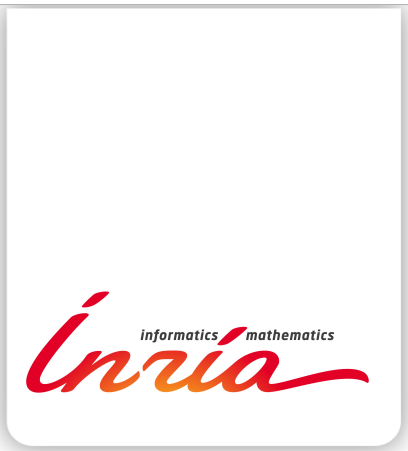
On analysis of partitioning models and metrics in parallel sparse matrix-vector multiplication

Umit V. Çatalyürek, Kamer Kaya, Bora Uçar

RESEARCH REPORT
N° 8301
February 2013
Project-Team ROMA

ISSN 0249-6399 ISRN INRIA/RR--8301--FR+ENG



On analysis of partitioning models and metrics in parallel sparse matrix-vector multiplication

Umit V. Çatalyürek* †, Kamer Kaya*, Bora Uçar‡

Project-Team ROMA

Research Report n° 8301 — February 2013 — 25 pages

Abstract: Graph/hypergraph partitioning models and methods have been successfully used to minimize the communication requirements among processors in several parallel computing applications. Parallel sparse matrix-vector multiplication (SpMxV) is one of the representative applications that renders these models and methods indispensable in many scientific computing contexts. We investigate the interplay of several partitioning metrics and execution times of SpMxV implementations in three libraries: Trilinos, PETSc, and an in-house one. We design and carry out experiments with up to 512 processors and investigate the results with regression analysis. Our experiments show that the partitioning metrics, although not an exact measure of communication cost, influence the performance greatly in a distributed memory setting. The regression analyses demonstrate which metric is the most influential for the execution time of the three libraries used.

Key-words: Parallel sparse-matrix vector multiplication, hypergraph partitioning

* Dept. of Biomedical Informatics, The Ohio State University (kamer, umit@bmi.osu.edu)
† Dept. of Electrical & Computer Engineering, The Ohio State University
‡ CNRS and LIP, ENS Lyon, France (bora.ucar@ens-lyon.fr)

RESEARCH CENTRE GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Sur l'analyse des modèles de partitionnement et des métriques lors de la multiplication d'une matrice creuse avec un vecteur dense en parallèle

Résumé : Les modèles et méthodes de partitionnement de graphes/hypergraphes ont été utilisés avec succès pour minimiser les besoins de communication entre processeurs dans de nombreuses applications de calcul parallèle. La multiplication d'une matrice creuse avec un vecteur dense (SpMxV) est une des applications représentatives qui rendent ces modèles et méthodes indispensables dans de nombreux contextes de calcul scientifique. Nous nous intéressons aux interactions entre plusieurs métriques de partitionnement et le temps d'exécution des implémentations de SpMxV dans les bibliothèques Trilinos, PETSc, et une bibliothèque à nous. Nous effectuons des expériences avec jusqu'à 512 processeurs, et nous étudions les résultats à l'aide d'analyse par régression. Nos expériences montrent que les métriques de partitionnement, bien qu'elles ne fournissent pas une mesure exacte du coût de communication, influent de façon significative sur la performance dans un système à mémoire distribuée. Les analyses par régression montrent quelle métrique a le plus d'influence sur le temps d'exécution pour les trois bibliothèques utilisées.

Mots-clés : Multiplication d'une matrice creuse avec un vecteur en parallèle, partitionnement d'hypergraphes

1 Introduction

Repeated sparse matrix-vector (SpMxV) and sparse matrix-transpose-vector multiplies that involve the same large, sparse matrix are the kernel operations in various iterative algorithms involving sparse linear systems. Such iterative algorithms include solvers for linear systems, eigenvalues, and linear programs. Efficient parallelization of SpMxV operations is therefore very important in virtually all large scale scientific computing applications. Quite a number of partitioning methods and models have been used to enable efficient parallelization of SpMxV. Some of the earlier methods try to balance the computational load of processors without paying attention to the effect of sparsity in the communication requirements. Some more recent methods use graph and hypergraph models to minimize some aspect of the communication cost while achieving load balance.

Hypergraph model-based partitioning methods have been proved to be very useful in implementing different partitioning schemes. These partitioning schemes address different communication cost metrics for the suitable variants of parallel SpMxV operations. In general, the importance of the communication cost metrics, such as the total volume of communication, the total number of messages and these two quantities on per processor basis, depends on the machine architecture, problem size, and the underlying parallel algorithm. In this study, we investigate the effects of the partitioning methods in order to identify the most relevant metrics in various configurations.

Hypergraph partitioning models try to reduce the total communication volume. Optimizing this metric seems to be more tractable than optimizing the others. However, in general, optimizing only this metric cannot always reduce the parallel execution time. This has been recognized by earlier models that implicitly and/or explicitly address other communication metrics. Here we mention two and defer the discussion of related work to Section 5.

Uçar and Aykanat [26] reduce the total number of messages by allowing controlled increase in the total communication volume with respect to the standard hypergraph models [10], while also trying to achieve balance on other communication cost metrics on per processor basis. They report practical experiments with parallel SpMxV computations showing that the method addressing the multiple communication cost metrics is more helpful than the original method that addresses only the total communication volume. Bisseling and Meesen [5] also propose methods addressing different communication cost metrics and report improvements with respect to an original method. Both of these studies demonstrate that different communication cost metrics and their interplay can be important to achieve scalable parallel algorithms. It is therefore important to understand the effects of different metrics (optimized by different partitioning models) on the running time of applications under different configurations.

The contribution of this paper is two-fold. We designed and conducted several experiments in a system with 512 processors to show the effects of partitioning models and metrics on SpMxV performance. As far as we know, this is the first work which evaluates the existing partitioning models and metrics in a modern architecture with modern software. Second, we showed that it is difficult, if not impossible, to define the correct partitioning model and metric without analyzing the characteristics of the input matrices and the SpMxV library being used. We experimented with three existing libraries, PETSc [1, 2],

Trilinos [18], and an in-house library SpMV [27]. In order to overcome the mentioned difficulty, we carefully analyze the results using regression analysis techniques and relate the execution time of SpMxV implementations to the different partitioning metrics. We portray this analysis in detail so as to suggest improved objective functions for partitioning software. Our experiments show that partitioning with the correct metric and model is important for good performance especially when the number of processors is large. Although, we only had an access to a 512-processor machine, the experiments and their analysis show that to scale larger systems, one needs to be more careful while partitioning the matrix—in our experiments the fact that the communication metrics greatly related to the execution time is observable starting from 64 processors.

The rest of the paper is organized as follows. In Section 2, we describe SpMxV in detail with three parallel algorithms using different approaches. Section 3 defines the partitioning models and metrics investigated in this work. Experimental results are given in Section 4 and some related work is mentioned in Section 5. Section 6 concludes the paper.

2 Parallel sparse matrix vector multiplication

Consider the sparse matrix-vector multiply of the form $\mathbf{y} \leftarrow \mathbf{Ax}$, where the nonzeros of the $m \times n$ matrix $\mathbf{A}$ are partitioned arbitrarily among K processors such that each processor P_k owns a mutually disjoint subset of nonzeros, $\mathbf{A}^{(k)}$ where $\mathbf{A} = \sum_k \mathbf{A}^{(k)}$. The vectors $\mathbf{y}$ and $\mathbf{x}$ are also partitioned among processors, where the processor P_k holds $\mathbf{x}^{(k)}$, a dense vector of size n_k, and it is responsible for computing $\mathbf{y}^{(k)}$, a dense vector of size m_k. Let $\mu(\cdot)$ denote the nonzero-to-processor and vector-entry-to-processor assignments. We note that the vectors $\mathbf{x}^{(k)}$ for $k = 1, \ldots, K$ are disjoint and hence $\sum_k n_k = n$; similarly the vectors $\mathbf{y}^{(k)}$ for $k = 1, \ldots, K$ are disjoint and hence $\sum_k m_k = m$. In this setting, the sparse matrix $\mathbf{A}^{(k)}$ owned by processor P_k can be permuted and written as

$$\mathbf{A}^{(k)} = \begin{bmatrix} \mathbf{A}^{(k)}_{11} & \cdots & \mathbf{A}^{(k)}_{1\ell} & \cdots & \mathbf{A}^{(k)}_{1K} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathbf{A}^{(k)}_{\ell 1} & \cdots & \mathbf{A}^{(k)}_{\ell\ell} & \cdots & \mathbf{A}^{(k)}_{\ell K} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathbf{A}^{(k)}_{K1} & \cdots & \mathbf{A}^{(k)}_{K\ell} & \cdots & \mathbf{A}^{(k)}_{KK} \end{bmatrix}. \tag{1}$$

Here, the blocks in the row-block stripe $\mathbf{A}^{(k)}_{k*} = \{\mathbf{A}^{(k)}_{k1}, \ldots, \mathbf{A}^{(k)}_{kk}, \ldots, \mathbf{A}^{(k)}_{kK}\}$ have row dimension of m_k, and similarly the blocks in the column-block stripe $\mathbf{A}^{(k)}_{*k} = \{\mathbf{A}^{(k)}_{1k}, \ldots, \mathbf{A}^{(k)}_{kk}, \ldots, \mathbf{A}^{(k)}_{Kk}\}$ have column dimension of n_k. The $\mathbf{x}$-vector entries that are needed by processor P_k are represented as $\hat{\mathbf{x}}^{(k)} = [\hat{\mathbf{x}}^{(k)}_1, \ldots, \hat{\mathbf{x}}^{(k)}_k, \ldots, \hat{\mathbf{x}}^{(k)}_K]$, a sparse column vector, where $\hat{\mathbf{x}}^{(k)}_\ell$ contains only those entries of $\mathbf{x}^{(\ell)}$ of processor P_ℓ corresponding to the nonzero columns in $\mathbf{A}^{(k)}_{*\ell}$. Here, the vector $\hat{\mathbf{x}}^{(k)}_k$ is equivalent to $\mathbf{x}^{(k)}$, defined according to the given partition on the $\mathbf{x}$-vector (hence the vector $\hat{\mathbf{x}}^{(k)}$ is of size at least n_k). The $\mathbf{y}$-vector entries for which the processor P_k computes partial results are represented as a sparse vector $\hat{\mathbf{y}}^{(k)} = [\hat{\mathbf{y}}^{(1)}_k, \ldots, \hat{\mathbf{y}}^{(k)}_k, \ldots, \hat{\mathbf{y}}^{(K)}_k]$, where $\hat{\mathbf{y}}^{(\ell)}_k$ contains only the partial results for $\mathbf{y}^{(\ell)}$ corresponding to the nonzero rows in $\mathbf{A}^{(k)}_{\ell*}$. If a row

i in $\mathbf{A}^{(k)}$ contains nonzeros only in columns corresponding to $\mathbf{x}^{(k)}$, that is if $\mu(a_{ij}) = P_k$ for all nonzero $a_{ij} \in \mathbf{A}^{(k)}$, and $\mu(x_i) = P_k$, the row i is called a local row from the perspective of the processor P_k. Similarly, if a column j in $\mathbf{A}^{(k)}$ contains nonzeros only in rows corresponding to $\hat{\mathbf{y}}_k^{(k)}$, that is if $\mu(a_{ij}) = P_k$ for all nonzero $a_{ij} \in \mathbf{A}^{(k)}$, and $\mu(y_i) = P_k$, the column j is called a local column, from the perspective of the processor P_k.

The standard parallel SpMxV algorithm [15, 27, 31] based on the described nonzero and vector entry partitioning is called the *row-column-parallel* algorithm. In this algorithm, each processor P_k executes the following steps.

1. For each $\ell \neq k$, form and send sparse vector $\hat{\mathbf{x}}_k^{(\ell)}$ to processor P_ℓ, where $\hat{\mathbf{x}}_k^{(\ell)}$ contains only those entries of $\mathbf{x}^{(k)}$ corresponding to the nonzero columns in $\mathbf{A}_{*k}^{(\ell)}$.
2. In order to form $\hat{\mathbf{x}}^{(k)} = [\hat{\mathbf{x}}_1^{(k)}, \ldots, \hat{\mathbf{x}}_k^{(k)}, \ldots, \hat{\mathbf{x}}_K^{(k)}]$, first define $\hat{\mathbf{x}}_k^{(k)} = \mathbf{x}^{(k)}$. Then, for each $\ell \neq k$ where $\mathbf{A}_{*\ell}^{(k)}$ contains nonzeros, receive $\hat{\mathbf{x}}_\ell^{(k)}$ from processor $P_\ell = \mu(\hat{\mathbf{x}}_\ell^{(k)})$, corresponding to the nonzero columns in $\mathbf{A}_{*\ell}^{(k)}$.
3. Compute $\hat{\mathbf{y}}^{(k)} \leftarrow \mathbf{A}^{(k)}\hat{\mathbf{x}}^{(k)}$.
4. For each $\ell \neq k$, send the sparse partial-results vector $\hat{\mathbf{y}}_k^{(\ell)}$ to processor $P_\ell = \mu(\hat{\mathbf{y}}_k^{(\ell)})$, where $\hat{\mathbf{y}}_k^{(\ell)}$ contains only those partial results for $\mathbf{y}^{(\ell)}$ corresponding to the nonzero rows in $\mathbf{A}_{\ell*}^{(k)}$.
5. Receive the partial-results vector $\hat{\mathbf{y}}_\ell^{(k)}$ from each processor P_ℓ which has computed a partial result for $\mathbf{y}^{(k)}$, i.e., from each processor P_ℓ where $\mathbf{A}_{k*}^{(\ell)}$ has nonzeros.
6. Compute $\mathbf{y}^{(k)} \leftarrow \sum_\ell \hat{\mathbf{y}}_\ell^{(k)}$, adding all the partial-results $\hat{\mathbf{y}}_\ell^{(k)}$ received in the previous step to its own partial results for $\mathbf{y}^{(k)}$.

There are two communication phases in this algorithm. The first one is just before the local matrix-vector multiply, and it is due to the communication of the **x**-vector entries (steps 1 and 2). This communication operations is referred to as *expand*. If the matrix **A** is distributed columnwise, and the **x**-vector entries are partitioned conformably with the column partition of **A**, then the expand operation is not needed. The resulting algorithm is called the *column-parallel* algorithm. The second communication phase is just after the local matrix-vector multiply (steps 4 and 5), and it is due to the communication of the partial results on the **y**-vector entries. This communication operation on the **y**-vector entries is referred to as *fold*. If the matrix **A** is distributed rowwise, and the **y**-vector entries are partitioned conformably with the rows of **A**, then the fold operation is not needed. The resulting algorithm is called the *row-parallel* algorithm.

There are different implementations of the above algorithm. We summarize three implementations with which we have experimented. Two of the implementations are in the well-known general libraries Trilinos [18] and PETSc [1, 2]; the third one, SpMV, is an in-house library [27].

Trilinos provides an implementation which can be described as in Algorithm 1 from the point of view of the processor P_k. In this implementation, the expand operations are finished before doing any computation. Then, all the scalar multiply-add operations are performed. Later on, the fold operations are completed. Trilinos uses `Irecv`/`Isend` and `waitall` communication primitives to handle the communications at steps 1 and 2 of Algorithm 1. It issues `Irecv`s,

Algorithm 1: ParSpMxV-Trilinos variant

Input: $\mathbf{A}, x, \mu$
Output: y

1 SEND and RECEIVE x vector entries so that each processor has the required x-vector entries
2 Compute $y_i^{(k)} \leftarrow a_{ij}\, x_j$ for the local nonzeros, i.e., the nonzeros for which $\mu(a_{ij}) = P_k$
3 SEND and RECEIVE local nonzero partial results $y_i^{(k)}$ to the processor $\mu(y_i) \neq P_k$, for all nonzero $y_i^{(k)}$
4 Compute $y_i \leftarrow \sum y_i^{\ell}$ for each y_i with $\mu(y_i) = P_k$

performs `Isend`s and then before commencing the computations ensures that all in the incoming data is received by using the `waitall` operation.

Algorithm 2: ParSpMxV-Overlap-PETSc variant

Input: $\mathbf{A}, x, \mu$
Output: y

1 SEND local x_j (i.e., $\mu(x_j) = P_k$) to those processors that have at least one nonzero in column j
2 Compute $y_i^k \leftarrow a_{ij}\, x_j$ for the local nonzeros and local x_j, i.e., the nonzeros for which $\mu(a_{ij}) = P_k$ and $\mu(x_j) = P_k$
3 RECEIVE **all** non-local x_j (i.e., $\mu(x_j) \neq P_k$)
4 Compute $y_i^k \leftarrow y_i^k + a_{ij}\, x_j$ for the local nonzeros and non-local x_j, i.e., the nonzeros for which $\mu(a_{ij}) = P_k$ and $\mu(x_j) \neq P_k$

PETSc provides an implementation of the above algorithm only for the row-parallel case. The Algorithm 2 summarizes that implementation from the point of view of P_k. First, the expand operation is initiated using `Irecv` and `Isend` primitives. Then, instead of waiting the reception of all necessary **x**-vector entries, it performs some local computations so as to overlap communication and computations. In particular, the processor P_k performs scalar multiply-add operations using local a_{ij}'s for which $\mu(x_j) = P_k$ and there is no $a_{i\ell}$ with $\mu(x_\ell) \neq P_k$. Then, upon verifying the reception of all needed **x**-vector entries using `waitall`, P_k continues with scalar multiply-add operations with the nonzeros on the rows that has at least one nonzero in a column j for which $\mu(x_\ell) \neq P_k$. The implementation can also be seen in an earlier technical report [24]. Figure 1 describes the algorithm pictorially. After issuing `Isend`s and `Irecv`s (for $\hat{x}_\ell^{(k)}$), processor P_k performs the computations associated with the horizontally shaded matrix zone. Then, `waitall` is executed to have all $x^{(k)}$ before continuing with the rows that are below the horizontal ones. Note that the local matrices are actually permuted into the displayed form (local rows and the interface rows). The advantage of this implementation with respect to the Algorithm 1 is that it allows overlap between the reception of messages for the expand operation and scalar multiply-add operations with the nonzeros in local rows.

Consider again the matrix $\mathbf{A}^{(k)}$ of processor P_k as shown in Fig. 1. Before executing the `waitall` operation, there are some more scalar multiply-add operations that P_k can perform before the reception of any $\hat{x}_\ell^{(k)}$. These operations are related to the nonzeros that are in the hatched zone in the figure. In order to

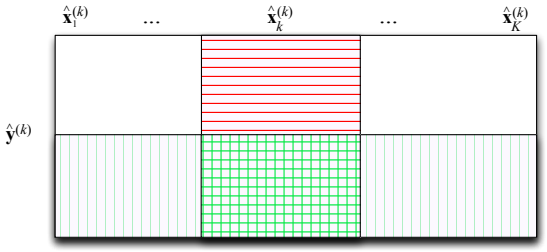


Figure 1: The zones of the matrix $\mathbf{A}^{(k)}$ of processor P_k with respect to the vector $\mathbf{x}$ assuming a row-parallel algorithm. PETSc exploits communication-computation overlap using only the horizontal zone. It is possible to use the hatched zone as well. Furthermore, upon the reception of a message we have association computations to do with the vertical zone.

exploit the hatched zone for communication computation overlap, one can store that zone in the compressed column storage (CCS) format. This way, one can delay the invocation of the `waitall` operation for some more time. In fact, we can get rid of the `waitall` operation and maximize the communication computation overlap by performing all scalar multiply-operations that involve a received $\mathbf{x}$-vector entry before waiting the reception of any other message. This requires storing the vertically shaded zones of the matrix in Fig. 1 in CCS format: with this, when P_k receives $\hat{x}_\ell^{(k)}$, it can visit the respective column and perform all operations. This way of storing the vertically shaded and hatched zones in CCS maximizes the amount of overlap in the strict sense (optimal amount of overlap) when a processor receives a single message from each sender (as should be the case in a proper SpMxV code). The third library that we investigate in this work SpMV [27] implements this approach for row-parallel and row-column parallel algorithms. The row-column-parallel algorithm's implementation of SpMV is shown in Algorithm 3 for processor P_k.

Algorithm 3: ParSpMxV-Overlap

Input: $\mathbf{A}, x, \mu$
Output: y

1 Send local x_j (i.e., $\mu(x_j) = P_k$) to those processors that have at least one nonzero in column j
2 Compute $y_i^k \leftarrow a_{ij}\, x_j$ for the local nonzeros and local x_j, i.e., the nonzeros for which $\mu(a_{ij}) = P_k$ and $\mu(x_j) = P_k$
3 **while** *not all non-local x_j received* **do**
4 Receive non-local x_j from **any** processor P_ℓ (i.e., $\mu(x_j) = P_\ell$)
5 Compute $y_i^k \leftarrow y_i^k + a_{ij}\, x_j$ for the local nonzeros and process P_ℓ's x_j, i.e., the nonzeros for which $\mu(a_{ij}) = P_k$ and $\mu(x_j) = P_\ell$
6 Send local nonzero partial results y_i^k to the processor $\mu(y_i) \neq P_k$, for all nonzero y_i^k
7 **while** *not all partial results y_i^ℓ received* **do**
8 Receive nonzero partial results set $\mathbf{y}^\ell$ from **any** processor P_ℓ
9 Compute $y_i \leftarrow y_i^k + y_i^\ell$ for each $y_i \in \mathbf{y}^\ell$

3 Summary of the existing partitioning models and metrics

3.1 Hypergraph-based models for SpMxV

3.1.1 Hypergraph partitioning

A hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ is defined as a set of vertices $\mathcal{V}$ and a set of nets (hyperedges) $\mathcal{N}$ among those vertices. A net $n_j \in \mathcal{N}$ is a subset of vertices, and the vertices in n_j are called its *pins*. A graph is a special hypergraph whose nets have size two. We use $\mathsf{pins}[n_j]$ and $\mathsf{nets}[v_i]$ to represent the pins of a net n_j, and the set of nets that contain vertex v_i, respectively. Vertices can be associated with weights, denoted with $\mathsf{w}[\cdot]$, and nets can be associated with costs, denoted with $\mathsf{c}[\cdot]$.

A *K-way partition* of a hypergraph $\mathcal{H}$ is denoted as $\Pi = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$ where each part is nonempty subset of $\mathcal{V}$, they are pairwise disjoint, and their union is equal to $\mathcal{V}$. In a partition Π, a net that has at least one pin (vertex) in a part is said to *connect* that part. The number of parts connected by a net n_j, i.e., *connectivity*, is denoted as λ_j. A net n_j is said to be *uncut* (*internal*) if it connects exactly one part (i.e., $\lambda_j = 1$), and *cut* (*external*), otherwise (i.e., $\lambda_j > 1$).

Let W_k denote the total vertex weight in $\mathcal{V}_k$ (i.e., $W_k = \sum_{v \in \mathcal{V}_k} \mathsf{w}[v]$) and W_{avg} denote the weight of each part when the total vertex weight is equally distributed (i.e., $W_{avg} = (\sum_{v_i \in \mathcal{V}} \mathsf{w}[v_i])/K$). If each part $\mathcal{V}_k \in \Pi$ satisfies the *balance criterion*

$$W_k \leq W_{avg}(1 + \varepsilon), \quad \text{for } k = 1, 2, \ldots, K \tag{2}$$

we say that Π is *balanced* where ε represents the maximum allowed imbalance ratio. The set of external nets of a partition Π is denoted as $\mathcal{N}_E$. Let $\chi(\Pi)$ denote the cost, i.e., *cutsize*, of a partition Π. There are various cutsize definitions [20]. The following two are the most relevant ones for this work:

$$\chi(\Pi) = \sum_{n_j \in \mathcal{N}_E} \mathsf{c}[n_j] \tag{3}$$

$$\chi(\Pi) = \sum_{n_j \in \mathcal{N}} \mathsf{c}[n_j](\lambda_j - 1) \; . \tag{4}$$

In (3) and (4), the contributions of each cut net n_j to the cutsize are $\mathsf{c}[n_j]$ and $\mathsf{c}[n_j](\lambda_j - 1)$, respectively. The cutsize metric given in (3) will be referred to here as *cut-net* metric and the one in (4) will be referred as *connectivity* metric. Given ε and an integer $K > 1$, the hypergraph partitioning problem can be defined as the task of finding a balanced partition Π with K parts such that $\chi(\Pi)$ is minimized. The hypergraph partitioning problem is NP-hard [20].

3.1.2 Hypergraph models and methods for sparse matrix partitioning

There are three basic hypergraph models for sparse matrix partitioning: the *column-net hypergraph model*, the *row-net hypergraph model*, and the *fine-grain model*. Each of these models also directly corresponds to a partitioning method.

We will briefly describe these models below. It is also possible to develop other partitioning methods, like *jagged-like*, *checker-board*, *Mondriaan* (a variant of *orthogonal recursive bisection*) using these three basic models [6, 15, 30, 31] with the standard hypergraph partitioning tools, such as PaToH [11] or Zoltan [7]. In this work, among other partitioning methods we will only use the checker-board method, because it is usually the best method to reduce the maximum number of messages sent and received by a processor.

In the *column-net hypergraph model* [10] used for 1D rowwise partitioning, an $m \times n$ matrix $\mathbf{A}$ with τ nonzeros is represented as a unit-cost hypergraph $\mathcal{H}_{\mathcal{R}} = (\mathcal{V}_{\mathcal{R}}, \mathcal{N}_{\mathcal{C}})$ with $|\mathcal{V}_R| = m$ vertices, $|\mathcal{N}_C| = n$ nets, and τ pins. In $\mathcal{H}_{\mathcal{R}}$, there exists one vertex $v_i \in \mathcal{V}_{\mathcal{R}}$ for each row i of matrix $\mathbf{A}$. Weight $w[v_i]$ of a vertex v_i is equal to the number of nonzeros in row i. The name of the model comes from the fact that the columns are represented as nets. That is, there exists one unit-cost net $n_j \in \mathcal{N}_{\mathcal{C}}$ for each column j of matrix $\mathbf{A}$. The net n_j connects the vertices corresponding to the rows that have a nonzero in column j. That is, $v_i \in n_j$ if and only if $a_{ij} \neq 0$.

In the *row-net hypergraph model* [10] used for 1D columnwise partitioning, an $m \times n$ matrix $\mathbf{A}$ with τ nonzeros is represented as a unit-cost hypergraph $\mathcal{H}_{\mathcal{C}} = (\mathcal{V}_{\mathcal{C}}, \mathcal{N}_{\mathcal{R}})$ with $|\mathcal{V}_C| = n$ vertices, $|\mathcal{N}_R| = m$ nets, and τ pins. In $\mathcal{H}_{\mathcal{C}}$, there exists one vertex $v_j \in \mathcal{V}_{\mathcal{C}}$ for each column j of matrix $\mathbf{A}$. Weight $w[v_j]$ of a vertex $v_j \in \mathcal{V}_{\mathcal{R}}$ is equal to the number of nonzeros in column j. The name of the model comes from the fact that the rows are represented as nets. That is, there exists one unit-cost net $n_i \in \mathcal{N}_{\mathcal{R}}$ for each row i of matrix $\mathbf{A}$. The net $n_i \subseteq \mathcal{V}_{\mathcal{C}}$ connects the vertices corresponding to the columns that have a nonzero in row i. That is, $v_j \in n_i$ if and only if $a_{ij} \neq 0$.

In the *fine-grain model* [12], otherwise known as the *column-row-net hypergraph model*, used for 2D nonzero-based fine-grain partitioning, an $m \times n$ matrix $\mathbf{A}$ with τ nonzeros is represented as a unit-weight and unit-cost hypergraph $\mathcal{H}_{\mathcal{Z}} = (\mathcal{V}_{\mathcal{Z}}, \mathcal{N}_{\mathcal{RC}})$ with $|\mathcal{V}_{\mathcal{Z}}| = \tau$ vertices, $|\mathcal{N}_{\mathcal{RC}}| = m + n$ nets and 2τ pins. In $\mathcal{V}_{\mathcal{Z}}$, there exists one unit-weight vertex v_{ij} for each nonzero a_{ij} of matrix $\mathbf{A}$. The name of the model comes from the fact that both rows and columns are represented as nets. That is, in $\mathcal{N}_{\mathcal{RC}}$, there exists one unit-cost row-net r_i for each row i of matrix $\mathbf{A}$ and one unit-cost column-net c_j for each column j of matrix A. The row-net r_i connects the vertices corresponding to the nonzeros in row i of matrix $\mathbf{A}$, and the column-net c_j connects the vertices corresponding to the nonzeros in column j of matrix $\mathbf{A}$. That is, $v_{ij} \in r_i$ and $v_{ij} \in c_j$ if and only if $a_{ij} \neq 0$.

Three standard partitioning methods are obtained by partitioning the vertices of the hypergraphs generated by one of these models using the connectivity metric (4): *Column-Net* (CN), *Row-Net* (RN) and *Fine-Grain* (FG). Each one of those methods minimizes the total communication volume, while keeping the balanced work.

The *checker-board partitioning method* (CB) [13] is a two-step method, where each step models either the expand phase or the fold phase of the parallel SpMxV. There are two alternative schemes for this partitioning method: the one which models the expands in the first step and the folds in the second step, and vice-versa. We describe the first one below (the other can be described similarly). Given an $m \times n$ matrix $\mathbf{A}$ and the number K of processors organized as a $M \times N$ mesh, first, $\mathbf{A}$ is partitioned row-wise into M parts using the column-net model, producing $\Pi_{\mathcal{R}} = \{\mathcal{R}_1, \ldots, \mathcal{R}_M\}$. In the second step, the matrix $\mathbf{A}$ is

partitioned columnwise into N parts by using the multi-constraint partitioning to obtain $\Pi_{\mathcal{C}} = \{\mathcal{C}_1, \ldots, \mathcal{C}_N\}$. The rowwise and columnwise partitions $\Pi_{\mathcal{R}}$ and $\Pi_{\mathcal{C}}$ together define a 2D partition of the matrix $\mathbf{A}$, where $\mu(a_{ij}) = P_{k,\ell} \Leftrightarrow r_i \in \mathcal{R}_k$ and $c_j \in \mathcal{C}_\ell$. The multi-constraint formulation is used to achieve load balance among processors. In this formulation, each vertex v_i of $\mathcal{H}_{\mathcal{C}}$ is assigned M weights $\mathtt{w}[i,k]$, for $k = 1, \ldots, M$. Here, $\mathtt{w}[i,k]$ is equal to the number of nonzeros of column c_i in rows $\mathcal{R}_k$. Consider an N-way partitioning of $\mathcal{H}_{\mathcal{C}}$ with M constraints using the vertex weight definition above. Maintaining the M balance constraints (each individual constraint treated as separate balance constraint) corresponds to maintaining computational load balance on the processors of each row of the processor mesh. Establishing the equivalence between the total communication volume and the sum of the cutsizes of the two partitions is fairly straightforward. The volume of communication for the fold operations corresponds exactly to the $\chi(\Pi_{\mathcal{C}})$. The volume of communication for the expand operations corresponds exactly to the $\chi(\Pi_{\mathcal{R}})$.

3.2 Partitioning metrics

We investigate the practical effects of partitioning models on several metrics. These metrics include the maximum number of nonzeros assigned to a processor (MaxNnz), the total communication volume (TotVol), the maximum send volume of a processor (MaxSV), the total number of messages (TotMsg), and the maximum number of messages sent by a processor (MaxMS). We will also analyze how these metrics are related with the SpMxV performance.

As can be expected, most of our results are based on experiments and their careful analysis (see the next section). Yet, there are some theoretical results worth mentioning. In 1D partitioning, the maximum number of messages can be as high as $K-1$, and the total number of messages can be K times larger. In the fine-grain model the maximum number of messages can be as high as $2 \times (K-1)$, and the total number of messages as can be, again, K times larger. When a checker-board partitioning is used for distributing a matrix into $K = MN$ processors, each processor only communicates with up to $M-1$ and $N-1$ processors in the expand and fold phases. Since the checker-board model is more restricted, the total communication volume is expected to be larger.

4 Experimental results

We carried our experiments on a 64-node cluster where each node has a 2.27GHz dual quad-core Intel Xeon (Bloomfield) CPU and 48GB main memory. Each core in a socket has 64KB L1 and 256KB L2 caches, and each socket has an 8MB L3 cache shared by 4 cores. The interconnection network is 20Gbps DDR InfiniBand. For parallelism, mvapich2 version 1.6 is used. We built SpMV, PETSc, and Trilinos with gcc 4.4.4 and used optimization flag -O3. For PETSc experiments, we used the matrix type `MPIAIJ` and the multiplication routine `MatMult`. Since each node has 8 cores, we have 512 processors in total. In the experiments, we use $K \in \{1, 8, 16, 32, 64, 128, 256, 512\}$. For an experiment with $K \neq 1$ processors, we fully utilize $K/8$ nodes of the cluster. To measure the time of one SpMxV operation (in secs), we do 500 multiplications for each execution. The values in the tables and figures are the averages of these 500 runs.

Matrix	Description	n	τ
atmosmodl	Atmospheric model.	1,489,752	10,319,760
TSOPF_RS	Optimal pow. flow	38,120	16,171,169
Freescale1	Semiconductor sim.	3,428,755	17,052,626
rajat31	Circuit sim.	4,690,002	20,316,253
RM07R	Comp. fluid dyn.	381,689	37,464,962
cage15	DNA electrophoresis	5,154,859	99,199,551
HV15R	3D engine fan	2,017,169	283,073,458
mesh-1024	5-point stencil	1,048,576	5,238,784
mesh-2048	5-point stencil	4,194,304	20,963,328
mesh-4096	5-point stencil	16,777,216	83,869,696

Table 1: Properties of the experiment matrices.

We used seven large real-life square matrices from different application domains that are available at the University of Florida (UFL) Sparse Matrix Collection (`http://www.cise.ufl.edu/research/sparse/matrices`) and three synthetically generated matrices corresponding to 5-point stencil meshes in 2D with sizes 1024×1024, 2048×2048, and 4096×4096. The properties of the matrices are given in Table 1.

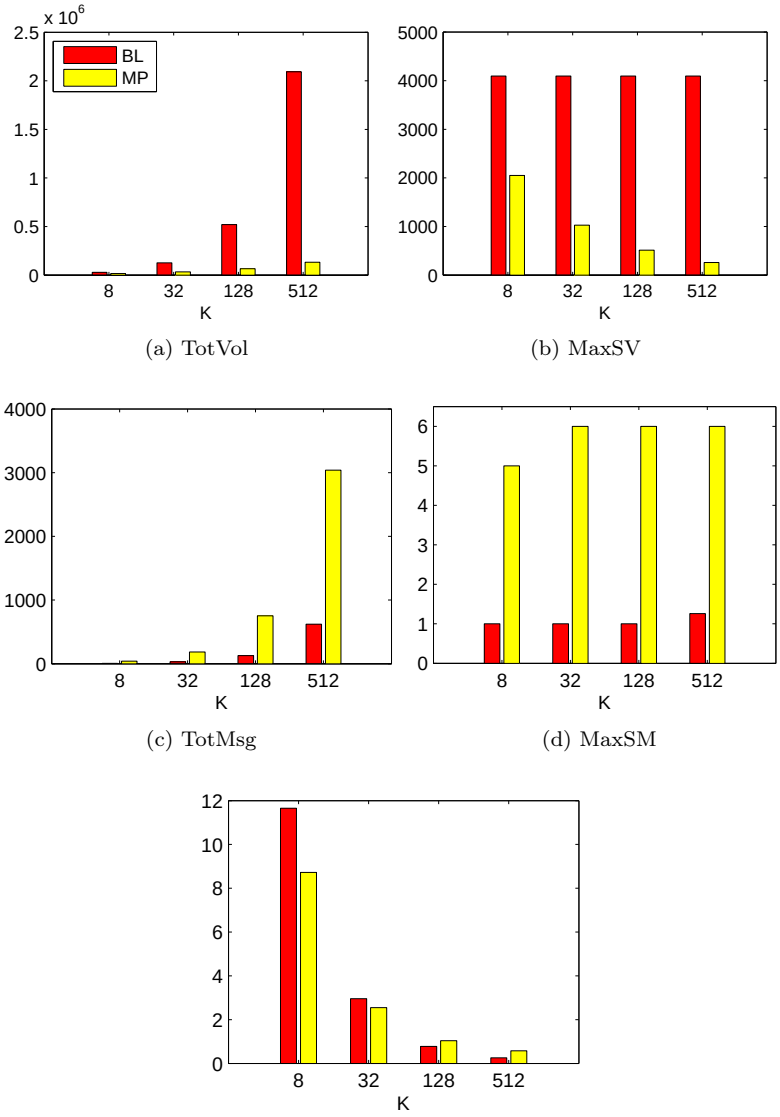
To generate the partitions with respect to column-net (CN), row-net (RN), fine grain (FG), and checkerboard (CB) models, we used PaToH [11] with the connectivity metric (4), i.e., aiming to minimize TotVol, and 0.03 as the maximum imbalance. We also used a block partitioning (BL) model to generate rowwise partitions of the matrices. In this model, we traverse the rows from 1 to m, generate a part with approximately τ/K nonzeros, and continue with the next part when this number is exceeded. Experimental results show that the average imbalance ratio of the partitions generated with BL is around 10^{-4} which is two order magnitude smaller than what we allowed in other partitioning methods. The model does not take any other partitioning metric into account. For matrices based on 2D meshes, we used another rowwise partitioning model called MP. This model tiles the 2D plane with a diamond-like shape [4, Section 4.8] and associates each shape with a processor. This approach balances the volume and the number of messages the processors send and receive. In the experiments on real-life matrices, we use PETSc with rowwise models CN and BL, and SpMV and Trilinos with all models except MP. For meshes, we added MP to each library's model set.

Let us start with two small experiments on mesh-based matrices to demonstrate that the partitioning metrics are relevant. In the first experiment, we use MP which allows us to obtain the same MaxNnz, MaxSV, and MaxSM values for different number of processors (K) and mesh sizes. We exploit this to observe the effect of TotVol and TotMsg on the execution time as Table 2 shows. In this table, from one row to the other, only TotVol and TotMsg change. Yet, the execution time of each library increases significantly when TotVol increases from 8,200 to 131,296, and TotMsg increases from 40 to 752. Hence, a good partitioning with minimized metrics would reduce the execution time.

In the second experiment, we compared the metrics and execution times of Trilinos on mesh-based matrices when models BL and MP are used. Since MP takes TotVol and MaxNnz into account, and BL only deals with the latter, there is a drastic difference between mean TotVol and MaxSV values of these two models in favor of MP as Figure 2 shows. However, due to the banded

mesh-	K	TotVol	TotMsg	SpMV	PETSc	Trilinos
1,024	8	8,200	40	2.24	2.03	2.20
2,048	32	32,816	184	2.26	2.07	2.49
4,096	128	131,296	752	2.71	2.45	2.75

Table 2: Effect of partitioning metrics TotVol and TotMsg on execution times (in sec) for mesh-based matrices using the MP partitioning model. From one row to the other, among the partitioning metrics only TotVol and TotMsg change.



(a) TotVol

(b) MaxSV

(c) TotMsg

(d) MaxSM

(e) Exec. Time

Figure 2: Partitioning metrics and corresponding mean execution time for an SpMxV operation in Trilinos on mesh-based matrices when models BL and MP are used.

	$8 \leq K \leq 64$			$128 \leq K \leq 512$		
Metric	SpMV	PETSc	Trilinos	SpMV	PETSc	Trilinos
MaxNnz	8.02	7.81	6.80	0.49	0.44	0.83
TotVol	0.18	0.38	1.00	0.39	0.36	1.06
MaxSV	1.66	1.53	2.20	0.00	0.00	0.11
TotMsg	0.15	0.28	0.00	7.90	8.03	4.51
MaxSM	0.00	0.00	0.00	1.22	1.18	3.49

Table 3: Regression analysis of SpMV, PETSc and Trilinos with all matrices and models CN and BL.

structure of mesh-based matrices, BL obtains much less TotMsg and MaxSM values. The mean execution times of Trilinos employing these models are given in Fig. 2e. As the experiment shows, the total latency has a larger performance impact on Trilinos on these matrices in our system for larger processor counts, whereas the total volume of messages seems to be important for smaller number of processors. Clearly, one has to determine which metric is the most important for a given number of processors for a given algorithm. Below, we carry out a careful regression analysis of the metrics for different algorithms.

4.1 Regression analysis

To evaluate the performance of the libraries with respect to the partitioning metrics, we use linear regression analysis techniques and solve the nonnegative least squares problem (NNLS). In NNLS, given a variable matrix $\mathbf{V}$ and a vector $\mathbf{t}$, we want to find a dependency vector $\mathbf{d}$ which minimizes $\|\mathbf{Vd}-\mathbf{t}\|$ s.t. $\mathbf{d} \geq 0$. In our case, $\mathbf{V}$ has five columns which correspond to the partitioning metrics MaxNnz, TotVol, MaxSV, TotMsg, and MaxSM. Each row of $\mathbf{V}$ corresponds to an SpMxV execution where the execution time is put to the corresponding entry of $\mathbf{t}$. Hence, we have the same $\mathbf{V}$ but a different $\mathbf{t}$ for each library. We apply a well-known technique in regression analysis and standardize each entry of $\mathbf{V}$ by subtracting its column's mean and dividing it to its column's standard deviation so that the mean and the standard deviation of each column become 0 and 1, respectively. This way, the units are removed and each column becomes equally important throughout the analysis. We then used MATLAB's `lsqnonneg` to solve NNLS. Each entry of the output d_i shows the dependency of the execution time to the partitioning metric corresponding to the ith column of $\mathbf{V}$. Tables 3, 4, and 5 show the dependency values found in various settings.

We first apply regression analysis to each library with all matrices and row-wise partitioning models CN (column-net) and BL (block). The analysis shows that when $K \leq 64$, SpMxV performance depends rigorously on the maximum number of nonzeros assigned to a processor. In this case, the dependency values for MaxNnz are 8.02, 7.81, and 6.80 for SpMV, PETSc, and Trilinos, respectively. As Table 3 shows, the next important metric is MaxSV with values 1.66, 1.53, and 2.20. The message-based partitioning metrics do not effect the performance for $K \leq 64$. However, when K gets larger, these metrics are of utmost importance. Furthermore, the importance of MaxNnz decreases drastically for all the libraries. For SpMV and PETSc, MaxNnz becomes the 3rd important variable, whereas for Trilinos, it is the 4th. This shows that SpMV and PETSc handle the increase in the communication metrics better than Trilinos.

When $K \geq 128$, the dependency of Trilinos to TotMsg is much less than that of SpMV and PETSc. On the contrary, Trilinos' MaxSM dependency is almost 1.75 times more than SpMV and PETSc. This is expected since Trilinos uses Algorithm 1 which has no communication-computation overlap due to the use of `waitall` primitive. Such primitives can cause close coupling among the processors. When MaxNnz and the variance on the number of messages per processor are large, the overhead due to the bottleneck processor can result in poor SpMxV performance. Note that the dependency profiles of SpMV and PETSc, which are similar due to the communication-computation overlap in Algorithms 2 and 3, do not point out a similar bottleneck.

	$8 \leq K \leq 32$		$64 \leq K \leq 128$		$256 \leq K \leq 512$	
Metric	SpMV	Trilinos	SpMV	Trilinos	SpMV	Trilinos
MaxNnz	8.43	7.54	2.75	2.52	0.00	0.02
TotVol	0.23	0.89	0.52	1.94	0.38	0.98
MaxSV	1.35	1.57	1.57	1.69	0.04	0.50
TotMsg	0.00	0.00	4.66	2.38	6.24	3.06
MaxSM	0.00	0.00	0.49	1.47	3.34	5.44

Table 4: Regression analysis of SpMV and Trilinos with all matrices and partitioning models. PETSc is not shown in this table because it cannot handle all partitioning schemes.

We extend the regression analysis to all matrices and all partitioning models and show the results in Table 4. The performance of SpMV and Trilinos rigorously depend on MaxNnz if $K \leq 32$, and on TotMsg and MaxSM when $K \geq 256$. Once again, Trilinos' MaxSM dependency is higher than that of SpMV due to the `waitall` primitive. To see the effect of matrix structure on regression analysis, we use only mesh-based matrices in the next experiment. As Table 5 shows, we observe that for these matrices, the performance of SpMV and Trilinos mostly depend on MaxNnz even when $K \geq 64$. Note that these matrices are banded and the communication metrics have relatively lower values compared to those of real-life matrices. Hence, the most dominant factor is MaxNnz.

	$8 \leq K \leq 32$		$64 \leq K \leq 128$		$256 \leq K \leq 512$	
Metric	SpMV	Trilinos	SpMV	Trilinos	SpMV	Trilinos
MaxNnz	8.97	9.38	8.83	9.05	5.10	5.47
TotVol	0.00	0.00	0.00	0.24	0.00	0.00
MaxSV	0.72	0.48	0.43	0.09	0.92	0.52
TotMsg	0.00	0.00	0.42	0.07	0.42	0.99
MaxSM	0.31	0.14	0.33	0.55	3.55	3.02

Table 5: Regression analysis of SpMV and Trilinos with mesh-based matrices and all partitioning models.

In the light of the regression analysis experiments, we can argue that partitioning metrics in fact effect the performance of parallel SpMxV libraries. However, the best metric (or function of metrics) that needs to be minimized depends on the number of processors, the size and structure of the matrix, which we are planning to investigate as a future work, and even the library itself. Although some of these variables are known while generating the partitions, predicting the others may need a preprocessing phase. For example, we already know that the

libraries in this paper are employing point-to-point communication primitives which makes the connectivity metric suitable. However, if collective communication primitives, e.g., `MPI_ALLGATHER`, had been used, it would be better to minimize the cut-net metric as the main partitioning objective (however, we should note that such collective operations introduce unnecessary synchronization and messages especially for large K values). On the other hand, the matrix structure can be different for each input and a partitioner needs either a manual direction or a preprocessing to predict the best metric for each matrix.

4.2 Effects of vector partitioning

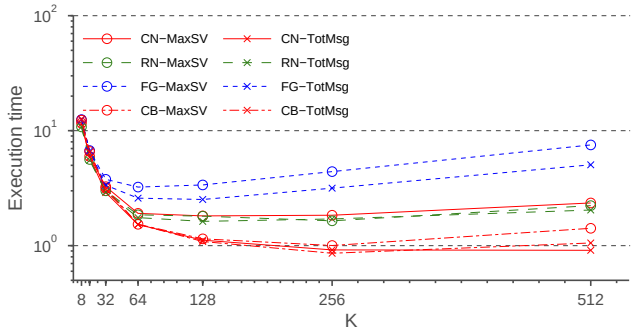
For parallel $\mathbf{y} \leftarrow \mathbf{Ax}$ with nonzero-to-processor and vector-entry-to-processor assignment μ, we assume the following: $\mu(x_j)$ (or $\mu(y_i)$) is equal to P_ℓ, then there exists a nonzero a_{ij} assigned to processor P_ℓ. Although vector-entry-to-processor assignments affect most of the communication metrics such as MaxSV and TotMsg, under this assumption, TotVol is not one of them. To minimize either of the partitioning metrics MaxSV and TotMsg while keeping TotVol constant, we use a simple vector-partitioning heuristic. The heuristic traverses each vector entry and sets $\mu(x_j)$ (or $\mu(y_i)$) to P_ℓ if the assignment causes the minimum increase in the concerned metric and P_ℓ is assigned at least one nonzero a_{ij}.

Figure 3 shows the mean execution times of the libraries with the vector-partitioning heuristic minimizing either MaxSV or TotMsg. We observed that nearly for all libraries and partitioning models, minimizing TotMsg is more important than minimizing MaxSV for reducing the execution time, especially when K is large. The difference is more clear in Fig. 3b. Starting from $K = 64$, the difference becomes obvious in favor of TotMsg which is concordant with the regression analyses. For $K \in \{8, 16\}$, minimizing MaxSV or TotMsg seem to be equally important.

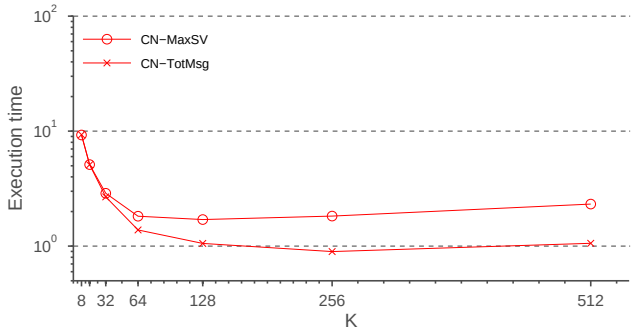
4.3 Effects of models on metrics

All the partitions used in this work are generated with the objective of minimizing TotVol and balancing the load for a good MaxNnz. To partition $\mathbf{x}$ and $\mathbf{y}$, we used the greedy heuristic described above. Since minimizing TotMsg during vector partitioning seems more promising, we will use that variant in the rest of the paper. Here, we analyze the effect of the partitioning models on the metrics. We do not include MaxNnz since it is in the $(1 + \epsilon)$ margin of W_{avg}.

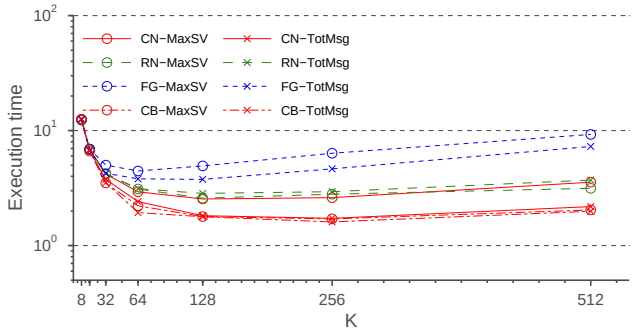
The communication metrics vary with respect to the partitioning model as Figure 4 shows for $K = \{32, 128, 512\}$. As expected, BL (block) partitioning produces high TotVol and MaxSV (maximum send volume). However, due to the irregular nonzero distributions in original matrices and with the help of the vector partitioning heuristic, its mean TotMsg becomes minimum and its MaxSM (maximum number of messages sent) is reduced to a number less than those of CN (column-net), RN (row-net), and FG (fine-grain). The only model which is better than BL is CB (checker-board) which also results in good volume-related metrics. As expected, its mean MaxSV (maximum send volume) and MaxSM values are much better than the others. The FG model has the worst number of messages and the maximum number of messages. Note that the relative performances of the models do not change with K. However, their



(a) SpMV

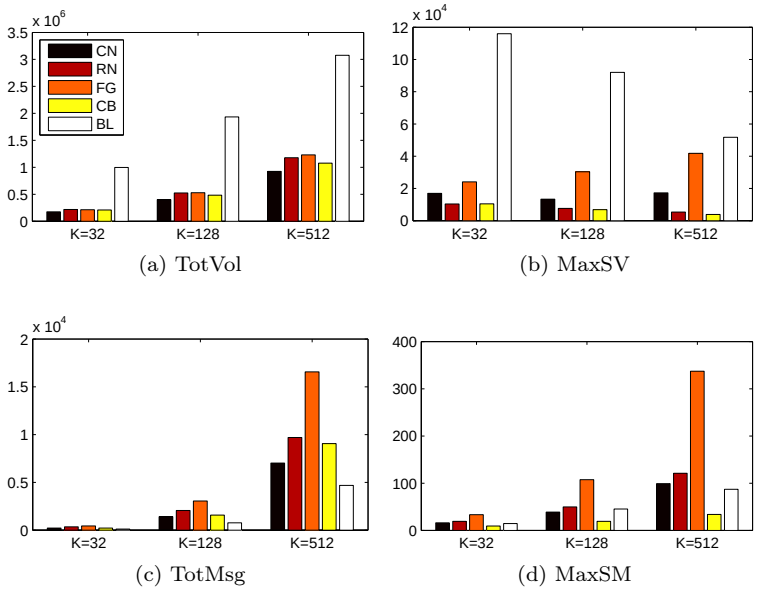


(b) PETSc



(c) Trilinos

Figure 3: Effect of the vector-partitioning heuristic minimizing MaxSV or TotMsg. The horizontal and vertical axes show the #processors and the geometric means of the execution times (secs) across all matrices in log scale, respectively.



(a) TotVol (b) MaxSV

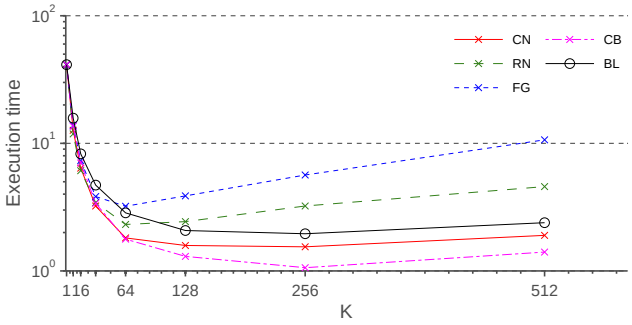
(c) TotMsg (d) MaxSM

Figure 4: Effects of partitioning models on metrics for matrices from real-life applications. The charts show the geometric mean of each metric w.r.t. the partitioning model.

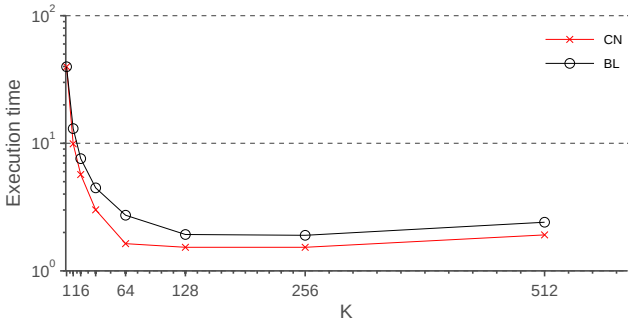
difference tend to increase and hence, the model used for partitioning becomes more important as the parallel matrix-vector multiplication times of the libraries show in Figure 5. As regression analysis experiments also prove, the models do not effect the execution time when K is low. However, when K increases, the effect becomes significant. In agreement with Fig. 5, for SpMV and Trilinos, the CB model is the best one. This is expected since CB obtains a good model for TotVol and TotMsg and is the best one for MaxSV and MaxSM. For PETSc, CN is better than BL since CN's volume-based performance is much better while CN and BL are comparable for the message-based metrics. When K is increased from 128 to 256, the only significant reduction on the execution time is obtained by SpMV with the CB model. This shows that minimizing the right metric with the right partitioning model may be crucial if one wants to use the available resources in their limits.

As Figure 6 shows, for mesh-based matrices, the effects of the partitioning models on TotVol and MaxSV are similar to the ones obtained for real-life matrices. The BL model produces large communication volumes and all the other models can obtain comparable TotVol values. But the relative MaxSV, TotMsg, and MaxSM performance of CB gets worse due to the band structure of the mesh-based matrices which is not suitable for a 2D distribution. For the same reason, the rowwise model BL becomes the best one w.r.t. the message-based metrics. With BL-based partitioning, each processor sends at most 1 message throughout an SpMxV operation. The runtimes in Fig. 7 are in agreement with these observations.

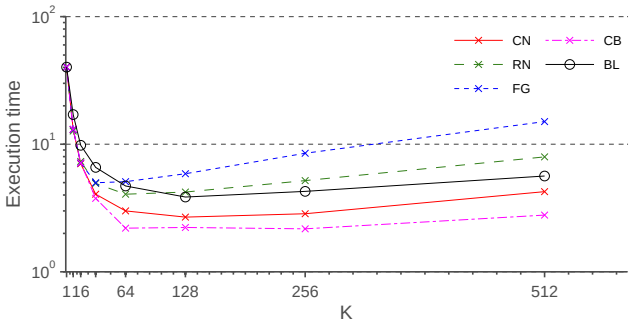
Unlike the experiment with the real-life matrices, the mean execution times



(a) SpMV

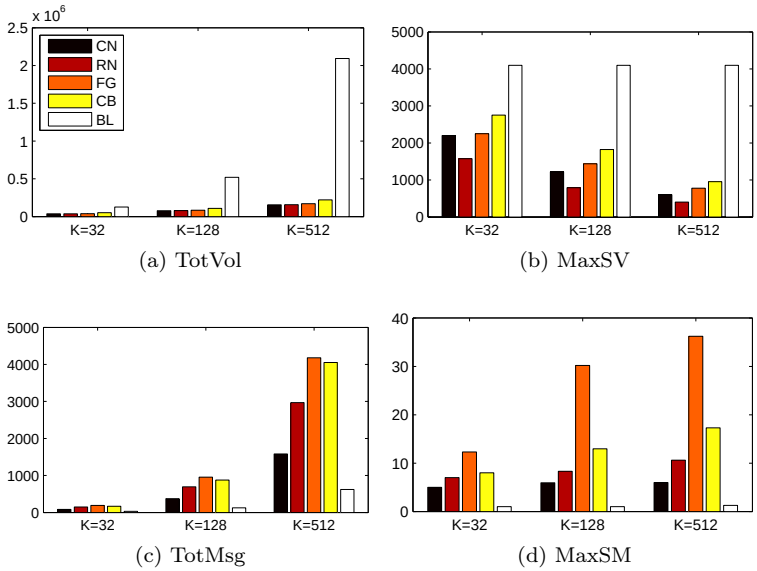


(b) PETSc



(c) Trilinos

Figure 5: Mean SpMxV times on real-life matrices in log scale for each library with respect to partitioning model. The vector-partitioning heuristic in this experiment tries to minimize TotMsg.



(a) TotVol (b) MaxSV

(c) TotMsg (d) MaxSM

Figure 6: Effects of partitioning models on metrics for mesh-based matrices. The charts show the geometric mean of each metric w.r.t. partitioning model.

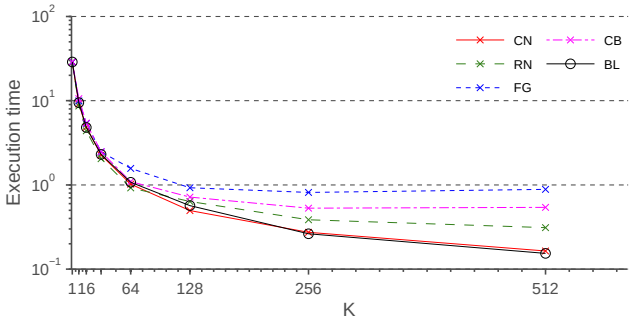
significantly decrease till K goes to 512. Note that the average number of nonzeros in most of the real-life matrices and mesh-based matrices are comparable to each other. However, the band structure of the mesh-based matrices yields lower metrics as shown in Figure 6. Hence, the communication overhead for these matrices is smaller compared with that of real-life matrices. We observed that this is the main reason of better scalability.

5 Related work

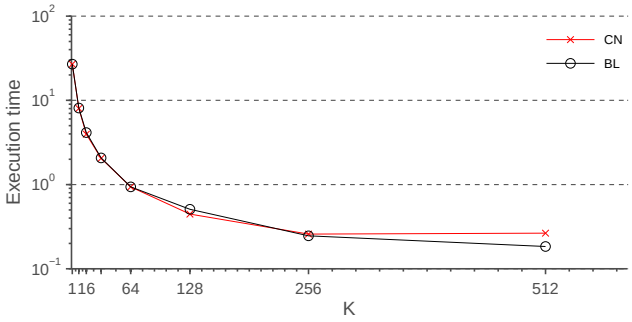
Partitioning literature is a rich one. Here we provide a small survey of related work that we believe characterizes major partitioning approaches. Earlier approaches to sparse matrix partitioning concern only the computational load of processors, without paying attention to the effect of sparsity in communication. These approaches are discussed in some recent as well as older studies [3, 17, 21–23, 25]. Even though those methods would yield fairly good load balance, they rely on inherent ordering of matrix rows and columns for reducing the communication cost—which can be arbitrary. To alleviate this problem, first graph-based models have been proposed, later hypergraph-based models have been developed to overcome limitations of graph models [16].

The first use of the hypergraph partitioning methods for efficient parallel sparse matrix-vector multiply operations dates from mid 90s [9]. A more comprehensive study [10] describes the use of the row-net and column-net hypergraph models in 1D sparse matrix partitioning. The vector partitioning problem that complements the hypergraph partitioning is a recent one [5, 26, 28].

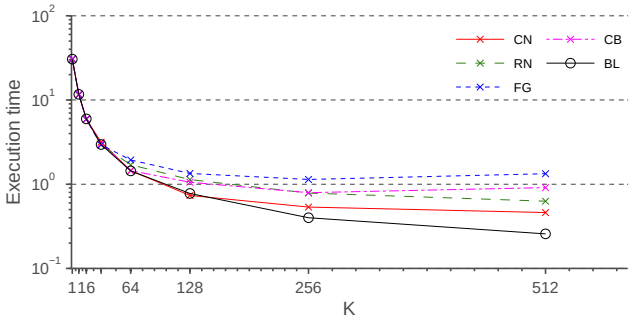
A fair treatment of parallel sparse matrix-vector multiplication, analysis and investigations on certain matrix types along with the use of hypergraph parti-



(a) SpMV



(b) PETSc



(c) Trilinos

Figure 7: Mean SpMxV times on mesh-based matrices in log scale for each library with respect to partitioning model. The vector-partitioning heuristic in this experiment tries to minimize TotMsg.

tioning is given in [4, Chapter 4]. Further analysis of hypergraph partitioning on some model problems is given in [29].

Some different methods for sparse matrix partitioning using hypergraphs can be found in [30], including jagged-like and checkerboard partitioning methods, and in [31], the orthogonal recursive bisection approach. A recipe to choose a partitioning method for a given matrix is given in [15].

All of the studies mentioned above uses static partitioning. A recent work [19] presents a dynamic partitioning but requires the matrix to be duplicated on all processors; which is infeasible in many applications and hence will not scale.

Most of these studies are more theoretical and comparisons presented on these studies are in terms of one or two partitioning metrics. Our work differs from those by mainly being a through experimental analysis paper. To the best of our knowledge, our paper is the first one that actually compares all such partitioning methods and metrics, using state-of-the-art software packages.

We did not take the performance difference between intra-node and inter-node communication operations. In recent work [14], Çatalyürek et al. discuss the relevance of hypergraph partitioning methods in shared memory parallelization of a SpMxV-like algorithm. In particular, they show that connectivity-1 metric (which corresponds to the total communication volume in a distributed memory SpMxV) corresponds to the total size of the private arrays used in a possible shared memory implementation. In other words, although there are differences between the metrics optimized by partitioning algorithms for different computing environments, the same partitioning make sense for both implementations, and hence for a hybrid implementation. We did not touch this issue yet but there are on going work by, among others, Byun et al. [8] and Yzelman and Roose [32].

6 Conclusion

We have carried out a detailed study to understand the importance of partitioning models and their effects in parallel SpMxV operations. As mentioned in the experiments, minimizing the right metric with the right partitioning model may be crucial to increase throughput. For example, for the real-life matrices in our test set, CB model is the only one which can obtain a significant reduction on the SpMxV time when K is increased from 128 to 256 (after that we did not see any speed up). It is obvious that the other models fail to obtain such a reduction since the gain by dividing MaxNnz by two does not compensate the communication overhead induced by multiplying K by two. Hence, assuming the communication overhead is doubled on the average, doubling K increases the relative importance of communication on SpMxV four times.

Matrices from today's scientific and industrial applications can be huge. If one has only a few processors, partitioning may not matter, since the contribution of communication to the execution time will be low and the overall improvement on SpMxV via a good partitioning will be insignificant. However, as the regression analyses of Section 4.1 show, after a number of processors, the communication overhead will start to dominate the SpMxV time. For our experiments, this number is somewhere between 32 and 64, and it depends on the characteristics of the matrix, the library and the architecture used for SpMxV operations. Although it may be more than 64, considering the advancements on

CPU hardware, we can easily argue that this number will remain highly practical and partitioning will matter more for systems that are larger than those considered here.

Bibliography

[1] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press, 1997.

[2] S. Balay, J. Brown, , K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.2, Argonne National Laboratory, 2011.

[3] R. H. Bisseling. Parallel iterative solution of sparse linear systems on a transputer network. In A. E. Fincham and B. Ford, editors, *Parallel Computation*, volume 46, pages 253–271. Oxford University Press, Oxford, UK, 1993.

[4] R. H. Bisseling. *Parallel Scientific Computation: A Structured Approach using BSP and MPI*. Oxford University Press, Oxford, UK, Mar. 2004.

[5] R. H. Bisseling and W. Meesen. Communication balancing in parallel sparse matrix-vector multiplication. *Electron. Trans. Numer. Anal.*, 21:47–65, 2005.

[6] R. H. Bisseling, B. O. F. Auer, A. Yzelman, T. van Leeuwen, and Ü. V. Çatalyürek. Two-dimensional approaches to sparse matrix partitioning. In U. Naumann and O. Schenk, editors, *Combinatorial Scientific Computing*, chapter 12. CRC Press, 2012.

[7] E. Boman, K. Devine, R. Heaphy, B. Hendrickson, V. Leung, L. A. Riesen, C. Vaughan, U. Catalyurek, D. Bozdag, W. Mitchell, and J. Teresco. *Zoltan 3.0: Parallel Partitioning, Load Balancing, and Data-Management Services; User's Guide*. Sandia National Laboratories, Albuquerque, NM, 2007. Tech. Report SAND2007-4748W.

[8] J.-H. Byun, R. Lin, J. W. Demmel, and K. A. Yelick. *pOSKI: Parallel Optimized Sparse Kernel Interface Library User's Guide for Version 1.0.0*. Berkeley Benchmarking and Optimization (BeBOP) Group University of California, Berkeley, 2012.

[9] Ü. V. Çatalyürek and C. Aykanat. A hypergraph model for mapping repeated sparse matrix-vector product computations onto multicomputers. In *HiPC'95*, 1995.

[10] Ü. V. Çatalyürek and C. Aykanat. Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. *IEEE T. Parall. Distr.*, 10(7):673–693, Jul 1999.

[11] Ü. V. Çatalyürek and C. Aykanat. *PaToH: A Multilevel Hypergraph Partitioning Tool, Version 3.0*. Bilkent University, Department of Computer Engineering, Ankara, 06533 Turkey. PaToH is available at `http://bmi.osu.edu/~umit/software.htm`, 1999.

[12] Ü. V. Çatalyürek and C. Aykanat. A fine-grain hypergraph model for 2D decomposition of sparse matrices. In *IPDPS'01*, San Francisco, CA, 2001.

[13] Ü. V. Çatalyürek and C. Aykanat. A hypergraph-partitioning approach for coarse-grain decomposition. In *Supercomputing'01*, 2001.

[14] Ü. V. Çatalyürek, K. Kaya, and B. Uçar. On shared-memory parallelization of a sparse matrix scaling algoritm. In *ICPP'12*, Pittsburgh, PA, 2012.

[15] Ü. V. Çatalyürek, C. Aykanat, and B. Uçar. On two-dimensional sparse matrix partitioning: Models, methods, and a recipe. *SIAM J. Sci. Comput.*, 32(2):656–683, 2010.

[16] B. Hendrickson and T. G. Kolda. Graph partitioning models for parallel computing. *Parallel Comput.*, 26:1519–1534, 2000.

[17] B. Hendrickson, R. W. Leland, and S. Plimpton. An efficient parallel algorithm for matrix-vector multiplication. *Int. J. High Speed Com.*, 7(1): 73–88, 1995.

[18] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley. An overview of the trilinos project. *ACM Trans. Math. Softw.*, 31(3):397–423, 2005.

[19] S. Lee and R. Eigenmann. Adaptive runtime tuning of parallel sparse matrix-vector multiplication on distributed memory systems. In *Supercomputing'08*, pages 195–204, New York, NY, USA, 2008. ACM.

[20] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout.* Wiley–Teubner, Chichester, U.K., 1990.

[21] J. G. Lewis and R. A. van de Geijn. Distributed memory matrix-vector multiplication and conjugate gradient algorithms. In *Supercomputing'93*, pages 484–492, New York, NY, USA, 1993. ACM.

[22] A. T. Ogielski and W. Aiello. Sparse matrix computations on parallel processor arrays. *SIAM J. Sci. Comput.*, 14(3):519–530, 1993.

[23] A. Pınar and C. Aykanat. Fast optimal load balancing algorithms for 1D partitioning. *J. Parallel Distr. Com.*, 64:974–996, 2004.

[24] Y. Saad and A. V. Malevsky. P-SPARSLIB: A portable library of distributed memory sparse iterative solvers. Technical Report umsi-95-180, Minnesota Supercomputer Institute, Minneapolis, MN, Sep. 1995.

[25] E. Saule, E. Ö. Baş, and Ü. V. Çatalyürek. Load-balancing spatially located computations using rectangular partitions. *J. Parallel Distr. Com.*, 72(10): 1201–1214, 2012.

[26] B. Uçar and C. Aykanat. Encapsulating multiple communication-cost metrics in partitioning sparse rectangular matrices for parallel matrix-vector multiplies. *SIAM J. Sci. Comput.*, 25(6):1837–1859, 2004.

[27] B. Uçar and C. Aykanat. A library for parallel sparse matrix-vector multiplies. Technical Report BU-CE-0506, Department of Computer Engineering, Bilkent University, Ankara, Turkey, 2005.

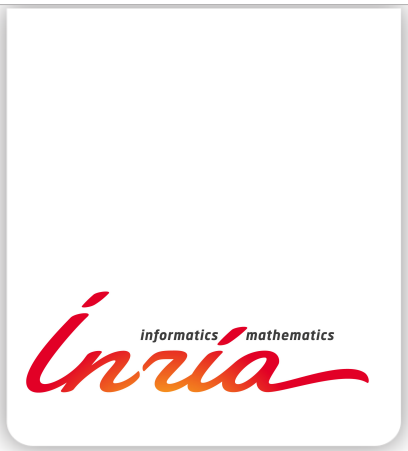
[28] B. Uçar and C. Aykanat. Revisiting hypergraph models for sparse matrix partitioning. *SIAM Rev.*, 49(4):595–603, 2007.

[29] B. Uçar and Ü. V. Çatalyürek. On the scalability of hypergraph models for sparse matrix partitioning. In M. Danelutto, J. Bourgeois, and T. Gross, editors, *Euromicro'10*, pages 593–600. IEEE Computer Society, Conference Publishing Services, 2010.

[30] B. Uçar, Ü. V. Çatalyürek, and C. Aykanat. A matrix partitioning interface to PaToH in MATLAB. *Parallel Comput.*, 36(5-6):254–272, 2010.

[31] B. Vastenhouw and R. H. Bisseling. A two-dimensional data distribution method for parallel sparse matrix-vector multiplication. *SIAM Rev.*, 47(1): 67–95, 2005.

[32] A. N. Yzelman and D. Roose. High-level strategies for parallel shared-memory sparse matrix–vector multiplication. Technical Report TW614, KU Leuven, June 2012.



RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399