Analysis of Partitioning Models and Metrics in Parallel Sparse Matrix-Vector Multiplication

Kamer Kaya[1], Bora Uçar[2(✉)], and Ümit V. Çatalyürek[1,3]

[1] Department of Biomedical Informatics,
The Ohio State University, Columbus, USA
{kamer,umit}@bmi.osu.edu
[2] CNRS and LIP, ENS Lyon, Lyon, France
bora.ucar@ens-lyon.fr
[3] Department of Electrical and Computer Engineering,
The Ohio State University, Columbus, USA

Abstract. Graph/hypergraph partitioning models and methods have been successfully used to minimize the communication 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 the partitioning metrics and execution times of SpMxV implementations in three libraries: Trilinos, PETSc, and an in-house one. We carry out experiments with up to 512 processors and investigate the results with regression analysis. Our experiments show that the partitioning metrics 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 libraries.

Keywords: Parallel sparse-matrix vector multiplication · Hypergraph partitioning

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. A number of partitioning methods and models based on hypergraphs have been used to enable efficient parallelization of SpMxV. These partitioning methods address different communication cost metrics for some 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

R. Wyrzykowski et al. (Eds.): PPAM 2013, Part II, LNCS 8385, pp. 174–184, 2014.
DOI: 10.1007/978-3-642-55195-6_16,

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 and quantify their effects in various configurations. Our aims are to help the partitioner developers identify the important metrics, and to help the users of those partitioners to identify the most suitable partitioning method for their use case.

The standard hypergraph based models minimize the total volume of communication explicitly [7]. Some more recent variants do that while imposing a limit on the total number of communication by a 2D partitioning approach [6,9]. More sophisticated approaches [3,12,13] minimize different communication cost metrics on top of the total volume. Experimental investigations in 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 compares the existing partitioning models and metrics in modern architectures with modern software following message-passing paradigm. Our experiments confirm 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 [10], and an in-house library SpMV [14]. 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 different partitioning metrics. We portray this analysis, which forms out the second contribution, in detail so as to suggest improved objective functions for partitioning software and a guideline to choose partitioning methods for practitioners. 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.

2 Parallel SpMxV Operation and Software

Consider the sparse matrix-vector multiply operation of the form $\mathbf{y} \leftarrow \mathbf{Ax}$, where the nonzeros of the $m \times n$ matrix $\mathbf{A}$ are partitioned 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.

The standard parallel SpMxV algorithm [9,14,15] based on the described onzero and vector entry partitioning is called the *row-column-parallel* algorithm. In this algorithm, each processor P_k executes the following steps:

1. **Expand:** Send entries of $\mathbf{x}^{(k)}$ that are needed by others. Receive entries of $\mathbf{x}$ that are needed but owned by others.
2. **Scalar multiply-adds:** Perform $\bar{\mathbf{y}} \leftarrow \mathbf{A}^{(k)}\bar{\mathbf{x}}$, where $\bar{\mathbf{x}}$ contains $\mathbf{x}^{(k)}$ and the received entries of $\mathbf{x}$.
3. **Fold:** Send partial results from $\bar{\mathbf{y}}$ to the responsible processors. Receive contributions to the $\mathbf{y}^{(k)}$ vector.

If $\mathbf{A}$ is distributed columnwise, and the $\mathbf{x}$-vector entries are partitioned conformably with the column partition of $\mathbf{A}$, then the expand operation is not needed. Similarly, if $\mathbf{A}$ is distributed rowwise, and the $\mathbf{y}$-vector entries are partitioned conformably with the rows of $\mathbf{A}$, then the fold operation is not needed.

2.1 Libraries

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 [10] and PETSc [1,2]; the third one, SpMV, is an in-house library [14].

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 $\mu(y_i) = P_k$

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

PETSc provides an implementation of the above algorithm only for the row-parallel case. 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 $\mathbf{x}$-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

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 $\mu(x_j) \neq P_k$

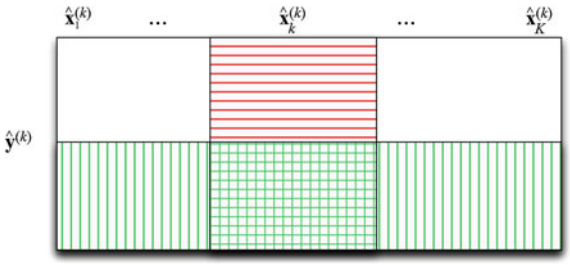

Fig. 1. The zones of the matrix $\mathbf{A}^{(k)}$ of processor P_k with respect to the vector $\mathbf{x}$ assuming a row-parallel algorithm.

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 $\mathbf{x}$-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 [11]. 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 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 **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 [14], implements this approach for row-parallel and row-column parallel algorithms (see descriptions in the accompanying technical report [8]).

2.2 Investigated Partitioning Metrics and Methods

We study the practical effects of five different metrics: the maximum number of nonzeros assigned to a processor (MaxNnz) which defines the load balance; 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). Our investigations are necessarily experimental, yet some a priori results can be told about these metrics, see the accompanying technical report [8] and the references therein.

We investigated the rowwise and columnwise partitioning methods CN and RN (the naming convention corresponds to the hypergraph models in the original paper [7]) among the one-dimensional partitioning approaches. Among the two-dimensional ones we investigated the fine grain (FG), and checkerboard (CB) partitioning models (see [9] and the references therein). These four methods try to reduce the total volume of communication and obtain load balance; the 2D methods implicitly reduce the total and maximum number of messages per processor. We also used a block partitioning (BL) model whose aim is just to have load balance in rowwise partitioning 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. 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, Sect. 4.8] and associates each shape (corresponding to a set of rows) with a processor. This approach balances the volume and the number of messages the processors send and receive.

3 Experimental Investigations

We carried our experiments on a 64-node cluster where each node has a 2.27 GHz dual quad-core Intel Xeon (Bloomfield) CPU and 48 GB main memory. Each core in a socket has 64 KB L1 and 256 KB L2 caches, and each socket has an 8 MB L3 cache shared by 4 cores. The interconnection network is 20 Gbps 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`.

Table 1. Properties of the experiment matrices.

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

We used PaToH [5] with default setting `quality` for partitioning the matrices (this allows 0.03 imbalance). 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 tables and figures show the averages of these 500 runs.

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.

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. The reason is technological: PETSc provides SpMxV routines only for rowwise partitioning.

We designed a set of experiments to show the effect of different partitioning metrics in the actual running time of SpMxV computations. Our first two sets of experiments (Fig. 2 and Table 2 of the accompanying technical report [8]) showed clearly that the total volume of communication, the total number of messages, the maximum number and volume of messages sent by a processor affect significantly the running time of SpMxV. Which one of these four communication metric is the most important in different libraries? To what extent? What about their combinations? In order to answer these questions we carried out the following regression analysis.

3.1 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

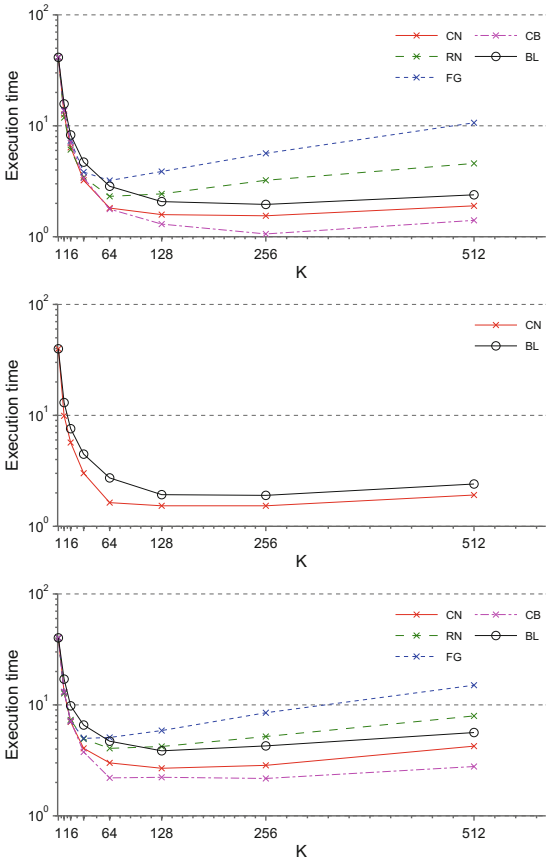

Fig. 2. Mean SpMxV times on real-life matrices in log scale for each library with respect to partitioning model.

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 2, 3, and 4 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

Table 2. Regression analysis of SpMV, PETSc and Trilinos with all matrices and models CN and BL.

	$8 \leq K \leq 64$			$128 \leq K \leq 512$		
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 and Trilinos with all matrices and partitioning models. PETSc is not shown in this table because it cannot handle all the schemes.

	$8 \leq K \leq 32$		$64 \leq K \leq 128$		$256 \leq K \leq 512$	
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 mesh-based matrices and all partitioning models.

	$8 \leq K \leq 32$		$64 \leq K \leq 128$		$256 \leq K \leq 512$	
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

for MaxNnz are 8.02, 7.81, and 6.80 for SpMV, PETSc, and Trilinos, respectively. As Table 2 shows, the next important metric is MaxSV with values 1.66, 1.53, and 2.20. The latency-based (TotMsg, MaxSM) 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, do not point out a similar bottleneck.

We extend the regression analysis to all matrices and all partitioning models and show the results in Table 3. 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 4 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.

In the light of the regression analysis experiments, we note that the partitioning metrics effect the performance of parallel SpMxV libraries. 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 in the future, 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.

3.2 Summary of Further Results

We provide summary of some further results and refer the interested reader to the accompanying technical report [8]. We observed that for all libraries and partitioning methods, minimizing TotMsg is more important than minimizing MaxSV for reducing the execution time, especially when K is large. 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 are equally important.

The checkerboard method (CB) which is demonstrated (see Fig. 4 in [8]) to reduce most of the communication cost metrics seems to be the method of choice when K is not small (when K is small, there is no much difference between the models as also revealed by the regression analysis before). The relative performances of the partitioning methods do not change with the increasing K.

However, their 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 Fig. 2. When K is 256, the only significant reduction on the execution time is obtained by SpMV with the CB model.

4 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 is 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 Sect. 3.1 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.

References

1. Balay, S., Brown, J., Buschelman, K., Eijkhout, V., Gropp, W.D., Kaushik, D., Knepley, M.G., McInnes, L.C., Smith, B.F., Zhang, H.: PETSc users manual. Technical report ANL-95/11 - Revision 3.2, Argonne National Laboratory (2011)
2. Balay, S., Gropp, W.D., McInnes, L.C., Smith, B.F.: Efficient management of parallelism in object oriented numerical software libraries. In: Arge, E., Bruaset, A.M., Langtangen, H.P. (eds.) Modern Software Tools in Scientific Computing, pp. 163–202. Birkhäuser Press, Basel (1997)
3. Bisseling, R.H., Meesen, W.: Communication balancing in parallel sparse matrix-vector multiplication. Electron. Trans. Numer. Anal. **21**, 47–65 (2005)
4. Bisseling, R.H.: Parallel Scientific Computation: A Structured Approach using BSP and MPI. Oxford University Press, Oxford (2004)
5. Çatalyürek, Ü.V., Aykanat, C.: PaToH: A multilevel hypergraph partitioning tool, Version 3.0. Bilkent University, Department of Computer Engineering, Ankara, 06533 Turkey. PaToH. http://bmi.osu.edu/umit/software.htm (1999)

6. Çatalyürek, Ü.V., Aykanat, C.: A hypergraph-partitioning approach for coarse-grain decomposition. In: Supercomputing'01 (2001)
7. Çatalyürek, Ü.V., Aykanat, C.: Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. IEEE Trans. Parall. Distr. **10**(7), 673–693 (1999)
8. Çatalyürek, Ü.V., Kaya, K., Uçar, B.: On analysis of partitioning models and metrics in parallel sparse matrix-vector multiplication. Technical report INRIA, France (2013)
9. Çatalyürek, Ü.V., Aykanat, C., Uçar, B.: On two-dimensional sparse matrix partitioning: models, methods, and a recipe. SIAM J. Sci. Comput. **32**(2), 656–683 (2010)
10. Heroux, M.A., Bartlett, R.A., Howle, V.E., Hoekstra, R.J., Hu, J.J., Kolda, T.G., Lehoucq, R.B., Long, K.R., Pawlowski, R.P., Phipps, E.T., Salinger, A.G., Thornquist, H.K., Tuminaro, R.S., Willenbring, J.M., Williams, A., Stanley, K.S.: An overview of the trilinos project. ACM Trans. Math. Softw. **31**(3), 397–423 (2005)
11. Saad, Y., Malevsky, A.V.: P-SPARSLIB: A portable library of distributed memory sparse iterative solvers. Technical report umsi-95-180, Minnesota Supercomputer Institute, Minneapolis, MN (1995)
12. Uçar, B., Aykanat, C.: Minimizing communication cost in fine-grain partitioning of sparse matrices. In: Yazıcı, A., Şener, C. (eds.) ISCIS 2003. LNCS, vol. 2869, pp. 926–933. Springer, Heidelberg (2003)
13. Uçar, B., Aykanat, C.: Encapsulating multiple communication-cost metrics in partitioning sparse rectangular matrices for parallel matrix-vector multiplies. SIAM J. Sci. Comput. **25**(6), 1837–1859 (2004)
14. Uçar, B., Aykanat, C.: A library for parallel sparse matrix-vector multiplies. Technical report BU-CE-0506, Department of Computer Engineering, Bilkent University, Ankara, Turkey (2005)
15. Vastenhouw, B., Bisseling, R.H.: A two-dimensional data distribution method for parallel sparse matrix-vector multiplication. SIAM Rev. **47**(1), 67–95 (2005)