



**HAL**  
open science

## 1.5D Parallel Sparse Matrix-Vector Multiply

Enver Kayaaslan, Cevdet Aykanat, Bora Uçar

► **To cite this version:**

Enver Kayaaslan, Cevdet Aykanat, Bora Uçar. 1.5D Parallel Sparse Matrix-Vector Multiply. SIAM Journal on Scientific Computing, 2018, 40 (1), pp.C25 - C46. 10.1137/16M1105591 . hal-01897555

**HAL Id: hal-01897555**

**<https://inria.hal.science/hal-01897555v1>**

Submitted on 17 Oct 2018

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# 1.5D PARALLEL SPARSE MATRIX-VECTOR MULTIPLY\*

ENVER KAYAASLAN<sup>†</sup>, CEVDET AYKANAT<sup>‡</sup>, AND BORA UÇAR<sup>§</sup>

**Abstract.** There are three common parallel sparse matrix-vector multiply algorithms: 1D row-parallel, 1D column-parallel and 2D row-column-parallel. The 1D parallel algorithms offer the advantage of having only one communication phase. On the other hand, the 2D parallel algorithm is more scalable but it suffers from two communication phases. Here, we introduce a novel concept of heterogeneous messages where a heterogeneous message may contain both input-vector entries and partially computed output-vector entries. This concept not only leads to a decreased number of messages, but also enables fusing the input- and output-communication phases into a single phase. These findings are exploited to propose a 1.5D parallel sparse matrix-vector multiply algorithm which is called local row-column-parallel. This proposed algorithm requires a constrained fine-grain partitioning in which each fine-grain task is assigned to the processor that contains either its input-vector entry, or its output-vector entry, or both. We propose two methods to carry out the constrained fine-grain partitioning. We conduct our experiments on a large set of test matrices to evaluate the partitioning qualities and partitioning times of these proposed 1.5D methods.

**Key words.** sparse matrix partitioning, parallel sparse matrix-vector multiplication, directed hypergraph model, bipartite vertex cover, combinatorial scientific computing

**AMS subject classifications.** 05C50, 05C65, 05C70, 65F10, 65F50, 65Y05

**1. Introduction.** Sparse matrix-vector multiply (SpMV) of the form  $\mathbf{y} \leftarrow \mathbf{Ax}$  is a fundamental operation in many iterative solvers for linear systems, eigensystems and least squares problems. This renders the parallelization of SpMV operation an important problem. In the literature, there are three SpMV algorithms: *row-parallel*, *column-parallel*, and *row-column-parallel*. Row-parallel and column-parallel (called 1D) algorithms have a single communication phase, in which either the  $\mathbf{x}$ -vector or partial results on the  $\mathbf{y}$ -vector entries are communicated. Row-column-parallel (2D) algorithms have two communication phases; first the  $\mathbf{x}$ -vector entries are communicated, then the partial results on the  $\mathbf{y}$ -vector entries are communicated. We propose another parallel SpMV algorithm in which both the  $\mathbf{x}$ -vector and the partial results on the  $\mathbf{y}$ -vector entries are communicated as in the 2D algorithms, yet the communication is handled in a single phase as in the 1D algorithms. That is why, the new parallel SpMV algorithm is dubbed 1.5D.

Partitioning methods based on graphs and hypergraphs are widely established to achieve 1D and 2D parallel algorithms. For 1D parallel SpMV, row-wise or column-wise partitioning methods are available. The scalability of 1D parallelism is limited especially when a row or a column has too many nonzeros in the row- and column-parallel algorithms, respectively. In such cases, the communication volume is high and the load balance is hard to achieve, severely reducing the solution space. The associated partitioning methods are usually the fastest alternatives. For 2D parallel SpMV, there are different partitioning methods. Among them, those that partition matrix entries individually, based on the fine-grain model [4], have the highest flexibility. That is why they usually obtain the lowest communication volume and achieve near perfect balance among nonzeros per processor [7]. However, the fine-grain partitioning approach usually results in higher number of messages; not surprisingly higher

---

\*A preliminary version appeared in IPDPSW [12].

<sup>†</sup>NTENT, Inc., USA

<sup>‡</sup>Bilkent University, Turkey

<sup>§</sup>CNRS and LIP (UMR5668 CNRS-ENS Lyon-INRIA-UCBL),  
46, allée d'Italie, ENS Lyon, Lyon, 69364, France.

44 number of messages hampers the parallel SpMV performance [11].

45 The parallel SpMV operation is composed of fine-grain tasks of multiply-and-add  
 46 operations of the form  $y_i \leftarrow y_i + a_{ij}x_j$ . Here, each fine-grain task is identified with a  
 47 unique nonzero and assumed to be performed by the processor that holds the associ-  
 48 ated nonzero by the *owner-computes rule*. The proposed 1.5D parallel SpMV imposes  
 49 a special condition on the operands of the fine-grain task  $y_i \leftarrow y_i + a_{ij}x_j$ : the proces-  
 50 sor that holds  $a_{ij}$  should also hold  $x_j$  or should be responsible for  $y_i$  (or both). The  
 51 standard rowwise and columnwise partitioning algorithms for 1D parallel algorithms  
 52 satisfy the condition, but they are too restrictive. The standard fine-grain partition-  
 53 ing approach does not necessarily satisfy the condition. Here we propose two methods  
 54 for partitioning for 1.5D parallel SpMV. With the proposed partitioning methods, the  
 55 overall 1.5D parallel SpMV algorithm inherits the important characteristics of 1D and  
 56 2D parallel SpMV and the associated partitioning methods. In particular, it has

- 57 • a single communication phase as in 1D parallel SpMV,
- 58 • the partitioning flexibility close to that of 2D fine-grain partitioning,
- 59 • much reduced number of messages compared to the 2D fine-grain partitioning,
- 60 • a partitioning time close to that of 1D partitioning.

61 We propose two methods (Section 4) to obtain a 1.5D local fine-grain partition  
 62 each with a different setting and approach where some preliminary studies on these  
 63 methods are given in our recent work [12]. The first method is developed by proposing  
 64 a directed hypergraph model. Since current partitioning tools cannot meet 1.5D  
 65 partitioning requirements, we adopt and adapt an approach similar to that of a recent  
 66 work by Pelt and Bisseling. [15]. The second method has two parts. The first part  
 67 applies a conventional 1D partitioning method but decodes this only as a partition  
 68 of the vectors  $x$  and  $y$ . The second part decides nonzero/task distribution under the  
 69 fixed partition of the input and output vectors.

70 The remainder of this paper is as follows. In Section 2, we give a background  
 71 on parallel SpMV. Section 3 presents the proposed 1.5D local row-column-parallel  
 72 algorithm and 1.5D local fine-grain partitioning. The two methods proposed to obtain  
 73 a local fine-grain partition are presented and discussed in Section 4. Section 5 gives a  
 74 brief review of recent related work. We display our experimental results in Section 6  
 75 and conclude the paper in Section 7.

## 76 2. Background on parallel sparse matrix-vector multiply.

77 **2.1. The anatomy of parallel sparse matrix-vector multiply.** Recall that  
 78  $\mathbf{y} \leftarrow \mathbf{A}\mathbf{x}$  can be cast as a collection of fine-grain tasks of multiply-and-add operations

$$79 \quad (1) \quad y_i \leftarrow y_i + a_{ij} \times x_j .$$

80 These tasks can share input and output-vector entries. When a task  $a_{ij}$  and the  
 81 input-vector entry  $x_j$  are assigned to different processors, say  $P_\ell$  and  $P_r$ , respectively,  
 82  $P_r$  sends  $x_j$  to  $P_\ell$ , which is responsible to carry out the task  $a_{ij}$ . An input-vector  
 83 entry  $x_j$  is not communicated multiple times between processor pairs. When a task  
 84  $a_{ij}$  and the output-vector entry  $y_i$  are assigned to different processors, say  $P_r$  and  $P_k$ ,  
 85 respectively, then  $P_r$  performs  $\hat{y}_i \leftarrow \hat{y}_i + a_{ij} \times x_j$  as well as all other multiply-and-add  
 86 operations that contribute to the partial result  $\hat{y}_i$  and then sends  $\hat{y}_i$  to  $P_k$ . The partial  
 87 results received by  $P_k$  from different processors are then summed to compute  $y_i$ .

88 **2.2. Task-and-data distributions.** Let  $\mathbf{A}$  be an  $m \times n$  sparse matrix and  $a_{ij}$   
 89 represent both a nonzero of  $\mathbf{A}$  and the associated fine-grain task of multiply-and-  
 90 add operation (1). Let  $\mathbf{x}$  and  $\mathbf{y}$  be the input- and output-vectors of size  $n$  and

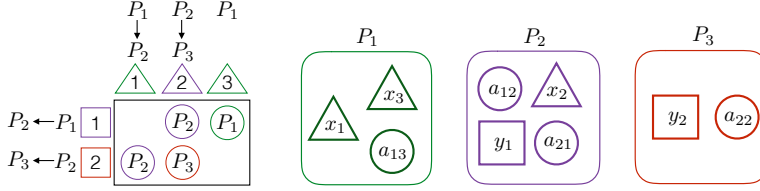


Fig. 1: A task-and-data distribution  $\Pi(\mathbf{y} \leftarrow \mathbf{A}\mathbf{x})$  of matrix-vector multiply with a  $2 \times 3$  sparse matrix  $\mathbf{A}$ .

91  $m$ , respectively, and  $K$  be the number of processors. We define a  $K$ -way task-and-  
 92 data distribution  $\Pi(\mathbf{y} \leftarrow \mathbf{A}\mathbf{x})$  of the associated SpMV as a 3-tuple  $\Pi(\mathbf{y} \leftarrow \mathbf{A}\mathbf{x}) =$   
 93  $(\Pi(\mathbf{A}), \Pi(\mathbf{x}), \Pi(\mathbf{y}))$ , where  $\Pi(\mathbf{A}) = \{\mathbf{A}^{(1)}, \dots, \mathbf{A}^{(K)}\}$ ,  $\Pi(\mathbf{x}) = \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(K)}\}$ , and  
 94  $\Pi(\mathbf{y}) = \{\mathbf{y}^{(1)}, \dots, \mathbf{y}^{(K)}\}$ . We can also represent  $\Pi(\mathbf{A})$  as a nonzero-disjoint summation

$$95 \quad (2) \quad \mathbf{A} = \mathbf{A}^{(1)} + \mathbf{A}^{(2)} + \dots + \mathbf{A}^{(K)}.$$

96 In  $\Pi(\mathbf{x})$  and  $\Pi(\mathbf{y})$ , each  $\mathbf{x}^{(k)}$  and  $\mathbf{y}^{(k)}$  is a disjoint subvector of  $\mathbf{x}$  and  $\mathbf{y}$ , respectively.  
 97 Figure 1 illustrates a sample 3-way task-and-data distribution of matrix-vector multiply  
 98 on a  $2 \times 3$  sparse matrix.

99 For given input- and output-vector distributions  $\Pi(\mathbf{x})$  and  $\Pi(\mathbf{y})$ , the columns and  
 100 rows of  $\mathbf{A}$  and those of  $\mathbf{A}^{(k)}$  can be permuted, conformably with  $\Pi(\mathbf{x})$  and  $\Pi(\mathbf{y})$ , to  
 101 form  $K \times K$  block structures:

$$102 \quad (3) \quad \mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} & \dots & \mathbf{A}_{1K} \\ \mathbf{A}_{21} & \mathbf{A}_{22} & \dots & \mathbf{A}_{2K} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}_{K1} & \mathbf{A}_{K2} & \dots & \mathbf{A}_{KK} \end{bmatrix}, \quad (4) \quad \mathbf{A}^{(k)} = \begin{bmatrix} \mathbf{A}_{11}^{(k)} & \mathbf{A}_{12}^{(k)} & \dots & \mathbf{A}_{1K}^{(k)} \\ \mathbf{A}_{21}^{(k)} & \mathbf{A}_{22}^{(k)} & \dots & \mathbf{A}_{2K}^{(k)} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}_{K1}^{(k)} & \mathbf{A}_{K2}^{(k)} & \dots & \mathbf{A}_{KK}^{(k)} \end{bmatrix}.$$

103 Note that the row and column orderings (4) of the individual  $\mathbf{A}^{(k)}$  matrices are in  
 104 compliance with the row and column orderings (3) of  $\mathbf{A}$ . Hence, each block  $\mathbf{A}_{k\ell}$  of  
 105 the block structure (3) of  $\mathbf{A}$  can be written as a nonzero-disjoint summation

$$106 \quad (5) \quad \mathbf{A}_{k\ell} = \mathbf{A}_{k\ell}^{(1)} + \mathbf{A}_{k\ell}^{(2)} + \dots + \mathbf{A}_{k\ell}^{(K)}.$$

107 Let  $\Pi(\mathbf{y} \leftarrow \mathbf{A}\mathbf{x})$  be any  $K$ -way task-and-data distribution. According to this  
 108 distribution, each processor  $P_k$  holds the submatrix  $\mathbf{A}^{(k)}$ , holds the input-subvector  
 109  $\mathbf{x}^{(k)}$  and is responsible for storing/computing the output subvector  $\mathbf{y}^{(k)}$ . The fine-  
 110 grain tasks (1) associated with the nonzeros of  $\mathbf{A}^{(k)}$  are to be carried out by  $P_k$ .  
 111 An input-vector entry  $x_j \in \mathbf{x}^{(k)}$  is sent from  $P_k$  to  $P_\ell$ , which is called an *input*  
 112 *communication*, if there is a task  $a_{ij} \in \mathbf{A}^{(\ell)}$  associated with a nonzero at column  $j$ .  
 113 On the other hand,  $P_k$  receives a partial result  $\hat{y}_i$  on an output-vector entry  $y_i \in \mathbf{y}^{(k)}$   
 114 from  $P_\ell$ , which is referred to as an *output communication*, if there is a task  $a_{ij} \in \mathbf{A}^{(\ell)}$   
 115 associated with a nonzero at row  $i$ . Therefore, the fine-grain tasks associated with the  
 116 nonzeros of the column stripe  $\mathbf{A}_{*k} = [\mathbf{A}_{1k}^T, \dots, \mathbf{A}_{Kk}^T]^T$  are the only ones that require  
 117 an input-vector entry of  $\mathbf{x}^{(k)}$  and the fine-grain tasks associated with the nonzeros  
 118 of the row stripe  $\mathbf{A}_{k*} = [\mathbf{A}_{k1}, \dots, \mathbf{A}_{kK}]$  are the only ones that contribute to the  
 119 computation of an output-vector entry of  $\mathbf{y}^{(k)}$ .

120 **2.3. 1D parallel sparse matrix-vector multiply.** There are two main alter-  
 121 natives for 1D parallel SpMV, row-parallel and column-parallel.

122 In the row-parallel SpMV, the basic computational units are the rows. For an  
 123 output-vector entry  $y_i$  assigned to processor  $P_k$ , the fine-grain tasks associated with  
 124 the nonzeros of  $\mathbf{A}_{i*} = \{a_{ij} \in \mathbf{A} : 1 \leq j \leq n\}$  are combined into a composite task of  
 125 inner product  $y_i \leftarrow \mathbf{A}_{i*}\mathbf{x}$  which is to be carried out by  $P_k$ . Therefore, for the row-  
 126 parallel algorithm, a task-and-data distribution  $\Pi(\mathbf{y} \leftarrow \mathbf{A}\mathbf{x})$  of matrix-vector multiply  
 127 on  $\mathbf{A}$  should satisfy the following condition:

$$128 \quad (6) \quad a_{ij} \in \mathbf{A}^{(k)} \text{ whenever } y_i \in \mathbf{y}^{(k)} .$$

129 Then,  $\Pi(\mathbf{A})$  coincides with the output-vector distribution  $\Pi(\mathbf{y})$ —each submatrix is  
 130 a row stripe of the block structure (3) of  $\mathbf{A}$ . In the row-parallel parallel SpMV, all  
 131 messages are communicated in an input-communication phase called *expand* where  
 132 each message contains only input-vector entries.

133 In the column-parallel SpMV, the basic computational units are the columns. For  
 134 an input-vector entry  $x_j$  assigned to processor  $P_k$ , the fine-grain tasks associated with  
 135 the nonzeros of  $\mathbf{A}_{*j} = \{a_{ij} \in \mathbf{A} : 1 \leq i \leq m\}$  are combined into a composite task of  
 136 “daxpy” operation  $\hat{\mathbf{y}}_k \leftarrow \hat{\mathbf{y}}_k + \mathbf{A}_{*j}x_j$  which is to be carried out by  $P_k$  where  $\hat{\mathbf{y}}_k$  is  
 137 the partially computed output-vector of  $P_k$ . As a result, a task-and-data distribution  
 138  $\Pi(\mathbf{y} \leftarrow \mathbf{A}\mathbf{x})$  of matrix-vector multiply on  $\mathbf{A}$  for the column-parallel algorithm should  
 139 satisfy the following condition:

$$140 \quad (7) \quad a_{ij} \in \mathbf{A}^{(k)} \text{ whenever } x_j \in \mathbf{x}^{(k)} .$$

141 Here,  $\Pi(\mathbf{A})$  coincides with the input-vector distribution  $\Pi(\mathbf{x})$ —each submatrix  $\mathbf{A}^{(k)}$   
 142 is a column stripe of the block structure (3) of  $\mathbf{A}$ . In the column-parallel SpMV, all  
 143 messages are communicated in an output-communication phase called *fold* where each  
 144 message contains only partially computed output-vector entries.

145 The column-net and row-net hypergraph models [3] can be respectively used to  
 146 obtain the required task-and-data partitioning for the row-parallel and column-parallel  
 147 SpMV.

148 **2.4. 2D parallel sparse matrix-vector multiply.** In the 2D parallel SpMV,  
 149 also referred to as the row-column-parallel, the basic computational units are nonze-  
 150 ros [4, 7]. The row-column-parallel algorithm requires fine-grain partitioning which  
 151 imposes no restriction on distributing tasks and data. The row-column-parallel algo-  
 152 rithm contains two communication and two computational phases in an interleaved  
 153 manner as shown in Algorithm 1. The algorithm starts with the expand phase where  
 154 the required input-subvector entries are communicated. The second step computes  
 155 only those partial results that are to be communicated in the following fold phase.  
 156 In the final step, each processor computes its own output-subvector. If we have a  
 157 rowwise partitioning, the steps 2, 3 and 4c are not needed and hence the algorithm  
 158 reduces to the row-parallel algorithm. Similarly, the algorithm without steps 1, 2b  
 159 and 4b, can be used when we have a columnwise partitioning. The row-column-net  
 160 hypergraph model [4, 7] can be used to obtain the required task-and-data partitioning  
 161 for row-column-parallel SpMV.

162 **3. 1.5D parallel sparse matrix-vector multiply.** In this section, we propose  
 163 the local row-column-parallel SpMV algorithm that exhibits 1.5D parallelism. The  
 164 proposed algorithm simplifies the row-column-parallel algorithm by combining the two  
 165 communication phases into a single expand-fold phase while attaining a flexibility on  
 166 nonzero/task distribution close to the flexibility attained by the row-column-parallel  
 167 algorithm.

**Algorithm 1** The row-column-parallel sparse matrix-vector multiply

For each processor  $P_k$ :

1. (*expand*) for each nonzero column stripe  $\mathbf{A}_{*k}^{(\ell)}$ , where  $\ell \neq k$ ;
  - (a) form vector  $\hat{\mathbf{x}}_\ell^{(k)}$  which contains only those entries of  $\mathbf{x}^{(k)}$  corresponding to nonzero columns in  $\mathbf{A}_{*k}^{(\ell)}$  and
  - (b) send vector  $\hat{\mathbf{x}}_\ell^{(k)}$  to  $P_\ell$ ,
2. for each nonzero row stripe  $\mathbf{A}_{\ell*}^{(k)}$ , where  $\ell \neq k$ ; compute
  - (a)  $\mathbf{y}_k^{(\ell)} \leftarrow \mathbf{A}_{\ell k}^{(k)} \mathbf{x}^{(k)}$  and
  - (b)  $\mathbf{y}_k^{(\ell)} \leftarrow \mathbf{y}_k^{(\ell)} + \sum_{r \neq k} \mathbf{A}_{\ell r}^{(k)} \hat{\mathbf{x}}_r^{(k)}$
3. (*fold*) for each nonzero row stripe  $\mathbf{A}_{\ell*}^{(k)}$ , where  $\ell \neq k$ ;
  - (a) form vector  $\hat{\mathbf{y}}_\ell^{(k)}$  which contains only those entries of  $\mathbf{y}_k^{(\ell)}$  corresponding to nonzero rows in  $\mathbf{A}_{\ell*}^{(k)}$  and
  - (b) send vector  $\hat{\mathbf{y}}_\ell^{(k)}$  to  $P_\ell$ ,
4. compute output-subvector
  - (a)  $\mathbf{y}^{(k)} \leftarrow \mathbf{A}_{kk}^{(k)} \mathbf{x}^{(k)}$ ,
  - (b)  $\mathbf{y}^{(k)} \leftarrow \mathbf{y}^{(k)} + \mathbf{A}_{k\ell}^{(k)} \hat{\mathbf{x}}_\ell^{(k)}$  and
  - (c)  $\mathbf{y}^{(k)} \leftarrow \mathbf{y}^{(k)} + \sum_{\ell \neq k} \hat{\mathbf{y}}_\ell^{(k)}$ .

168 In the well-known parallel SpMV, the messages are homogenous in the sense that  
 169 they pertain to either  $\mathbf{x}$ - or  $\mathbf{y}$ -vector entries. In the proposed row-column-parallel  
 170 SpMV algorithm, the number of messages are reduced with respect to the row-column-  
 171 parallel algorithm by making the messages heterogenous (pertaining to both  $\mathbf{x}$ - and  
 172  $\mathbf{y}$ -vector entries), and by communicating them in a single expand-fold phase. If a  
 173 processor  $P_\ell$  sends a message to processor  $P_k$  in both of the expand and fold phases,  
 174 then the number of messages required from  $P_\ell$  to  $P_k$  reduces from two to one. However,  
 175 if a message from  $P_\ell$  to  $P_k$  is sent only in the expand phase or only in the fold phase,  
 176 then there is no reduction in the number of such messages.

177 **3.1. A Task categorization.** We introduce a two-way categorization of input-  
 178 and output-vector entries and a four-way categorization of fine-grain tasks (1) accord-  
 179 ing to a task-and-data distribution  $\Pi(\mathbf{y} \leftarrow \mathbf{A}\mathbf{x})$  of matrix-vector multiply on  $\mathbf{A}$ . For  
 180 a task  $a_{ij}$ , the input-vector entry  $x_j$  is said to be *local* if both  $a_{ij}$  and  $x_j$  are assigned  
 181 to the same processor; the output-vector entry  $y_i$  is said to be *local* if both  $a_{ij}$  and  $y_i$   
 182 are assigned to the same processor. With this definition, the tasks can be classified  
 183 into four groups. The task

$$184 \quad y_i \leftarrow y_i + a_{ij} \times x_j \text{ on } P_k \text{ is } \begin{cases} \text{input-output-local} & \text{if } x_j \in \mathbf{x}^{(k)} \text{ and } y_i \in \mathbf{y}^{(k)}, \\ \text{input-local} & \text{if } x_j \in \mathbf{x}^{(k)} \text{ and } y_i \notin \mathbf{y}^{(k)}, \\ \text{output-local} & \text{if } x_j \notin \mathbf{x}^{(k)} \text{ and } y_i \in \mathbf{y}^{(k)}, \\ \text{nonlocal} & \text{if } x_j \notin \mathbf{x}^{(k)} \text{ and } y_i \notin \mathbf{y}^{(k)}. \end{cases}$$

185 Recall that an input-vector entry  $x_j \in \mathbf{x}^{(\ell)}$  is sent from  $P_\ell$  to  $P_k$  if there exists a task  
 186  $a_{ij} \in \mathbf{A}^{(k)}$  at column  $j$ , which implies that the task  $a_{ij}$  of  $P_k$  is either output-local or  
 187 nonlocal since  $x_j \notin \mathbf{x}^{(k)}$ . Similarly, for an output-vector entry  $y_i \in \mathbf{y}^{(\ell)}$ ,  $P_\ell$  receives

188 a partial result  $\hat{y}_i$  from  $P_k$  if a task  $a_{ij} \in \mathbf{A}^{(k)}$ , which implies that the task  $a_{ij}$  of  
 189  $P_k$  is either input-local or nonlocal since  $y_i \notin \mathbf{y}^{(k)}$ . We can also infer from that the  
 190 input-output-local tasks neither depend on the input-communication phase nor incur  
 191 a dependency on the output-communication phase. However, the nonlocal tasks are  
 192 linked with both communication phases.

193 In the row-parallel algorithm, each of the fine-grain tasks is either input-output-  
 194 local or output-local due to the rowwise partitioning condition (6). For this reason,  
 195 no partial result is computed for other processors, and thus no output communication  
 196 is incurred. In the column-parallel algorithm, each of the fine-grain tasks is either  
 197 input-output-local or input-local due to the columnwise partitioning condition (7). In  
 198 the row-column-parallel algorithm, the input and output communications have to be  
 199 carried out in separate phases. The reason is that the partial results on the output-  
 200 vector entries to be sent are partially derived by performing nonlocal tasks that rely  
 201 on the input-vector entries received.

202 **3.2. Local fine-grain partitioning.** In order to remove the dependency be-  
 203 tween the two communication phases in the row-column-parallel algorithm, we pro-  
 204 pose the local fine-grain partitioning where “locality” refers to the fact that each fine-  
 205 grain task is input-local, output-local or input-output-local. In other words, there is  
 206 no nonlocal fine-grain task.

207 A task-and-data distribution  $\Pi(\mathbf{y} \leftarrow \mathbf{A}\mathbf{x})$  of matrix-vector multiply on  $\mathbf{A}$  is said  
 208 to be a local fine-grain partition if the following condition is satisfied:

$$209 \quad (8) \quad a_{ij} \in \mathbf{A}^{(k)} + \mathbf{A}^{(\ell)} \text{ whenever } y_i \in \mathbf{y}^{(k)} \text{ and } x_j \in \mathbf{x}^{(\ell)}.$$

210 Notice that this condition is equivalent to

$$211 \quad (9) \quad \text{if } a_{ij} \in \mathbf{A}^{(k)} \text{ then either } x_j \in \mathbf{y}^{(k)}, \text{ or } y_i \in \mathbf{x}^{(k)}, \text{ or both.}$$

212 Due to (4) and (9), each submatrix  $\mathbf{A}^{(k)}$  becomes of the following form

$$213 \quad (10) \quad \mathbf{A}^{(k)} = \begin{bmatrix} 0 & \cdots & \mathbf{A}_{1k}^{(k)} & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathbf{A}_{k1}^{(k)} & \cdots & \mathbf{A}_{kk}^{(k)} & \cdots & \mathbf{A}_{kK}^{(k)} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & \mathbf{A}_{Kk}^{(k)} & \cdots & 0 \end{bmatrix}.$$

214 In this form, the tasks associated with the nonzeros of diagonal block  $\mathbf{A}_{kk}^{(k)}$ , the off-  
 215 diagonal blocks of the row stripe  $\mathbf{A}_{k*}^{(k)}$ , and the off-diagonal blocks of the column-stripe  
 216  $\mathbf{A}_{*k}^{(k)}$  are input-output-local, output-local and input-local, respectively. Furthermore,  
 217 due to (5) and (8), each off-diagonal block  $\mathbf{A}_{k\ell}$  of the block structure (3) induced by  
 218 the vector distribution  $(\Pi(\mathbf{x}), \Pi(\mathbf{y}))$  becomes

$$219 \quad (11) \quad \mathbf{A}_{k\ell} = \mathbf{A}_{k\ell}^{(k)} + \mathbf{A}_{k\ell}^{(\ell)},$$

220 and for each diagonal block we have  $\mathbf{A}_{kk} = \mathbf{A}_{kk}^{(k)}$ .

221 In order to clarify Equations (8)–(11), we provide the following 4-way local fine-  
 222 grain partition on  $\mathbf{A}$  as permuted into a  $4 \times 4$  block structure.



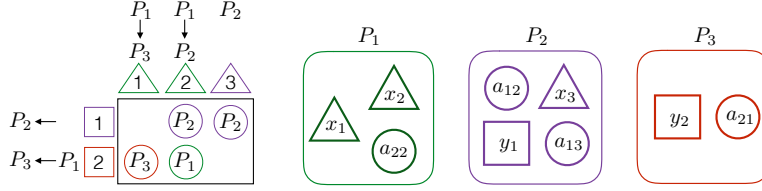


Fig. 2: A sample local fine-grain partition. Here,  $a_{12}$  is an output-local task,  $a_{13}$  is an input-output-local task,  $a_{21}$  is an output-local task, and  $a_{22}$  is an input-local task.

$$\begin{aligned}
 \mathbf{A} &= \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12}^{(1)} & \mathbf{A}_{13}^{(1)} & \mathbf{A}_{14}^{(1)} \\ \mathbf{A}_{21}^{(1)} & 0 & 0 & 0 \\ \mathbf{A}_{31}^{(1)} & 0 & 0 & 0 \\ \mathbf{A}_{41}^{(1)} & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & \mathbf{A}_{12}^{(2)} & 0 & 0 \\ \mathbf{A}_{21}^{(2)} & \mathbf{A}_{22}^{(2)} & \mathbf{A}_{23}^{(2)} & \mathbf{A}_{24}^{(2)} \\ 0 & \mathbf{A}_{32}^{(2)} & 0 & 0 \\ 0 & \mathbf{A}_{42}^{(2)} & 0 & 0 \end{bmatrix} + \\
 & \begin{bmatrix} 0 & 0 & \mathbf{A}_{13}^{(3)} & 0 \\ 0 & 0 & \mathbf{A}_{23}^{(3)} & 0 \\ \mathbf{A}_{31}^{(3)} & \mathbf{A}_{32}^{(3)} & \mathbf{A}_{33}^{(3)} & \mathbf{A}_{34}^{(3)} \\ 0 & 0 & \mathbf{A}_{43}^{(3)} & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & \mathbf{A}_{14}^{(4)} \\ 0 & 0 & 0 & \mathbf{A}_{24}^{(4)} \\ 0 & 0 & 0 & \mathbf{A}_{34}^{(4)} \\ \mathbf{A}_{41}^{(4)} & \mathbf{A}_{42}^{(4)} & \mathbf{A}_{43}^{(4)} & \mathbf{A}_{44} \end{bmatrix}.
 \end{aligned}$$

For instance,  $\mathbf{A}_{42} = \mathbf{A}_{42}^{(2)} + \mathbf{A}_{42}^{(4)}$ ,  $\mathbf{A}_{23} = \mathbf{A}_{23}^{(2)} + \mathbf{A}_{23}^{(3)}$ ,  $\mathbf{A}_{31} = \mathbf{A}_{31}^{(1)} + \mathbf{A}_{31}^{(3)}$ ,  $\dots$ , etc.

Figure 2 displays a sample 3-way local fine-grain partition on the same sparse matrix used in Figure 1. In this figure,  $a_{13} \in \mathbf{A}^{(1)}$  where  $y_1 \in \mathbf{y}^{(2)}$  and  $x_3 \in \mathbf{x}^{(1)}$  and thus  $a_{13}$  is an input-local task of  $P_1$ . Also,  $a_{21} \in \mathbf{A}^{(3)}$  where  $y_2 \in \mathbf{y}^{(3)}$  and  $x_1 \in \mathbf{x}^{(1)}$  and thus  $a_{21}$  is an output-local task of  $P_3$ .

**3.3. Local row-column-parallel sparse matrix-vector multiply.** As there is no nonlocal tasks, the output-local tasks depend on input communication, and the output communication depends on the input-local tasks. Therefore, the tasks groups and communication phases can be arranged as: (i) input-local tasks; (ii) output-communication, input-communication; (iii) output-local tasks and input-output-local tasks. The input and output communication phases can be combined into the expand-fold phase, and the output-local and input-output-local task groups can be combined into a single computation phase to simplify the overall execution.

The local row-column-parallel algorithm is composed of three steps as shown in Algorithm 2. In the first step, processors concurrently perform their input-local tasks which contribute to partially computed output-vector entries for other processors. In the expand-fold phase, for each nonzero off-diagonal block  $\mathbf{A}_{\ell k} = \mathbf{A}_{\ell k}^{(k)} + \mathbf{A}_{\ell k}^{(\ell)}$ ,  $P_k$  prepares a message  $[\hat{\mathbf{x}}_\ell^{(k)}, \hat{\mathbf{y}}_k^{(\ell)}]$  for  $P_\ell$ . Here,  $\hat{\mathbf{x}}_\ell^{(k)}$  contains the input-vector entries of  $\mathbf{x}^{(k)}$  that are required by the output-local tasks of  $P_\ell$ , whereas  $\hat{\mathbf{y}}_k^{(\ell)}$  contains the partial results on the output-vector entries of  $\mathbf{y}^{(\ell)}$ , where the partial results are derived by performing the input-local tasks of  $P_k$ . In the last step, each processor  $P_k$  computes output-subvector  $\mathbf{y}^{(k)}$  by summing the partial results computed locally by its own input-output-local tasks (step 3a) and output-local tasks (step 3b) as well as the partial results received from other processors due to their input-local tasks (step 3c).

For a message  $[\hat{\mathbf{x}}_\ell^{(k)}, \hat{\mathbf{y}}_k^{(\ell)}]$  from processor  $P_k$  to  $P_\ell$ , the input-vector entries of  $\hat{\mathbf{x}}_\ell^{(k)}$  correspond to the nonzero columns of  $\mathbf{A}_{\ell k}^{(\ell)}$ , whereas the partially computed output-



**Algorithm 2** The local row-column-parallel sparse matrix-vector multiply

For each processor  $P_k$ :

1. for each nonzero block  $\mathbf{A}_{\ell k}^{(k)}$ , where  $\ell \neq k$ ;  
 compute  $\mathbf{y}_k^{(\ell)} \leftarrow \mathbf{A}_{\ell k}^{(k)} \mathbf{x}^{(k)}$ , ► input-local tasks of  $P_k$
2. (*expand-fold*) for each nonzero block  $\mathbf{A}_{\ell k} = \mathbf{A}_{\ell k}^{(k)} + \mathbf{A}_{\ell k}^{(\ell)}$ , where  $\ell \neq k$ ;  
  - (a) form vector  $\hat{\mathbf{x}}_\ell^{(k)}$ , which contains only those entries of  $\mathbf{x}^{(k)}$  corresponding to nonzero columns in  $\mathbf{A}_{\ell k}^{(\ell)}$ ,
  - (b) form vector  $\hat{\mathbf{y}}_k^{(\ell)}$ , which contains only those entries of  $\mathbf{y}_k^{(\ell)}$  corresponding to nonzero rows in  $\mathbf{A}_{\ell k}^{(k)}$ ,
  - (c) send vector  $[\hat{\mathbf{x}}_\ell^{(k)}, \hat{\mathbf{y}}_k^{(\ell)}]$  to processor  $P_\ell$ .
3. compute output-subvector  
  - (a)  $\mathbf{y}^{(k)} \leftarrow \mathbf{A}_{kk}^{(k)} \mathbf{x}^{(k)}$ , ► input-output-local tasks of  $P_k$
  - (b)  $\mathbf{y}^{(k)} \leftarrow \mathbf{y}^{(k)} + \mathbf{A}_{k\ell}^{(k)} \hat{\mathbf{x}}_\ell^{(k)}$  and ► output-local tasks of  $P_k$
  - (c)  $\mathbf{y}^{(k)} \leftarrow \mathbf{y}^{(k)} + \sum_{\ell \neq k} \hat{\mathbf{y}}_\ell^{(k)}$ . ► input-local tasks of other processors

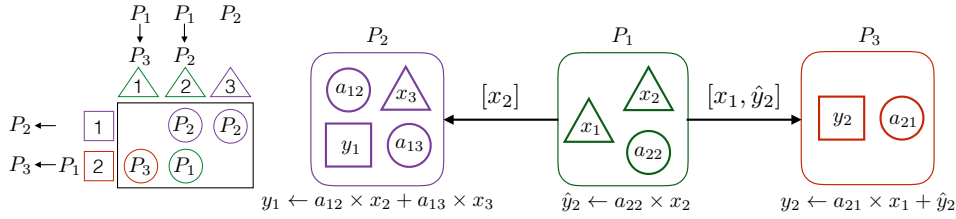


Fig. 3: An illustration of Algorithm 2 for the local fine-grain partition in Figure 2.

251 vector entries of  $\hat{\mathbf{y}}_k^{(\ell)}$  correspond to the nonzero rows of  $\mathbf{A}_{\ell k}^{(k)}$ . That is,  $\hat{\mathbf{x}}_\ell^{(k)} = [x_j :$   
 252  $a_{ij} \in \mathbf{A}_{\ell k}^{(\ell)}]$  and  $\hat{\mathbf{y}}_k^{(\ell)} = [\hat{y}_i : a_{ij} \in \mathbf{A}_{\ell k}^{(k)}]$ . This message is heterogeneous if  $\mathbf{A}_{\ell k}^{(k)}$  and  
 253  $\mathbf{A}_{\ell k}^{(\ell)}$  are both nonzero and homogeneous otherwise. We also note that the number of  
 254 messages is equal to the number of nonzero off-diagonal blocks of the block structure  
 255 (3) of  $\mathbf{A}$  induced by the vector distribution  $(\Pi(\mathbf{x}), \Pi(\mathbf{y}))$ . Figure 3 illustrates the  
 256 steps of Algorithm 2 on the sample local fine-grain partition given in Figure 2. As  
 257 seen in the figure, there are only two messages to be communicated. One message  
 258 is homogeneous, which is from  $P_1$  to  $P_2$  and contains only an input-vector entry  $x_2$ ,  
 259 whereas the other message is heterogeneous, which is from  $P_1$  to  $P_3$  and contains an  
 260 input-vector entry  $x_1$  and a partially computed output-vector entry  $\hat{y}_2$ .

#### 261 4. Two proposed methods for local row-column-parallel partitioning.

262 We propose two methods to find a local row-column-parallel partition that is required  
 263 for 1.5D local row-column-parallel SpMV. One method finds vector and nonzero dis-  
 264 tributions simultaneously, whereas the other one has two parts in which vector and  
 265 nonzero distributions are found separately.

266 **4.1. A directed hypergraph model for simultaneous vector and nonzero**  
 267 **distribution.** In this method, we adopt the elementary hypergraph model for the  
 268 fine-grain partitioning [16] and introduce an additional locality constraint on par-  
 269 tioning in order to obtain a local fine-grain partition. In this hypergraph model

270  $\mathcal{H}_{2D} = (\mathcal{V}, \mathcal{N})$ , there is an input-data vertex for each input-vector entry, an output-  
 271 data vertex for each output-vector entry and a task vertex for each fine-grain task (or  
 272 per matrix nonzero) for a given matrix  $\mathbf{A}$ . That is,

$$273 \quad \mathcal{V} = \{v_x(j) : x_j \in \mathbf{x}\} \cup \{v_y(i) : y_i \in \mathbf{y}\} \cup \{v_z(ij) : a_{ij} \in \mathbf{A}\} .$$

274 The input- and output-data vertices have zero weights, whereas the task vertices have  
 275 unit weights. In  $\mathcal{H}_{2D}$ , there is an input-data net for each input-vector entry, and an  
 276 output-data net for each output-vector entry. An input-data net  $n_x(j)$ , corresponding  
 277 to the input-vector entry  $x_j$ , connects all task vertices associated with the nonzeros at  
 278 column  $j$  as well as the input-data vertex  $v_x(j)$ . Similarly, an output-data net  $n_y(i)$ ,  
 279 corresponding to the output-vector entry  $y_i$ , connects all task vertices associated with  
 280 the nonzeros at row  $i$  as well as the output-data vertex  $v_y(i)$ . That is

$$281 \quad \mathcal{N} = \{n_x(j) : x_j \in \mathbf{x}\} \cup \{n_y(i) : y_i \in \mathbf{y}\} ,$$

$$282 \quad n_x(j) = \{v_x(j)\} \cup \{v_z(ij) : a_{ij} \in \mathbf{A}, 1 \leq i \leq m\} , \text{ and}$$

$$283 \quad n_y(i) = \{v_y(i)\} \cup \{v_z(ij) : a_{ij} \in \mathbf{A}, 1 \leq j \leq n\} .$$

284 Note that each input-data net connects a separate input-data vertex, whereas  
 285 each output-data net connects a separate output-data vertex. We associate nets with  
 286 their respective data vertices.

287 We enhance the elementary row-column-net hypergraph model by imposing di-  
 288 rections on the nets; this is required for modeling the dependencies and their nature.  
 289 Each input-data net  $n_x(j)$  is directed from the input-data vertex  $v_x(j)$  to the task  
 290 vertices connected by  $n_x(j)$ , and each output-data net  $n_y(i)$  is directed from the task  
 291 vertices connected by  $n_y(i)$  to the output-data vertex  $v_y(i)$ . Each task vertex  $v_z(ij)$   
 292 is connected by a single input-data-net  $n_x(j)$  and a single output-data-net  $n_y(i)$ .

293 In order to impose the locality in the partitioning, we introduce the following  
 294 constraint for vertex partitioning on the directed hypergraph model  $\mathcal{H}_{2D}$ : each task  
 295 vertex  $v_z(ij)$  should be assigned to the part that contains either input-data vertex  
 296  $v_x(j)$ , or output-data vertex  $v_y(i)$ , or both. Figure 4a displays a sample  $6 \times 7$  sparse  
 297 matrix. Figure 4b illustrates the associated directed hypergraph model. Figure 4c  
 298 shows a 3-way vertex partition of this directed hypergraph model satisfying the locality  
 299 constraint, and Fig. 4d shows the local fine-grain partition decoded by this partition.

300 Instead of developing a partitioner for this particular directed hypergraph model,  
 301 we propose a task-vertex amalgamation procedure which will help in meeting the  
 302 described locality constraint by using a standard hypergraph partitioning tool. For  
 303 this, we adopt and adapt a simple-yet-effective approach of Pelt and Bisseling [15]. In  
 304 our adaptation, we amalgamate each task vertex  $v_z(ij)$  into either input-data vertex  
 305  $v_x(j)$  or output-data vertex  $v_y(i)$  according to the number of task vertices connected  
 306 by  $n_x(j)$  and  $n_y(i)$ , respectively. That is,  $v_z(ij)$  is amalgamated into  $v_x(j)$  if column  $j$   
 307 has a smaller number of nonzeros than row  $i$ , and otherwise it is amalgamated into  
 308  $v_y(i)$ , where the ties are broken arbitrarily. The result is a reduced hypergraph that  
 309 contains only the input- and output-data vertices amalgamated with the task vertices  
 310 where the weight of a data vertex is equal to the number of task vertices amalgamated  
 311 into that data vertex. As a result, the locality constraint on vertex partitioning of the  
 312 initial directed hypergraph naturally holds on any vertex partitioning on the reduced  
 313 hypergraph. It so happens that after this process, the net directions become irrelevant  
 314 for partitioning, and hence one can use the standard hypergraph partitioning tools.

315 Figure 5 illustrates how to obtain a local fine-grain partition through the described  
 316 task-vertex amalgamation procedure. In Figure 5a, the up and left arrows imply that

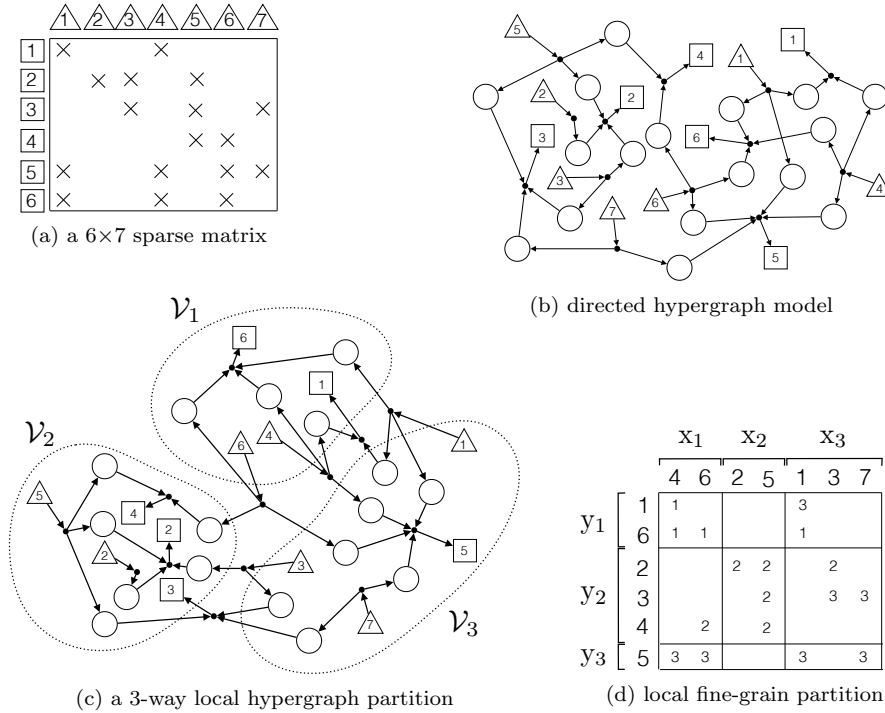


Fig. 4: An illustration of attaining a local fine-grain partition through vertex partitioning of the directed hypergraph model that satisfies locality constraints. The input- and output-data vertices are drawn with triangles and rectangles, respectively.

317 a task vertex  $v_z(ij)$  is amalgamated into input-data vertex  $v_x(j)$  and output-data  
 318 vertex  $v_y(i)$ , respectively. The reduced hypergraph obtained by these task-vertex  
 319 amalgamations is shown in Figure 5b. Figures 5c and 5d show a 3-way vertex partition  
 320 of this reduced hypergraph and the obtained local fine-grain partition, respectively. As  
 321 seen in these figures, task  $a_{35}$  is assigned to processor  $P_2$  since  $v_z(3, 5)$  is amalgamated  
 322 into  $v_x(5)$ , and  $v_x(5)$  is assigned to  $\mathcal{V}_2$ .

323 We emphasize here that the reduced hypergraph constructed as above is equiv-  
 324 alent to the hypergraph model of Pelt and Bisseling [15]. In that original work, the  
 325 use of this model was only for two-way partitioning (of the fine grain model) which is  
 326 then used for  $K$ -way fine-grain partitioning recursively. But this distorts the locality  
 327 of task vertices so that a partition obtained in further recursive steps is no more a  
 328 local fine-grain partition. That is why the adaptation was necessary.

329 **4.2. Nonzero distribution to minimize the total communication vol-**  
 330 **ume.** This method is composed of two parts. The first part finds a vector distri-  
 331 bution  $(\Pi(\mathbf{x}), \Pi(\mathbf{y}))$ . The second part finds a nonzero/task distribution  $\Pi(\mathbf{A})$  that  
 332 exactly minimizes the total communication volume over all possible local fine-grain  
 333 partitions which abide by  $(\Pi(\mathbf{x}), \Pi(\mathbf{y}))$  of the first part. The first part can be ac-  
 334 complished by any conventional data partitioning method such as 1D partitioning.  
 335 Therefore, this section is devoted to the second part.

336 Consider the block structure (3) of  $\mathbf{A}$  induced by  $(\Pi(\mathbf{x}), \Pi(\mathbf{y}))$ . Recall that in  
 337 a local fine-grain partition, due (11), the nonzero/task distribution is such that each

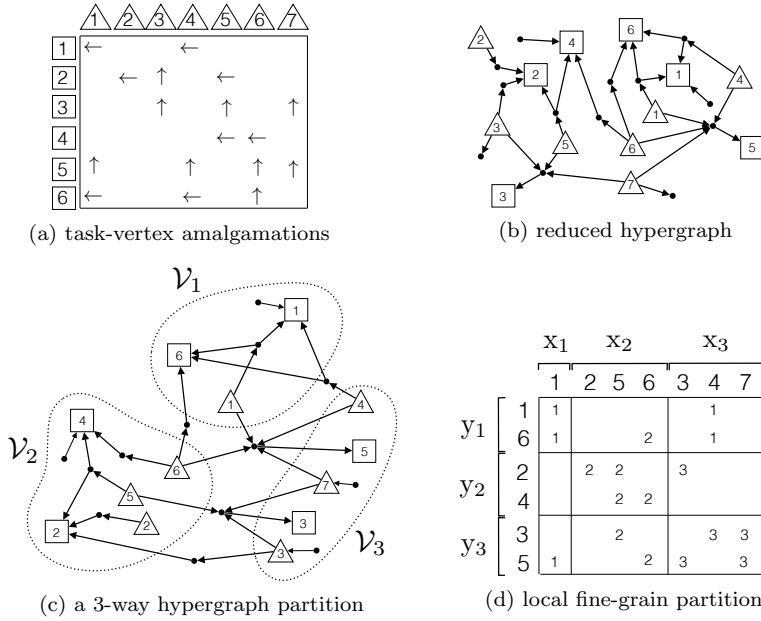


Fig. 5: An illustration of local fine-grain partitioning through task-vertex amalgamations. The input- and output-data vertices are drawn with triangles and rectangles, respectively. The figure on the bottom right shows the fine-grain partition.

338 diagonal block  $\mathbf{A}_{kk} = \mathbf{A}_{kk}^{(k)}$ , and each off-diagonal block  $\mathbf{A}_{k\ell}$  is a nonzero-disjoint  
 339 summation of the form  $\mathbf{A}_{k\ell} = \mathbf{A}_{k\ell}^{(k)} + \mathbf{A}_{k\ell}^{(\ell)}$ . This corresponds to assigning each  
 340 nonzero of  $\mathbf{A}_{kk}$  to  $P_k$ , for each diagonal block  $\mathbf{A}_{kk}$ , and assigning each nonzero of  $\mathbf{A}_{k\ell}$   
 341 to either  $P_k$  or  $P_\ell$ . Figure 6 illustrates a sample  $10 \times 12$  sparse matrix and its block  
 342 structure induced by a sample 3-way vector distribution which incurs four messages:  
 343 from  $P_3$  to  $P_1$ , from  $P_1$  to  $P_2$ , from  $P_3$  to  $P_2$ , and from  $P_2$  to  $P_3$  due to  $\mathbf{A}_{13}$ ,  $\mathbf{A}_{21}$ ,  
 344  $\mathbf{A}_{23}$  and  $\mathbf{A}_{32}$ , respectively.

345 Since diagonal blocks and zero off-diagonal blocks do not incur any communica-  
 346 tion, we focus on the nonzero off-diagonal blocks. Consider a nonzero off-diagonal  
 347 block  $\mathbf{A}_{k\ell}$  which incurs a message from  $P_\ell$  to  $P_k$ . The volume of this message is  
 348 determined by the distribution of tasks of  $\mathbf{A}_{k\ell}$  between  $P_k$  and  $P_\ell$ . This in turn im-  
 349 plies that distributing the tasks of each nonzero off-diagonal block can be performed  
 350 independently for minimizing the total communication volume.

351 In the local row-column-parallel algorithm,  $P_\ell$  sends  $[\hat{\mathbf{x}}_\ell^{(k)}, \hat{\mathbf{y}}_k^{(\ell)}]$  to  $P_k$ . Here,  $\hat{\mathbf{x}}_\ell^{(k)}$   
 352 corresponds to the nonzero columns of  $\mathbf{A}_{\ell k}^{(\ell)}$ , and  $\hat{\mathbf{y}}_k^{(\ell)}$  corresponds to the nonzero rows  
 353 of  $\mathbf{A}_{\ell k}^{(k)}$ , for a nonzero/task distribution  $\mathbf{A}_{k\ell} = \mathbf{A}_{k\ell}^{(k)} + \mathbf{A}_{k\ell}^{(\ell)}$ . Then, we can derive the  
 354 following formula for the communication volume  $\phi_{k\ell}$  from  $P_\ell$  to  $P_k$ :

$$355 \quad (12) \quad \phi_{k\ell} = \hat{n}(\mathbf{A}_{k\ell}^{(k)}) + \hat{m}(\mathbf{A}_{k\ell}^{(\ell)}),$$

356 where  $\hat{n}(\cdot)$  and  $\hat{m}(\cdot)$  refer to the number of nonzero columns and nonzero rows of the  
 357 input submatrix, respectively. The total communication volume  $\phi$  is then computed  
 358 by summing the communication volumes incurred by each nonzero off-diagonal block  
 359 of the block structure. Then, the problem of our interest can be described as follows.

360

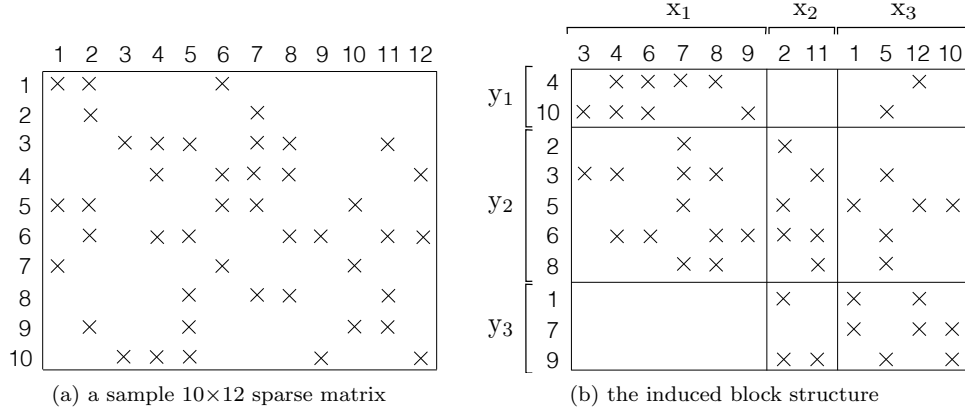


Fig. 6: A sample 10×12 sparse matrix  $\mathbf{A}$  and its block structure induced by input-data distribution  $\Pi(\mathbf{x}) = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{x}^{(3)}\}$  and output-data distribution  $\Pi(\mathbf{y}) = \{\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \mathbf{y}^{(3)}\}$ , where  $\mathbf{x}^{(1)} = \{x_3, x_4, x_6, x_7, x_8, x_9\}$ ,  $\mathbf{x}^{(2)} = \{x_2, x_{11}\}$ ,  $\mathbf{x}^{(3)} = \{x_1, x_5, x_{12}, x_{10}\}$ ,  $\mathbf{y}^{(1)} = \{y_4, y_{10}\}$ ,  $\mathbf{y}^{(2)} = \{y_2, y_3, y_5, y_6, y_8\}$ , and  $\mathbf{y}^{(3)} = \{y_1, y_7, y_9\}$ .

361        **PROBLEM 1.** Given  $\mathbf{A}$  and a vector distribution  $(\Pi(\mathbf{x}), \Pi(\mathbf{y}))$ , find a nonzero/task  
362 distribution  $\Pi(\mathbf{A})$  such that (i) each nonzero off-diagonal block has the form  $\mathbf{A}_{k\ell} =$   
363  $\mathbf{A}_{k\ell}^{(k)} + \mathbf{A}_{k\ell}^{(\ell)}$ ; (ii) each diagonal block  $\mathbf{A}_{kk} = \mathbf{A}_{kk}^{(k)}$  in the block structure induced by  
364  $(\Pi(\mathbf{x}), \Pi(\mathbf{y}))$ ; and (iii) the total communication volume  $\phi = \sum_{k \neq \ell} \phi_{k\ell}$  is minimized.

365        Let  $G_{k\ell} = (\mathcal{U}_{k\ell} \cup \mathcal{V}_{k\ell}, \mathcal{E}_{k\ell})$  be the bipartite graph representation of  $\mathbf{A}_{k\ell}$ , where  
366  $\mathcal{U}_{k\ell}$  and  $\mathcal{V}_{k\ell}$  are the set of vertices corresponding to the rows and columns of  $\mathbf{A}_{k\ell}$ ,  
367 respectively, and  $\mathcal{E}_{k\ell}$  is the set of edges corresponding to the nonzeros of  $\mathbf{A}_{k\ell}$ . Based  
368 on this notation, the following theorem states a correspondence between the problem  
369 of distributing nonzeros/tasks of  $\mathbf{A}_{k\ell}$  to minimize the communication volume  $\phi_{k\ell}$  from  
370  $P_\ell$  to  $P_k$  and the problem of finding a minimum vertex cover of  $G_{k\ell}$ . Before stating  
371 the theorem we give a brief definition of vertex covers for the sake of completeness.  
372 A subset of vertices of a graph is called *vertex cover* if each of the graph edges is  
373 incident to any of the vertices in this subset. A vertex cover is minimum if its size  
374 is the least possible. In bipartite graphs, the problem of finding a minimum vertex  
375 cover is equivalent to the problem of finding a maximum matching [13]. Aschraft and  
376 Liu [1] describe a similar application of vertex covers.

377        **THEOREM 1.** Let  $\mathbf{A}_{k\ell}$  be a nonzero off-diagonal block and  $G_{k\ell} = (\mathcal{U}_{k\ell} \cup \mathcal{V}_{k\ell}, \mathcal{E}_{k\ell})$   
378 be its bipartite graph representation.

- 379        1. For any vertex cover  $S_{k\ell}$  of  $G_{k\ell}$ , there is a nonzero distribution  $\mathbf{A}_{k\ell} = \mathbf{A}_{k\ell}^{(k)} +$   
380  $\mathbf{A}_{k\ell}^{(\ell)}$  such that  $|S_{k\ell}| \geq \hat{n}(\mathbf{A}_{k\ell}^{(k)}) + \hat{m}(\mathbf{A}_{k\ell}^{(\ell)})$ ,  
381        2. For any nonzero distribution  $\mathbf{A}_{k\ell} = \mathbf{A}_{k\ell}^{(k)} + \mathbf{A}_{k\ell}^{(\ell)}$ , there is a vertex cover  $S_{k\ell}$   
382 of  $G_{k\ell}$  such that  $|S_{k\ell}| = \hat{n}(\mathbf{A}_{k\ell}^{(k)}) + \hat{m}(\mathbf{A}_{k\ell}^{(\ell)})$ .

383        *Proof.* We prove the two parts of the theorem separately.

- 384        1) Take any vertex cover  $S_{k\ell}$  of  $G_{k\ell}$ . Consider any nonzero distribution  $\mathbf{A}_{k\ell} =$

385  $\mathbf{A}_{k\ell}^{(k)} + \mathbf{A}_{k\ell}^{(\ell)}$  such that

$$386 \quad (13) \quad a_{ij} \in \begin{cases} \mathbf{A}_{k\ell}^{(k)} & \text{if } v_j \in S_{k\ell} \text{ and } u_i \notin S_{k\ell}, \\ \mathbf{A}_{k\ell}^{(\ell)} & \text{if } v_j \notin S_{k\ell} \text{ and } u_i \in S_{k\ell}, \\ \mathbf{A}_{k\ell}^{(k)} \text{ or } \mathbf{A}_{k\ell}^{(\ell)} & \text{if } v_j \in S_{k\ell} \text{ and } u_i \in S_{k\ell}. \end{cases}$$

387 Since  $v_j \in S_{k\ell}$  for every  $a_{ij} \in \mathbf{A}_{k\ell}^{(k)}$  and  $u_i \in S_{k\ell}$  for every  $a_{ij} \in \mathbf{A}_{k\ell}^{(\ell)}$ ,  $|S_{k\ell} \cap \mathcal{V}_{k\ell}| \geq$   
 388  $\hat{n}(\mathbf{A}_{k\ell}^{(k)})$  and  $|S_{k\ell} \cap \mathcal{U}_{k\ell}| \geq \hat{m}(\mathbf{A}_{k\ell}^{(\ell)})$ , which in turn leads to

$$389 \quad (14) \quad |S_{k\ell}| \geq \hat{n}(\mathbf{A}_{k\ell}^{(k)}) + \hat{m}(\mathbf{A}_{k\ell}^{(\ell)}).$$

390 2) Take any nonzero distribution  $\mathbf{A}_{k\ell} = \mathbf{A}_{k\ell}^{(k)} + \mathbf{A}_{k\ell}^{(\ell)}$ . Consider  $S_{k\ell} = \{u_i \in U_{k\ell} :$   
 391  $a_{ij} \in \mathbf{A}_{k\ell}^{(\ell)}\} \cup \{v_j \in V_{k\ell} : a_{ij} \in \mathbf{A}_{k\ell}^{(k)}\}$  where  $|S_{k\ell}| = \hat{n}(\mathbf{A}_{k\ell}^{(k)}) + \hat{m}(\mathbf{A}_{k\ell}^{(\ell)})$ . Now, consider  
 392 a nonzero  $a_{ij} \in \mathbf{A}_{k\ell}$  and its corresponding edge  $\{u_i, v_j\} \in \mathcal{E}_{k\ell}$ . If  $a_{ij} \in \mathbf{A}_{k\ell}^{(k)}$ , then  
 393  $v_j \in S_{k\ell}$ . Otherwise,  $u_i \in S_{k\ell}$  since  $a_{ij} \in \mathbf{A}_{k\ell}^{(\ell)}$ . So,  $S_{k\ell}$  is a vertex cover of  $G_{k\ell}$ .  $\square$

394 At this point, however, it is still not clear how the reduction from the problem  
 395 of distributing the nonzeros/tasks to the problem of finding the minimum vertex  
 396 cover holds. For this purpose, using Theorem 1, we show that a minimum vertex  
 397 cover of  $G_{k\ell}$  can be decoded as a nonzero distribution of  $\mathbf{A}_{k\ell}$  with the minimum  
 398 communication volume  $\phi_{k\ell}$  as follows. Let  $S_{k\ell}^*$  be a minimum vertex cover of  $G_{k\ell}$  and  
 399  $\phi_{k\ell}^*$  be the minimum communication volume incurred by a nonzero/task distribution  
 400 of  $\mathbf{A}_{k\ell}$ . Then,  $|S_{k\ell}^*| = \phi_{k\ell}^*$ , since the first and second parts of Theorem 1 imply  $|S_{k\ell}^*| \geq$   
 401  $\phi_{k\ell}^*$  and  $|S_{k\ell}^*| \leq \phi_{k\ell}^*$ , respectively. We decode an optimal nonzero/task distribution  
 402  $\mathbf{A}_{k\ell} = \mathbf{A}_{k\ell}^{(k)} + \mathbf{A}_{k\ell}^{(\ell)}$  out of  $S_{k\ell}^*$  according to (13) where one such distribution is

$$403 \quad (15) \quad \mathbf{A}_{k\ell}^{(k)} = \{a_{ij} \in \mathbf{A}_{k\ell} : v_j \in S_{k\ell}^*\} \text{ and } \mathbf{A}_{k\ell}^{(\ell)} = \{a_{ij} \in \mathbf{A}_{k\ell} : v_j \notin S_{k\ell}^*\}.$$

404 Let  $\phi_{k\ell}$  be the communication volume incurred by this nonzero/task distribution.  
 405 Then,  $|S_{k\ell}^*| \geq \phi_{k\ell}$  due to (14), and  $\phi_{k\ell} = \phi_{k\ell}^*$  since  $\phi_{k\ell}^* = |S_{k\ell}^*| \geq \phi_{k\ell} \geq \phi_{k\ell}^*$ .

406 Figure 7 illustrates the reduction on a sample  $5 \times 6$  nonzero off-diagonal block  
 407  $\mathbf{A}_{k\ell}$ . The left side and middle of this figure respectively display  $\mathbf{A}_{k\ell}$  and its bipartite  
 408 graph representation  $G_{k\ell}$ , which contains 5 row vertices and 6 column vertices. On  
 409 the middle of the figure, a minimum vertex cover  $S_{k\ell}$  that contains two row vertices  
 410  $\{u_3, u_6\}$  and two column vertices  $\{v_7, v_8\}$  is also shown. The right side of the figure  
 411 displays how this minimum vertex cover is decoded as a nonzero/task distribution  
 412  $\mathbf{A}_{k\ell} = \mathbf{A}_{k\ell}^{(k)} + \mathbf{A}_{k\ell}^{(\ell)}$ . As a result of this decoding,  $P_\ell$  sends  $[x_7, x_8, \hat{y}_3, \hat{y}_6]$  to  $P_k$  in a  
 413 single message. Note that a nonzero corresponding to an edge connecting two cover  
 414 vertices can be assigned to either  $\mathbf{A}_{k\ell}^{(k)}$  or  $\mathbf{A}_{k\ell}^{(\ell)}$  without changing the communication  
 415 volume from  $P_\ell$  to  $P_k$ . The only change that may occur is in the values of partially  
 416 computed output-vector entries to be communicated. For instance, in the figure,  
 417 nonzero  $a_{37}$  is assigned to  $\mathbf{A}_{k\ell}^{(k)}$ . Since both  $u_3$  and  $v_7$  are cover vertices,  $a_{37}$  could be  
 418 assigned to  $\mathbf{A}_{k\ell}^{(\ell)}$  with no change in the communicated entries but the value of  $\hat{y}_3$ .

419 Algorithm 3 gives a sketch of our method to find a nonzero/task distribution that  
 420 minimizes the total communication volume based on Theorem 1. For each nonzero off-  
 421 diagonal block  $\mathbf{A}_{k\ell}$ , the algorithm first constructs  $G_{k\ell}$ , then obtains a minimum vertex  
 422 cover  $S_{k\ell}$ , and then decodes  $S_{k\ell}$  as a nonzero/task distribution  $\mathbf{A}_{k\ell} = \mathbf{A}_{k\ell}^{(k)} + \mathbf{A}_{k\ell}^{(\ell)}$   
 423 according to (15). Hence, the communication volume incurred by  $\mathbf{A}_{k\ell}$  is equal to the  
 424 size of the cover  $|S_{k\ell}|$ . In detail, each row vertex  $u_i$  on the cover incurs an output

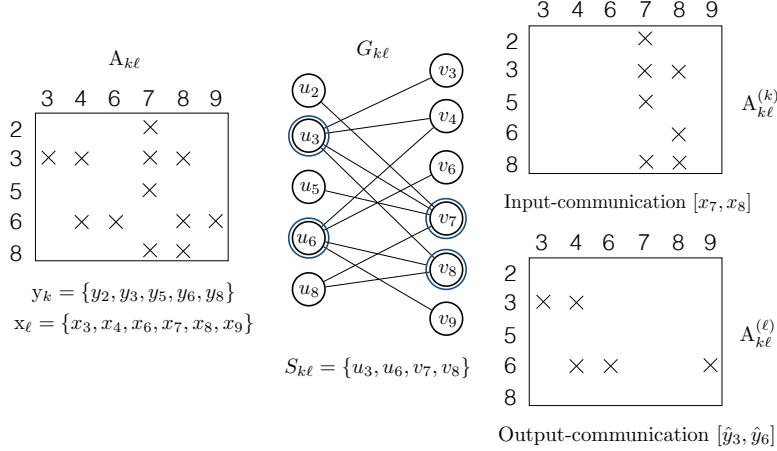


Fig. 7: The minimum vertex cover model for minimizing the communication volume  $\phi_{k\ell}$  from  $P_\ell$  to  $P_k$ . According to the vertex cover  $S_{k\ell}$ ,  $P_\ell$  sends  $[x_7, x_8, \hat{y}_3, \hat{y}_6]$  to  $P_k$ .

425 communication of  $\hat{y}_i \in \hat{\mathbf{y}}_\ell^{(k)}$ , and each column vertex  $v_j$  on the cover incurs an input  
 426 communication of  $x_j \in \hat{\mathbf{x}}_k^{(\ell)}$ . We recall that  $P_\ell$  sends  $\hat{\mathbf{y}}_\ell^{(k)}$  and  $\hat{\mathbf{x}}_k^{(\ell)}$  to  $P_k$  in a single  
 427 message in the proposed row-column-parallel sparse matrix-vector multiply algorithm.

---

**Algorithm 3** Nonzero/task distribution to minimize the total communication volume

---

- 1: **procedure** NONZEROTASKDISTRIBUTEVOLUME( $\mathbf{A}, \Pi(\mathbf{x}), \Pi(\mathbf{y})$ )
  - 2:   **for each** nonzero off-diagonal block  $\mathbf{A}_{k\ell}$  **do** ▶ See (3)
  - 3:     Construct  $G_{k\ell} = (\mathcal{U}_{k\ell} \cup \mathcal{V}_{k\ell}, \mathcal{E}_{k\ell})$  ▶ Bipartite graph representation
  - 4:      $S_{k\ell} \leftarrow \text{MINIMUMVERTEXCOVER}(G_{k\ell})$
  - 5:     **for each** nonzero  $a_{ij} \in \mathbf{A}_{k\ell}$  **do**
  - 6:       **if**  $v_j \in S_{k\ell}$  **then** ▶  $v_j \in \mathcal{V}_{k\ell}$  is a column vertex and  $v_j \in S_{k\ell}$
  - 7:           $\mathbf{A}_{k\ell}^{(k)} \leftarrow \mathbf{A}_{k\ell}^{(k)} \cup \{a_{ij}\}$
  - 8:       **else** ▶  $u_i \in \mathcal{U}_{k\ell}$  is a row vertex and  $u_i \in S_{k\ell}$
  - 9:           $\mathbf{A}_{k\ell}^{(\ell)} \leftarrow \mathbf{A}_{k\ell}^{(\ell)} \cup \{a_{ij}\}$
- 

428 Figure 8 illustrates the steps of Algorithm 3 on the block structure given in  
 429 Figure 6b. Figure 8a shows four bipartite graphs each corresponding to a nonzero  
 430 off-diagonal block. In this figure, a minimum vertex cover for each bipartite graph  
 431 is also shown. Figure 8b illustrates how to decode a local fine-grain partition from  
 432 those minimum vertex covers. In this figure, the nonzeros are represented with the  
 433 processor to which they are assigned. As seen in the figure, the number of entries sent  
 434 from  $P_1$  to  $P_2$  is four, that is,  $\phi_{21} = 4$ , and the number of entries sent from  $P_3$  to  $P_1$ ,  
 435 from  $P_3$  to  $P_2$  and from  $P_2$  to  $P_3$  are all two, that is,  $\phi_{13} = \phi_{23} = \phi_{32} = 2$ .

436 We note here that the objective of this method is to minimize the total com-  
 437 munication volume under a given vector distribution. Since blocks of nonzeros are  
 438 assigned, a strict load balance cannot be always maintained.

439 **5. Related work.** Here we review recent related work on matrix partitioning  
 440 for parallel SpMV.

441 Kuhlemann and Vassilevski [14] recognize the need to reduce the number of mes-  
 442 sages in parallel sparse matrix vector multiply operations with matrices corresponding



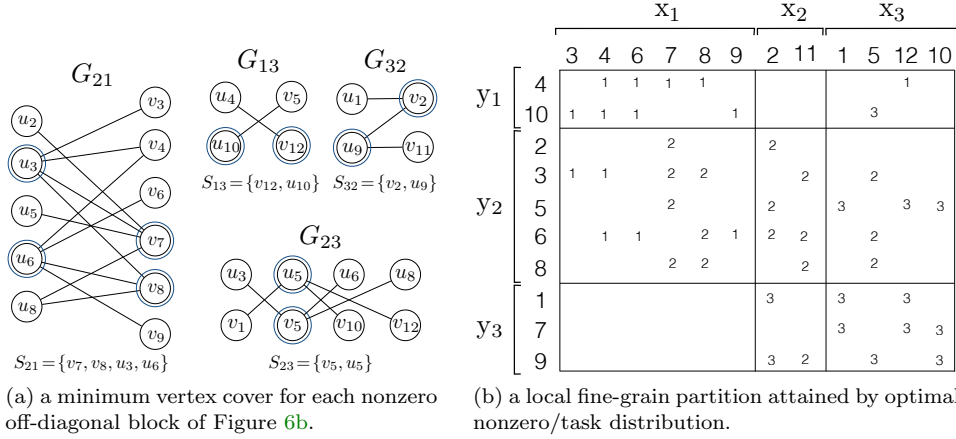


Fig. 8: An optimal nonzero distribution minimizing the total communication volume obtained by Algorithm 3. The matrix nonzeros are represented with the processors they are assigned to. The total communication volume is 10, where  $P_1$  sends  $[x_7, x_8, \hat{y}_3, \hat{y}_6]$  to  $P_2$ ;  $P_3$  sends  $[x_{12}, \hat{y}_{10}]$  to  $P_1$ ;  $P_3$  sends  $[x_2, \hat{y}_9]$  to  $P_1$ ; and  $P_3$  sends  $[x_5, \hat{y}_5]$  to  $P_2$ .

443 to scale-free graphs. They present methods to embed the given graph in a bigger one  
 444 to reduce the number of messages. The gist of the method is to split a vertex into  
 445 a number of copies (the number is determined with a simple calculation to limit the  
 446 maximum number of messages per processor). In such a setting, the SpMV opera-  
 447 tions with the matrix associated with the original graph,  $\mathbf{y} \leftarrow \mathbf{A}\mathbf{x}$ , is then cast as triple  
 448 sparse matrix vector products of the form  $\mathbf{y} \leftarrow \mathbf{Q}^T(\mathbf{B}(\mathbf{Q}\mathbf{x}))$ . This original work can  
 449 be extended to other matrices (not necessarily symmetric, nor square) by recognizing  
 450 the triplet product as a communication on  $\mathbf{x}$  for duplication (for the columns that  
 451 are split), communication of  $\mathbf{x}$  vector entries (duplicates are associated with different  
 452 destinations), multiplication, and as a communication on the output vector (for the  
 453 rows that are split) to gather results. This exciting extension requires further analysis.

454 Boman et al. [2] propose a 2D partitioning method obtained by post-processing  
 455 a 1D partition. Given a 1D partition among  $P$  processors, the method maps the  
 456  $P \times P$  block structure to a virtual mesh of size  $P_r \times P_c$  and reassigns the off-diagonal  
 457 blocks so as to limit the number of messages per processor by  $P_r + P_c$ . The post-  
 458 processing is fast, and hence the method is as nearly efficient as a 1D partitioning  
 459 method. However, the communication volume and the computational load balance  
 460 obtained in the 1D partitioning phase are disturbed and the method does not have  
 461 any means to control the perturbation. The proposed two-part method (Section 4.2),  
 462 is similar to this work in this aspect; a strict balance cannot always be achieved; yet  
 463 a finer approach is discussed in the preliminary version of the paper [12].

464 Pelt and Bisseling [15] propose a model to partition sparse matrices into two  
 465 parts (which then can be used recursively to partition into any number of parts). The  
 466 essential idea has two steps. First, the nonzeros of a given matrix  $\mathbf{A}$  are split into  
 467 two different matrices (of the same size as the original matrix), say  $\mathbf{A} = \mathbf{A}_r + \mathbf{A}_c$ .  
 468 Second,  $\mathbf{A}_r$  and  $\mathbf{A}_c$  are partitioned together, where  $\mathbf{A}_r$  is partitioned rowwise, and  
 469  $\mathbf{A}_c$  is partitioned columnwise. As all nonzeros of  $\mathbf{A}$  are in only one of  $\mathbf{A}_r$  or  $\mathbf{A}_c$ , the  
 470 final result is a two-way partitioning of the nonzeros of  $\mathbf{A}$ . The resulting partition on

471 **A** achieves load balance and reduces the total communication volume by the standard  
 472 hypergraph partitioning techniques.

473 Two-dimensional partitioning methods that bound the maximum number of mes-  
 474 sages per processor, such as the checkerboard [5, 8] and orthogonal recursive bisection  
 475 [17] based methods, have been used in modern applications [18, 20], sometimes  
 476 without graph/hypergraph partitioning [19]. In almost all cases, inadequacy of 1D  
 477 partitioning schemes are confirmed.

478 All previous work (including those that were summarized above) assumes the  
 479 standard SpMV algorithm based on expanding  $\mathbf{x}$ -vector entries, performing multiplies  
 480 with matrix entries, and folding  $\mathbf{y}$ -vector entries. Compared to all these previous work,  
 481 ours has therefore a distinctive characteristic. In this work, we introduce the novel  
 482 concept of heterogeneous messages where  $\mathbf{x}$ -vector and partially computed  $\mathbf{y}$ -vector  
 483 entries are possibly communicated within the same message packet. In order to make  
 484 use of this, we search for a special 2D partition on the matrix nonzeros in which a  
 485 nonzero is assigned to a processor holding either the associated input-vector entry, or  
 486 the associated output-vector entry, or both. The implication is that the proposed local  
 487 row-column-parallel SpMV algorithm requires only a single communication phase (all  
 488 the previous algorithms based on 2D partitions require two communication phases)  
 489 as is the case for the parallel algorithms based on 1D partitions; yet the proposed  
 490 algorithm achieves a greater flexibility to reduce the communication volume than the  
 491 1D methods.

492 **6. Experiments.** We performed our experiments on a large selection of sparse  
 493 matrices obtained from the University of Florida (UFL) sparse matrix collection [9].  
 494 We used square and structurally symmetric matrices with 500–10M nonzeros. At the  
 495 time of experiments, we had 904 such matrices. We discarded 14 matrices as they  
 496 contain diagonal entries only, and we also excluded one matrix (`kron_g500-logn16`)  
 497 because it took extremely long to have a partition with the hypergraph partitioning  
 498 tool used in the experiments. We conducted our experiments for  $K = 64$  and  $K =$   
 499  $1024$  and omit the cases when the number of rows is less than  $50 \times K$ . As a result, we  
 500 had 566 and 168 matrices for the experiments with  $K = 64$  and  $1024$ , respectively.  
 501 We separate all our test matrices into two groups according to the maximum number  
 502 of nonzeros per row/column, more precisely, according to whether the test matrix  
 503 contains a dense row/column or not. We say a row/column dense if it contains at  
 504 least  $10\sqrt{m}$  nonzeros, where  $m$  denotes the number of rows/columns. Hence, for  
 505  $K = 64$  and  $1024$ , the first group respectively contains 477 and 142 matrices that  
 506 have no dense rows/columns out of 566 and 168 test matrices. The second group  
 507 contains the remaining 89 and 26 matrices, each having some dense rows/column, for  
 508  $K = 64$  and  $1024$ , respectively.

509 In the experiments, we evaluated the partitioning qualities of the local fine-grain  
 510 partitioning methods proposed in Section 4 against 1D rowwise (1D-H [3]), the 2D  
 511 fine-grain (2D-H [4]), and two checkerboard partitioning methods (2D-B [2], 2D-C [5]).  
 512 For the method proposed in Section 4.1, we obtain a local fine-grain partition through  
 513 the directed hypergraph model (1.5D-H) using the procedure described at the end of  
 514 that subsection. For the method proposed in Section 4.2 (1.5D-V), the required vector  
 515 distribution is obtained by 1D rowwise partitioning using the column-net hypergraph  
 516 model. Then, we obtain a local fine-grain partition on this vector distribution with a  
 517 nonzero/task distribution that minimizes the total communication volume.

518 The 1D-H, 2D-H, 2D-C and 1.5D-H methods are based on hypergraph models. Al-  
 519 though all these models allow arbitrary distribution of the input- and output-vectors,

520 in the experiments, we consider conformal partitioning of input and output vectors,  
 521 by using vertex amalgamation of the input- and output-vector entries [16]. We used  
 522 PaToH [3, 6] with default parameters where the maximum allowable imbalance ratio  
 523 is 3% for partitioning. We also notice that the 1.5D-V and 2D-B methods are based  
 524 on 1D-H and keeps the vector distribution obtained from 1D-H intact. Hence, in the  
 525 experiments, the input and output vectors for those methods are conformal as well.  
 526 Finally, since PaToH depends on randomization, we report the geometric mean of ten  
 527 different runs for each partitioning instance.

528 In all experiments, we report the results using performance profiles [10] which is  
 529 very helpful in comparing multiple methods over a large collection of test cases. In a  
 530 performance profile, we compare methods according to the best performing method  
 531 for each test case and measure in what fraction of the test cases a method performs  
 532 within a factor of the best observed performance. For example, a point (abscissa =  
 533 1.05, ordinate = 0.30) on the performance curve of a given method refers to the fact  
 534 that for 30% of the test cases, the method performs within a factor of 1.05 of the best  
 535 observed performance. As a result, a method that is closer to top-left corner is better.  
 536 In the load balancing performance profiles displayed in Figures 9b, 9d, 10b and 10d,  
 537 we compare performance results with respect to the performance of perfect balance  
 538 instead best observed performance. That is, a point (abscissa = 6% and ordinate =  
 539 0.40) on the performance curve of a given method means that for 40% of the test  
 540 cases, the method produces a load imbalance ratio less than or equal to 6%.

541 Figures 9 and 10 both display performance profiles of four task-and-data distri-  
 542 bution methods in terms of the total communication volume and the computational  
 543 load imbalance. Figure 9 displays performance profiles for the set of matrices with  
 544 no dense rows/columns, whereas Figure 10 displays performance profiles for the set  
 545 of matrices containing dense rows/columns.

546 As seen in Figure 9, for the set of matrices with no dense rows/columns, the  
 547 relative performances of all methods are similar for  $K = 64$  and  $K = 1024$  in terms  
 548 of both communication volume and load imbalance. As seen in Figures 9a and 9c,  
 549 all methods except the 1.5D-H method achieve a total communication volume at most  
 550 30% more than the best in almost 80% of the cases in this set of matrices. As seen  
 551 in these two figures, the proposed 1.5D-V method performs significantly better than  
 552 all other methods, whereas the 2D-H method is the second best performing method.  
 553 As also seen in the figures, 1D-H displays the third best performance, whereas 1.5D-H  
 554 shows the worst performance. As seen in Figures 9b and 9d, in terms of load balance,  
 555 the 2D-H method is the best performing method. As also seen in the figures, the  
 556 proposed 1.5D-V method displays considerably worse performance than the others.  
 557 Specifically, all methods except 1.5D-V achieve a load imbalance below 3% in almost all  
 558 test cases. In terms of the total communication volume, 2D checkerboard partitioning  
 559 methods perform considerably worse than 1.5D-V, 2D-H and 1D-H methods. The first  
 560 alternative 2D-B obtains better results than 2D-C. For load balance, 2D-C behaves  
 561 similar to 1D-H, 2D-H and 1.5D-H methods except that 2D-C achieves a load imbalance  
 562 below 5% (instead of 3%) for almost all instances. 2D-B behaves similar to 1.5D-V,  
 563 and does not achieve a good load balance.

564 As seen in Figure 10, for the set of matrices with some dense rows/columns, all  
 565 methods display a similar performance for  $K = 64$  and  $K = 1024$  in terms of the  
 566 total communication volume. As in the previous dataset, in terms of the total com-  
 567 munication volume, the 1.5D-V and 2D-H methods are again the best and second best  
 568 methods, respectively, as seen in Figures 10a and 10c. As also seen in these figures,  
 569 1.5D-H is the third best performing method in terms of the total communication vol-

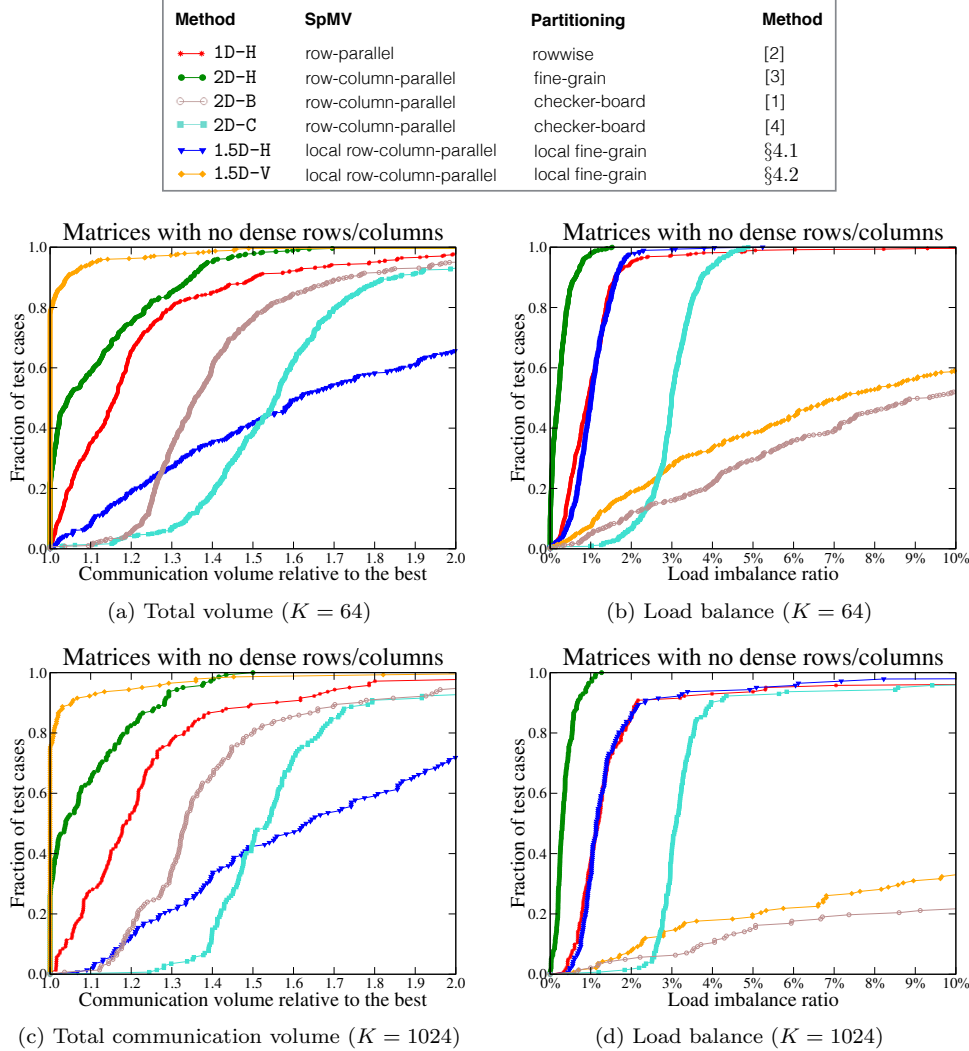


Fig. 9: Performance profiles comparing the total communication volume and load balance using test matrices with no dense rows/columns for  $K = 64$  and  $1024$ .

570 ume, whereas 1D-H shows considerably worse performance. The 2D-H method achieves  
 571 near-to-perfect load balance in almost all cases, as seen in Figures 10b and 10d. As  
 572 also seen in these figures, the 1.5D-H method displays a load imbalance lower than ap-  
 573 proximately 6% and 14% for all test matrices for  $K = 64$  and  $1024$ , respectively. This  
 574 shows the success of the vertex amalgamation procedure within the context of the  
 575 directed hypergraph model described in Section 4.1. As seen in Figure 10c, the total  
 576 communication volume does not exceed the best method by 40% in about 75% and  
 577 85% of the test cases for the 1.5D-H and 2D-H methods, respectively, for  $K = 1024$ .  
 578 The two 2D checkerboard methods display considerably worse performance than the  
 579 others (except 1D-H, which also shows a poor performance) in terms of the total com-  
 580 munication volume. When  $K = 64$ , 2D-C shows an acceptable performance however  
 581 when  $K = 1024$  its performance considerably deteriorates in terms of load balance.  
 582 2D-B obtains worse results. This not surprising since 2D-B is a modification of 1D-H

Method	SpMV	Partitioning	Method
1D-H	row-parallel	rowwise	[2]
2D-H	row-column-parallel	fine-grain	[3]
2D-B	row-column-parallel	checker-board	[1]
2D-C	row-column-parallel	checker-board	[4]
1.5D-H	local row-column-parallel	local fine-grain	§4.1
1.5D-V	local row-column-parallel	local fine-grain	§4.2

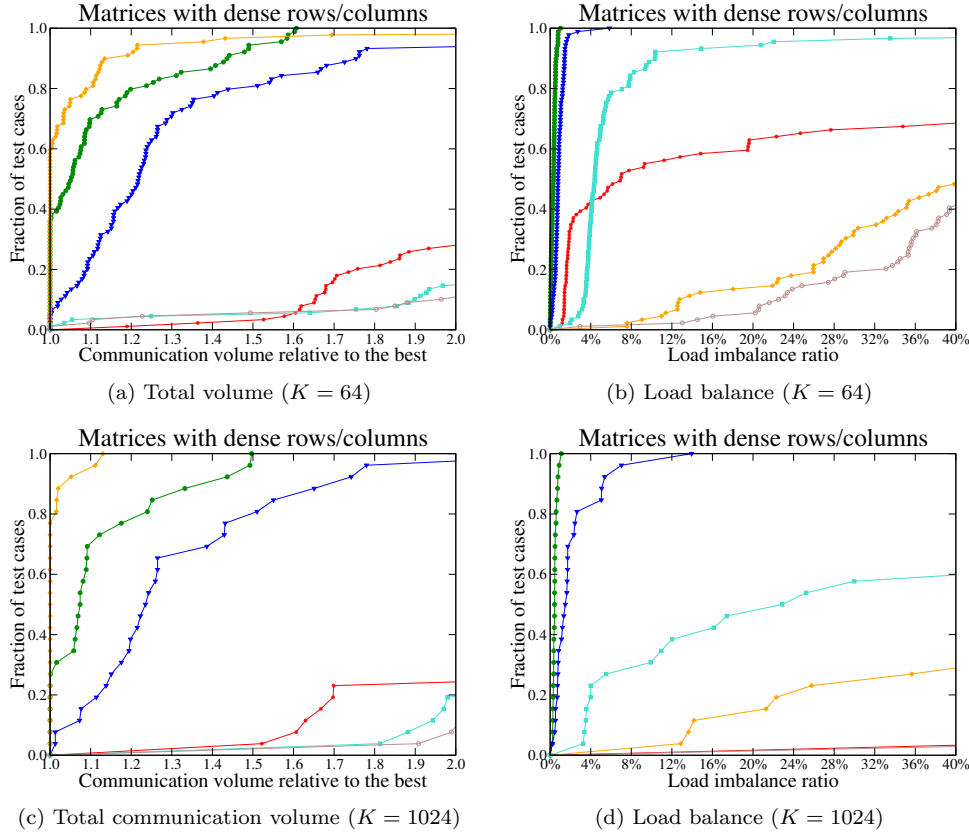


Fig. 10: Performance profiles comparing the total communication volume and the load balance on test matrices with dense rows/columns for  $K = 64$  and  $1024$ .

583 whose load balance performance is already very poor.

584 Figures 11a and 11b compare the methods in terms of the total and maximum  
585 message counts, respectively, using all test matrices for  $K = 1024$ . We note that  
586 these are secondary metrics and none of the methods addresses them explicitly as the  
587 main objective function. Since 1.5D-V uses the conformal distribution of the input-  
588 and output-vectors obtained from 1D-H, the total and the maximum message count  
589 of 1.5D-V are equivalent to those of 1D-H in these experiments. As seen in the figures,  
590 in terms of the total and the maximum message counts, 2D-B, 2D-C and 1D-H (also  
591 1.5D-V) display the best performance, 2D-H performs considerably poor and 1.5D-H  
592 performs in between. At a finer look, the method 2D-B is the winner with both  
593 metrics. 1.5D-V (as 1D-H) and the other checkerboard method 2D-C follows it, where  
594 2D checkerboard methods show clearer advantage.

595 Figure 11c compares all four methods in terms of the maximum communication

596 volume sent from a processor for  $K = 1024$ . The 1.5D-V method performs significantly  
 597 better than all others, 2D-H is the second best performing method, 1D displays the  
 598 third best, and 1.5D-H displays the worst performance. These relative performances of  
 599 the methods in terms of the maximum communication volume resemble their relative  
 600 performances in terms of the total communication volume as expected.

601 Figure 11d compares the methods in terms of partitioning times for  $K = 1024$ .  
 602 The run time of the 1.5D-V method involves the time spent for obtaining the vector  
 603 distribution, which is the run time of the 1D-H method in our case. As seen in  
 604 the figure, the 1D-H, 1.5D-V and 1.5D-H methods display comparable performances,  
 605 whereas the 2D-H method takes significantly longer. The longer run time of 2D-H stems  
 606 from the large size of the hypergraph model. 2D-B displays comparable performance  
 607 (in terms of running-time) with that of 1D-H, 1.5D-V and 1.5D-H methods. Meanwhile,  
 608 2D-C is considerably slower than all others except 2D-H.

609 In summary, the 1.5D-H method is a promising alternative for sparse matrices  
 610 with dense rows/columns. It obtains a total communication volume close to 2D-H,  
 611 near-perfect balance, considerably lower message count than 2D-H, and has short par-  
 612 titioning time. The 1.5D-V method performs at the extremes: the best for the total  
 613 communication volume, and the worst for the load balance, especially for matrices  
 614 with dense rows/columns. Nevertheless, 1.5D-V could still be favorable to other meth-  
 615 ods for particular matrices due to lower communication volume. In short, if a sparse  
 616 matrix contains dense rows/columns, then 1.5D-H seems to be the method of choice in  
 617 general; otherwise, 1.5D-V and 1D-H are reasonable alternatives competing with each  
 618 other. The 2D checkerboard based methods perform worse than the 1.5D methods,  
 619 but they have good performance in terms of the message count based metrics. In  
 620 particular, 2D-B is a fast method with a striking performance in reducing the latency,  
 621 but load balance can be an issue. These could be deciding factors for large scale  
 622 systems. On the other hand, 2D-C obtains better balance than 2D-B, but is slower.

623 **7. Conclusion and further discussions.** This paper introduced 1.5D paral-  
 624 lelism for the sparse matrix-vector multiply (SpMV) operations. We presented the  
 625 local row-column parallel SpMV that uses this novel parallelism. This multiply algo-  
 626 rithm is the fourth one in the literature for SpMV in addition to the well-known 1D  
 627 row-parallel, 1D column-parallel and 2D row-column-parallel ones. In this paper, we  
 628 also proposed two methods (1.5D-H and 1.5D-V) to distribute tasks and data in accor-  
 629 dance with the requirements of the proposed 1.5D parallel algorithm. Using a large  
 630 set of matrices from the UFL sparse matrix collection, we compared the partitioning  
 631 qualities of these two methods against the standard 1D and 2D methods.

632 The experiments suggest the use of the local row-column-parallel SpMV with a  
 633 local fine-grain partition obtained by the proposed directed hypergraph model for  
 634 matrices with dense rows/columns. This is because the performance of the proposed  
 635 1.5D partitioning is close to that of 2D fine-grain partitioning (2D-H) in terms of  
 636 the partitioning quality, with considerably less number of messages and much faster  
 637 execution.

638 We considered the problem mainly from a theoretical point of interest and leave  
 639 the performance of 1.5D parallel SpMV algorithms in terms of the parallel multiply  
 640 timings as a future work. We note that the main ideas behind the proposed 1.5D  
 641 parallelism, such as heterogeneous messaging and avoiding nonlocal tasks by a locality  
 642 constraint on partitioning, are of course not restricted to the parallel SpMV operation  
 643 and these ideas can be extended to other parallel computations as well.



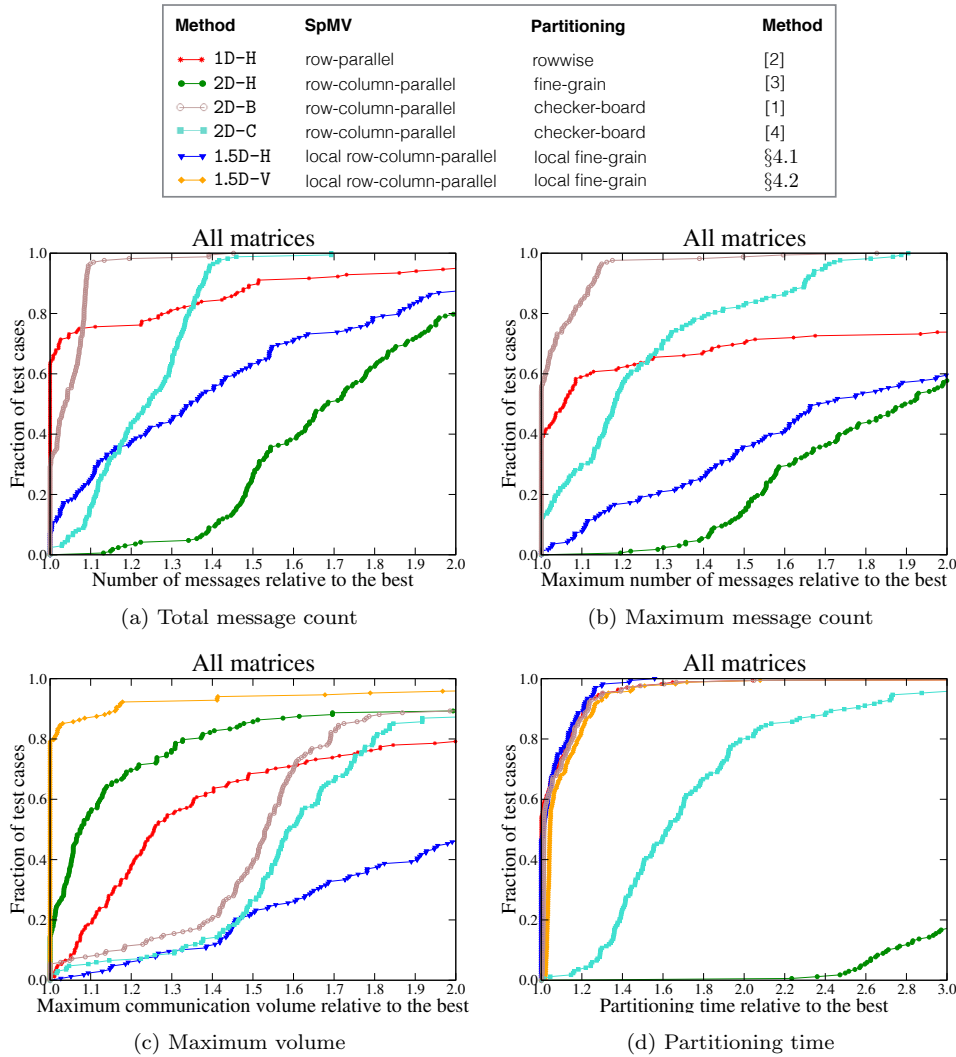


Fig. 11: Performance profiles comparing the total message count and the maximum message count for three methods 1D-H, 2D-H and 1.5D-H, maximum communication volume per processor and partitioning time for all methods on all test matrices for  $K = 1024$ . In 11a and 11b, 1.5D-V's profiles are identical to that of 1D-H, and hence not shown.

644

## REFERENCES

- 645 [1] C. ASHCRAFT AND J. W. H. LIU, *Applications of the Dulmage-Mendelsohn decomposition and*  
646 *network flow to graph bisection improvement*, SIAM Journal on Matrix Analysis and Ap-  
647 *plications*, 19 (1998), pp. 325–354.
- 648 [2] E. G. BOMAN, K. D. DEVINE, AND S. RAJAMANICKAM, *Scalable matrix computations on large*  
649 *scale-free graphs using 2d graph partitioning*, in Proceedings of the International Conference  
650 *on High Performance Computing, Networking, Storage and Analysis, SC '13*, New York,  
651 *NY, USA, 2013*, ACM, pp. 50:1–50:12.
- 652 [3] Ü. ÇATALYÜREK AND C. AYKANAT, *Hypergraph-partitioning-based decomposition for parallel*  
653 *sparse-matrix vector multiplication*, Parallel and Distributed Systems, IEEE Transactions  
654 *on*, 10 (1999), pp. 673–693.



- 655 [4] Ü. ÇATALYÜREK AND C. AYKANAT, *A fine-grain hypergraph model for 2d decomposition of*  
656 *sparse matrices*, Parallel and Distributed Processing Symposium, International, 3 (2001),  
657 p. 30118b.
- 658 [5] Ü. ÇATALYÜREK AND C. AYKANAT, *A hypergraph-partitioning approach for coarse-grain decom-*  
659 *position*, in Supercomputing, ACM/IEEE 2001 Conference, IEEE, 2001, pp. 42–42.
- 660 [6] Ü. ÇATALYÜREK AND C. AYKANAT, *Patoh (partitioning tool for hypergraphs)*, in Encyclopedia  
661 of Parallel Computing, Springer, 2011, pp. 1479–1487.
- 662 [7] Ü. ÇATALYÜREK, C. AYKANAT, AND B. UÇAR, *On two-dimensional sparse matrix partitioning:*  
663 *Models, methods, and a recipe*, SIAM Journal on Scientific Computing, 32 (2010), pp. 656–  
664 683.
- 665 [8] Ü. V. ÇATALYÜREK, C. AYKANAT, AND B. UÇAR, *On two-dimensional sparse matrix partition-*  
666 *ing: Models, methods, and a recipe*, SIAM J. Sci. Comput., 32 (2010), pp. 656–683.
- 667 [9] T. A. DAVIS AND Y. HU, *The University of Florida sparse matrix collection*, ACM Trans. Math.  
668 Softw., 38 (2011), pp. 1:1–1:25.
- 669 [10] E. D. DOLAN AND J. J. MORÉ, *Benchmarking optimization software with performance profiles*,  
670 Mathematical programming, 91 (2002), pp. 201–213.
- 671 [11] K. KAYA, B. UÇAR, AND U. V. ÇATALYÜREK, *Analysis of partitioning models and metrics in*  
672 *parallel sparse matrix-vector multiplication*, in Parallel Processing and Applied Mathemat-  
673 ics (PPAM2014), R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Waśniewski, eds.,  
674 Lecture Notes in Computer Science, Warsaw, Poland, 2014, Springer Berlin Heidelberg,  
675 pp. 174–184.
- 676 [12] E. KAYAASLAN, B. UÇAR, AND C. AYKANAT, *Semi-two-dimensional partitioning for parallel*  
677 *sparse matrix-vector multiplication*, in Parallel and Distributed Processing Symposium  
678 Workshop (IPDPSW), 2015 IEEE International, IEEE, 2015, pp. 1125–1134.
- 679 [13] D. KONIG, *Gráfok és mátrixok. matematikai és fizikai lapok*, 38: 116–119, 1931.
- 680 [14] V. KUHLEMANN AND P. S. VASSILEVSKI, *Improving the communication pattern in matrix-vector*  
681 *operations for large scale-free graphs by disaggregation*, SIAM Journal on Scientific Com-  
682 puting, 35 (2013), pp. S465–S486.
- 683 [15] D. M. PELT AND R. H. BISSELING, *A medium-grain method for fast 2d bipartitioning of sparse*  
684 *matrices*, in Parallel and Distributed Processing Symposium, 2014 IEEE 28th International,  
685 IEEE, 2014, pp. 529–539.
- 686 [16] B. UÇAR AND C. AYKANAT, *Revisiting hypergraph models for sparse matrix partitioning*, SIAM  
687 review, 49 (2007), pp. 595–603.
- 688 [17] B. VASTENHOUW AND R. H. BISSELING, *A two-dimensional data distribution method for parallel*  
689 *sparse matrix-vector multiplication*, SIAM Review, 47 (2005), pp. 67–95.
- 690 [18] R. S. XIN, J. E. GONZALEZ, M. J. FRANKLIN, AND I. STOICA, *Graphx: A resilient distributed*  
691 *graph system on spark*, in First International Workshop on Graph Data Management Ex-  
692 periences and Systems, GRADES '13, New York, NY, USA, 2013, ACM, pp. 2:1–2:6.
- 693 [19] A. YOO, A. H. BAKER, R. PEARCE, AND V. E. HENSON, *A scalable eigensolver for large*  
694 *scale-free graphs using 2D graph partitioning*, in Proc. International Conference for High  
695 Performance Computing, Networking, Storage and Analysis, ACM, 2011, pp. 63:1–63:11.
- 696 [20] A. YOO, E. CHOW, K. HENDERSON, W. MCLENDON, B. HENDRICKSON, AND U. CATALYUREK,  
697 *A scalable distributed parallel breadth-first search algorithm on bluegene/l*, in Proceedings  
698 of the 2005 ACM/IEEE Conference on Supercomputing, SC '05, Washington, DC, USA,  
699 2005, IEEE Computer Society, p. 25.