Addressing Volume and Latency Overheads in 1D-parallel Sparse Matrix-Vector Multiplication

Seher Acer, Oguz Selvitopi, and Cevdet Aykanat(✉)

Bilkent University, 06800 Ankara, Turkey
{acer,reha,aykanat}@cs.bilkent.edu.tr

Abstract. The scalability of sparse matrix-vector multiplication (SpMV) on distributed memory systems depends on multiple factors that involve different communication cost metrics. The irregular sparsity pattern of the coefficient matrix manifests itself as high bandwidth (total and/or maximum volume) and/or high latency (total and/or maximum message count) overhead. In this work, we propose a hypergraph partitioning model which combines two earlier models for one-dimensional partitioning, one addressing total and maximum volume, and the other one addressing total volume and total message count. Our model relies on the recursive bipartitioning paradigm and simultaneously addresses three cost metrics in a single partitioning phase in order to reduce volume and latency overheads. We demonstrate the validity of our model on a large dataset that contains more than 300 matrices. The results indicate that compared to the earlier models, our model significantly improves the scalability of SpMV.

Keywords: Communication cost · Sparse matrix-vector multiplication · Hypergraph partitioning · One-dimensional partitioning

1 Introduction

A key building block found in many applications is the ubiquitous sparse matrix-vector multiplication (SpMV) operation. The scalability of this kernel operation on distributed memory systems heavily depends on the communication overheads. The irregular sparsity pattern of the coefficient matrix may cause high volume and/or latency overhead and necessitate addressing multiple communication cost metrics for efficient parallel performance.

There are several communication cost metrics that determine the volume overhead such as total volume and maximum volume of data communicated by a processor. Similarly, the latency overhead is determined by cost metrics such as total message count and maximum message count. As the communication cost of SpMV generally depends on more than one of these metrics, solely minimizing a single one of them may not always lead to a scalable performance.

In this work, we propose a hypergraph partitioning model for one-dimensional-parallel (1D-parallel) SpMV, which reduces three important communication cost metrics simultaneously: total volume, maximum volume, and

F.F. Rivera et al. (Eds.): Euro-Par 2017, LNCS 10417, pp. 625–637, 2017.
DOI: 10.1007/978-3-319-64203-1_45

total message count. Our model utilizes two earlier models [1,9], where [1] addresses multiple volume-based cost metrics, whereas [9] addresses total volume and message count. The proposed model achieves partitioning in a single phase and exploits the recursive bipartitioning (RB) paradigm in order to target the cost metrics other than total volume. In our model, the maximum volume is addressed by representing the amount of communicated data with vertex weights while the total message count is addressed by encapsulating the communicated messages as message nets. We present our model for rowwise partitioning with conformal partitions on input and output vectors, however, it can easily be adapted to columnwise partitioning.

There are a few early works [2,5,12] as well as some recent works [1,3,7,9,10] that focus on reducing multiple communication cost metrics. Among these, the works in [2,3,12] are two-phase methods, where different cost metrics are handled in distinct phases. The disadvantage of these two-phase methods is that each phase is oblivious to the metrics handled in the other phase. Our model is able to address all cost metrics in a single phase. In [5], the checkerboard hypergraph model is proposed for reducing total volume and bounding message count. This work differs from ours in the sense that it achieves a nonconformal partition on vectors. UMPa [7] is a single-phase hypergraph partitioning tool that can handle multiple metrics. Despite this, it imposes a prioritization on the metrics in which the secondary metrics are considered only in the tie-breaking cases in the refinement algorithm. This may lead to poor optimization of the secondary metrics. There are very recent works [1,9] that are both single-phase and based on the RB paradigm. Our work builds upon these two works.

The rest of the paper is organized as follows. Section 2 provides the background material. We present the proposed hypergraph partitioning model in Sect. 3. Section 4 gives the experimental results and Sect. 5 concludes.

2 Background

2.1 Hypergraph Partitioning

A hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ is a general type of graph that consists of vertices and nets where edges/nets can connect more than two vertices. $\mathcal{V}$ and $\mathcal{N}$ respectively denote the sets of vertices and nets. The set of vertices connected by net $n \in \mathcal{N}$ is denoted with $Pins(n)$. Each vertex $v \in \mathcal{V}$ is assigned a weight denoted with $w(v)$. Similarly, each net $n \in \mathcal{N}$ is assigned a cost denoted with $c(n)$.

$\Pi(\mathcal{H}) = \{\mathcal{V}_1, \ldots, \mathcal{V}_K\}$ is a K-way vertex partition of $\mathcal{H}$, if each vertex part $\mathcal{V}_k$ is nonempty, parts are pairwise disjoint, and the union of the parts gives $\mathcal{V}$. In $\Pi(\mathcal{H})$, $\lambda(n)$ denotes the number of parts in which net n connects vertices, i.e., the number of parts connected by n. A net $n \in \mathcal{N}$ is a cut net if it connects at least two parts, i.e., $\lambda(n) > 1$. The cutsize of a partition $\Pi(\mathcal{H})$ is defined as

$$cut(\Pi(\mathcal{H})) = \sum_{n \in \mathcal{N}} (\lambda(n) - 1)c(n).$$

The weight $W(\mathcal{V}_k)$ of part $\mathcal{V}_k$ is the sum of the weights of the vertices in $\mathcal{V}_k$. $\Pi(\mathcal{H})$ is said to be balanced if $W(\mathcal{V}_k) \leq W_{avg}(1+\epsilon)$ for all $k = 1, \ldots, K$, where W_{avg} and ϵ respectively denote the average part weight and a maximum allowed imbalance ratio. Then, the hypergraph partitioning problem is defined as obtaining a K-way partition of a given hypergraph with the objective of minimizing cutsize and the constraint of maintaining balance on the part weights.

2.2 Reducing Total Volume via Hypergraph Partitioning

There are two hypergraph models [4] (column-net and row-net) for obtaining one-dimensional (1D) partitioning of a given SpMV of the form $y = Ax$. The column-net and row-net models are used for obtaining rowwise and columnwise partitions, respectively. We only discuss the column-net model since they are dual of each other.

In the column-net hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$, $\mathcal{V}$ contains a vertex v_i for each row i of A, whereas $\mathcal{N}$ contains a net n_j for each column j. n_j connects v_i if and only if $a_{ij} \neq 0$. In a conformal partition, x_i and y_i are assigned to the same processor for each i. To achieve a conformal partition, v_i represents row i, x_i, y_i, and the inner product associated with row i, i.e., $y_i = \langle a_{i*} \cdot x \rangle$, where a_{i*} denotes row i. n_j represents the dependency of the inner products on x_j. Note that an inner product $\langle a_{i*} \cdot x \rangle$ depends on x_j if and only if $a_{ij} \neq 0$. The weight $w(v_i)$ of $v_i \in \mathcal{V}$ is the number of nonzeros in row i, which is the number multiply-and-add operations in $\langle a_{i*} \cdot x \rangle$. Each $n_j \in \mathcal{N}$ is assigned a unit cost. A K-way partition $\Pi(\mathcal{H})$ is decoded as assigning row i, x_i, and y_i to processor P_k, for each $v_i \in \mathcal{V}_k$. This is often visualized as block-partitioned matrix

$$A^{\pi} = QAQ^T = \begin{bmatrix} R_1 \\ \vdots \\ R_K \end{bmatrix} = \begin{bmatrix} A_{11} & \cdots & A_{1K} \\ \vdots & \ddots & \vdots \\ A_{K1} & \cdots & A_{K1} \end{bmatrix},$$

where Q is the permutation matrix. Here, row stripe R_k and the corresponding y- and x-vector elements are assigned to processor P_k. The processor that owns a row performs the computations regarding its nonzeros due to the owner-computes rule [8]. In A^{π}, each nonzero segment of column j in off-diagonal block $A_{\ell k}$ incurs a unit communication as P_k sends x_j to P_ℓ. Then, the volume of communication incurred by sending x_j is equal to the number of nonzero segments of column j in off-diagonal blocks of A^{π}. The segment of column j in block $A_{\ell k}$ is a nonzero segment if and only if net n_j connects part $\mathcal{V}_\ell$. Assuming all the diagonal entries are nonzero in A, the total volume then amounts to the cutsize of $\Pi(\mathcal{H})$. Hence, the objective of minimizing cutsize corresponds to minimizing total volume. Since each processor P_k performs multiply-and-add operations proportional to the number of nonzeros in R_k, maintaining balance on the part weights corresponds to maintaining balance on the computational loads of the processors.

3 Simultaneous Reduction of Maximum Volume, Total Volume and Total Message Count

The proposed model relies on the recursive bipartitioning (RB) paradigm to address the cost metrics other than total volume. In RB, a given hypergraph $\mathcal{H}$ is recursively bipartitioned until the desired number of parts is reached. This process induces a full binary tree in which nodes represent hypergraphs. The rth level of the RB tree contains 2^r hypergraphs: $\mathcal{H}_0^r, \ldots, \mathcal{H}_{2^r-1}^r$. Note that the level that a hypergraph belongs to is indicated in the superscript. Bipartitioning $\mathcal{H}_k^r = (\mathcal{V}_k^r, \mathcal{N}_k^r)$ generates hypergraphs $\mathcal{H}_{2k}^{r+1}$ and $\mathcal{H}_{2k+1}^{r+1}$. At the end of the RB process, vertex sets of the hypergraphs in the $\lg_2 K$th level induce the resulting K-way partition of the given hypergraph $\mathcal{H}$ as $\Pi(\mathcal{H}) = \{\mathcal{V}_0^{\lg K}, \ldots, \mathcal{V}_{K-1}^{\lg K}\}$.

Our model is summarized in Algorithm 1. As inputs, it takes the column-net hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ of a given $y = Ax$, the number of processors K, the maximum allowable imbalance ratio ϵ, and coefficients α and β. We first compute the imbalance ratio ϵ' used in each bipartitioning in order to result in an imbalance ratio not exceeding ϵ in the final K-way partition (line 1). We start the RB process with the given column-net hypergraph $\mathcal{H}$ as $\mathcal{H}_0^0 = \mathcal{H}$ (line 2). The nets in $\mathcal{H}$ are referred to as volume nets as they capture the total communication volume of the corresponding parallel SpMV. The bipartitionings in the RB process are carried out in breadth-first order, as seen in lines 3–4 of Algorithm 1. At each RB step, after obtaining bipartition $\Pi(\mathcal{H}_k^r) = \{\mathcal{V}_{2k}^{r+1}, \mathcal{V}_{2k+1}^{r+1}\}$ (line 8), hypergraphs $\mathcal{H}_{2k}^{r+1}$ and $\mathcal{H}_{2k+1}^{r+1}$ belonging to the next level of the RB tree are immediately formed with volume nets via cut-net splitting technique (lines 9–12). The function calls in lines 6–7 enable the simultaneous reduction of cost metrics. These function calls introduce an additional cost of $O(V \lg_2 K)$ to the overall partitioning.

In our model, the matrix rows and x- and y-vector elements corresponding to the vertices in $\mathcal{H}_k^r$ are assumed to be assigned to processor group $\mathcal{P}_k^r$, for each hypergraph $\mathcal{H}_k^r$ in the RB tree. We also assume that the RB process is currently at the beginning of the for-loop iteration in which hypergraph $\mathcal{H}_k^r$ is bipartitioned. In the current RB tree, the leaf hypergraphs are listed from left to right as $\mathcal{H}_0^{r+1}, \ldots, \mathcal{H}_{2k-1}^{r+1}, \mathcal{H}_k^r, \ldots, \mathcal{H}_{2^r-1}^r$.

3.1 Reducing Maximum Volume

We formulate the objective of minimizing the maximum volume of processors as additional constraints [1]. These constraints are satisfied by maintaining balance on the communication loads of processor groups $\mathcal{P}_{2k}^{r+1}$ and $\mathcal{P}_{2k}^{r+1}$ for each bipartition $\Pi(\mathcal{H}_k^r)$. To do so, in addition to the standard vertex weights that capture the computational loads of processors, we utilize vertex weights that capture the communication loads.

ADD-COMMUNICATION-WEIGHTS function assigns the communication loads to the vertices in $\mathcal{H}_k^r$ (line 6 of Algorithm 1). The details of this function are given in Algorithm 2. Here, we consider the maximum volume as the maximum

Algorithm 1. The Proposed Hypergraph Partitioning Model

Input : Column-net hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$, number of processors K, imbalance ratio ϵ, coefficients α and β.
Output: K-way partition of $\mathcal{H}$.

1 $\epsilon' \leftarrow (1+\epsilon)^{\frac{1}{\lg K}} - 1$
2 $\mathcal{H}_0^0 \leftarrow \mathcal{H}$ ▷ $\mathcal{N}$ contains only volume nets
3 **for** $r \leftarrow 0$ **to** $\lg K - 1$ **do**
4 **for** $k \leftarrow 0$ **to** $2^r - 1$ **do**
5 **if** $r > 0$ **then**
 ▷ Addressing maximum volume
6 ADD-COMMUNICATION-WEIGHTS($\mathcal{H}$, $\mathcal{V}_k^r$, α)
 ▷ Addressing total message count
7 ADD-MESSAGE-NETS($\mathcal{H}$, $\mathcal{H}_k^r$, β)
8 $\Pi(\mathcal{H}_k^r) = \{\mathcal{V}_{2k}^{r+1}, \mathcal{V}_{2k+1}^{r+1}\} \leftarrow$ HypergraphPartitioning($\mathcal{H}_k^r, 2, \epsilon'$)
 ▷ Form $\mathcal{H}_{2k}^{r+1}$ and $\mathcal{H}_{2k+1}^{r+1}$ with volume nets by net splitting
9 $\mathcal{N}_{2k}^{r+1} \leftarrow$ Split volume nets of $\mathcal{H}_k^r$ in $\mathcal{V}_{2k}^{r+1}$
10 $\mathcal{N}_{2k+1}^{r+1} \leftarrow$ Split volume nets of $\mathcal{H}_k^r$ in $\mathcal{V}_{2k+1}^{r+1}$
11 $\mathcal{H}_{2k}^{r+1} \leftarrow (\mathcal{V}_{2k}^{r+1}, \mathcal{N}_{2k}^{r+1})$
12 $\mathcal{H}_{2k+1}^{r+1} \leftarrow (\mathcal{V}_{2k+1}^{r+1}, \mathcal{N}_{2k+1}^{r+1})$
13 **return** $\Pi(\mathcal{H}) = \{\mathcal{V}_0^{\lg K}, \ldots, \mathcal{V}_{K-1}^{\lg K}\}$

send volume of the processors. Recall that processor group $\mathcal{P}_k^r$ owns x_i for each $v_i \in \mathcal{V}_k^r$. Hence, $\mathcal{P}_k^r$ sends x_i to each processors group $\mathcal{P}_\ell^q$ that needs x_i, where $q \in \{r, r+1\}$. Note that $\mathcal{P}_\ell^q$ needs x_i if it is assigned a row j with $a_{ji} \neq 0$. This situation is captured by net n_i connecting vertex v_j where $v_j \in \mathcal{V}_\ell^q$ (lines 3–4). Here, we utilize the global view of net n_i of the initial column-net hypergraph $\mathcal{H}$ to determine the communications between $\mathcal{P}_k^r$ and the other processor groups.

The communication volume incurred by sending x_i amounts to the number of parts connected by n_i different than $\mathcal{V}_k^r$. This value is denoted with $|Con(n_i)|$ in Algorithm 2 and computed in lines 2–5.

The communication weight $|Con(n_i)|$ associated with vertex v_i is unified to its computational weight (line 6). This unification scheme is proven to be more successful than assigning the communication weights as separate second weights [1]. The unification scheme scales the communication weight by a coefficient α which denotes the ratio of the per-word transfer time to the per-word multiply-and-add time in the parallel system. As a result, the unified weight gives the time required to send x_i and to compute inner product of row i with x in terms of the time of an individual multiply-and-add operation.

With the unified communication and computation vertex weights, maintaining balance on the part weights while bipartitioning $\mathcal{H}_k^r$ corresponds to maintaining a unified balance on the computational and communication loads of processor groups $\mathcal{P}_{2k}^{r+1}$ and $\mathcal{P}_{2k+1}^{r+1}$. Balancing the communication volumes of processors corresponds to minimizing the maximum volume of processors under the condition that the total communication volume is minimized.

Algorithm 2. ADD-COMMUNICATION-WEIGHTS

Input : Original hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$, vertex set $\mathcal{V}_k^r$, coefficient α

foreach $v_i \in \mathcal{V}_k^r$ **do**
  $Con(n_i) \leftarrow \emptyset$
  foreach $v_j \in Pins(n_i)$ in $\mathcal{H}$ **do**
    if $v_j \notin \mathcal{V}_k^r$ **then**
      ▷ `Let` $v_j \in \mathcal{V}_\ell^q$
      $Con(n_i) \leftarrow Con(n_i) \cup \{\mathcal{V}_\ell^q\}$
  ▷ $|Con(n_i)|$ `is the communication load due to sending` x_i
  $w(v_i) \leftarrow w(v_i) + \alpha|Con(n_i)|$

Algorithm 3. ADD-MESSAGE-NETS

Input : Original hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$, hypergraph $\mathcal{H}_k^r$ to be bipartitioned, message net cost β

foreach $v_i \in \mathcal{V}_k^r$ **do**
  foreach $v_j \in Pins(n_i)$ in $\mathcal{H}$ **do**
    if $v_j \notin \mathcal{V}_k^r$ **then**
      ▷ `Let` $v_j \in \mathcal{V}_\ell^q$
      if message net $s_\ell^q \in \mathcal{N}_k^r$ **then**
        $Pins(s_\ell^q) \leftarrow Pins(s_\ell^q) \cup \{v_i\}$
      else
        $c(s_\ell^q) \leftarrow \beta$
        $Pins(s_\ell^q) \leftarrow \{v_i\}$ and $\mathcal{N}_k^r \leftarrow \mathcal{N}_k^r \cup \{s_\ell^q\}$
  foreach n_j in $\mathcal{H}$ with $v_i \in Pins(n_j)$ **do**
    if $v_j \notin \mathcal{V}_k^r$ **then**
      ▷ `Let` $v_j \in \mathcal{V}_\ell^q$
      if message net $r_\ell^q \in \mathcal{N}_k^r$ **then**
        $Pins(r_\ell^q) \leftarrow Pins(r_\ell^q) \cup \{v_i\}$
      else
        $c(r_\ell^q) \leftarrow \beta$
        $Pins(r_\ell^q) \leftarrow \{v_i\}$ and $\mathcal{N}_k^r \leftarrow \mathcal{N}_k^r \cup \{r_\ell^q\}$

3.2 Reducing Total Message Count

We use message nets in order to encapsulate the messages sent and received [9]. A message net connects the vertices that represent the rows or vector elements that require a message together. To encapsulate the up-to-date messages among processor groups in the RB process, the message nets are formed and added to the hypergraphs just prior to their bipartitioning (line 7 in Algorithm 1). Note that on the contrary, since volume nets do not depend on the state of the other parts, we form them as soon as their vertex set is formed (lines 9–10).

ADD-MESSAGE-NETS function adds message nets to hypergraph $\mathcal{H}_k^r$, which contains only volume nets before the respective function call. The details of this function are given in Algorithm 3. There are two types of message nets: send nets and receive nets. For each processor group $\mathcal{P}_\ell^q$ that $\mathcal{P}_k^r$ sends a mes-

sage to, we add a send net s_ℓ^q to $\mathcal{H}_k^r$. Net s_ℓ^q connects vertices that represent the x-vector elements to be sent to $\mathcal{P}_\ell^q$. $\mathcal{P}_k^r$ sends x_i to $\mathcal{P}_\ell^q$ if a row j with $a_{ji} \neq 0$ is assigned to $\mathcal{P}_\ell^q$. Then, the set of vertices connected by net s_ℓ^q is formulated as

$$Pins(s_\ell^q) = \{v_i : n_i \text{ of } \mathcal{H} \text{ connects } \mathcal{V}_\ell^q\},$$

as computed in lines 2–8. As in Sect. 3.1, we make use of the global view of n_i of the initial column-net hypergraph $\mathcal{H}$ to determine the communications $\mathcal{P}_k^r$ performs. Similarly, for each processor group $\mathcal{P}_\ell^q$ that $\mathcal{P}_k^r$ receives a message from, we add a receive net r_ℓ^q to $\mathcal{H}_k^r$. Net r_ℓ^q connects vertices that represent the A-matrix rows whose multiplications need x-vector elements to be received from $\mathcal{P}_\ell^q$. $\mathcal{P}_k^r$ receives x_j from $\mathcal{P}_\ell^q$ if row i and x_j are respectively assigned to $\mathcal{P}_k^r$ and $\mathcal{P}_\ell^q$, where $a_{ij} \neq 0$. Then, the set of vertices connected by net r_ℓ^q (computed in lines 9–15) is formulated as

$$Pins(r_\ell^q) = \{v_i : n_j \text{ of } \mathcal{H} \text{ connects } \mathcal{V}_k^r \text{ due to } v_i \text{ and } v_j \in \mathcal{V}_\ell^q\}.$$

The message nets are assigned a cost of β whereas the volume nets are assigned unit cost. Here, coefficient β denotes the ratio of per-message startup time to per-word transfer time in the parallel system. With both volume and message nets having the mentioned costs, minimizing the cutsize in each bipartitioning throughout the RB process corresponds to minimizing total volume and total message count in 1D-parallel SpMV.

4 Experiments

4.1 Setting

We consider a total of four schemes for comparison. The total volume metric is common to all schemes and it is addressed by default in all schemes. One scheme addresses a single metric, two schemes address two metrics and the proposed scheme addresses three metrics simultaneously. These schemes are listed as follows:

- `BL`: Proposed in [4], this scheme solely addresses total volume (Sect. 2.2).
- `MV`: Proposed recently in [1], this scheme considers two metrics related to volume: total volume and maximum send volume. α is set to 10.
- `TM`: Proposed in another recent work [9], this scheme considers one metric related to volume and one metric related to latency: total volume and total message count. β is set to 50.
- `MVTM`: This scheme is the one proposed in this work (Sect. 3) and considers all three metrics: total volume, maximum volume and total message count.

The values of α and β are respectively picked in the light of the experiments of [1] and [9]. For a more detailed discussion on these parameters, we refer the reader to these two studies. Note that `MV` and `TM` are special cases of `MVTM`, with $\alpha = 10$ and $\beta = 0$ for `MV`, and $\alpha = 0$ and $\beta = 50$ for `TM`.

We test for five different number of processors: $K \in \{64, 128, 256, 512, 1024\}$. The partitioning experiments are conducted on an extensive set of matrices from the SuiteSparse Matrix Collection [6]. We selected the square matrices that have more than 5,000 rows/columns and nonzeros between 50,000 and 50,000,000, resulting in 964 matrices. Among these, in order to select the matrices that have high volume and/or latency overhead, we used the following two criteria considering the partitioning statistics of `BL` for any tested K: (i) the partitions whose maximum volume is greater than or equal to 1.5 times the average volume and (ii) the partitions whose average message count is greater than or equal to $1.3 \lg_2 K$. The first criterion aims to include the matrices that are volume bound, i.e., the matrices with more than 50% imbalance in volume when partitioned with `BL`. The second criterion aims to include the matrices that are latency bound. We empirically found out that the matrices having around $\lg_2 K$ number of messages per processor exhibit insignificant latency overhead. By multiplying this value with a coefficient of 1.3 we were able to filter out such matrices. Note that our aim in this work is not to show the proposed scheme is better than the other tested three schemes for *any* matrix, but for the matrices that are bound by both volume and latency, hence the motivation to the selection criteria. After filtering, there exist respectively 317, 335, 363, 374 and 373 matrices for 64, 128, 256, 512 and 1024 processors. Partitionings regarding the four schemes are performed on these sets of matrices. Parallel runtime experiments with the SpMV operation are performed on a set of 15 matrices for 64, 128, 256, and 512 processors.

The schemes are realized using the hypergraph partitioner PaToH [4] (line 8 of Algorithm 1). The parallel SpMV is realized in C using the message passing paradigm [11]. The parallel experiments are performed on a Lenovo NeXtScale supercomputer[1] that consists of 1512 nodes. A node on this system has two 18-core Intel Xeon E5-2697 Broadwell processors clocked at 2.30 GHz each with 64 GB of RAM. The network topology of this system is a fat tree.

4.2 Partitioning and Parallel Runtime Results

Table 1 presents the average values obtained by the compared schemes in terms of three different communication cost metrics for 64, 128, 256, 512 and 1024 processors. These metrics are total volume, maximum volume and total message count, which are respectively denoted in the table as "tot vol.", "max vol." and "tot msg.". Total and maximum volume are in terms of number of words. The table consists of two column groups. In the first group, the actual values obtained by the schemes are given. In the second group, the values obtained by `MV`, `TM` and `MVTM` are normalized with respect to those obtained by `BL`. Each value is the geometric mean of the values obtained for the matrices in the respective dataset.

Considering maximum volume and total message count metrics, the best values obtained in these metrics belong to `MV` and `TM`, respectively, as expected. For example for 512 processors, `MV` obtains an improvement of 26% in maximum volume compared to `BL`, while `TM` obtains an improvement of 24% in message

[1] https://www.cineca.it/en/content/marconi.

Table 1. Partition statistics of four schemes.

K	Scheme	Actual values			Normalized w.r.t. BL		
		Tot vol.	Max vol.	Tot msg.	Tot vol.	Max vol.	Tot msg.
64 (317 matrices)	BL	52331	1757	1316	–	–	–
	MV	51250	1454	1344	0.98	0.83	1.02
	TM	64242	2279	887	1.23	1.30	0.67
	MVTM	62788	1855	911	1.20	1.06	0.69
128 (335 matrices)	BL	67310	1253	3298	–	–	–
	MV	65940	991	3419	0.98	0.79	1.04
	TM	87462	1732	2219	1.30	1.38	0.67
	MVTM	85248	1342	2296	1.27	1.07	0.70
256 (363 matrices)	BL	92008	944	7556	–	–	–
	MV	90013	728	7846	0.98	0.77	1.04
	TM	122337	1379	5306	1.33	1.46	0.70
	MVTM	118801	967	5546	1.29	1.02	0.73
512 (374 matrices)	BL	129345	792	17174	–	–	–
	MV	125915	589	17869	0.97	0.74	1.04
	TM	171887	1145	13030	1.33	1.45	0.76
	MVTM	165680	733	13712	1.28	0.93	0.80
1024 (373 matrices)	BL	176058	735	35768	–	–	–
	MV	170016	518	37364	0.97	0.71	1.04
	TM	228866	1036	29073	1.30	1.41	0.81
	MVTM	217871	613	30996	1.24	0.83	0.87

count compared to BL. Since these two schemes address solely one of these metrics along with total volume, they are clear winners in those metrics. MVTM reveals itself as a tradeoff between MV and TM by ranking second among these three schemes in both metrics. In other words, its maximum volume is worse than MV but better than TM, while its total message count is worse than TM but better than MV.

Another important aspect of MVTM is that, when we compare MV, TM and MVTM in maximum volume and total message count metrics in Table 1, MVTM always appears to be the second best scheme and the difference between MVTM and the best scheme is generally smaller than the difference between MVTM and the third best scheme. For example for 256 processors, MVTM's maximum volume is 33% worse than MV's while TM's maximum volume is 89% worse than MV's, and MVTM's message count is 5% worse than TM's while MV's message count is 48% worse than TM's. For these reasons, MVTM is expected to be a better remedy compared to MV and TM for the matrices with high volume and latency overhead, which is validated by the parallel experiments given in the rest of the section.

The performances of four schemes are compared in terms of parallel SpMV runtimes for 15 matrices on 64, 128, 256 and 512 processors. These matrices and the obtained parallel runtimes are presented in Table 2. The times are in microseconds and correspond to a single SpMV operation. We only give the detailed results for 128 and 512 processors, as similar improvements are observed for 64 and 256 processors. On 128 processors, `MVTM` obtains the best runtimes in 11 of 15 matrices, while `MV` obtains the best runtimes in three matrices and `TM` obtains in only one. On 512 processors, `MVTM` obtains the best runtimes in all matrices. These results indicate that `MVTM` is more successful than the other three schemes in addressing volume and latency overheads.

In Table 3, we present the parallel SpMV runtime averages (geometric means) for four schemes for these 15 matrices on 64, 128, 256 and 512 processors. The first column group of the table gives the actual values obtained by the schemes while the second column group gives the normalized values of `MV`, `TM` and `MVTM` with respect to those of `BL`. In any number of processors, the best scheme is

Table 2. Detailed parallel SpMV runtimes (microseconds).

matrix	#rows/ #columns	#nonzeros	128 processors				512 processors			
			BL	MV	TM	MVTM	BL	MV	TM	MVTM
144	144, 649	2, 148, 786	139.5	135.1	153.9	**116.8**	91.8	83.5	91.7	**79.5**
598a	110, 971	1, 483, 868	93.8	92.6	93.7	**77.4**	77.8	67.6	58.4	**54.9**
ASIC_680ks	682, 712	2, 329, 176	184.1	165.9	154.1	**131.8**	128.6	127.2	153.2	**106.3**
cage13	445, 315	7, 479, 343	453.0	**413.0**	445.0	416.5	336.4	332.2	288.9	**284.7**
cfd1	70, 656	1, 828, 364	97.1	94.0	88.8	**86.7**	65.5	60.2	60.9	**54.2**
crystk03	24, 696	1, 751, 178	87.3	88.6	83.9	**82.6**	67.8	67.6	52.3	**50.5**
Ga19As19H42	133, 123	8, 884, 839	484.3	437.2	**427.1**	440.8	417.4	281.0	281.4	**256.0**
gas_sensor	66, 917	1, 703, 365	90.6	90.8	84.5	**78.5**	64.4	67.9	46.7	**44.3**
kkt_power	2, 063, 494	14, 612, 663	604.0	**582.5**	610.7	586.9	261.8	234.7	244.2	**198.9**
m14b	214, 765	3, 358, 036	162.5	169.4	165.2	**146.0**	105.8	95.5	95.7	**74.5**
offshore	259, 789	4, 242, 673	202.1	182.5	202.7	**178.4**	113.1	114.1	116.1	**97.4**
pre2	659, 033	5, 959, 282	246.3	**240.3**	245.2	243.3	120.2	117.5	109.7	**91.5**
raefsky4	19, 779	1, 328, 611	69.3	65.8	67.4	**62.8**	63.9	63.9	46.8	**44.4**
Si34H36	97, 569	5, 156, 379	315.2	300.1	303.2	**289.2**	348.1	258.9	243.7	**212.6**
webbase-1M	1, 000, 005	3, 105, 536	248.4	271.3	210.5	**192.0**	274.0	301.1	243.1	**173.7**

Table 3. Average parallel SpMV runtimes (microseconds).

K	Actual values				Normalized w.r.t. BL		
	BL	MV	TM	MVTM	MV	TM	MVTM
64	280.1	277.2	275.7	268.8	0.99	0.98	0.96
128	185.7	179.7	178.1	163.5	0.97	0.96	0.88
256	144.9	136.6	134.8	115.9	0.94	0.93	0.80
512	134.4	124.8	115.0	99.2	0.93	0.86	0.74

MVTM, followed by TM, MV and BL. For example on 512 processors, MVTM obtains a 26% improvement over BL, while MV and TM respectively obtain 7% and 14% improvement over BL. Observe that with increasing number of processors, the improvements obtained by all schemes over BL become more pronounced. This can be attributed to the increased importance of different communication cost metrics with increasing number of processors, which implies that addressing more cost metrics leads to better parallel runtime performance.

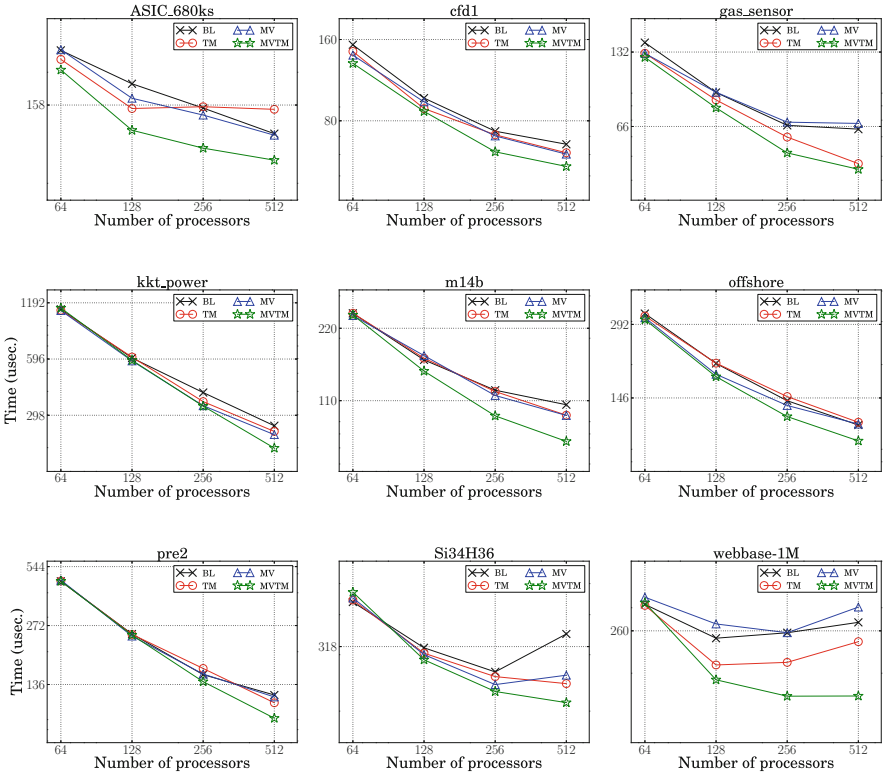

Fig. 1. Strong scaling plots of four schemes.

In Fig. 1, we plot the parallel SpMV runtimes obtained by the schemes for nine matrices. These nine matrices are a subset of the matrices given in Table 2. The x and y axes in the plots are both log-scale and respectively denote the number of processors and SpMV runtime (microseconds). As seen from these plots, MVTM scales significantly better than the other three schemes. These plots validate the claim that MVTM handles the tradeoff between volume and latency overheads better than TM and MV.

5 Conclusion

This work focused on the aspects of reducing communication bottlenecks of a key kernel, sparse matrix-vector multiplication. We argued that there exist several communication cost metrics that affect the parallel performance and proposed a model to reduce three such metrics simultaneously: total volume, maximum volume and total message count. With extensive experiments, it is shown that the proposed model strikes a better tradeoff between these volume- and latency-related cost metrics compared to the other models that address only one or two cost metrics. Realistic experiments up to 512 processors on a large-scale system showed that our model leads to better scalability and validated that it is a better remedy for the SpMV instances that are bound by both volume and latency.

Acknowledgments. We acknowledge PRACE for awarding us access to resource Marconi (Lenovo NextScale) based in Italy at CINECA Supercomputing Centre. This work was supported by The Scientific and Technological Research Council of Turkey (TUBITAK) under Grant EEEAG-114E545. This article is also based upon work from COST Action CA 15109 (COSTNET).

References

1. Acer, S., Selvitopi, O., Aykanat, C.: Improving performance of sparse matrix dense matrix multiplication on large-scale parallel systems. Parallel Comput. **59**, 71–96 (2016). Theory and Practice of Irregular Applications
2. Bisseling, R.H., Meesen, W.: Communication balancing in parallel sparse matrix-vector multiply. Electron. Trans. Numer. Anal. **21**, 47–65 (2005)
3. Boman, E.G., Devine, K.D., Rajamanickam, S.: Scalable matrix computations on large scale-free graphs using 2D graph partitioning. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis SC 2013, NY, USA, pp. 50:1–50:12. ACM, New York (2013)
4. Çatalyürek, U.V., Aykanat, C.: Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. IEEE Trans. Parallel Distrib. Syst. **10**(7), 673–693 (1999)
5. Çatalyürek, U., Aykanat, C.: A hypergraph-partitioning approach for coarse-grain decomposition. In: Proceedings of the 2001 ACM/IEEE Conference on Supercomputing SC 2001, NY, USA, pp. 28–28. ACM, New York (2001)
6. Davis, T.A., Hu, Y.: The University of Florida sparse matrix collection. ACM Trans. Math. Softw. **38**(1), 1:1–1:25 (2011)
7. Deveci, M., Kaya, K., Uçar, B., Çatalyürek, U.: Hypergraph partitioning for multiple communication cost metrics: model and methods. J. Parallel Distrib. Comput. **77**, 69–83 (2015)
8. Kumar, V.: Introduction to Parallel Computing, 2nd edn. Addison-Wesley Longman Publishing Co., Inc., Boston (2002)
9. Selvitopi, O., Acer, S., Aykanat, C.: A recursive hypergraph bipartitioning framework for reducing bandwidth and latency costs simultaneously. IEEE Trans. Parallel Distrib. Syst. **28**(2), 345–358 (2017)
10. Slota, G.M., Madduri, K., Rajamanickam, S.: PuLP: Scalable multi-objective multi-constraint partitioning for small-world networks. In: 2014 IEEE International Conference on Big Data (Big Data), pp. 481–490, October 2014

11. Uçar, B., Aykanat, C.: A library for parallel sparse matrix vector multiplies. Technical report BU-CE-0506, Bilkent University (2005)
12. Uçar, B., Aykanat, C.: Encapsulating multiple communication-cost metrics in partitioning sparse rectangular matrices for parallel matrix-vector multiplies. SIAM J. Sci. Comput. **25**(6), 1837–1859 (2004)