Optimizing nonzero-based sparse matrix partitioning models via reducing latency ☆

Seher Acer[a], Oguz Selvitopi[a], Cevdet Aykanat[a,*]

[a]*Bilkent University, Computer Engineering Department, 06800, Ankara, TURKEY*

Abstract

Nonzero-based fine-grain and medium-grain sparse matrix partitioning models attain the lowest communication volume and computational imbalance among all partitioning models due to their larger solution space. This usually comes, however, at the expense of a high message count, i.e., high latency overhead. This work addresses this shortcoming by proposing new fine-grain and medium-grain models that are able to minimize communication volume and message count in a single partitioning phase. The new models utilize message nets in order to encapsulate the minimization of total message count. We further fine-tune these models by proposing delayed addition and thresholding for message nets in order to establish a trade-off between the conflicting objectives of minimizing communication volume and message count. The experiments on an extensive dataset of nearly one thousand matrices show that the proposed models improve the total message count of the original nonzero-based models by up to 27% on the average, which is reflected on the parallel runtime of sparse matrix-vector multiplication as an average reduction of 15% on 512 processors.

Keywords: sparse matrix, sparse matrix-vector multiplication, row-column-parallel SpMV, load balancing, communication overhead, hypergraph, fine-grain partitioning, medium-grain partitioning, recursive bipartitioning.

1. Introduction

Sparse matrix partitioning plays a pivotal role in scaling applications that involve irregularly sparse matrices on distributed memory systems. Several decades of research on this subject led to elegant combinatorial partitioning models that are able to address the needs of these applications.

A key operation in sparse applications is the sparse matrix-vector multiplication (SpMV). The irregular sparsity pattern of the coefficient matrix in SpMV necessitates a non-trivial parallelization, usually achieved through combinatorial models based on graph and hypergraph partitioning. Graph and hypergraph models prove to be powerful tools in their immense ability to represent applications with the aim of optimizing desired parallel performance metrics. The literature is rich in terms of such models for parallelizing SpMV [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]. We focus on the hypergraph models as they correctly encapsulate the total communication volume in SpMV and the proposed models in this work rely on hypergraphs. The hypergraph models for SpMV are grouped into two depending on how they distribute the nonzeros of individual rows/columns of the matrix among processors: if all nonzeros that belong to a row/column are assigned to a single processor, then they are called one-dimensional (1D) models [1], otherwise, they are called two-dimensional (2D) models. The 2D models are generally superior to the 1D models in terms of parallel performance due to their higher flexibility in distributing the matrix nonzeros. Examples of 2D models include checkerboard [8, 14], jagged [14], fine-grain [14, 15], and medium-grain [10] models. Among these, the fine-grain and medium-grain models are referred to as nonzero-based models as they obtain nonzero-based matrix partitions, which are the most general possible [7].

Among all models, the fine-grain model adopts the finest partitioning granularity by treating the nonzeros of the matrix as individual units, which leads it to have the largest solution space. For this reason, it achieves the lowest communication volume and the lowest imbalance on computational loads of the processors [14]. Since the nonzeros of the matrix are treated individually in the fine-grain model, the nonzeros that belong to the same row/column are more likely to be scattered to multiple processors compared to the other models. This may result in a high message count and hinder scalability. The fine-grain hypergraphs have the largest size for the same reason, causing this model to have the highest partitioning overhead. The recently proposed medium-grain model [10] alleviates this issue by operating on groups of nonzeros instead of individual nonzeros. The medium-grain model's partitioning overhead is comparable to those of the 1D models, (i.e., quite low), while its communication volume is comparable to that of the fine-grain model.

The nonzero-based models attain the lowest communication volume among all 1D and 2D models, however, the overall communication cost is not determined by the volume only, but better formulated as a function of multiple communication cost metrics. Another important cost metric is the total message count, which is not only overlooked by both the fine-grain and medium-grain models, but also exacerbated due to the having nonzero-based partitions. Among the two basic components of

☆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).

*Corresponding author

Email addresses: acer@cs.bilkent.edu.tr (Seher Acer), reha@cs.bilkent.edu.tr (Oguz Selvitopi), aykanat@cs.bilkent.edu.tr (Cevdet Aykanat)

the communication cost, the total communication volume determines the bandwidth component, whereas the total message count determines the latency component.

In this work, we propose a novel fine-grain model and a novel medium-grain model to simultaneously reduce the bandwidth and latency costs of parallel SpMV. The original fine-grain [15] and medium-grain [10] models already encapsulate the bandwidth cost. We use message nets to incorporate the minimization of the latency cost into the partitioning objective of these models. Message nets aim to group the matrix nonzeros and/or the vector entries in the SpMV that necessitate a message together. The formation of message nets relies on the recursive bipartitioning paradigm, which is shown to be a powerful approach to optimize multiple communication cost metrics in recent studies [16, 17]. Message nets are recently proposed for certain types of iterative applications that involve a computational phase either preceded or followed by a communication phase with a restriction of conformal partitions on input and output data [17]. 1D row-parallel and column-parallel SpMV operations constitute examples for these applications. This work differs from [17] in the sense that the nonzero-based partitions necessitate a parallel SpMV that involves *two* communication phases with *no* restriction of conformal partitions. We also propose two enhancements concerning the message nets to better exploit the trade-off between the bandwidth and latency costs for the proposed models.

The existing partitioning models that address the bandwidth and latency costs in the literature can be grouped into two according to whether they explicitly address the latency cost (the bandwidth cost is usually addressed explicitly). The models that do not explicitly address the latency cost provide an upper bound on the message counts [8, 14, 18]. We focus on the works that explicitly address the latency cost [17, 19, 20], which is also the case in this work. Among these works, the one proposed in [19] is a two-phase approach which addresses the bandwidth cost in the first phase with the 1D models and the latency cost in the second phase with the communication hypergraph model. In the two-phase approaches, since different cost metrics are addressed in separate phases, a metric minimized in a particular phase may get out of control in the other phase. Our models fall into the category of single-phase approaches. The other two works also adopt a single-phase approach to address multiple communication cost metrics, where UMPa [20] uses a direct K-way partitioning approach, while [17] exploits the recursive bipartitioning paradigm. UMPa is rather expensive as it introduces an additional cost involving a quadratic factor in terms of the number of processors to each refinement pass. Our approach introduces an additional cost involving a mere logarithmic factor in terms of the number of processors to the entire partitioning.

The rest of the paper is organized as follows. Section 2 gives background on parallel SpMV, the fine-grain model, recursive bipartitioning, and the medium-grain model. Sections 3 and 4 present the proposed fine-grain and medium-grain models, respectively. Section 5 describes practical enhancements to these models. Section 6 gives the experimental results and Section 7 concludes.

Algorithm 1 Row-column-parallel SpMV as performed by processor P_k

Require: $\mathcal{A}_k$, $\mathcal{X}_k$

$\triangleright$ *Pre-communication phase — expands on x-vector entries*
Receive the needed x-vector entries that are not in $\mathcal{X}_k$
Send the x-vector entries in $\mathcal{X}_k$ needed by other processors

$\triangleright$ *Computation phase*
$y_i^{(k)} \leftarrow y_i^{(k)} + a_{i,j}x_j$ for each $a_{i,j} \in \mathcal{A}_k$

$\triangleright$ *Post-communication phase — folds on y-vector entries*
Receive the partial results for y-vector entries in $\mathcal{Y}_k$ and compute $y_i \leftarrow \sum y_i^{(\ell)}$ for each partial result $y_i^{(\ell)}$
Send the partial results for y-vector entries not in $\mathcal{Y}_k$

return $\mathcal{Y}_k$

2. Preliminaries

2.1. Row-column-parallel SpMV

We consider the parallelization of SpMV of the form $y = Ax$ with a nonzero-based partitioned matrix A, where $A = (a_{i,j})$ is an $n_r \times n_c$ sparse matrix with n_{nz} nonzero entries, and x and y are dense vectors. The ith row and the jth column of A are respectively denoted by r_i and c_j. The jth entry of x and the ith entry of y are respectively denoted by x_j and y_i. Let $\mathcal{A}$ denote the set of nonzero entries in A, that is, $\mathcal{A} = \{a_{i,j} : a_{i,j} \neq 0\}$. Let $\mathcal{X}$ and $\mathcal{Y}$ respectively denote the sets of entries in x and y, that is, $\mathcal{X} = \{x_1, \ldots, x_{n_c}\}$ and $\mathcal{Y} = \{y_1, \ldots, y_{n_r}\}$. Assume that there are K processors in the parallel system denoted by $P_1, \ldots, P_K$. Let $\Pi_K(\mathcal{A}) = \{\mathcal{A}_1, \ldots, \mathcal{A}_K\}$, $\Pi_K(\mathcal{X}) = \{\mathcal{X}_1, \ldots, \mathcal{X}_K\}$, and $\Pi_K(\mathcal{Y}) = \{\mathcal{Y}_1, \ldots, \mathcal{Y}_K\}$ denote K-way partitions of $\mathcal{A}$, $\mathcal{X}$, and $\mathcal{Y}$, respectively.

Given partitions $\Pi_K(\mathcal{A})$, $\Pi_K(\mathcal{X})$, and $\Pi_K(\mathcal{Y})$, without loss of generality, the nonzeros in $\mathcal{A}_k$ and the vector entries in $\mathcal{X}_k$ and $\mathcal{Y}_k$ are assigned to processor P_k. For each $a_{i,j} \in \mathcal{A}_k$, P_k is held responsible for performing the respective multiply-and-add operation $y_i^{(k)} \leftarrow y_i^{(k)} + a_{i,j}x_j$, where $y_i^{(k)}$ denotes the partial result computed for y_i by P_k. Algorithm 1 displays the basic steps performed by P_k in parallel SpMV for a nonzero-based partitioned matrix A. This algorithm is called the *row-column-parallel SpMV* [19]. In this algorithm, P_k first receives the needed x-vector entries that are not in $\mathcal{X}_k$ from their owners and sends its x-vector entries to the processors that need them in a *pre-communication* phase. Sending x_j to possibly multiple processors is referred to as the *expand* operation on x_j. When P_k has all needed x-vector entries, it performs the local SpMV by computing $y_i^{(k)} \leftarrow y_i^{(k)} + a_{i,j}x_j$ for each $a_{i,j} \in \mathcal{A}_k$. P_k then receives the partial results for the y-vector entries in $\mathcal{Y}_k$ from other processors and sends its partial results to the processors that own the respective y-vector entries in a *post-communication* phase. Receiving partial result(s) for y_i from possibly multiple processors is referred to as the *fold* operation on y_i. Note overlapping of computation and communication is not considered in this algorithm for the sake of clarity.

For an efficient row-column-parallel SpMV, the goal is to find $\Pi_K(\mathcal{A})$, $\Pi_K(\mathcal{X})$, and $\Pi_K(\mathcal{Y})$ with low communication overhead

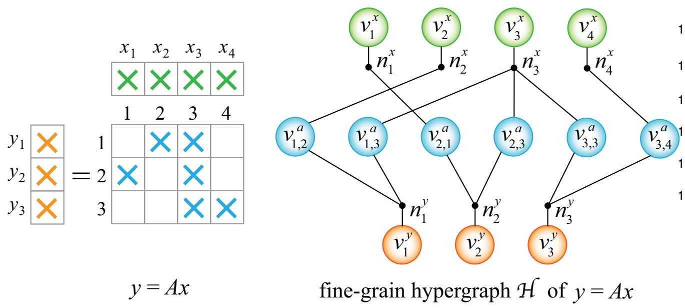

Figure 1: A sample $y = Ax$ and the corresponding fine-grain hypergraph.

and good balance on computational loads of processors. Sections 2.2 and 2.4 respectively describe the fine-grain [8] and medium-grain [10] hypergraph partitioning models, in which the goal of reducing communication overhead is met partially by only minimizing the bandwidth cost, i.e., the total communication volume. Vector partitions $\Pi_K(\mathcal{X})$ and $\Pi_K(\mathcal{Y})$ can also be found after finding $\Pi_K(\mathcal{A})$ [19, 21]. This work, on the other hand, finds all partitions at once in a single partitioning phase.

2.2. *Fine-grain hypergraph model*

In the fine-grain hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$, each entry in $\mathcal{A}$, $\mathcal{X}$, and $\mathcal{Y}$ is represented by a different vertex. Vertex set $\mathcal{V}$ contains a vertex $v^a_{i,j}$ for each $a_{i,j} \in \mathcal{A}$, a vertex v^x_j for each $x_j \in \mathcal{X}$, and a vertex v^y_i for each $y_i \in \mathcal{Y}$. That is,

$$\mathcal{V} = \{v^a_{i,j} : a_{i,j} \neq 0\} \cup \{v^x_1, \ldots, v^x_{n_c}\} \cup \{v^y_1, \ldots, v^y_{n_r}\}.$$

$v^a_{i,j}$ represents both the data element $a_{i,j}$ and the computational task $y_i \leftarrow y_i + a_{i,j}x_j$ associated with $a_{i,j}$, whereas v^x_j and v^y_i only represent the input and output data elements x_j and y_i, respectively.

The net set $\mathcal{N}$ contains two different types of nets to represent the dependencies of the computational tasks on x- and y-vector entries. For each $x_j \in \mathcal{X}$ and $y_i \in \mathcal{Y}$, $\mathcal{N}$ respectively contains the nets n^x_j and n^y_i. That is,

$$\mathcal{N} = \{n^x_1, \ldots, n^x_{n_c}\} \cup \{n^y_1, \ldots, n^y_{n_r}\}.$$

Net n^x_j represents the input dependency of the computational tasks on x_j, hence, it connects the vertices that represent these tasks and v^x_j. Net n^y_i represents the output dependency of the computational tasks on y_i, hence, it connects the vertices that represent these tasks and v^y_i. The sets of vertices connected by n^x_j and n^y_i are respectively formulated as

$$Pins(n^x_j) = \{v^x_j\} \cup \{v^a_{t,j} : a_{t,j} \neq 0\} \text{ and}$$
$$Pins(n^y_i) = \{v^y_i\} \cup \{v^a_{i,t} : a_{i,t} \neq 0\}.$$

$\mathcal{H}$ contains $n_{nz}+n_c+n_r$ vertices, n_c+n_r nets and $2n_{nz}+n_c+n_r$ pins. Figure 1 displays a sample SpMV instance and its corresponding fine-grain hypergraph. In $\mathcal{H}$, the vertices are assigned the weights that signify their computational loads. Hence, $w(v^a_{i,j}) = 1$ for each $v^a_{i,j} \in \mathcal{V}$ as $v_{i,j}$ represents a single multiply-and-add operation, whereas $w(v^x_j) = w(v^y_i) = 0$ for each $v^x_j \in \mathcal{V}$ and $v^y_i \in \mathcal{V}$ as they do not represent any computation. The nets are assigned unit costs, i.e., $c(n^x_j) = c(n^y_i) = 1$ for each $n^x_j \in \mathcal{N}$ and $n^y_i \in \mathcal{N}$.

A K-way vertex partition $\Pi_K(\mathcal{H}) = \{\mathcal{V}_1, \ldots, \mathcal{V}_K\}$ can be decoded to obtain $\Pi_K(\mathcal{A})$, $\Pi_K(\mathcal{X})$, and $\Pi_K(\mathcal{Y})$ by assigning the entries represented by the vertices in part $\mathcal{V}_k$ to processor P_k. That is,

$$\mathcal{A}_k = \{a_{i,j} : v^a_{i,j} \in \mathcal{V}_k\},$$
$$\mathcal{X}_k = \{x_j : v^x_j \in \mathcal{V}_k\}, \text{ and}$$
$$\mathcal{Y}_k = \{y_i : v^y_i \in \mathcal{V}_k\}.$$

Let $\lambda(n)$ denote the number of parts connected by net n in $\Pi_K(\mathcal{H})$, where a net is said to connect a part if it connects at least one vertex in that part. A net n is called cut if it connects at least two parts, i.e., $\lambda(n) > 1$, and uncut, otherwise. The cutsize of $\Pi_K(\mathcal{H})$ is defined as

$$cutsize(\Pi_K(\mathcal{H})) = \sum_{n \in \mathcal{N}} c(n)(\lambda(n) - 1). \tag{1}$$

For a given $\Pi_K(\mathcal{H})$, a cut net n^x_j (n^y_i) incurs an expand (fold) operation on x_j (y_i) with a volume of $\lambda(n^x_j) - 1$ ($\lambda(n^y_i) - 1$). Hence, $cutsize(\Pi_K(\mathcal{H}))$ is equal to the total communication volume in parallel SpMV. Therefore, minimizing $cutsize(\Pi_K(\mathcal{H}))$ corresponds to minimizing the total communication volume.

In $\Pi_K(\mathcal{H})$, the weight $W(\mathcal{V}_k)$ of part $\mathcal{V}_k$ is defined as the sum of the weights of the vertices in $\mathcal{V}_k$, i.e., $W(\mathcal{V}_k) = \sum_{v \in \mathcal{V}_k} w(v)$, which is equal to the total computational load of processor P_k. Then, maintaining the balance constraint

$$W(\mathcal{V}_k) \leq W_{avg}(1 + \epsilon), \text{ for } k = 1, \ldots, K,$$

corresponds to maintaining balance on the computational loads of the processors. Here, W_{avg} and ϵ denote the average part weight and a maximum imbalance ratio, respectively.

2.3. *Recursive bipartitioning (RB) paradigm*

In RB, a given domain is first bipartitioned and then this bipartition is used to form two new subdomains. In our case, a domain refers to a hypergraph ($\mathcal{H}$) or a set of matrix and vector entries ($\mathcal{A}$, $\mathcal{X}$, $\mathcal{Y}$). The newly-formed subdomains are recursively bipartitioned until K subdomains are obtained. This procedure forms a hypothetical full binary tree, which contains $\lceil \log K \rceil + 1$ levels. The root node of the tree represents the given domain, whereas each of the remaining nodes represents a subdomain formed during the RB process. At any stage of the RB process, the subdomains represented by the leaf nodes of the RB tree collectively induce a partition of the original domain.

The RB paradigm is successfully used for hypergraph partitioning. Figure 2 illustrates an RB tree currently in the process of partitioning a hypergraph. The current leaf nodes induce a four-way partition $\Pi_4(\mathcal{H}) = \{\mathcal{V}_1, \mathcal{V}_2, \mathcal{V}_3, \mathcal{V}_4\}$ and each node in the RB tree represents both a hypergraph and its vertex set. While forming two new subhypergraphs after each RB step, the cut-net splitting technique is used [1] to encapsulate the cutsize in (1). The sum of the cutsizes incurred in all RB steps is equal to the cutsize of the resulting K-way partition.

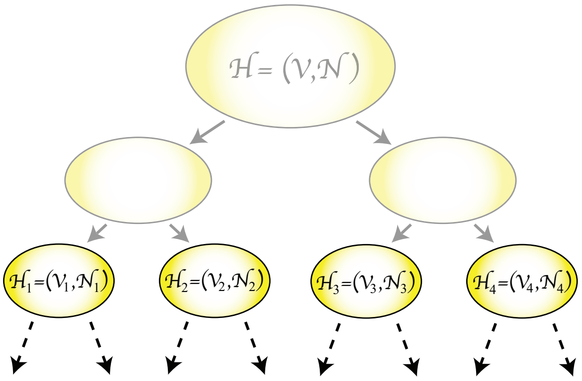

Figure 2: The RB tree during partitioning $\mathcal{H} = (\mathcal{V}, \mathcal{N})$. The current RB tree contains four leaf hypergraphs with the hypergraph to be bipartitioned next being $\mathcal{H}_1 = (\mathcal{V}_1, \mathcal{N}_1)$.

2.4. Medium-grain hypergraph model

In the medium-grain hypergraph model, the sets $\mathcal{A}$, $\mathcal{X}$ and $\mathcal{Y}$ are partitioned into K parts using RB. The medium-grain model uses a mapping for a subset of the nonzeros at each RB step. Because this mapping is central to the model, we focus on a single bipartitioning step to explain the medium-grain model. Before each RB step, the nonzeros to be bipartitioned are first mapped to their rows or columns by a heuristic and a new hypergraph is formed according to this mapping.

Consider an RB tree for the medium-grain model with K' leaf nodes, where $K' < K$, and assume that the kth node from the left is to be bipartitioned next. This node represents $\mathcal{A}_k$, $\mathcal{X}_k$, and $\mathcal{Y}_k$ in the respective K'-way partitions $\{\mathcal{A}_1, \ldots, \mathcal{A}_{K'}\}$, $\{\mathcal{X}_1, \ldots, \mathcal{X}_{K'}\}$, and $\{\mathcal{Y}_1, \ldots, \mathcal{Y}_{K'}\}$. First, each $a_{i,j} \in \mathcal{A}_k$ is mapped to either r_i or c_j, where this mapping is denoted by $map(a_{i,j})$. With a heuristic, $a_{i,j} \in \mathcal{A}_k$ is mapped to r_i if r_i has fewer nonzeros than c_j in $\mathcal{A}_k$, and to c_j if c_j has fewer nonzeros than r_i in $\mathcal{A}_k$. After determining $map(a_{i,j})$ for each nonzero in $\mathcal{A}_k$, the medium-grain hypergraph $\mathcal{H}_k = (\mathcal{V}_k, \mathcal{N}_k)$ is formed as follows. Vertex set $\mathcal{V}_k$ contains a vertex v^x_j if x_j is in $\mathcal{X}_k$ or there exists at least one nonzero in $\mathcal{A}_k$ mapped to c_j. Similarly, $\mathcal{V}_k$ contains a vertex v^y_i if y_i is in $\mathcal{Y}_k$ or there exists at least one nonzero in $\mathcal{A}_k$ mapped to r_i. Hence, v^x_j represents x_j and/or the nonzero(s) assigned to c_j, whereas v^y_i represents y_i and/or the nonzero(s) assigned to r_i. That is,

$$\mathcal{V}_k = \{v^x_j : x_j \in \mathcal{X}_k \text{ or } \exists a_{t,j} \in \mathcal{A}_k \text{ s.t. } map(a_{t,j}) = c_j\} \cup \{v^y_i : y_i \in \mathcal{Y}_k \text{ or } \exists a_{i,t} \in \mathcal{A}_k \text{ s.t. } map(a_{i,t}) = r_i\}.$$

Besides the data elements, vertex v^x_j/v^y_i represents the group of computational tasks associated with the nonzeros mapped to them, if any.

The net set $\mathcal{N}_k$ contains a net n^x_j if $\mathcal{A}_k$ contains at least one nonzero in c_j, and a net n^y_i if $\mathcal{A}_k$ contains at least one nonzero in r_i. That is,

$$\mathcal{N}_k = \{n^x_j : \exists a_{t,j} \in \mathcal{A}_k\} \cup \{n^y_i : \exists a_{i,t} \in \mathcal{A}_k\}.$$

n^x_j represents the input dependency of the groups of computational tasks on x_j, whereas n^y_i represents the output dependency of the groups of computational tasks on y_i. Hence, the sets of vertices connected by n^x_j and n^y_i are respectively formulated by

$$Pins(n^x_j) = \{v^x_j\} \cup \{v^y_t : map(a_{t,j}) = r_t\} \text{ and}$$
$$Pins(n^y_i) = \{v^y_i\} \cup \{v^x_t : map(a_{i,t}) = c_t\}.$$

In $\mathcal{H}_k$, each net is assigned a unit cost, i.e., $c(n^x_j) = c(n^y_i) = 1$ for each $n^x_j \in \mathcal{N}$ and $n^y_i \in \mathcal{N}$. Each vertex is assigned a weight equal to the number of nonzeros represented by that vertex. That is,

$$w(v^x_j) = |\{a_{t,j} : map(a_{t,j}) = c_j\}| \text{ and}$$
$$w(v^y_i) = |\{a_{i,t} : map(a_{i,t}) = r_i\}|.$$

$\mathcal{H}_k$ is bipartitioned with the objective of minimizing the cutsize and the constraint of maintaining balance on the part weights. The resulting bipartition is further improved by an iterative refinement algorithm. In every RB step, minimizing the cutsize corresponds to minimizing the total volume of communication, whereas maintaining balance on the weights of the parts corresponds to maintaining balance on the computational loads of the processors.

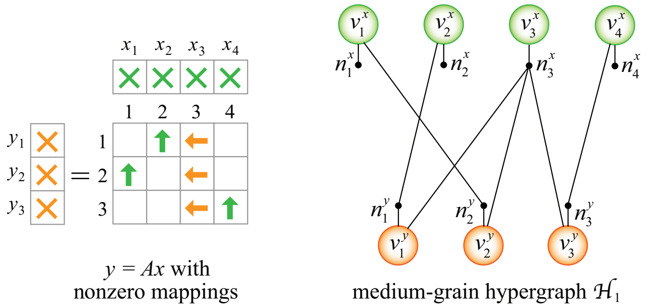

Figure 3: The nonzero assignments of the sample $y = Ax$ and the corresponding medium-grain hypergraph.

Figure 3 displays a sample SpMV instance with nonzero mapping information and the corresponding medium-grain hypergraph. This example illustrates the first RB step, hence, $\mathcal{A}_1 = \mathcal{A}$, $\mathcal{X}_1 = \mathcal{X}$, $\mathcal{Y}_1 = \mathcal{Y}$, and $K' = k = 1$. Each nonzero in A is denoted by an arrow, where the direction of the arrow shows the mapping for that nonzero. For example, n^x_3 connects v^x_3, v^y_1, v^y_2, and v^y_3 since $map(a_{1,3}) = r_1$, $map(a_{2,3}) = r_2$, and $map(a_{3,3}) = r_3$.

3. Optimizing fine-grain partitioning model

In this section, we propose a fine-grain hypergraph partitioning model that simultaneously reduces the bandwidth and latency costs of the row-column-parallel SpMV. Our model is built upon the original fine-grain model (Section 2.2) via utilizing the RB paradigm. The proposed model contains two different types of nets to address the bandwidth and latency costs. The nets of the original fine-grain model already address the bandwidth cost and they are called "volume nets" as they encapsulate the minimization of the total communication volume. At each RB step, our model forms and adds new nets to the hypergraph to be bipartitioned. These new nets address the latency cost and they are called "message nets" as they encapsulate the minimization of the total message count.

Message nets aim to group the matrix nonzeros and vector entries that altogether necessitate a message. The formation and addition of message nets rely on the RB paradigm. To determine the existence and the content of a message, a partition information is needed first. At each RB step, prior to bipartitioning the current hypergraph that already contains the volume nets, the message nets are formed using the K'-way partition information and added to this hypergraph, where K' is the number of leaf nodes in the current RB tree. Then this hypergraph is bipartitioned, which results in a $(K'+1)$-way partition as the number of leaves becomes $K'+1$ after bipartitioning. Adding message nets just before each bipartitioning allows us to utilize the most recent global partition information at hand. In contrast to the formation of the message nets, the formation of the volume nets via cut-net splitting requires only the local bipartition information.

3.1. Message nets in a single RB step

Consider an SpMV instance $y = Ax$ and its corresponding fine-grain hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ with the aim of partitioning $\mathcal{H}$ into K parts to parallelize $y = Ax$. The RB process starts with bipartitioning $\mathcal{H}$, which is represented by the root node of the corresponding RB tree. Assume that the RB process is at the state where there are K' leaf nodes in the RB tree, for $1 < K' < K$, and the hypergraphs corresponding to these nodes are denoted by $\mathcal{H}_1, \ldots, \mathcal{H}_{K'}$ from left to right. Let $\Pi_{K'}(\mathcal{H}) = \{\mathcal{V}_1, \ldots, \mathcal{V}_{K'}\}$ denote the K'-way partition induced by the leaf nodes of the RB tree. $\Pi_{K'}(\mathcal{H})$ also induces K'-way partitions $\Pi_{K'}(\mathcal{A})$, $\Pi_{K'}(\mathcal{X})$, and $\Pi_{K'}(\mathcal{Y})$ of sets $\mathcal{A}$, $\mathcal{X}$, and $\mathcal{Y}$, respectively. Without loss of generality, the entries in $\mathcal{A}_k$, $\mathcal{X}_k$, and $\mathcal{Y}_k$ are assigned to processor group $\mathcal{P}_k$. Assume that $\mathcal{H}_k = (\mathcal{V}_k, \mathcal{N}_k)$ is next to be bipartitioned among these hypergraphs. $\mathcal{H}_k$ initially contains only the volume nets. In our model, we add message nets to $\mathcal{H}_k$ to obtain the augmented hypergraph $\mathcal{H}_k^M = (\mathcal{V}_k, \mathcal{N}_k^M)$. Let $\Pi(\mathcal{H}_k^M) = \{\mathcal{V}_{k,L}, \mathcal{V}_{k,R}\}$ denote a bipartition of $\mathcal{H}_k^M$, where L and R in the subscripts refer to left and right, respectively. $\Pi(\mathcal{H}_k^M)$ induces bipartitions $\Pi(\mathcal{A}_k) = \{\mathcal{A}_{k,L}, \mathcal{A}_{k,R}\}$, $\Pi(\mathcal{X}_k) = \{\mathcal{X}_{k,L}, \mathcal{X}_{k,R}\}$, and $\Pi(\mathcal{Y}_k) = \{\mathcal{Y}_{k,L}, \mathcal{Y}_{k,R}\}$ on $\mathcal{A}_k$, $\mathcal{X}_k$, and $\mathcal{Y}_k$, respectively. Let $\mathcal{P}_{k,L}$ and $\mathcal{P}_{k,R}$ denote the processor groups to which the entries in $\{\mathcal{A}_{k,L}, \mathcal{X}_{k,L}, \mathcal{Y}_{k,L}\}$ and $\{\mathcal{A}_{k,R}, \mathcal{X}_{k,R}, \mathcal{Y}_{k,R}\}$ are assigned.

Algorithm 2 displays the basic steps of forming message nets and adding them to $\mathcal{H}_k$. For each processor group $\mathcal{P}_\ell$ that $\mathcal{P}_k$ communicates with, four different message nets may be added to $\mathcal{H}_k$: expand-send net, expand-receive net, fold-send net and fold-receive net, respectively denoted by s_ℓ^e, r_ℓ^e, s_ℓ^f and r_ℓ^f. Here, s and r respectively denote the messages sent and received, the subscript ℓ denotes the id of the processor group communicated with, and the superscripts e and f respectively denote the expand and fold operations. These nets are next explained in detail.

Algorithm 2 ADD-MESSAGE-NETS

Require: $\mathcal{H}_k = (\mathcal{V}_k, \mathcal{N}_k)$, $\Pi_{K'}(\mathcal{A}) = \{\mathcal{A}_1, \ldots, \mathcal{A}_{K'}\}$, $\Pi_{K'}(\mathcal{X}) = \{\mathcal{X}_1, \ldots, \mathcal{X}_{K'}\}$, $\Pi_{K'}(\mathcal{Y}) = \{\mathcal{Y}_1, \ldots, \mathcal{Y}_{K'}\}$.

```
 1: N_k^M ← N_k
    ▷ Expand-send nets
 2: for each x_j ∈ X_k do
 3:    for each a_{t,j} ∈ A_{ℓ≠k} do
 4:       if s_ℓ^e ∉ N_k^M then
 5:          Pins(s_ℓ^e) ← {v_j^x}, N_k^M ← N_k^M ∪ {s_ℓ^e}
 6:       else
 7:          Pins(s_ℓ^e) ← Pins(s_ℓ^e) ∪ {v_j^x}
    ▷ Expand-receive nets
 8: for each a_{t,j} ∈ A_k do
 9:    for each x_j ∈ X_{ℓ≠k} do
10:       if r_ℓ^e ∉ N_k^M then
11:          Pins(r_ℓ^e) ← {v_{t,j}^a}, N_k^M ← N_k^M ∪ {r_ℓ^e}
12:       else
13:          Pins(r_ℓ^e) ← Pins(r_ℓ^e) ∪ {v_{t,j}^a}
    ▷ Fold-send nets
14: for each a_{i,t} ∈ A_k do
15:    for each y_i ∈ Y_{ℓ≠k} do
16:       if s_ℓ^f ∉ N_k^M then
17:          Pins(s_ℓ^f) ← {v_{i,t}^a}, N_k^M ← N_k^M ∪ {s_ℓ^f}
18:       else
19:          Pins(s_ℓ^f) ← Pins(s_ℓ^f) ∪ {v_{i,t}^a}
    ▷ Fold-receive nets
20: for each y_i ∈ Y_k do
21:    for each a_{i,t} ∈ A_{ℓ≠k} do
22:       if r_ℓ^f ∉ N_k^M then
23:          Pins(r_ℓ^f) ← {v_i^y}, N_k^M ← N_k^M ∪ {r_ℓ^f}
24:       else
25:          Pins(r_ℓ^f) ← Pins(r_ℓ^f) ∪ {v_i^y}
26: return H_k^M = (V_k, N_k^M)
```

- **expand-send net** s_ℓ^e: Net s_ℓ^e represents the message sent from $\mathcal{P}_k$ to $\mathcal{P}_\ell$ during the expand operations on x-vector entries in the pre-communication phase. This message consists of the x-vector entries owned by $\mathcal{P}_k$ and needed by $\mathcal{P}_\ell$. Hence, s_ℓ^e connects the vertices that represent the x-vector entries required by the computational tasks in $\mathcal{P}_\ell$. That is,

$$Pins(s_\ell^e) = \{v_j^x : x_j \in \mathcal{X}_k \text{ and } \exists a_{t,j} \in \mathcal{A}_\ell\}.$$

The formation and addition of expand-send nets are performed in lines 2–7 of Algorithm 2. After bipartitioning $\mathcal{H}_k^M$, if s_ℓ^e becomes cut in $\Pi(\mathcal{H}_k^M)$, both $\mathcal{P}_{k,L}$ and $\mathcal{P}_{k,R}$ send a message to $\mathcal{P}_\ell$, where the contents of the messages sent from $\mathcal{P}_{k,L}$ and $\mathcal{P}_{k,R}$ to $\mathcal{P}_\ell$ are $\{x_j : v_j^x \in \mathcal{V}_{k,L}$ and $a_{t,j} \in \mathcal{A}_\ell\}$ and $\{x_j : v_j^x \in \mathcal{V}_{k,R}$ and $a_{t,j} \in \mathcal{A}_\ell\}$, respectively. The overall number of messages in the pre-communication phase increases by one in this case since $\mathcal{P}_k$ was sending a single message to $\mathcal{P}_\ell$ and it is split into two messages after bipartitioning. If s_ℓ^e becomes uncut, the overall number of messages does not change since only one of $\mathcal{P}_{k,L}$ and $\mathcal{P}_{k,R}$ sends a message to $\mathcal{P}_\ell$.

- **expand-receive net** r_ℓ^e: Net r_ℓ^e represents the message received by $\mathcal{P}_k$ from $\mathcal{P}_\ell$ during the expand operations on

x-vector entries in the pre-communication phase. This message consists of the x-vector entries owned by $\mathcal{P}_\ell$ and needed by $\mathcal{P}_k$. Hence, r^e_ℓ connects the vertices that represent the computational tasks requiring x-vector entries from $\mathcal{P}_\ell$. That is,

$$Pins(r^e_\ell) = \{v^a_{t,j} : a_{t,j} \in \mathcal{A}_k \text{ and } x_j \in \mathcal{X}_\ell\}.$$

The formation and addition of expand-receive nets are performed in lines 8–13 of Algorithm 2. After bipartitioning $\mathcal{H}^M_k$, if r^e_ℓ becomes cut in $\Pi(\mathcal{H}^M_k)$, both $\mathcal{P}_{k,L}$ and $\mathcal{P}_{k,R}$ receive a message from $\mathcal{P}_\ell$, where the contents of the messages received by $\mathcal{P}_{k,L}$ and $\mathcal{P}_{k,R}$ from $\mathcal{P}_\ell$ are $\{x_j : v^a_{t,j} \in \mathcal{V}_{k,L} \text{ and } x_j \in \mathcal{X}_\ell\}$ and $\{x_j : v^a_{t,j} \in \mathcal{V}_{k,R} \text{ and } x_j \in \mathcal{X}_\ell\}$, respectively. The overall number of messages in the pre-communication phase increases by one in this case and does not change if r^e_ℓ becomes uncut.

- **fold-send net** s^f_ℓ: Net s^f_ℓ represents the message sent from $\mathcal{P}_k$ to $\mathcal{P}_\ell$ during the fold operations on y-vector entries in the post-communication phase. This message consists of the partial results computed by $\mathcal{P}_k$ for the y-vector entries owned by $\mathcal{P}_\ell$. Hence, s^f_ℓ connects the vertices that represent the computational tasks whose partial results are required by $\mathcal{P}_\ell$. That is,

$$Pins(s^f_\ell) = \{v^a_{i,t} : a_{i,t} \in \mathcal{A}_k \text{ and } y_i \in \mathcal{Y}_\ell\}.$$

The formation and addition of fold-send nets are performed in lines 14–19 of Algorithm 2. After bipartitioning $\mathcal{H}^M_k$, if s^f_ℓ becomes cut in $\Pi(\mathcal{H}^M_k)$, both $\mathcal{P}_{k,L}$ and $\mathcal{P}_{k,R}$ send a message to $\mathcal{P}_\ell$, where the contents of the messages sent from $\mathcal{P}_{k,L}$ and $\mathcal{P}_{k,R}$ to $\mathcal{P}_\ell$ are $\{y^{(k,L)}_i : v^a_{i,t} \in \mathcal{V}_{k,L} \text{ and } y_i \in \mathcal{Y}_\ell\}$ and $\{y^{(k,R)}_i : v^a_{i,t} \in \mathcal{V}_{k,R} \text{ and } y_i \in \mathcal{Y}_\ell\}$, respectively. The overall number of messages in the post-communication phase increases by one in this case and does not change if s^f_ℓ becomes uncut.

- **fold-receive net** r^f_ℓ: Net r^f_ℓ represents the message received by $\mathcal{P}_k$ from $\mathcal{P}_\ell$ during the fold operations on y-vector entries in the post-communication phase. This message consists of the partial results computed by $\mathcal{P}_\ell$ for the y-vector entries owned by $\mathcal{P}_k$. Hence, r^f_ℓ connects the vertices that represent the y-vector entries for which $\mathcal{P}_\ell$ produces partial results. That is,

$$Pins(r^f_\ell) = \{v^y_i : y_i \in \mathcal{Y}_k \text{ and } \exists a_{i,t} \in \mathcal{A}_\ell\}.$$

The formation and addition of fold-receive nets are performed in lines 20–25 of Algorithm 2. After bipartitioning $\mathcal{H}^M_k$, if r^f_ℓ becomes cut in $\Pi(\mathcal{H}^M_k)$, both $\mathcal{P}_{k,L}$ and $\mathcal{P}_{k,R}$ receive a message from $\mathcal{P}_\ell$, where the contents of the messages received by $\mathcal{P}_{k,L}$ and $\mathcal{P}_{k,R}$ from $\mathcal{P}_\ell$ are $\{y^{(\ell)}_i : v^y_i \in \mathcal{V}_{k,L} \text{ and } a_{i,t} \in \mathcal{A}_\ell\}$ and $\{y^{(\ell)}_i : v^y_i \in \mathcal{V}_{k,R} \text{ and } a_{i,t} \in \mathcal{A}_\ell\}$, respectively. The overall number of messages in the post-communication phase increases by one in this case and does not change if r^f_ℓ becomes uncut.

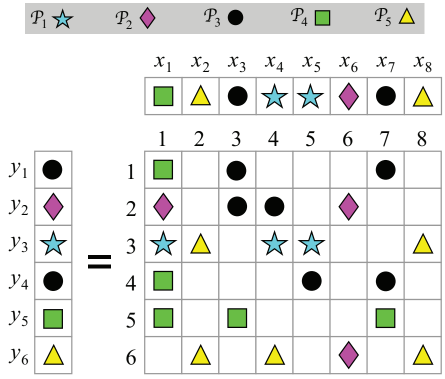

Figure 4: A 5-way nonzero-based partition of an SpMV instance $y = Ax$.

Note that at most four message nets are required to encapsulate the messages between processor groups $\mathcal{P}_k$ and $\mathcal{P}_\ell$. The message nets in $\mathcal{H}^M_k$ encapsulate all the messages that $\mathcal{P}_k$ communicates with other processor groups. Since the number of leaf hypergraphs is K', $\mathcal{P}_k$ may communicate with at most $K'-1$ processor groups, hence the maximum number of message nets that can be added to $\mathcal{H}_k$ is $4(K'-1)$.

Figure 4 displays an SpMV instance with a 6×8 matrix A, which is being partitioned by the proposed model. The RB process is at the state where there are five leaf hypergraphs $\mathcal{H}_1, \ldots, \mathcal{H}_5$, and the hypergraph to be bipartitioned next is $\mathcal{H}_3$. The figure displays the assignments of the matrix nonzeros and vector entries to the corresponding processor groups $\mathcal{P}_1, \ldots, \mathcal{P}_5$. Each symbol in the figure represents a distinct processor group and a symbol inside a cell signifies the assignment of the corresponding matrix nonzero or vector entry to the processor group represented by that symbol. For example, the nonzeros in $\mathcal{A}_3 = \{a_{1,3}, a_{1,7}, a_{2,3}, a_{2,4}, a_{4,5}, a_{4,7}\}$, x-vector entries in $\mathcal{X}_3 = \{x_3, x_7\}$, and y-vector entries in $\mathcal{Y}_3 = \{y_1, y_4\}$ are assigned to $\mathcal{P}_3$. The left of Figure 5 displays the augmented hypergraph $\mathcal{H}^M_3$ that contains volume and message nets. In the figure, the volume nets are illustrated by small black circles with thin lines, whereas the message nets are illustrated by the respective processor's symbol with thick lines.

The messages communicated by $\mathcal{P}_3$ under the assignments given in Figure 4 are displayed at the top half of Table 1. In the pre-communication phase, $\mathcal{P}_3$ sends a message to $\mathcal{P}_4$ and receives a message from $\mathcal{P}_1$, and in the post-communication phase, it sends a message to $\mathcal{P}_2$ and receives a message from $\mathcal{P}_4$. Hence, we add four message nets to $\mathcal{H}_3$: expand-send net s^e_4, expand-receive net r^e_1, fold-send net s^f_2, and fold-receive net r^f_4. In Figure 5, for example, r^e_1 connects the vertices $v^a_{2,4}$ and $v^a_{4,5}$ since it represents the message received by $\mathcal{P}_3$ from $\mathcal{P}_1$ containing $\{x_4, x_5\}$ due to nonzeros $a_{2,4}$ and $a_{4,5}$. The right of Figure 5 displays a bipartition $\Pi(\mathcal{H}^M_3)$ and the messages that $\mathcal{P}_{3,L}$ and $\mathcal{P}_{3,R}$ communicate with the other processor groups due to

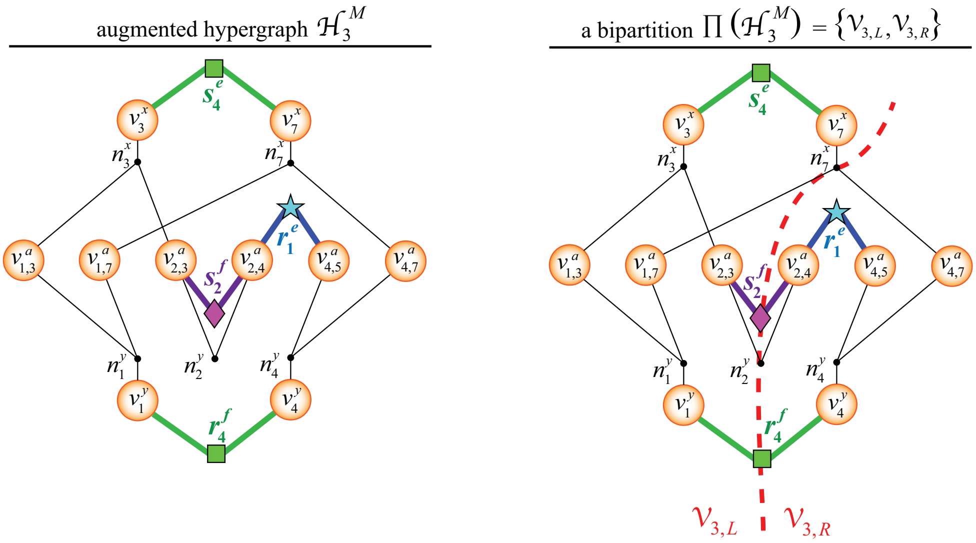

Figure 5: Left: Augmented hypergraph $\mathcal{H}_3^M$ with 5 volume and 4 message nets. Right: A bipartition $\Pi(\mathcal{H}_3^M)$ with two cut message nets (s_2^f, r_4^f) and two cut volume nets (n_7^x, n_2^y).

Table 1: The messages communicated by $\mathcal{P}_3$ in pre- and post-communication phases before and after bipartitioning $\mathcal{H}_3^M$. The number of messages communicated by $\mathcal{P}_3$ increases from 4 to 6 due to two cut message nets in $\Pi(\mathcal{H}_3^M)$.

RB state	phase	message	due to
before $\Pi(\mathcal{H}_3^M)$	pre	$\mathcal{P}_3$ sends $\{x_3, x_7\}$ to $\mathcal{P}_4$	$a_{5,3}, a_{5,7}$
		$\mathcal{P}_3$ receives $\{x_4, x_5\}$ from $\mathcal{P}_1$	$a_{2,4}, a_{4,5}$
	post	$\mathcal{P}_3$ sends $\{y_2^{(3)}\}$ to $\mathcal{P}_2$	$a_{2,3}, a_{2,4}$
		$\mathcal{P}_3$ receives $\{y_1^{(4)}, y_4^{(4)}\}$ from $\mathcal{P}_4$	$a_{1,1}, a_{4,1}$
after $\Pi(\mathcal{H}_3^M)$	pre	$\mathcal{P}_{3,L}$ sends $\{x_3, x_7\}$ to $\mathcal{P}_4$	$a_{5,3}, a_{5,7}$
		$\mathcal{P}_{3,R}$ receives $\{x_4, x_5\}$ from $\mathcal{P}_1$	$a_{2,4}, a_{4,5}$
	post	$\mathcal{P}_{3,L}$ sends $\{y_2^{(3,L)}\}$ to $\mathcal{P}_2$	$a_{2,3}$
		$\mathcal{P}_{3,R}$ sends $\{y_2^{(3,R)}\}$ to $\mathcal{P}_2$	$a_{2,4}$
		$\mathcal{P}_{3,L}$ receives $\{y_1^{(4)}\}$ from $\mathcal{P}_4$	$a_{1,1}$
		$\mathcal{P}_{3,R}$ receives $\{y_4^{(4)}\}$ from $\mathcal{P}_4$	$a_{4,1}$

$\Pi(\mathcal{H}_3^M)$ are given in the bottom half of Table 1. Since s_4^e and r_1^e are uncut, only one of $\mathcal{P}_{3,L}$ and $\mathcal{P}_{3,R}$ participates in sending or receiving the corresponding message. Since s_2^f is cut, both $\mathcal{P}_{3,L}$ and $\mathcal{P}_{3,R}$ send a message to $\mathcal{P}_2$, and since r_4^f is cut, both $\mathcal{P}_{3,L}$ and $\mathcal{P}_{3,R}$ receive a message from $\mathcal{P}_4$.

In $\mathcal{H}_k^M$, each volume net is assigned the cost of the per-word transfer time, t_w, whereas each message net is assigned the cost of the start-up latency, t_{su}. Let v and m respectively denote the number of volume and message nets that are cut in $\Pi(\mathcal{H}_k^M)$. Then,

$$cutsize(\Pi(\mathcal{H}_k^M)) = vt_w + mt_{su}.$$

Here, v is equal to the increase in the total communication volume incurred by $\Pi(\mathcal{H}_k^M)$ [1]. Recall that each cut message net increases the number of messages that $\mathcal{P}_k$ communicates with the respective processor group by one. Hence, m is equal to the increase in the number of messages that $\mathcal{P}_k$ communicates with other processor groups. The overall increase in the total message count due to $\Pi(\mathcal{H}_k^M)$ is $m + \delta$, where δ denotes the number of messages between $\mathcal{P}_{k,L}$ and $\mathcal{P}_{k,R}$, and is bounded by two (empirically found to be almost always two). Hence, minimizing the cutsize of $\Pi(\mathcal{H}_k^M)$ corresponds to simultaneously reducing the increase in the total communication volume and the total message count in the respective RB step. Therefore, minimizing the cutsize in all RB steps corresponds to reducing the total communication volume and the total message count simultaneously.

After obtaining a bipartition $\Pi(\mathcal{H}_k^M) = \{\mathcal{V}_{k,L}, \mathcal{V}_{k,R}\}$ of the augmented hypergraph $\mathcal{H}_k^M$, the new hypergraphs $\mathcal{H}_{k,L} = (\mathcal{V}_{k,L}, \mathcal{N}_{k,L})$ and $\mathcal{H}_{k,R} = (\mathcal{V}_{k,R}, \mathcal{N}_{k,R})$ are immediately formed with only volume nets. Recall that the formation of the volume nets of $\mathcal{H}_{k,L}$ and $\mathcal{H}_{k,R}$ is performed with the cut-net splitting technique and it can be performed using the local bipartition information $\Pi(\mathcal{H}_k^M)$.

3.2. The overall RB

After completing an RB step and obtaining $\mathcal{H}_{k,L}$ and $\mathcal{H}_{k,R}$, the labels of the hypergraphs represented by the leaf nodes of the RB tree are updated as follows. For $1 \leq i < k$, the label of $\mathcal{H}_i = (\mathcal{V}_i, \mathcal{N}_i)$ does not change. For $k < i < K'$, $\mathcal{H}_i = (\mathcal{V}_i, \mathcal{N}_i)$ becomes $\mathcal{H}_{i+1} = (\mathcal{V}_{i+1}, \mathcal{N}_{i+1})$. Hypergraphs $\mathcal{H}_{k,L} = (\mathcal{V}_{k,L}, \mathcal{N}_{k,L})$ and $\mathcal{H}_{k,R} = (\mathcal{V}_{k,R}, \mathcal{N}_{k,R})$ become $\mathcal{H}_k = (V_k, \mathcal{N}_k)$ and $\mathcal{H}_{k+1} = (\mathcal{V}_{k+1}, \mathcal{N}_{k+1})$, respectively. As a result, the vertex sets corresponding to the updated leaf nodes induce a $(K'+1)$-way partition $\Pi_{K'+1}(\mathcal{H}) = \{\mathcal{V}_1, \ldots, \mathcal{V}_{K'+1}\}$. The RB process then continues with the next hypergraph $\mathcal{H}_{k+2}$ to be bipartitioned, which was labeled with $\mathcal{H}_{k+1}$ in the previous RB state.

We next provide the cost of adding message nets through Algorithm 2 in the entire RB process. For the addition of expand-send nets, all nonzeros $a_{t,j} \in \mathcal{A}_{\ell \neq k}$ with $x_j \in \mathcal{X}_k$ are visited once (lines 2–7). Since $\mathcal{X}_k \cap \mathcal{X}_\ell = \emptyset$ for $1 \leq k \neq \ell \leq K'$ and

$\mathcal{X} = \bigcup_{k=1}^{K'} \mathcal{X}_k$, each nonzero of A is visited once. For the addition of expand-receive nets, all nonzeros in $\mathcal{A}_k$ are visited once (lines 8–13). Hence, each nonzero of A is visited once during the bipartitionings in a level of the RB tree since $\mathcal{A}_k \cap \mathcal{A}_\ell = \emptyset$ for $1 \le k \ne \ell \le K'$ and $\mathcal{A} = \bigcup_{k=1}^{K'} \mathcal{A}_k$. Therefore, the cost of adding expand-send and expand-receive nets is $O(n_{nz})$ in a single level of the RB tree. A dual discussion holds for the addition of fold-send and fold-receive nets. Since the RB tree contains $\lceil \log K \rceil$ levels in which bipartitionings take place, the overall cost of adding message nets is $O(n_{nz} \log K)$.

3.3. Adaptation for conformal partitioning

Partitions on input and output vectors x and y are said to be conformal if x_i and y_i are assigned to the same processor, for $1 \le i \le n_r = n_c$. Note that conformal vector partitions are valid for $y = Ax$ with a square matrix. The motivation for a conformal partition arises in iterative solvers in which the y_i in an iteration is used to compute the x_i of the next iteration via linear vector operations. Assigning x_i and y_i to the same processor prevents the redundant communication of y_i to the processor that owns x_i.

Our model does not impose conformal partitions on vectors x and y, i.e., x_i and y_i can be assigned to different processors. However, it is possible to adapt our model to obtain conformal partitions on x and y using the vertex amalgamation technique proposed in [9]. To assign x_i and y_i to the same processor, the vertices v_i^x and v_i^y are amalgamated into a new vertex $v_i^{x/y}$, which represents both x_i and y_i. The weight of $v_i^{x/y}$ is set to be zero since the weights of v_i^x and v_i^y are zero. In $\mathcal{H}_k^M$, each volume/message net that connects v_i^x or v_i^y now connects the amalgamated vertex $v_i^{x/y}$. At each RB step, x_i and y_i are both assigned to the processor group corresponding to the leaf hypergraph that contains $v_i^{x/y}$.

4. Optimizing medium-grain partitioning model

In this section, we propose a medium-grain hypergraph partitioning model that simultaneously reduces the bandwidth and latency costs of the row-column-parallel SpMV. Our model is built upon the original medium-grain partitioning model (Section 2.4). The medium-grain hypergraphs in RB are augmented with the message nets before they are bipartitioned as in the fine-grain model proposed in Section 3. Since the fine-grain and medium-grain models both obtain nonzero-based partitions, the types and meanings of the message nets used in the medium-grain model are the same as those used in the fine-grain model. However, forming message nets for a medium-grain hypergraph is more involved due to the mappings used in this model.

Consider an SpMV instance $y = Ax$ and the corresponding sets $\mathcal{A}$, $\mathcal{X}$, and $\mathcal{Y}$. Assume that the RB process is at the state before bipartitioning the kth leaf node where there are K' leaf nodes in the current RB tree. Recall from Section 2.4 that the leaf nodes induce K'-way partitions $\Pi_{K'}(\mathcal{A}) = \{\mathcal{A}_1, \ldots, \mathcal{A}_{K'}\}$, $\Pi_{K'}(\mathcal{X}) = \{\mathcal{X}_1, \ldots, \mathcal{X}_{K'}\}$ and $\Pi_{K'}(\mathcal{Y}) = \{\mathcal{Y}_1, \ldots, \mathcal{Y}_{K'}\}$, and the kth leaf node represents $\mathcal{A}_k$, $\mathcal{X}_k$, and $\mathcal{Y}_k$. To obtain bipartitions of $\mathcal{A}_k$, $\mathcal{X}_k$, and $\mathcal{Y}_k$, we perform the following four steps.

1) Form the medium-grain hypergraph $\mathcal{H}_k = (\mathcal{V}_k, \mathcal{N}_k)$ using $\mathcal{A}_k$, $\mathcal{X}_k$, and $\mathcal{Y}_k$. This process is the same with that in the original medium-grain model (Section 2.4). Recall that the nets in the medium-grain hypergraph encapsulate the total communication volume. Hence, these nets are assigned a cost of t_w.

2) Add message nets to $\mathcal{H}_k$ to obtain augmented hypergraph $\mathcal{H}_k^M$. For each processor group $\mathcal{P}_\ell$ other than $\mathcal{P}_k$, there are four possible message nets that can be added to $\mathcal{H}_k$:

- **expand-send net** s_ℓ^e: The set of vertices connected by s_ℓ^e is the same with that of the expand-send net in the fine-grain model.
- **expand-receive net** r_ℓ^e: The set of vertices connected by r_ℓ^e is given by

$$
\begin{aligned}
Pins(r_\ell^e) = &\{v_j^x : \exists a_{t,j} \in \mathcal{A}_k \text{ s.t. } map(a_{t,j}) = c_j \text{ and } x_j \in \mathcal{X}_\ell\} \cup \\
&\{v_t^y : \exists a_{t,j} \in \mathcal{A}_k \text{ s.t. } map(a_{t,j}) = r_t \text{ and } x_j \in \mathcal{X}_\ell\}.
\end{aligned}
$$

- **fold-send net** s_ℓ^f: The set of vertices connected by s_ℓ^f is given by

$$
\begin{aligned}
Pins(s_\ell^f) = &\{v_t^x : \exists a_{i,t} \in \mathcal{A}_k \text{ s.t. } map(a_{i,t}) = c_t \text{ and } y_i \in \mathcal{Y}_\ell\} \cup \\
&\{v_i^y : \exists a_{i,t} \in \mathcal{A}_k \text{ s.t. } map(a_{i,t}) = r_i \text{ and } y_i \in \mathcal{Y}_\ell\}.
\end{aligned}
$$

- **fold-receive net** r_ℓ^f: The set of vertices connected by r_ℓ^f is the same with that of the fold-receive net in the fine-grain model.

The message nets are assigned a cost of t_{su} as they encapsulate the latency cost.

3) Obtain a bipartition $\Pi(\mathcal{H}_k^M)$. $\mathcal{H}_k^M$ is bipartitioned to obtain $\Pi(\mathcal{H}_k^M) = \{\mathcal{V}_{k,L}, \mathcal{V}_{k,R}\}$.

4) Derive bipartitions $\Pi(\mathcal{A}_k) = \{\mathcal{A}_{k,L}, \mathcal{A}_{k,R}\}$, $\Pi(\mathcal{X}_k) = \{\mathcal{X}_{k,L}, \mathcal{X}_{k,R}\}$ and $\Pi(\mathcal{Y}_k) = \{\mathcal{Y}_{k,L}, \mathcal{Y}_{k,R}\}$ from $\Pi(\mathcal{H}_k^M)$. For each nonzero $a_{i,j} \in \mathcal{A}_k$, $a_{i,j}$ is assigned to $\mathcal{A}_{k,L}$ if the vertex that represents $a_{i,j}$ is in $\mathcal{V}_{k,L}$, and to $\mathcal{A}_{k,R}$, otherwise. That is,

$$
\begin{aligned}
\mathcal{A}_{k,L} = \{a_{i,j} : \; &map(a_{i,j}) = c_j \text{ with } v_j^x \in \mathcal{V}_{k,L} \text{ or} \\
&map(a_{i,j}) = r_i \text{ with } v_i^y \in \mathcal{V}_{k,L}\} \text{ and} \\
\mathcal{A}_{k,R} = \{a_{i,j} : \; &map(a_{i,j}) = c_j \text{ with } v_j^x \in \mathcal{V}_{k,R} \text{ or} \\
&map(a_{i,j}) = r_i \text{ with } v_i^y \in \mathcal{V}_{k,R}\}.
\end{aligned}
$$

For each x-vector entry $x_j \in \mathcal{X}_k$, x_j is assigned to $\mathcal{X}_{k,L}$ if $v_j^x \in \mathcal{V}_{k,L}$, and to $\mathcal{X}_{k,R}$, otherwise. That is,

$$\mathcal{X}_{k,L} = \{x_j : v_j^x \in \mathcal{V}_{k,L}\} \text{ and } \mathcal{X}_{k,R} = \{x_j : v_j^x \in \mathcal{V}_{k,R}\}.$$

Similarly, for each y-vector entry $y_i \in \mathcal{Y}_k$, y_i is assigned to $\mathcal{Y}_{k,L}$ if $v_i^y \in \mathcal{V}_{k,L}$, and to $\mathcal{Y}_{k,R}$, otherwise. That is,

$$\mathcal{Y}_{k,L} = \{y_i : v_i^y \in \mathcal{V}_{k,L}\} \text{ and } \mathcal{Y}_{k,R} = \{y_i : v_i^y \in \mathcal{V}_{k,R}\}.$$

Figure 6 displays the medium-grain hypergraph $\mathcal{H}_3^M = (\mathcal{V}_3, \mathcal{N}_3^M)$ augmented with message nets, which is formed during bipartitioning $\mathcal{A}_3$, $\mathcal{X}_3$ and $\mathcal{Y}_3$ given in Figure 4. The table in the figure displays $map(a_{i,j})$ value for each nonzero in $\mathcal{A}_3$ computed by the heuristic described in Section 2.4. Augmented medium-grain hypergraph $\mathcal{H}_3$ has four message nets.

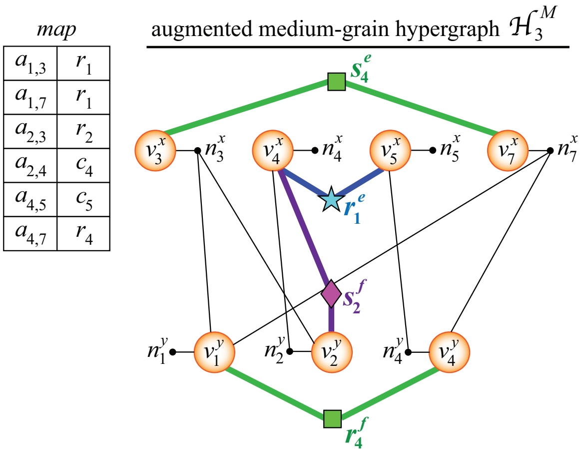

Figure 6: The augmented medium-grain hypergraph $\mathcal{H}_3^M$ formed during the RB process for the SpMV instance given in Figure 4.

Observe that the sets of vertices connected by expand-send net s_4^e and fold-receive net r_4^f are the same for the fine-grain and medium-grain hypergraphs, which are respectively illustrated in Figures 5 and 6. Expand-receive net r_1^e connects v_4^x and v_5^x since $\mathcal{P}_3$ receives $\{x_4, x_5\}$ due to nonzeros in $\{a_{2,4}, a_{4,5}\}$ with $map(a_{2,4}) = c_4$ and $map(a_{4,5}) = c_5$. Fold-send net s_2^f connects v_4^x and v_2^y since $\mathcal{P}_3$ sends partial result $y_2^{(3)}$ due to nonzeros in $\{a_{2,3}, a_{2,4}\}$ with $map(a_{2,3}) = r_2$ and $map(a_{2,4}) = c_4$.

Similar to Section 3, after obtaining bipartitions $\Pi(\mathcal{A}_k) = \{\mathcal{A}_{k,L}, \mathcal{A}_{k,R}\}$, $\Pi(\mathcal{X}_k) = \{\mathcal{X}_{k,L}, \mathcal{X}_{k,R}\}$, and $\Pi(\mathcal{Y}_k) = \{\mathcal{Y}_{k,L}, \mathcal{Y}_{k,R}\}$, the labels of the parts represented by the leaf nodes are updated in such a way that the resulting $(K'+1)$-way partitions are denoted by $\Pi_{K'+1}(\mathcal{A}) = \{\mathcal{A}_1, \ldots, \mathcal{A}_{K'+1}\}$, $\Pi_{K'+1}(\mathcal{X}) = \{\mathcal{X}_1, \ldots, \mathcal{X}_{K'+1}\}$, and $\Pi_{K'}(\mathcal{Y}) = \{\mathcal{Y}_1, \ldots, \mathcal{Y}_{K'+1}\}$.

4.1. Adaptation for conformal partitioning

Adapting the medium-grain model for a conformal partition on vectors x and y slightly differs from adapting the fine-grain model. Vertex set $\mathcal{V}_k$ contains an amalgamated vertex $v_i^{x/y}$ if at least one of the following conditions holds:

- $x_i \in \mathcal{X}_k$, or equivalently, $y_i \in \mathcal{Y}_k$.
- $\exists a_{t,i} \in \mathcal{A}_k$ s.t. $map(a_{t,i}) = c_i$.
- $\exists a_{i,t} \in \mathcal{A}_k$ s.t. $map(a_{i,t}) = r_i$.

The weight of v_i is assigned as

$$w(v_i) = |\{a_{t,i} : a_{t,i} \in \mathcal{A}_k \text{ and } map(a_{t,i}) = c_i\}| + |\{a_{i,t} : a_{i,t} \in \mathcal{A}_k \text{ and } map(a_{i,t}) = r_i\}|.$$

Each volume/message net that connects v_i^x or v_i^y in $\mathcal{H}_k^M$ now connects the amalgamated vertex $v_i^{x/y}$.

5. Delayed addition and thresholding for message nets

Utilization of the message nets decreases the importance attributed to the volume nets in the partitioning process and this may lead to a relatively high bandwidth cost compared to the case where no message nets are utilized. The more the number of RB steps in which the message nets are utilized, the higher the total communication volume. A high bandwidth cost can especially be attributed to the bipartitionings in the early levels of the RB tree. There are only a few nodes in the early levels of the RB tree compared to the late levels and each of these nodes represents a large processor group. The messages among these large processor groups are difficult to refrain from. In terms of hypergraph partitioning, since the message nets in the hypergraphs at the early levels of the RB tree connect more vertices and the cost of the message nets is much higher than the cost of the volume nets ($t_{su} \gg t_w$), it is very unlikely for these message nets to be uncut. While the partitioner tries to save these nets from the cut in the early bipartitionings, it may cause high number of volume nets to be cut, which in turn are likely to introduce new messages in the late levels of the RB tree. Therefore, adding message nets in the early levels of the RB tree adversely affects the overall partition quality in multiple ways.

The RB approach provides the ability to adjust the partitioning parameters in the individual RB steps for the sake of the overall partition quality. In our model, we use this flexibility to exploit the trade-off between the bandwidth and latency costs by selectively deciding whether to add message nets in each bipartitioning. To make this decision, we use the level information of the RB steps in the RB tree. For a given $L < \log K$, the addition of the message nets is delayed until the Lth level of the RB tree, i.e., the bipartitionings in level ℓ are performed only with the volume nets for $0 \leq \ell < L$. Thus, the message nets are included in the bipartitionings in which they are expected to connect relatively fewer vertices.

Using a delay parameter L aims to avoid large message nets by not utilizing them in the early levels of the RB tree. However, there may still exist such nets in the late levels depending on the structure of the matrix being partitioned. Another idea is to eliminate the message nets whose size is larger than a given threshold. That is, for a given threshold $T > 0$, a message net n with $|Pins(n)| > T$ is excluded from the corresponding bipartition. This approach also enables a selective approach for send and receive message nets. In our implementation of the row-column-parallel SpMV, the receive operations are performed by non-blocking MPI functions (i.e., `MPI_Irecv`), whereas the send operations are performed by blocking MPI functions (i.e., `MPI_Send`). When the maximum message count or the maximum communication volume is considered to be a serious bottleneck, blocking send operations may be more limiting compared to non-blocking receive operations. Note that saving message nets from the cut tends to assign the respective communication operations to fewer processors, hence the maximum message count and maximum communication volume may increase. Hence, a smaller threshold is preferable for the send message nets while a higher threshold is preferable for the receive nets.

6. Experiments

We consider a total of five partitioning models for evaluation. Four of them are nonzero-based partitioning models: the fine-grain model (FG), the medium-grain model (MG), and the proposed models which simultaneously reduce the bandwidth and latency costs, as described in Section 3 (FG-LM) and Section 4 (MG-LM). The last partitioning model tested is the one-dimensional model (1D-LM) that simultaneously reduces the bandwidth and latency costs [17]. Two of these five models (FG and MG) encapsulate a single communication cost metric, i.e., total volume, while three of them (FG-LM, MG-LM, and 1D-LM) encapsulate two communication cost metrics, i.e., total volume and total message count. The partitioning constraint of balancing part weights in all these models corresponds to balancing of the computational loads of processors. In the models that address latency cost with the message nets, the cost of the volume nets is set to 1 while the cost of the message nets is set to 50, i.e., it is assumed $t_{su} = 50t_w$, which is also the setting recommended in [17].

The performance of the compared models are evaluated in terms of the partitioning cost metrics and the parallel SpMV runtime. The partitioning cost metrics include total volume, total message count, load imbalance, etc. (these are explained in detail in following sections) and they are helpful to test the validity of the proposed models. The hypergraphs in all models are partitioned using PaToH [1] in the default settings. An imbalance ratio of 10% is used in all models, i.e., $\epsilon = 0.10$. We test for five different number of parts/processors, $K \in \{64, 128, 256, 512, 1024\}$. The parallel SpMV is implemented using the PETSc toolkit [22] and run on a Blue Gene/Q system using the partitions provided by these five models. A node on Blue Gene/Q system consists of 16 PowerPC A2 processors with 1.6 GHz clock frequency and 16 GB memory.

The experiments are performed on an extensive dataset containing matrices from the SuiteSparse Matrix Collection [23]. We consider the case of conformal vector partitioning as it is more common for the applications in which SpMV is use as a kernel operation. Hence, only the square matrices are considered. We use the following criteria for the selection of test matrices: (i) the minimum and maximum number of nonzeros per processor are respectively set to 100 and 100,000, (ii) the matrices that have more than 50 million nonzeros are excluded, and (iii) the minimum number of rows/columns per processor is set to 50. The resulting number of matrices are 833, 730, 616, 475, and 316 for K = 64, 128, 256, 512, and 1024 processors, respectively. The union of these sets of matrices makes up to a total of 978 matrices.

6.1. Tuning parameters for nonzero-based partitioning models

There are two important issues described in Section 5 regarding the addition of the message nets for the nonzero-based partitioning models. We next discuss setting these parameters.

6.1.1. Delay parameter (L)

We investigate the effect of the delay parameter L on four different communication cost metrics for the fine-grain and medium-grain models with the message nets. These cost metrics are maximum volume, total volume, maximum message count, and total message count. The volume metrics are in terms of number of words communicated. We compare FG-LM with delay against FG, as well as MG-LM with delay against MG. We only present the results for $K = 256$ since the observations made for the results of different K values are similar. Note that there are $\log 256 = 8$ bipartitioning levels in the corresponding RB tree. The tested values of the delay parameter L are 1, 4, 5, 6, and 7. Note that the message nets are added in a total of 4, 3, 2, and 1 levels for the L values of 4, 5, 6, and 7, respectively. When $L = 1$, it is equivalent to adding message nets throughout the whole partitioning without any delay. Note that it is not possible to add message nets at the root level (i.e., by setting $L = 0$) since there is no partition available yet to form the message nets. The results for the remaining values of L are not presented as the tested values contain all the necessary insight for picking a value for L. Table 2 presents the results obtained. The value obtained by a partitioning model for a specific cost metric is the geometric mean of the values obtained for the matrices by that partitioning model (i.e., the mean of the results for 616 matrices). We also present two plots in Figure 7 to provide a visual comparison of the values presented in Table 2. The plot at the top belongs to the fine-grain models and each different cost metric is represented by a separate line in which the values are normalized with respect to those of the standard fine-grain model FG. Hence, a point on a line below $y = 1$ indicates the variants of FG-LM attaining a better performance in the respective metric compared to FG, whereas a point in a line above indicates a worse performance. For example, FG-LM with $L = 7$ attains 0.72 times the total message count of FG, which corresponds to the second point of the line marked with a filled circle. The plot at the bottom compares the medium-grain models in a similar fashion.

Table 2: The communication cost metrics obtained by the nonzero-based partitioning models with varying delay values (L).

		volume		message	
model	L	max	total	max	total
FG	-	567	52357	60	5560
FG-LM	1	2700	96802	56	2120
FG-LM	4	2213	94983	49	2186
FG-LM	5	1818	90802	46	2317
FG-LM	6	1346	82651	46	2694
FG-LM	7	926	69572	49	3574
MG	-	558	49867	57	5103
MG-LM	1	1368	77479	50	2674
MG-LM	4	1264	77227	48	2735
MG-LM	5	1148	74341	47	2809
MG-LM	6	969	69159	47	3066
MG-LM	7	776	61070	50	3695

It can be seen from Figure 7 that, compared to FG, FG-LM attains better performance in maximum and total message count, and a worse performance in maximum and total volume. A similar observation is also valid for comparing MG with MG-LM.

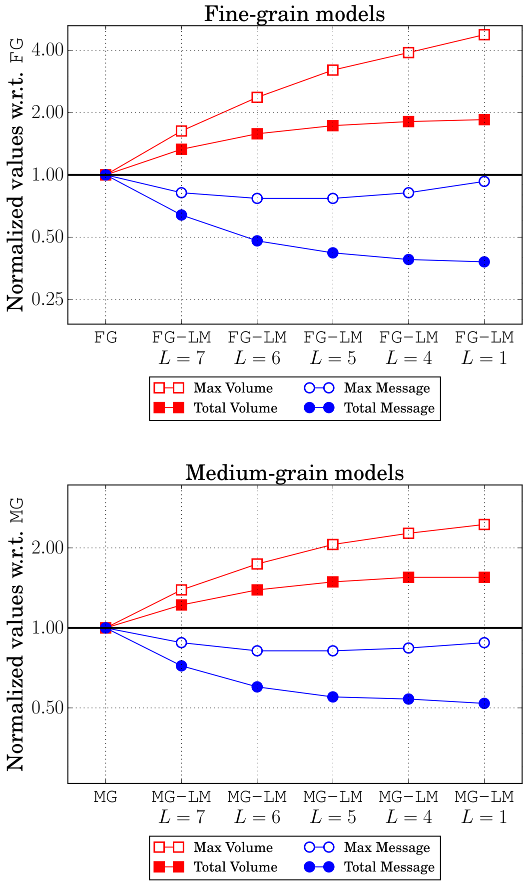

Figure 7: The effect of the delay parameter on nonzero-based partitioning models in four different communication metrics.

As the number of RB tree levels in which the message nets are added increases, FG-LM and MG-LM obtain lower latency and higher bandwidth overheads compared to FG and MG, respectively. The improvement rates in latency cost obtained by the partitioning models utilizing the message nets saturate around $L = 6$ or $L = 5$, whereas the deterioration rates in bandwidth cost continue to increase. In other words, adding message nets in the bipartitionings other than those in the last two or three levels of the RB tree has small benefits in terms of improving the latency cost but it has a substantial negative effect on the bandwidth cost, especially on maximum volume. For this reason, we choose FG-LM and MG-LM with $L = 6$, i.e., add message nets in the last two levels of the RB tree.

6.1.2. Message net threshold parameters (T_S,T_R)

The message net threshold parameters for the send and receive message nets are respectively denoted with T_S and T_R. The tested values are set based upon the average degree of the message nets throughout the partitioning, which is found to be close to 30. We evaluate threshold values smaller than, roughly equal to, and greater than this average degree: $T_S, T_R \in \{15, 30, 50\}$. We follow a similar experimental setting as for the delay parameter and only present the results for $K = 256$. In addition, we omit the discussions for the medium-grain models as the observations made for the fine-grain and medium-grain models are alike. Table 3 presents the values for four different cost metrics obtained by FG-LM and FG-LM with nine different threshold settings. Note that the delay value of $L = 6$ is utilized in all these experiments.

Table 3: The communication cost metrics of FG-LM with varying message net thresholds (T_S,T_R).

		volume		message	
T_S	T_R	max	total	max	total
-	-	1346	82651	46	2694
15	15	706	56218	58	4539
15	30	773	58452	56	4258
15	50	835	60864	54	4043
30	15	793	58418	59	4251
30	30	827	60086	57	4087
30	50	900	62393	55	3879
50	15	879	61099	59	4037
50	30	908	62516	58	3877
50	50	952	64041	56	3729

The partitionings without large message nets lead to lower bandwidth and higher latency costs as seen in Table 3 compared to the case without any threshold, i.e., FG-LM. The more the number of eliminated message nets, the higher the latency cost and the lower the bandwidth cost. Among the nine combinations for T_S and T_R in the table, we pick $T_S = 15$ and $T_R = 50$ due to its reasonable maximum volume and maximum message count values for the reasons described in Section 5.

6.2. Comparison of all partitioning models

6.2.1. Partitioning cost metrics

We present the values obtained by the four nonzero-based partitioning models in six different partitioning cost metrics in Table 4. These cost metrics are computational imbalance (indicated in the column titled "imb (%)"), maximum and total volume, maximum and total message count, and partitioning time in seconds. Each entry in the table is the geometric mean of the values for the matrices that belong to the respective value of K. The columns three to eight in the table display the actual values, whereas the columns nine to fourteen display the normalized values, where the results obtained by FG-LM and MG-LM at each K value are normalized with respect to those obtained by FG and MG at that K value, respectively. The top half of the table displays the results obtained by the fine-grain models, whereas the bottom half displays the results obtained by the medium-grain models.

Among the four nonzero-based partitioning models compared in Table 4, the models that consider both the bandwidth and latency overheads achieve better total and maximum message counts compared to the models that solely consider the bandwidth overhead. For example at $K = 256$, FG-LM attains 27% improvement in total message count compared to FG,

Table 4: Comparison of nonzero-based partitioning models in six cost metrics.

		actual values						normalized values w.r.t. FG/MG					
			volume		message				volume		message		
K	model	imb (%)	max	total	max	total	part. time	imb	max	total	max	total	part. time
64	FG	0.91	413	11811	32	968	7.7	-	-	-	-	-	-
	FG-LM	0.88	542	13267	29	753	7.4	0.97	1.31	1.12	0.91	0.78	0.97
128	FG	1.11	484	24670	45	2332	16.4	-	-	-	-	-	-
	FG-LM	1.01	669	28159	40	1751	16.3	0.91	1.38	1.14	0.89	0.75	1.00
256	FG	1.36	567	52357	60	5560	40.9	-	-	-	-	-	-
	FG-LM	1.21	835	60864	54	4043	40.8	0.89	1.47	1.16	0.90	0.73	1.00
512	FG	1.67	584	92141	72	11186	77.9	-	-	-	-	-	-
	FG-LM	1.61	863	108497	66	8218	77.2	0.96	1.48	1.18	0.92	0.73	0.99
1024	FG	1.87	530	165923	69	20209	156.2	-	-	-	-	-	-
	FG-LM	1.81	811	196236	66	15415	159.6	0.97	1.53	1.18	0.96	0.76	1.02
64	MG	0.90	412	11655	31	928	3.9	-	-	-	-	-	-
	MG-LM	0.87	521	13205	28	732	4.1	0.97	1.26	1.13	0.90	0.79	1.06
128	MG	1.13	482	24256	44	2217	8.1	-	-	-	-	-	-
	MG-LM	1.08	634	27799	39	1690	8.4	0.96	1.32	1.15	0.89	0.76	1.04
256	MG	1.48	558	49867	57	5103	19.1	-	-	-	-	-	-
	MG-LM	1.39	766	58981	52	3876	20.6	0.94	1.37	1.18	0.91	0.76	1.08
512	MG	1.91	588	91856	67	10265	39.7	-	-	-	-	-	-
	MG-LM	1.80	785	108128	62	7878	43.7	0.94	1.34	1.18	0.93	0.77	1.10
1024	MG	2.05	530	165722	65	18692	82.2	-	-	-	-	-	-
	MG-LM	2.00	724	196443	61	14827	87.5	0.98	1.37	1.19	0.94	0.79	1.06

while MG-LM attains 24% improvement in total message count compared to MG. On the other hand, the two models that solely consider the bandwidth overhead achieve better total and maximum volume compared to the two models that also consider the latency overhead. This is because FG and MG optimize a single cost metric, while FG-LM and MG-LM aim to optimize two cost metrics at once. At $K = 256$, FG-LM causes 16% deterioration in total volume compared to FG, while MG-LM causes 18% deterioration in total volume compared to MG. Note that the models behave accordingly in maximum volume and maximum message count metrics as although these metrics are not directly addressed by any of the models, the former one is largely dependent on the total volume while the latter one is largely dependent on the total message count. FG-LM and MG-LM have slightly lower imbalance compared to FG and MG, respectively. Addition of the message nets does not seem to change the partitioning overhead, a result likely to be a consequence of the choice of the delay and net threshold parameters.

Another observation worth discussion is the performance of the medium-grain models against the performance of the fine-grain models. When MG is compared to FG or MG-LM is compared to FG-LM, the medium-grain models achieve slightly better results in volume and message cost metrics, and slightly worse results in imbalance. However, the partitioning overhead of the medium-grain models is much lower than the partitioning overhead of the fine-grain models: the medium grain models are 1.8-2.2x faster. This is also one of the main findings of [10], which makes the medium-grain model a better alternative for obtaining nonzero-based partitions.

Table 5: Comparison of partitioning models in six cost metrics at $K = 256$.

		volume		message		
model	imb (%)	max	total	max	total	part. time
1D-LM	2.50	968	101565	33	2448	13.2
FG	1.36	567	52357	60	5560	40.9
FG-LM	1.21	835	60864	54	4043	40.8
MG	1.48	558	49867	57	5103	19.1
MG-LM	1.39	766	58981	52	3876	20.6

1D-LM and nonzero-based partitioning models are compared in Table 5 at $K = 256$. 1D-LM has higher total volume and imbalance, and lower total message count compared to the nonzero-based partitioning models. The nonzero-based models have broader search space due to their representation of the SpMV via smaller units, which allows them to attain better volume and imbalance. The latency overheads of FG and MG are higher than the latency overhead of 1D-LM simply because latency is not addressed in the former two. Although FG-LM and MG-LM may as well obtain comparable latency overheads with 1D-LM (e.g., compare total message count of FG-LM with $L = 1$ in Table 2 against total message count of 1D-LM in Table 5), we favor a decrease in volume-related cost metrics at the expense of a small deterioration in latency-related cost metrics in these two models. 1D-LM has the lowest partitioning overhead due to having the smallest hypergraph among the five models. A similar discussion follows for the maximum volume and maximum message count metrics as for the total volume and total message count metrics.

In the rest of the paper, we use MG and MG-LM among the nonzero-based models for evaluation due to their lower partitioning overhead and slightly better performance compared to FG and FG-LM, respectively, in the remaining metrics.

6.2.2. Parallel SpMV performance

We compare 1D-LM, MG, and MG-LM in terms of parallel SpMV runtime. Parallel SpMV is run with the partitions obtained through these three models. There are 12 matrices tested, listed with their types as follows: eu-2005 (web graph), ford2 (mesh), Freescale1 (circuit simulation), invextr1_new (computational fluid dynamics), k1_san (2D/3D), LeGresley_87936 (power network), mouse_gene (gene network), olesnik0 (2D/3D), tuma1 (2D/3D), turon_m (2D/3D), usroads (road network), web-Google (web graph). Number of nonzeros in these matrices varies between 87,760 and 28,967,291. These 12 matrices are the subset of 978 matrices for which the partitioning models are compared in terms of partitioning cost metrics in the preceding sections. Four different number of processors (i.e., K) are tested: 64, 128, 256, and 512. We did not test for 1024 processors as in most of the tested matrices SpMV could not scale beyond 512 processors. We only consider the strong-scaling case. The parallel SpMV is run for 100 times and the average runtime (in milliseconds) is reported. The obtained results are presented in Figure 8.

The plots in Figure 8 show that both MG and MG-LM scale usually better than 1D-LM. It is known the nonzero-based partitioning models scale better than the 1D models due to their lower communication overheads and computational imbalance. In difficult instances such as invextr1_new or mouse_gene at which 1D-LM does not scale, using a nonzero-based model such as MG or MG-LM successfully scales the parallel SpMV. MG-LM improves the scalability of MG in most of the test instances. Apart from the instances Freescale1, invextr1_new, and turon_m, MG-LM performs significantly better than MG. MG-LM's performance especially gets more prominent with increasing number of processors, which is due to the fact that the latency overheads are more critical in the overall communication costs in high processor counts since the message size usually decreases with increasing number of processors. These plots show that using a nonzero-based partitioning model coupled with the addressing of multiple communication cost metrics yields the best parallel SpMV performance.

7. Conclusion

We proposed two novel nonzero-based matrix partitioning models, a fine-grain and a medium-grain model, that simultaneously address the bandwidth and latency costs of parallel SpMV. These models encapsulate two communication cost metrics at once as opposed to their existing counterparts which only address a single cost metric regarding the bandwidth cost. Our approach exploits the recursive bipartitioning paradigm to incorporate the latency minimization into the partitioning objective via message nets. In addition, we proposed two practical enhancements to find a good balance between reducing the bandwidth and the latency costs. The experimental results obtained on an extensive dataset show that the proposed models attain up to 27% improvement in latency-related cost metrics over their existing counterparts on average and the scalability of parallel SpMV can substantially be improved with the proposed models.

Acknowledgment

We acknowledge PRACE for awarding us access to resources Juqueen (Blue Gene/Q) based in Germany at Jülich Supercomputing Centre.

References

[1] U. V. Çatalyürek, C. Aykanat, Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication, Parallel and Distributed Systems, IEEE Transactions on 10 (7) (1999) 673–693. doi:10.1109/71.780863.

[2] B. Hendrickson, Graph partitioning and parallel solvers: Has the emperor no clothes? (extended abstract), in: Proceedings of the 5th International Symposium on Solving Irregularly Structured Problems in Parallel, IRREGULAR '98, Springer-Verlag, London, UK, UK, 1998, pp. 218–225. URL http://dl.acm.org/citation.cfm?id=646012.677019

[3] B. Hendrickson, T. G. Kolda, Graph partitioning models for parallel computing, Parallel Comput. 26 (12) (2000) 1519–1534. doi:10.1016/S0167-8191(00)00048-X. URL http://dx.doi.org/10.1016/S0167-8191(00)00048-X

[4] B. Hendrickson, T. G. Kolda, Partitioning rectangular and structurally unsymmetric sparse matrices for parallel processing, SIAM J. Sci. Comput. 21 (6) (1999) 2048–2072. doi:10.1137/S1064827598341475. URL http://dx.doi.org/10.1137/S1064827598341475

[5] G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, SIAM J. Sci. Comput. 20 (1) (1998) 359–392. doi:10.1137/S1064827595287997. URL http://dx.doi.org/10.1137/S1064827595287997

[6] K. Schloegel, G. Karypis, V. Kumar, Parallel multilevel algorithms for multi-constraint graph partitioning, in: A. Bode, T. Ludwig, W. Karl, R. Wismller (Eds.), Euro-Par 2000 Parallel Processing, Vol. 1900 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2000, pp. 296–310. doi:10.1007/3-540-44520-X_39. URL http://dx.doi.org/10.1007/3-540-44520-X_39

[7] B. Vastenhouw, R. H. Bisseling, A two-dimensional data distribution method for parallel sparse matrix-vector multiplication, SIAM Rev. 47 (2005) 67–95. doi:10.1137/S0036144502409019. URL http://portal.acm.org/citation.cfm?id=1055334.1055397

[8] U. Çatalyürek, C. Aykanat, A hypergraph-partitioning approach for coarse-grain decomposition, in: Proceedings of the 2001 ACM/IEEE Conference on Supercomputing, SC '01, ACM, New York, NY, USA, 2001, pp. 28–28. doi:10.1145/582034.582062. URL http://doi.acm.org/10.1145/582034.582062

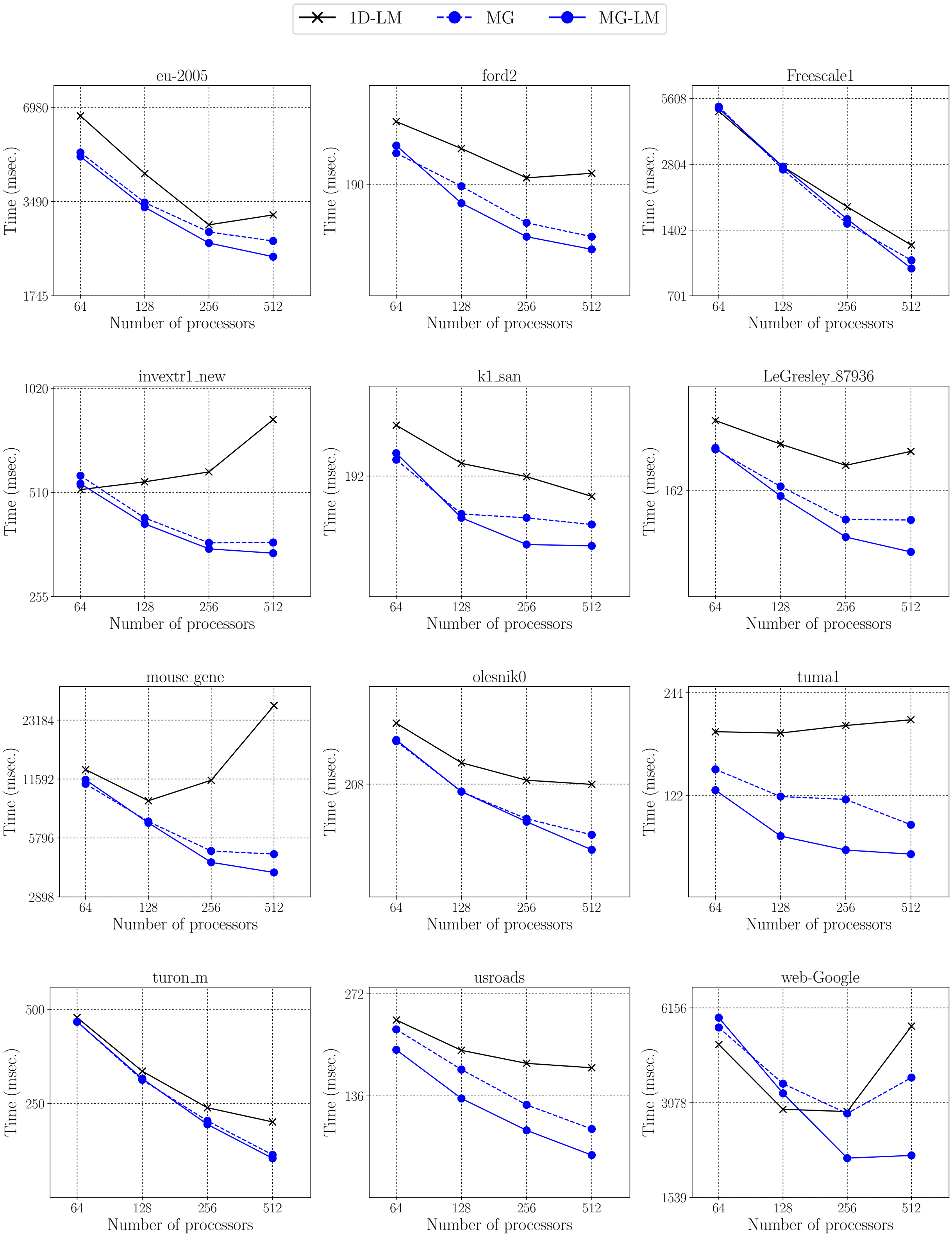

Figure 8: Comparison of partitioning models in terms of parallel SpMV runtime.

[9] B. Uçar, C. Aykanat, Revisiting hypergraph models for sparse matrix partitioning, SIAM Rev. 49 (2007) 595–603. doi:10.1137/060662459. URL http://portal.acm.org/citation.cfm?id=1330215.1330219
[10] D. Pelt, R. Bisseling, A medium-grain method for fast 2d bipartitioning of sparse matrices, in: Parallel and Distributed Processing Symposium, 2014 IEEE 28th International, 2014, pp. 529–539. doi:10.1109/IPDPS.2014.62.
[11] E. Kayaaslan, B. Ucar, C. Aykanat, Semi-two-dimensional partitioning for parallel sparse matrix-vector multiplication, in: Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International, 2015, pp. 1125–1134. doi:10.1109/IPDPSW.2015.20.
[12] E. Kayaaslan, B. Uçar, C. Aykanat, 1.5D parallel sparse matrix-vector multiply, SIAM J. Sci. Comput., *in press*.
[13] A. N. Yzelman, R. H. Bisseling, Cache-oblivious sparse matrix-vector multiplication by using sparse matrix partitioning methods, SIAM J. Sci. Comput. 31 (4) (2009) 3128–3154. doi:10.1137/080733243. URL http://dx.doi.org/10.1137/080733243
[14] U. V. Çatalyürek, C. Aykanat, B. Uçar, On two-dimensional sparse matrix partitioning: Models, methods, and a recipe, SIAM J. Sci. Comput. 32 (2) (2010) 656–683. doi:10.1137/080737770. URL http://dx.doi.org/10.1137/080737770
[15] U. Çatalyürek, C. Aykanat, A fine-grain hypergraph model for 2d decomposition of sparse matrices, in: Proceedings of the 15th International Parallel and Distributed Processing Symposium, IPDPS '01, IEEE Computer Society, Washington, DC, USA, 2001, pp. 118–. URL http://dl.acm.org/citation.cfm?id=645609.663255
[16] S. Acer, O. Selvitopi, C. Aykanat, Improving performance of sparse matrix dense matrix multiplication on large-scale parallel systems, Parallel Computing 59 (2016) 71 – 96, theory and Practice of Irregular Applications.
[17] O. Selvitopi, S. Acer, C. Aykanat, A recursive hypergraph bipartitioning framework for reducing bandwidth and latency costs simultaneously, IEEE Transactions on Parallel and Distributed Systems 28 (2) (2017) 345–358. doi:10.1109/TPDS.2016.2577024.
[18] E. G. Boman, K. D. Devine, S. Rajamanickam, 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 '13, ACM, New York, NY, USA, 2013, pp. 50:1–50:12. doi:10.1145/2503210.2503293. URL http://doi.acm.org/10.1145/2503210.2503293
[19] B. Uçar, C. Aykanat, Encapsulating multiple communication-cost metrics in partitioning sparse rectangular matrices for parallel matrix-vector multiplies, SIAM J. Sci. Comput. 25 (6) (2004) 1837–1859. doi:http://dx.doi.org/10.1137/S1064827502410463.
[20] M. Deveci, K. Kaya, B. Uçar, Ümit Çatalyürek, Hypergraph partitioning for multiple communication cost metrics: Model and methods, Journal of Parallel and Distributed Computing 77 (0) (2015) 69 – 83. doi:http://dx.doi.org/10.1016/j.jpdc.2014.12.002. URL http://www.sciencedirect.com/science/article/pii/S0743731514002275
[21] R. H. Bisseling, W. Meesen, Communication balancing in parallel sparse matrix-vector multiply, Electronic Transactions on Numerical Analysis 21 (2005) 47–65.
[22] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, K. Rupp, B. F. Smith, H. Zhang, PETSc users manual, Tech. Rep. ANL-95/11 - Revision 3.5, Argonne National Laboratory (2014). URL http://www.mcs.anl.gov/petsc
[23] T. A. Davis, Y. Hu, The University of Florida sparse matrix collection, ACM Trans. Math. Softw. 38 (1) (2011) 1:1–1:25. doi:10.1145/2049662.2049663. URL http://doi.acm.org/10.1145/2049662.2049663