW` schemes, we group the results into three variants for the ease of presentation as `STFW-L`, `STFW-M`, and `STFW-H`, which signify the communication operations performed on low-, medium-, and high-dimensional VPTs (the columns LOW, MED, and HIGH in Table IV). We conduct our experiments on 512 processes, for which the VPT dimensions are $\{T_2, T_3, \ldots, T_9\}$. Here, `STFW-L`, `STFW-M`, and `STFW-H` stand for the best runtime obtained with VPT dimensions $\{T_2, T_3, T_4\}$, $\{T_5, T_6, T_7\}$, and $\{T_8, T_9\}$, respectively.

Our experiments on synthetic communication graphs (Sections V-B and V-C1) are conducted on a parallel system equipped with AMD EPYC CPUs clocked at 2.45 GHz with 64 cores. The nodes in the system are connected in Dragonfly network topology. All codes are developed with C++ and the runs are repeated 50 times in addition to a warmup stage. MPI version used in the experiments is Cray MPICH 8.1.28.

We present the obtained communication times in Table IV comparing 7 schemes, 4 of which are MPI routines and 3 of which are `STFW` routines. To aid the table, we also illustrate the schemes that obtain the best communication time in Fig. 6 on a Cartesian system. In this figure, the x axis contains the edge weight distributions, where the uniform and exponential edge weight distributions are plotted on the left and right of the origin, respectively. The y axis contains the graph structures, where the RMAT and GNP graphs are plotted to the up and down of the origin, respectively. A point in the plot indicates the scheme that attained the best communication time in that specific instance. For instance, on RMAT matrices with scale parameter 0.2 and exponential edge distribution with 1 KB (exp/1 KB), the scheme that obtains the lowest communication time is `STFW-L`, which is indicated with an upside down triangle.

The results clearly show that as the communication structure gets more irregular, the proposed scheme works better: in all but one RMAT graph structures the `STFW` schemes obtain the lowest communication time, whereas in GNP instances they perform better than the MPI routines in more than half of the test instances. The collective MPI routines do not perform well, although often the neighbor routines are preferable to the general collective communication: in 33 out of 48 instances NA2Av performs better than A2Av. Enabling the reorder option in the neighbor collective routine often improves the communication time (25 out of 48 instances NA2Av-R is better than NA2Av) although the amount of improvements are not significant. The comparisons against various MPI routines show that applications that harbor irregular communication patterns with small to medium message sizes can benefit and reduce their communication time by replacing those calls with the `STFW` schemes.

In Fig. 6, among the `STFW` schemes, low-dimensional VPTs tend to work better as message sizes increase and when there is a variance in message sizes, whereas high-dimensional VPTs prefer small message sizes and less variance in message sizes. In other words, if we think of the communication as a spectrum between completely latency-bound and completely volume-bound extremes, the high-dimensional VPTs would be preferred for the latency-bound extreme of the spectrum and as we move from that extreme towards the volume-bound spectrum, the preferred VPT dimensions would move from high to medium, and then medium to low, after which some point the proposed scheme would not bring any benefit as latency costs become negligible towards the volume-bound extreme of the spectrum. The reason behind this is that high-dimensional VPTs are more aggressive

TABLE IV
COMMUNICATION TIME (μsec) COMPARISON OF MPI AND STFW ON SYNTHETIC COMM. GRAPHS ON 512 PROCESSES.

			MPI				STFW		
Structure	Scale	Edge weights	A2Av	NA2Av	NA2Av-R	P2P	LOW	MED	HIGH
GNP	0.1	uni/0.1KB	523.8	119.4	126.5	103.8	108.6	93.5	**86.5**
		uni/0.5KB	832.0	765.9	587.6	**114.7**	140.1	137.0	444.0
		uni/1KB	1278.3	1410.9	1604.2	**124.5**	192.6	195.2	201.6
		uni/2KB	1192.9	1806.7	2218.7	**130.1**	303.2	296.1	315.8
		exp/0.1KB	508.2	115.9	116.3	102.5	104.5	87.1	**81.5**
		exp/0.5KB	571.9	137.2	149.0	106.6	114.7	98.5	**93.3**
		exp/1KB	724.6	571.6	555.6	117.4	124.4	**110.3**	214.0
		exp/2KB	1029.4	887.1	1611.0	**120.6**	157.6	155.2	575.0
	0.2	uni/0.1KB	627.5	241.0	313.1	193.9	118.4	102.6	**98.2**
		uni/0.5KB	1505.4	1269.5	1831.6	248.3	**192.0**	202.4	203.9
		uni/1KB	1322.8	1363.1	2142.9	**275.5**	295.0	301.4	311.3
		uni/2KB	1461.0	2173.8	2389.9	**319.4**	511.4	509.7	565.9
		exp/0.1KB	613.2	213.2	255.2	178.6	109.5	92.9	**87.5**
		exp/0.5KB	808.1	708.5	1027.1	221.0	128.7	113.9	**110.9**
		exp/1KB	965.1	1347.8	1464.5	250.6	**158.4**	161.2	476.0
		exp/2KB	1598.7	1921.9	1742.4	260.1	234.0	**225.8**	232.5
	0.4	uni/0.1KB	1047.4	1120.7	972.6	382.9	134.2	**126.5**	426.0
		uni/0.5KB	1660.3	1846.1	2336.6	508.4	**288.4**	297.6	307.3
		uni/1KB	1396.7	2380.6	2566.4	**498.7**	515.5	524.7	573.9
		uni/2KB	1454.8	2497.8	542.3	**509.4**	880.4	963.6	1131.5
		exp/0.1KB	794.3	478.9	457.0	392.5	117.4	105.4	**99.2**
		exp/0.5KB	1535.4	1708.1	1561.9	427.7	**155.0**	167.7	373.6
		exp/1KB	2042.6	1604.2	2187.4	492.7	243.4	**229.7**	236.3
		exp/2KB	1350.4	2124.4	2575.2	519.4	371.6	**370.6**	396.5
RMAT	0.1	uni/0.1KB	662.4	371.3	582.3	334.1	123.8	96.2	**94.2**
		uni/0.5KB	1143.0	827.2	763.7	415.4	**332.4**	430.0	517.0
		uni/1KB	1855.9	950.8	954.6	**455.3**	471.8	579.3	768.9
		uni/2KB	3031.2	1238.7	1140.3	594.2	**542.4**	654.3	918.3
		exp/0.1KB	625.6	366.4	354.5	288.5	155.7	87.6	**83.1**
		exp/0.5KB	840.4	513.9	513.1	350.9	133.5	**117.1**	207.6
		exp/1KB	987.7	671.2	655.2	401.5	197.0	387.8	**186.7**
		exp/2KB	1231.0	859.2	786.9	439.1	**392.2**	497.9	496.3
	0.2	uni/0.1KB	874.9	844.0	650.2	506.3	**130.2**	143.6	346.8
		uni/0.5KB	2049.5	1527.7	1668.1	616.1	**373.8**	592.8	881.8
		uni/1KB	3387.6	2064.7	1587.9	976.5	486.0	**470.1**	497.3
		uni/2KB	4844.0	2144.4	1803.4	960.8	**770.0**	947.9	1137.4
		exp/0.1KB	798.2	520.1	494.4	456.1	110.8	96.3	**92.9**
		exp/0.5KB	1112.1	976.7	708.9	552.7	**158.1**	364.4	268.2
		exp/1KB	1795.8	1354.7	1295.5	617.2	**253.5**	333.6	623.7
		exp/2KB	2453.8	1833.2	1853.8	711.5	**411.1**	559.4	559.7
	0.4	uni/0.1KB	1165.4	1643.3	1660.7	654.3	**148.8**	177.1	795.5
		uni/0.5KB	4239.4	2531.9	2234.5	938.5	414.0	**399.4**	508.7
		uni/1KB	5867.4	2532.2	2247.9	1325.6	**758.7**	858.6	951.9
		uni/2KB	8078.1	2747.7	2624.2	1673.4	**1209.1**	1446.3	1740.7
		exp/0.1KB	1086.2	1494.7	1686.5	674.8	123.1	108.7	**108.6**
		exp/0.5KB	1768.3	2283.1	2217.7	730.8	**199.3**	517.4	731.7
		exp/1KB	3244.1	2270.0	2184.6	806.2	**343.6**	441.7	541.1
		exp/2KB	4579.8	2494.1	2206.4	726.3	**546.3**	569.3	670.3

LOW, MED, and HIGH respectively denote the best runtime obtained with VPT dimensions $\{T_2, T_3, T_4\}$, $\{T_5, T_6, T_7\}$, and $\{T_8, T_9\}$.

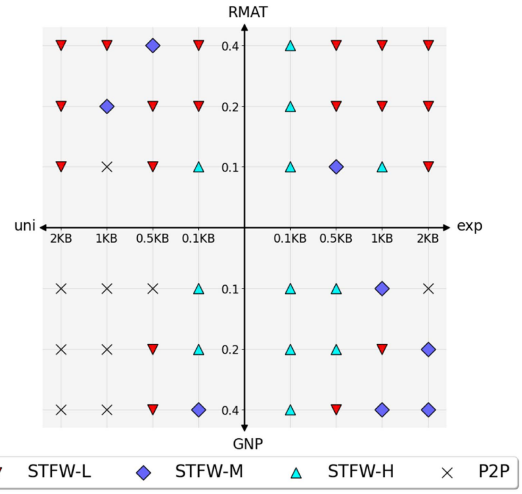


Fig. 6. The best time obtained by the communication routine in a specific experiment instance presented in Table IV.

in reducing the latency at the expense of increasing the volume due to increased STFW overhead.

For the GNP graphs, compared to the MPI routines, the STFW schemes tend to work better when there is a high skew in message sizes than they do when message sizes are uniform. This is because a power-law distribution on the message sizes result in many small messages and a small number of large messages, which can be said to be more latency-bound than where the message sizes are uniform, and hence the better performance of STFW in the former scenario. In the GNP graphs, MPI routines become more preferable to the STFW routines as the message sizes increase. This is because with a mild irregular structure of communication, the communication tends to become volume-bound with a few KBs of message sizes.

2) Parallel Sparse Tensor Decomposition: To assess the impact of replacing MPI routines with the STFW communication scheme, we evaluate the communication in the context of nonzero-based distributed-memory parallelization of Canonical Polyadic Decomposition [24] of sparse tensors. The CPD is a widely-known tensor decomposition method to analyze multi-dimensional data. The CPD decomposes a given tensor of m modes into m dense factor matrices, where the number of rows of each dense matrix equals to the dimension size of the respective tensor mode. The number of columns in each factor matrix equals to the decomposition rank R. The alternating least squares (ALS) method is a popular method for computing CPD. In CPD-ALS, Matricised Tensor Times Khatri-Rao Product (MTTKRP) is the bottleneck operation, which is performed repeatedly for each mode. In parallel MTTKRP, tensor nonzeros are distributed among processes and processes locally compute (partial) results for factor matrices using the nonzeros according to the owner computes rule. Shared factor-matrix rows incur sparse and irregular P2P communication operations as follows: Partial results computed for each shared factor-matrix row are reduced in the process handling that matrix row, and then that process expands the reduced result for the MTTKRP operation to be performed for the following mode(s).

In our evaluation, we perform the above-mentioned communication operations with the MPI P2P routines (which are found to be the most effective routine among compared MPI routines in the previous section) and the STFW scheme. The works in the literature usually vary the decomposition rank R between 4 and 64. So, the MTTKRP operation may become latency-bound for small R and bandwidth-bound for large R. We conduct experiments for parallel tensor decomposition of eight tensors (given in Table II) on 512 processes for $R \in \{4, 16, 32, 64\}$ on three different VPT dimensions T_2, T_5, and T_9 to capture low-, medium-, and high-dimensional VPTs, respectively. Our experiments on parallel tensor decomposition are performed on a parallel system equipped with Intel Xeon(R) Platinum 8480 CPUs at 2.00 GHz with 56 cores. The nodes in the system are connected with 200 Gbit NDR Infiniband NDR interconnect. All codes are developed with C++ and version 5.0 of OpenMPI distribution is used.

Fig. 7 presents the results of our experiments in terms of maximum message count handled by a process (indicated in the

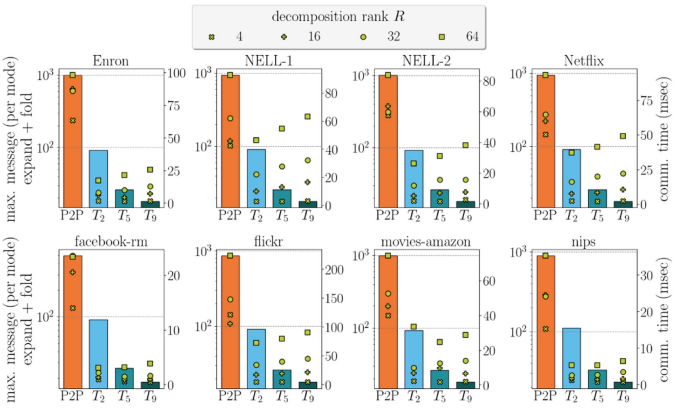


Fig. 7. Communication in parallel CP sparse tensor decomposition performed by MPI P2P routine and the STFW scheme on VPTs T_2, T_5, and T_9. Left y axis in log-scale.

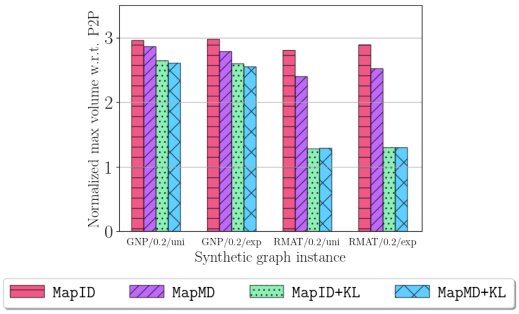


Fig. 8. Maximum send+receive volume values obtained by the mapping heuristics using STFW scheme normalized with respect to those of directly communicating the messages (i.e., P2P) for a specific subset of GNP and RMAT instances.

figure with bars using left y axis) and parallel communication time during the execution of m MTTKRP operations in a single CPD-ALS iteration (indicated in the figure with points for each R using right y axis).

As seen in Fig. 7, STFW schemes attain significantly smaller maximum message counts than P2P and the maximum message counts decrease with increasing VPT dimension. The high maximum message count of P2P shows that the communication operations are latency-bound and utilizing VPT leads to improvements more than an order of magnitude. For example, in facebook-rm, P2P's maximum message count is 1481, whereas STFW's are 276, 78, and 54 for VPTs T_2, T_5, and T_9, respectively. This significant improvement is reflected in the communication times. For small R, medium- and high-dimensional VPTs tend to perform better and as R gets larger, low-dimensional VPTs tend to perform better. This is in agreement with the findings in Section V-B: completely latency-bound communication are favored by high-dimensional VPTs and as communication move from being latency-bound towards bandwidth-bound, medium- and low-dimensional VPTs become favorable.

TABLE V
COMMUNICATION TIME (μsec) COMPARISON OF MAPPING HEURISTICS ON STFW (WITH VPT T_5) ON 512 PROCESSES

Structure	Scale	Edge weights	MPI	STFW			
			P2P	MapID	MapID+KL	MapMD	MapMD+KL
GNP	0.1	uni/0.1KB	103.8	**86.5**	100.6	100.8	101.1
		uni/0.5KB	**114.7**	137.0	148.1	145.3	148.5
		uni/1KB	**124.5**	192.6	198.8	197.4	197.0
		uni/2KB	**130.1**	296.1	289.1	299.5	307.5
		exp/0.1KB	102.5	**81.5**	98.1	96.6	97.6
		exp/0.5KB	106.6	**93.3**	104.2	105.8	106.5
		exp/1KB	117.4	**110.3**	121.3	132.9	132.5
		exp/2KB	**120.6**	155.2	158.6	162.4	162.0
	0.2	uni/0.1KB	193.9	**98.2**	109.6	110.8	109.9
		uni/0.5KB	248.3	192.0	189.5	**188.3**	196.8
		uni/1KB	**275.5**	295.0	285.4	291.3	293.2
		uni/2KB	**319.4**	509.7	472.2	485.1	487.6
		exp/0.1KB	178.6	**87.5**	102.5	104.2	103.1
		exp/0.5KB	221.0	**110.9**	135.2	134.8	135.5
		exp/1KB	250.6	**158.4**	162.8	165.7	161.5
		exp/2KB	260.1	**225.8**	235.5	239.3	239.0
	0.4	uni/0.1KB	382.9	**126.5**	141.4	133.6	138.7
		uni/0.5KB	508.4	288.4	**275.0**	283.8	279.4
		uni/1KB	498.7	515.5	**487.5**	496.8	499.3
		uni/2KB	**509.4**	880.4	804.3	808.7	806.9
		exp/0.1KB	392.5	**99.2**	111.8	112.9	113.4
		exp/0.5KB	427.7	**155.0**	168.3	163.4	167.0
		exp/1KB	492.7	**229.7**	235.8	239.4	240.1
		exp/2KB	519.4	370.6	**355.7**	362.2	357.4
RMAT	0.1	uni/0.1KB	334.1	**94.2**	100.7	109.5	103.3
		uni/0.5KB	415.4	332.4	**181.1**	286.6	181.1
		uni/1KB	455.3	471.8	455.5	444.1	**416.2**
		uni/2KB	594.2	542.4	496.8	459.3	**402.0**
		exp/0.1KB	288.5	**83.1**	96.3	98.9	97.3
		exp/0.5KB	350.9	117.1	115.1	118.8	**107.1**
		exp/1KB	401.5	186.7	168.9	295.2	**142.7**
		exp/2KB	439.1	392.2	**198.2**	307.9	300.1
	0.2	uni/0.1KB	506.3	130.2	122.5	129.3	**114.4**
		uni/0.5KB	616.1	373.8	260.5	489.9	**238.1**
		uni/1KB	976.5	470.1	**344.1**	424.3	357.8
		uni/2KB	960.8	770.0	**577.9**	693.3	589.8
		exp/0.1KB	456.1	**92.9**	104.5	107.9	104.7
		exp/0.5KB	552.7	158.1	145.4	166.8	**143.8**
		exp/1KB	617.2	253.5	288.0	324.6	**184.9**
		exp/2KB	711.5	411.1	284.8	349.4	**279.3**
	0.4	uni/0.1KB	654.3	148.8	**141.3**	161.6	168.2
		uni/0.5KB	938.5	399.4	**325.7**	376.6	330.4
		uni/1KB	1325.6	758.7	**548.7**	676.5	555.8
		uni/2KB	1673.4	1209.1	**898.1**	1064.5	906.4
		exp/0.1KB	674.8	**108.6**	119.1	120.3	116.0
		exp/0.5KB	730.8	**199.3**	211.3	217.6	211.5
		exp/1KB	806.2	343.6	**264.0**	404.1	411.1
		exp/2KB	726.3	546.3	**396.8**	456.4	403.3

C. *Effect of Mapping on VPT*

1) Communication Graphs: We investigate how addressing the increase in communication volume incurred due to forwarding messages affects the communication time by utilizing different mappings of processes onto the VPT. To this end, we compare the identical mapping (i.e., P_i is assigned to node i on VPT), MapMD (Section IV-A), and the proposed heuristic (Section IV-B) applied on top of these two mappings. We only present MapMD among the constructive heuristics as it is found to be the best performing one among three constructive heuristics tested. We utilize the same experimental setting in Section V-B (i.e., synthetic communication graphs on 512 processes). We only include the non-blocking P2P routine for MPI as the other tested MPI routines were found to be slower in Section V-B. The proposed improvement heuristic is abbreviated with KL when applied. The obtained results are presented in Fig. 8 and Table V.

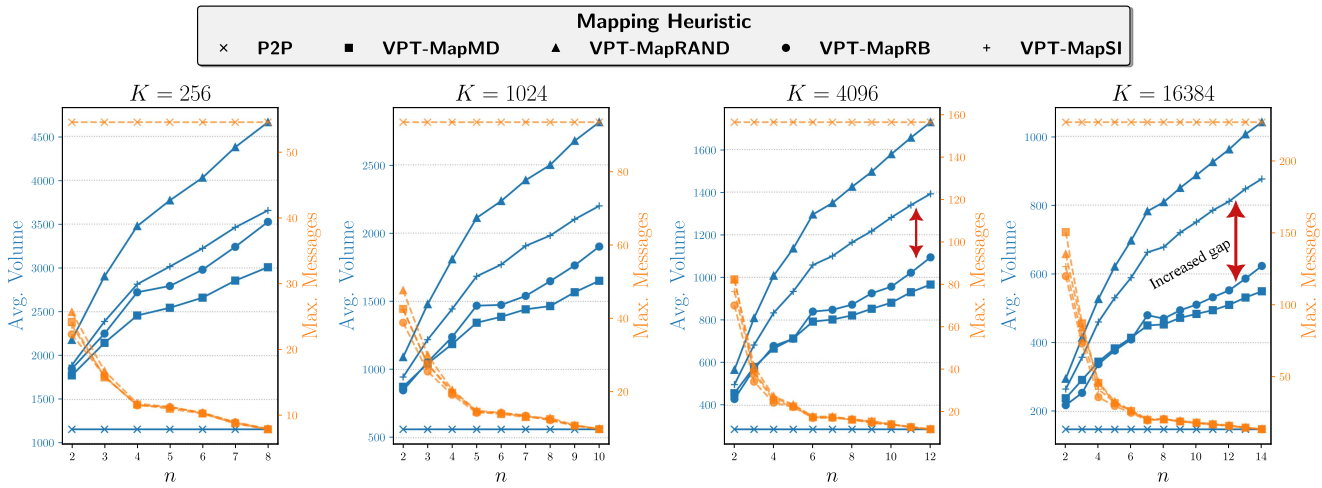


Fig. 9. Comparison of P2P and STFW schemes with varying VPT dimension (n) under different process mappings on SpMM dataset.

In Fig. 8, we compare the optimized metric in the KL heuristic to assess how successful this heuristic is in optimizing it. Recall that we address the reduction of maximum of summation of send and receive volume with this heuristic. We only present the GNP and RMAT instances with density 0.2 as other results are similar. The figure illustrates the values obtained by the mapping heuristics using STFW normalized against the values obtained when directly communicating the messages, i.e., P2P. The closer the obtained value is to 1.0, the better it is. The KL heuristic slightly improves the mappings in the GNP instances while the improvements are significant in RMAT instances. The reason that the proposed heuristic is more effective in RMAT instances can be attributed to the following: higher skew in the communication structure in the RMAT instances lead MapID and MapMD to attain much higher communication volume imbalance. This, in turn, provides more room for improvement for KL heuristic to attain in RMAT instances than in GNP instances. As a further interesting experimental finding, the values obtained by MapID+KL and MapMD+KL are quite close to each other, showing that the proposed approach is effective independent of its initial mapping.

As seen in Table V, utilizing KL is beneficial as it improves the communication time in 4 of 24 GNP instances and in 19 of 24 RMAT instances. The obtained improvements by KL in the optimized metric (i.e., maximum volume) are around 10% in GNP instances whereas they are around 50% in RMAT instances (Fig. 8), which are reflected in lower communication times in RMAT instances. It can also be said that KL tends to get more effective with increasing average message size, i.e., when the instances start moving from being latency-bound: in RMAT instances, there is not a single case where utilizing KL on top of MapID or MapMD does not improve the communication time for messages of average sizes 1 KB or 2 KB.

2) Parallel Sparse Matrix-Dense Matrix Multiplication: We evaluate the effect of mapping on volume incurred due to the STFW communication scheme in parallel SpMM. This operation, already prevalent in computational linear algebra, is also a kernel operation in graph analytics and recently popularized graph neural network training and inference. In SpMM, the dense matrix is tall and skinny. Beside the pattern of the sparse matrix, the dimension of the dense matrix in parallel SpMM is also detrimental in this operation's communication characteristics. We use a 1D row-parallel SpMM and perform the communication in parallel SpMM using MPI P2P routine and the proposed STFW scheme. For the STFW scheme, we consider the mapping heuristics described in Section IV-A in mapping processes onto the VPT. In addition to the three described mapping heuristics (abbreviated with MapSI, MapMD, and MapRB), we also evaluate random mapping as a baseline, which we denote with MapRAND. Recall that these mapping heuristics can be utilized to reduce the increase in the communication volume due to forwarding messages on the VPT.

Our experiments on SpMM are performed on a parallel system equipped with Intel Xeon(R) E5 CPUs at 2.10 GHz with 18 cores. The nodes in the system are connected with a Dragonfly-topology-based interconnect.

We illustrate the impact of process mappings in Fig. 9 on the communication cost metrics of average volume (left y axis) and maximum message count (right y axis) and plot them against VPT dimension for four different number of processes $\{256, 1024, 4096, 16384\}$. The values in the plots are the geometric averages of matrices given in Table III for parallel SpMM. In terms of average volume, P2P clearly obtains the lowest values as it does not cause any extra forwarding overhead. The mapping heuristics clearly reduce the forwarding overhead as MapSI, MapMD, and MapRB attain significantly lower average volume than MapRAND. Among the mapping heuristics, MapMD performs the best in terms of reducing volume, followed by MapRB. Both MapMD and MapRB scale much better than the rest, which is clear from the increasing gap between them as K increases beyond 4096. Different mappings have no considerable impact on the latency as maximum message counts across different mappings are close to each other. These results show that one can map processes onto the VPT to reduce the forwarding overhead

TABLE VI
PARALLEL SPMM WITH P2P VERSUS STFW UTILIZING VARIOUS PROCESS MAPPINGS ON VPT T_5

Matrix	Scheme	Mapping	Message		Time (msec)		
			Volume (avg)	Count (max)	Computation	Communication	Total
Bump_2911	P2P		1,864	28	1.29	6.02	7.31
	STFW	MapRAND	6,992	14	1.29	3.12	4.41
		MapRB	4,915	11	1.30	2.59	3.89
		MapSI	4,970	11	1.43	1.89	3.32
		MapMD	3,922	9	1.32	1.73	**3.04**
coPapersCiteseer	P2P		964	324	0.32	1.80	2.12
	STFW	MapRAND	3,628	14	0.33	2.10	2.42
		MapRB	2,882	14	0.32	1.64	1.96
		MapSI	3,319	14	0.31	1.72	2.03
		MapMD	2,963	14	0.32	1.47	**1.79**
coPapersDBLP	P2P		1,988	652	0.36	4.06	4.42
	STFW	MapRAND	7,464	15	0.31	2.88	3.19
		MapRB	6,239	15	0.35	3.15	3.50
		MapSI	7,055	15	0.32	2.89	3.21
		MapMD	6,550	14	0.32	2.85	**3.17**
dielFilterV3real	P2P		1,290	32	0.92	1.72	2.64
	STFW	MapRAND	4,833	14	0.92	2.24	3.16
		MapRB	2,590	9	0.95	0.83	**1.78**
		MapSI	3,176	11	0.92	1.41	2.33
		MapMD	2,435	9	0.95	0.98	1.93
Ge99H100	P2P		928	203	0.09	1.66	1.74
	STFW	MapRAND	3,477	15	0.09	1.63	1.72
		MapRB	2,438	13	0.09	1.06	1.15
		MapSI	3,117	14	0.09	1.51	1.60
		MapMD	2,345	14	0.09	1.01	**1.10**
rajat31	P2P		177	27	0.22	1.67	1.90
	STFW	MapRAND	662	7	0.23	0.92	1.15
		MapRB	488	5	0.22	0.83	1.06
		MapSI	397	4	0.23	0.91	1.13
		MapMD	319	3	0.24	0.51	**0.76**
vas_stokes_2M	P2P		1,552	75	0.66	2.00	2.66
	STFW	MapRAND	5,807	14	0.68	2.96	3.65
		MapRB	3,071	10	0.67	1.07	**1.73**
		MapSI	3,590	12	0.69	2.29	2.98
		MapMD	3,036	11	0.67	1.37	2.04
wikipedia...	P2P		9,231	1022	0.91	12.23	13.14
	STFW	MapRAND	34,647	15	0.74	13.26	14.00
		MapRB	30,492	15	1.40	11.34	12.75
		MapSI	32,399	15	1.43	10.94	**12.37**
		MapMD	32,727	15	1.41	13.03	14.43

without disrupting the latency properties obtained by using the VPT.

In Table VI we present the parallel SpMM runtime on 1024 processes obtained by using MPI P2P routine and `STFW` scheme with mentioned mapping heuristics on VPT T_5. The table presents two communication cost metrics (average volume in terms of communicated unit of data, which is a row of the dense matrix, and maximum message count) and time spent in computation and communication in parallel SpMM. By using mapping the communication time is improved compared to P2P and naïve mapping MapRAND. MapMD obtains the lowest parallel runtime in 5 instances, followed by MapRB in 3 instances. The better performance of MapMD in communication statistics in Fig. 9 is reflected with lower communication and hence parallel SpMM runtimes.

VI. RELATED WORK

We investigate algorithmic techniques on the software level to improve the latency costs by reducing the message counts between processes. MPI collectives contain a rich history in this aspect [2], [3], [4], [5], [6], [7]. It is possible to attain logarithmic bounds on the latency costs using algorithms like bidirectional exchange for collective communication operations such as broadcast, scatter, etc [25]. Furthermore, MPI implementations such as MPICH [26] switch between multiple algorithms depending on the message size to optimize latency or bandwidth costs. Related to our work are the sparse neighborhood collectives [14], [27], [28], [29] in MPI, with which one can define a restricted set of neighbor processes and perform collectives on them. In our methodology, the neighboring processes in distinct communication stages may or may not communicate with each other depending on what submessages they forward. In other words, our VPT indicates which processes *may* communicate. Hence, it may not be feasible to perform sparse collectives in the case where there are no or only a few communicating neighbor processes. Nonetheless, we compare the proposed store-and-forward scheme's efficiency on VPT by comparing it against general MPI collective `MPI_Alltoallv` and sparse neighborhood MPI collective `MPI_Neighbor_alltoallv` in Section V-B.

In MPI, one can define a virtual topology to indicate how processes communicate. The two types of virtual topologies supported by MPI are Cartesian and graph topologies. The provided virtual topology and additional information such as edge weights can effectively be used for the ordering of process ranks in mapping to physical topology. In our work, we assume that the proposed VPT is completely independent of the physical topology. We utilized MPI graph topology to setup the neighborhood collective communication operations in `MPI_Neighbor_alltoallv` when validating the proposed method's effectiveness on communication graphs in Section V-B.

It is also possible to reduce the latency costs with a careful distribution of data that will be communicated throughout the application. These often involve a preprocessing phase where the application is modeled with graphs/hypergraphs that are able to capture the communication requirements of the application. There are several works in this direction [30], [31], [32], [33], [34], [35], [36] and they often strive for better modeling of communication costs in the application.

The proposed VPT harbors similarities with the k-ary n-cube networks. The two common switching techniques in networks are the store-and-forward and wormhole switching [37], [38]. Our VPT borrows ideas from store-and-forward switching in the sense that there is no message fragmentation and multiple submessages may need to be stored in and forwarded by multiple processes in their buffers before reaching their destination. The submessages in our VPT refer to the original data the processes want to send, and they are contained in direct larger messages between processes. The important difference between the store-and-forward routing and the proposed store-and-forward scheme for our VPT is that multiple submessages received by a process in earlier communication stages are gathered into a single message to be forwarded in later stages. That is, at any stage of our algorithm, a message communicated between a pair of processes contains multiple submessages, where these submessages possibly differ in their source and destinations.

VII. CONCLUSION

Parallel applications whose communication involve many small-sized messages suffer from latency, which has the

potential to render the application unscalable. In our approach, we tackled this issue by developing a communication algorithm based on aggregating and forwarding messages on a regular structure called virtual process topology (VPT). With VPT, one need not rely on collective operations or specifics of the underlying parallel architecture. Moreover, the dimension parameter of the VPT, which can be provided by the user while forming it, can be used to adjust regularity of the topology. In this way, one can optimize the communication depending on how much latency-bound the application is: for instance, for relatively light latency-bound communication one can use a low-dimensional VPT while for heavy latency-bound communication high-dimensional VPT would be utilized.

We demonstrated our approach performs better than various MPI collective routines and MPI P2P routine on latency-bound scenarios. Forwarding messages on the VPT increases the volume of communicated data compared to directly communicating messages. We limited this increase with existing topology-aware mapping heuristics by adapting them to the proposed VPT and also proposed a novel mapping heuristic to limit the increase in maximum volume, which is not addressed by the existing heuristics. These mappings further improve the viability of our approach for communication scenarios that fall between latency-bound and bandwidth-bound.

REFERENCES

[1] O. Selvitopi and C. Aykanat, "Regularizing irregularly sparse point-to-point communications," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, New York, NY, USA, 2019, pp. 1–14.

[2] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of collective communication operations in MPICH," *Int. J. High Perform. Comput. Appl.*, vol. 19, no. 1, pp. 49–66, Feb. 2005.

[3] G. Almási et al., "Optimization of MPI collective communication on bluegene/l systems," in *Proc. 19th Annu. Int. Conf. Supercomputing*, New York, NY, USA, 2005, pp. 253–262.

[4] S. Kumar, A. Mamidala, P. Heidelberger, D. Chen, and D. Faraj, "Optimization of MPI collective operations on the IBM blue gene/Q supercomputer," *Int. J. High Perform. Comput. Appl.*, vol. 28, no. 4, pp. 450–464, Nov. 2014.

[5] J. Pjesivac-Grbovic, T. Angskun, G. Bosilca, G. E. Fagg, E. Gabriel, and J. J. Dongarra, "Performance analysis of MPI collective operations," in *19th IEEE Int. Parallel Distrib. Process. Symp.*, 2005, pp. 8 pp.–.

[6] A. Faraj and X. Yuan, "Automatic generation and tuning of MPI collective communication routines," in *Proc. 19th Annu. Int. Conf. Supercomput.*, New York, NY, USA, 2005, pp. 393–402.

[7] J. Bruck, C.-T. Ho, E. Upfal, S. Kipnis, and D. Weathersby, "Efficient algorithms for all-to-all communications in multiport message-passing systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 8, no. 11, pp. 1143–1156, Nov. 1997.

[8] B. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Tech. J.*, vol. 49, pp. 291–307, 1970.

[9] H. Sullivan and T. R. Bashkow, "A large scale, homogeneous, fully distributed parallel machine, I," *SIGARCH Comput. Archit. News*, vol. 5, no. 7, pp. 105–117, Mar. 1977.

[10] T. Agarwal, A. Sharma, A. Laxmikant, and L. V. Kalé, "Topology-aware task mapping for reducing communication contention on large parallel machines," in *Proc. 20th IEEE Int. Parallel Distrib. Process. Symp.*, 2006, pp. 10–pp.

[11] B. Brandfass, T. Alrutz, and T. Gerhold, "Rank reordering for MPI communication optimization," *Comput. Fluids*, vol. 80, pp. 372–380, 2013.

[12] E. Cela, *The Quadratic Assignment Problem: Theory and Algorithms*, vol. 1, Berlin, Germany: Springer Science & Business Media, 2013.

[13] S. Sahni and T. Gonzalez, "P-complete approximation problems," *J. ACM*, vol. 23, no. 3, pp. 555–565, 1976.

[14] T. Hoefler, R. Rabenseifner, H. Ritzdorf, B. R. de Supinski, R. Thakur, and J. L. Träff, "The scalable process topology interface of mpi 2.2," *Concurr. Comput. : Pract. Exper.*, vol. 23, no. 4, pp. 293–310, Mar. 2011.

[15] R. Glantz, H. Meyerhenke, and A. Noe, "Algorithms for mapping parallel processes onto grid and torus architectures," in *Proc. 23rd Euromicro Int. Conf. Parallel, Distrib., Netw.-Based Process.*, 2015, pp. 236–243.

[16] D. Chakrabarti, Y. Zhan, and C. Faloutsos, "R-MAT: A recursive model for graph mining," in *Proc. SIAM Int. Conf. Data Mining*, SIAM, 2004, pp. 442–446.

[17] J. Shetty and J. Adibi, "The Enron email dataset database schema and brief statistical report," Inf. Sci. Inst., Univ. Southern California, Los Angeles, CA, USA, Rep. 4(1), pp. 120–128, 2004.

[18] O. Görlitz, S. Sizov, and S. Staab, "Pints: Peer-to-peer infrastructure for tagging systems," in *Proc. 7th Int. Conf. Peer-to-peer Syst.*, 2008, p. 19. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855641.1855660

[19] J. McAuley and J. Leskovec, "Hidden factors and hidden topics: Understanding rating dimensions with review text," in *Proc. 7th ACM Conf. Recommender Syst.*, 2013, pp. 165–172.

[20] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka Jr., and T. M. Mitchell, "Toward an architecture for never-ending language learning," in *Proc. AAAI Conf. Artif. Intell.*, 2010, pp. 1306–1313.

[21] A. Globerson, G. Chechik, F. Pereira, and N. Tishby, "Euclidean embedding of co-occurrence data," *J. Mach. Learn. Res.*, vol. 8, pp. 2265–2295, 2007.

[22] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi, "On the evolution of user interaction in facebook," in *Proc. 2nd ACM Workshop Online Social Netw.*, New York, NY, USA, 2009, pp. 37–42.

[23] T. A. Davis and Y. Hu, "The university of Florida sparse matrix collection," *ACM Trans. Math. Softw.*, vol. 38, no. 1, pp. 1:1–1:25, Dec. 2011.

[24] M. O. Karsavuran, S. Acer, and C. Aykanat, "Partitioning models for general medium-grain parallel sparse tensor decomposition," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 1, pp. 147–159, Jan. 2021.

[25] E. Chan, M. Heimlich, A. Purkayastha, and R. van de Geijn, "Collective communication: Theory, practice, and experience: Research articles," *Concurr. Comput. : Pract. Exper.*, vol. 19, no. 13, pp. 1749–1783, Sep. 2007.

[26] MPICH 4.3.0rc4 - A portable implementation of MPI, 2019. [Online]. Available: https://www.mpich.org

[27] T. Hoefler and J. L. Traff, "Sparse collective operations for mpi," in *Proc. IEEE Int. Symp. Parallel Distrib. Process.*, Washington, DC, USA, 2009, pp. 1–8.

[28] T. Hoefler and T. Schneider, "Optimization principles for collective neighborhood communications," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Los Alamitos, CA, USA, 2012, pp. 98:1–98:10.

[29] S. H. Mirsadeghi, J. L. Träff, P. Balaji, and A. Afsahi, "Exploiting common neighborhoods to optimize MPI neighborhood collectives," in *Proc. IEEE 24th Int. Conf. High Perform. Comput.*, 2017, pp. 348–357.

[30] B. Uçar and C. Aykanat, "Encapsulating multiple communication-cost metrics in partitioning sparse rectangular matrices for parallel matrix-vector multiplies," *SIAM J. Sci. Comput.*, vol. 25, no. 6, pp. 1837–1859, 2004.

[31] M. Deveci, K. Kaya, B. Uçar, and Ümit Çatalyürek, "Hypergraph partitioning for multiple communication cost metrics: Model and methods," *J. Parallel Distrib. Comput.*, vol. 77, pp. 69–83, 2015.

[32] R. Selvitopi, M. Ozdal, and C. Aykanat, "A novel method for scaling iterative solvers: Avoiding latency overhead of parallel sparse-matrix vector multiplies," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 3, pp. 632–645, Mar. 2015.

[33] O. Selvitopi and C. Aykanat, "Reducing latency cost in 2D sparse matrix partitioning models," *Parallel Comput.*, vol. 57, pp. 1–24, 2016.

[34] O. Selvitopi, S. Acer, and C. Aykanat, "A recursive hypergraph bipartitioning framework for reducing bandwidth and latency costs simultaneously," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 2, pp. 345–358, Feb. 2017.

[35] K. Akbudak and C. Aykanat, "Simultaneous input and output matrix partitioning for outer-product–parallel sparse matrix-matrix multiplication," *SIAM J. Sci. Comput.*, vol. 36, no. 5, pp. C568–C590, 2014.

[36] E. G. Boman, K. D. Devine, and S. Rajamanickam, "Scalable matrix computations on large scale-free graphs using 2D graph partitioning," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, New York, NY, USA, 2013, pp. 50:1–50:12.

[37] W. J. Dally and C. L. Seitz, "The torus routing chip," *Distrib. Comput.*, vol. 1, no. 4, pp. 187–196, Dec. 1986.

[38] L. M. Ni and P. K. McKinley, "A survey of wormhole routing techniques in direct networks," *Computer*, vol. 26, no. 2, pp. 62–76, Feb. 1993.

Oguz Selvitopi received the PhD degree from Bilkent University, in 2016. He is currently a research scientist with Lawrence Berkeley National Laboratory. His research interests include high performance computing, parallel graph algorithms, and bioinformatics.

Nabil Abubaker received his PhD degree in computer engineering from Bilkent University in 2022. After that, he was an SNSF Postdoctoral Fellow at the Scalable Parallel Computing Laboratory (SPCL) of ETH Zürich, and he is currently a Computer Scientist at the Jülich Supercomputing Centre (JSC). His research interests include Parallel and High-Performance Computing, communication-efficient algorithms, and large-scale sparse matrix and tensor computations.

Erkin Aydin received the BS degree in computer engineering from Bilkent University, Ankara, Turkey in 2024. He is currently working toward the MS degree with Bilkent University. His research interests include parallel computing and high performance computing.

Cevdet Aykanat (Member, IEEE) received the BS and MS degrees in electrical engineering from Middle East Technical University, Turkey, and the PhD degree in electrical and computer engineering from Ohio State University, Columbus. He worked with the Intel Supercomputer Systems Division, Beaverton, Oregon, as a research associate. Since 1989, he has been affiliated with the Department of Computer Engineering, Bilkent University, Turkey, where he is currently a professor. His research interests mainly include parallel computing and its combinatorial aspects. He is the recipient of the 1995 Investigator Award of The Scientific and Technological Research Council of Turkey and 2007 Parlar Science Award. He has served as an associate editor of *IEEE Transactions of Parallel and Distributed Systems* between 2009 and 2013.