Chapter 17
Parallelization of Sparse Matrix Kernels for Big Data Applications

Oguz Selvitopi, Kadir Akbudak and Cevdet Aykanat

17.1 Introduction

There is a growing interest in scientific computing community for big data analytics. Recent approaches aim to benefit the big data analytics with the methods and techniques that are very common in the mature field of optimization for high performance computing (HPC) [20]. These efforts rely on the observation that graphs often constitute the spine of the data structures used in analyzing big data (as data is almost always sparse), and the adjacency list representation of a graph actually corresponds to a sparse matrix. Hence, analysis operations on big data can be expressed in terms of basic sparse matrix kernels. For example, the popular graph mining library PEGASUS (A Peta-Scale Graph Mining System) [17] uses an optimized sparse matrix vector multiplication kernel, called GIM-V, as the basic operation in several graph mining algorithms such as PageRank, spectral clustering, finding connected components, etc. This work focuses on efficient parallelization of two other important sparse kernels on distributed systems: sparse matrix–matrix multiplication (SpGEMM) of the form $C = AB$ and sparse matrix–dense matrix multiplication (SpMM) of the form $Y = AX$.

SpGEMM kernel finds its application in a wide range of domains such as finding all-pair shortest paths (APSP) [11], finite element simulations based on domain decomposition (e.g., finite element tearing and interconnect (FETI) [13]), molecular dynamics (e.g., CP2K [10]), computational fluid dynamics [19], climate

O. Selvitopi · K. Akbudak · C. Aykanat (✉)
Department of Computer Engineering, Bilkent University, 06800, Cankaya, Ankara, Turkey
e-mail: aykanat@cs.bilkent.edu.tr
URL: http://www.cs.bilkent.edu.tr

O. Selvitopi
e-mail: reha@cs.bilkent.edu.tr

K. Akbudak
e-mail: kadir@cs.bilkent.edu.tr

F. Pop et al. (eds.), *Resource Management for Big Data Platforms*,
Computer Communications and Networks, DOI 10.1007/978-3-319-44881-7_17

simulation [25], and interior point methods [6]. These applications necessitate efficient large-scale parallelization in order to obtain shorter running times for processing today's rapidly growing "Big Data". There exist several software packages that provide SpGEMM computation for distributed-memory architectures such as Trilinos [15] and Combinatorial BLAS [7]. Trilinos uses one-dimensional (1D) partitioning of input matrices. Matrices A and C are stationary, whereas matrix B is communicated in K stages for a parallel system with K processors. This algorithm corresponds to replicating B matrix among K processors in K stages. Combinatorial BLAS uses the parallel matrix multiplication algorithm (SUMMA [30]) based on dense matrices. The motivation of Combinatorial BLAS is large-scale graph analytic for "Big Data". It also contains scalable implementations of kernel operations such as sparse matrix–vector multiplication (SpMV) and subgraph extraction. Recently, a matrix-partitioning method based on two-constraint hypergraph partitioning is proposed in [3] for reducing total message volume during outer-product–parallel SpGEMM. [3] is known to be the first work that proposes to preprocess the sparsity patterns of the matrices in order to reduce parallelization overheads. In [3], it is also proposed that the input and output matrices can be simultaneously partitioned.

SpMM is also an important kernel and many graph analysis techniques such as centrality measures use it as a building block. Apart from its popularity in block methods in linear algebra [14, 22, 23], it is also a very basic kernel in graph analysis as several works pointed out its relation to graph algorithms [2, 8, 17, 24, 28]. In these methods, the dimensions of the dense matrices X and Y are usually very small compared to the dimensions of the sparse matrix A. The importance of this kernel is also acknowledged by vendors Intel MKL [1] and Nvidia cuSPARSE [21], being realized respectively for multi-core/many-core and GPU architectures.

Our contributions in this work are centered around partitioning models to efficiently parallelize these two kernels on distributed systems. In order to do so we aim at reducing communication overheads. The proposed model for the SpGEMM kernel aims to reduce total message volume while the proposed model for the SpMM kernel consists of two phases and it strives for reducing both total and maximum message volume. The experiments on up to 1024 processes show that scalability can be drastically improved using the proposed models.

The rest of the paper is organized as follows: Section 17.2 describes the SpGEMM kernel and the proposed model for parallelization. Section 17.3 describes the SpMM kernel and the two-phase methodology to achieve parallelization. Section 17.4 presents our experiments separately for these two kernels and Sect. 17.5 concludes.

17.2 Parallelization of the SpGEMM Kernel

We investigate efficient parallelization of the SpGEMM operation on distributed-memory architectures. The communication overhead and imbalance on computational loads of processors become significant bottleneck in large-scale parallelization. Thus, we propose an intelligent matrix-partitioning method that achieve reducing total message volume while maintaining balance on computational loads of processors in outer-product-based parallelization of the SpGEMM operation.

In Sect. 17.2.1, the outer-product-based parallelization of the SpGEMM operation is presented. We present the hypergraph model for this outer-product-based parallelization in Sect. 17.2.2. Section 17.2.3 describes how to decode a hypergraph partition as a matrix partition.

17.2.1 Outer-Product–Parallel SpGEMM Algorithm

We consider the SpGEMM computation of the form $C = AB$. Here, A and B denote the input matrices, and C denotes the output matrix, where all of these matrices are sparse.

Outer-product–parallel SpGEMM operation uses 1D columnwise and 1D rowwise partitioning of the input A and B matrices, respectively, as follows:

$$\hat{A} = AP = [A_1 A_2 \cdots A_K] \text{ and } \hat{B} = PB = \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_K \end{bmatrix}. \tag{17.1}$$

In Eq. (17.1), K denotes the number of parts, which is in turn equal to the number of processors of the parallel system, P denotes the permutation matrix obtained from partitioning. The same permutation matrix is used for reordering columns of matrix A and rows of matrix B so that outer products are performed without any computation.

According to the input matrix partitioning given in Eq. (17.1), the SpGEMM computation is performed in two steps. The first step consists of local outer-product computations performed as follows by each processor P_k:

$$C^k = A_k B_k \text{ where } k = 1, 2, \ldots, K.$$

The second step consists of summing low-rank C^k matrices, which incur communication. The following operation is performed by all processors as follows:

$$C = C^1 + C^2 + \cdots + C^K \text{ where } k = 1, 2, \ldots, K.$$

17.2.2 Hypergraph Model

We propose a hypergraph partitioning (HP) based method to reduce the total message volume that occur in the second step of the outer-product–parallel SpGEMM algorithm (Sect. 17.2.1) while maintaining balance on computation loads of outer products performed by processors in the first step of the parallel algorithm. The objective in this partitioning is to cluster columns of matrix A and rows of matrix B that contribute to the same nonzeros of matrix C into the same parts as much as possible. In other words, the outer-product computations that contribute to the same C-matrix nonzeros are likely to be performed by the same processor without any communication.

We model an SpGEMM instance $C = AB$ as a hypergraph $\mathscr{H}(C, A, B) = (\mathscr{V}, \mathscr{N})$. $\mathscr{V}$ contains a vertex v_x for each outer product of xth column of A with xth row of B. v_x represents the task of computing this outer product without any communication in the first step of the parallel SpGEMM algorithm given in Sect. 17.2.1. $\mathscr{N}$ contains a net (hyperedge) $n_{i,j}$ for each nonzero $c_{i,j}$ of C. Net $n_{i,j}$ connects the vertices representing the outer products producing scalar partial results for $c_{i,j}$. That is,

$$Pins(n_{i,j}) = \{v_x : x \in cols(a_{i,*}) \wedge x \in rows(b_{*,j})\} \cup \{v_{i,j}\}.$$

Here, $cols(a_{i,*})$ denotes the column indices of nonzeros in row i $(a_{i,*})$, whereas $rows(b_{*,j})$ denotes the row indices of nonzeros in column j $(b_{*,j})$. Hence, $n_{i,j}$ represents the summation operation of scalar partial results to obtain final result of $c_{i,j}$ in the second step of the SpGEMM algorithm. Each vertex $v_x \in \mathscr{V}$ is associated with a weight $w(v_x)$ as follows:

$$w(v_x) = |Nets(v_x)|,$$

where $Nets(v_x)$ denotes the set of nets that connect vertex v_x. This vertex weight definition encodes the amount of computation performed for the outer products. Each net $n_{i,j} \in \mathscr{N}$ is associated with a unit weight, i.e.,

$$w(n_{i,j}) = 1.$$

This net weight definition encodes the multi-way relation between the outer products regarding a single nonzero $c_{i,j}$.

17.2.3 Decoding Hypergraph Partitioning as Matrix Partitioning

A vertex partition $\Pi(\mathscr{V}) = \{\mathscr{V}_1, \mathscr{V}_2, \ldots, \mathscr{V}_K\}$ can be used to obtain a conformal columnwise and rowwise partition of A and B. That is, $v_x \in \mathscr{V}_k$ is decoded as assigning the outer product of xth column of A with xth row of B to processor P_k. If all the pins of $n_{i,j}$ reside in the same part $\mathscr{V}_k$ (i.e., the net is uncut), the summation operation regarding $c_{i,j}$ is performed locally by P_k. Otherwise, it means that the net is cut and this summation operation is assigned to one of those processors that yield a scalar partial result for $c_{i,j}$. Hence, Π induces a 1D partition of A and B, and a 2D nonzero-based partition of C.

In the proposed HP-based method, the partitioning constraint used while obtaining Π corresponds to maintaining balance on computational loads of processors. Each cut net incurs communication of scalar partial results. The amount of communication due to a nonzero $c_{i,j}$ is equal to one less of the number of parts in which $n_{i,j}$ has pins. This in turn corresponds to one less of the number of processors that send scalar partial results to the processor responsible for $c_{i,j}$. Thus the partitioning objective of minimizing the cutsize according to "connectivity-1" metric [9] corresponds to minimizing the total message volume.

17.3 Parallelization of the SpMM Kernel

We consider the SpMM operation of the form $Y = AX$, where A is an $n \times n$ sparse matrix and X and Y are $n \times s$ dense matrices. Whatever the context, SpMM often reveals itself as an expensive operation and hence parallelization of this kernel must be handled with care in order to squeeze the best performance out of it. In this regard, communication metrics centered around volume play a crucial role. Assuming $Y = AX$ is performed in a repeated/iterative manner, where the elements of X change in each iteration and the elements of A remain the same, the partitions on X and Y matrices should be conformable in order to avoid unnecessary communication during the parallel operations.

In a system with K processors, we consider the problem of obtaining a rowwise partition of A, where processor P_k stores the submatrix blocks $A_{k\ell}$, for $1 \leq \ell \leq K$, where size of $A_{k\ell}$ is $n_k \times n_\ell$ These submatrix blocks form the row stripe R_k and P_k is held responsible for computing $Y_k = R_k X$. Since P_k only stores X_k, it needs to receive the corresponding elements of X from other processors to compute Y_k. This necessitates point-to-point communication between processors. This scheme is called *one-dimensional row-parallel algorithm* and it consists of the following steps for any processor P_k with $1 \leq k \leq K$:

1. For each off-diagonal block $A_{\ell k}$, for $1 \leq \ell \leq K$, with at least one nonzero in it, P_k sends the respective elements of X_k to processor P_ℓ. If a_{ij} is a nonzero in this off-diagonal block, then jth row of X need to be communicated.
2. Perform computations on local submatrix A_{kk} using X_k. Local computations do not necessitate communication. Y_k is first set with the result of this computation.
3. For each off-diagonal block $A_{k\ell}$, for $1 \leq \ell \leq K$, with at least one nonzero in it, P_k receives the respective elements of X from P_ℓ in order to perform computations on the respective nonlocal submatrix block. Y_k is updated with the results of nonlocal computations and its final value is computed.

Then, with some possible dense matrix operations that involve Y, new X is computed and used in the upcoming iteration. A local submatrix block is simply the diagonal block owned by the respective processor (A_{kk}), whereas nonlocal submatrix blocks are the off-diagonal ones ($A_{k\ell}$, with $k \neq \ell$). As hinted above, computations involving local submatrix blocks can be carried out without communication, whereas the computations involving nonlocal ones may necessitate communication if they are nonempty. In the following sections, we describe how to distribute these three matrices to processors via a hypergraph partitioning model that minimizes total communication volume and another model that reduces maximum volume applied on top of the former, hence able to address two important communication cost metrics that contain message volume.

In order to parallelize the SpMM kernel, we utilize the concept of an atomic task, which signifies the smallest computational granularity that cannot further be divided, hence, an atomic task shall be executed by a single processor. In SpMM, the atomic task is defined to be the multiplication of row $a_{i,*}$ with whole X. The result of this multiplication are the elements of row $y_{i,*}$. In the hypergraph model, the atomic tasks are represented by vertices.

17.3.1 Hypergraph Model

In the hypergraph model $\mathscr{H} = (\mathscr{V}, \mathscr{N})$, the vertices represent the atomic tasks of computing the multiplication of rows of A with X, i.e., $v_i \in V$ represents $a_{i,*}X$. Note that v_i also represents the row $a_{i,*}$ as well as the computations associated with this row. The computational load of this task is the number of multiply-and-add operations. The weight of v_i is assigned the computational load associated with the corresponding multiplication, i.e.,

$$w(v_i) = nnz(a_{i,*}) \cdot nnz(X).$$

The dependencies among the computational tasks are captured by the nets. For each row of X, there exists a net n_j in the hypergraph and it captures the dependency of the computational tasks to the row j of X. In other words, n_j connects the vertices corresponding to the tasks that need row j of X in their computations. Hence, the vertices connected by n_j are given by

$$Pins(n_j) = \{v_i : a_{ij} \neq 0\}.$$

n_j effectively represents column j of A as well. Note that $|Pins(n_j)| = nnz(a_{*,j})$, where $nnz(\cdot)$ denotes the number of nonzeros in a row or column of matrix. Since the number of elements in a row of X is s, n_j is associated with a cost $c(n_j)$ proportional to s. In other words,

$$c(n_j) = s.$$

This quantity signifies the number of elements required by the computational tasks corresponding to the vertices in $Pins(n_j)$. As a result, there are m vertices, n nets and $nnz(A)$ pins in $\mathscr{H}$. This model is a simple extension of the model used for sparse matrix vector multiplication [9].

17.3.2 Partitioning and Decoding

The formed hypergraph $\mathscr{H} = (\mathscr{V}, \mathscr{N})$ is then partitioned into K vertex parts to obtain $\Pi = \{\mathscr{V}_1, \ldots, \mathscr{V}_K\}$. Without loss of generality, the set of rows corresponding to the vertices in $\mathscr{V}_k$ and the respective computations involving these rows are assigned to processor P_k. A net n_j in the cut necessitates communication of the elements of row j of X and this communication operation involves the processors corresponding to the parts in the connectivity set of this net. Specifically, if $\mathscr{V}_k \in \Lambda(n_j)$ is the owner of row j of X, then it needs to send this row to the remaining processors, i.e., to each processor P_ℓ such that $\mathscr{V}_\ell \in \Lambda(n_j) - \{\mathscr{V}_k\}$, amounting to a volume of $s \cdot (|\Lambda(n_j)| - 1)$, s being the number of elements in row j. An internal net does not necessitate communication as all the rows corresponding to the vertices connected by this net belong to the same processor. The objective of minimizing cutsize according to the connectivity metric [9] in the partitioning hence encodes the total volume of communication. The constraint of maintaining balance on the vertex part weights corresponds to maintaining balance on the computational loads of the processors.

Aforementioned formulation strives for reducing total communication volume. However, the high volume overhead of SpMM kernel makes another related volume-related metric maximum communication volume also important, which we address next.

17.3.3 A Volume-Balancing Extension for SpMM

The formulation used to balance the volume loads of processors is based on a hypergraph model. This model was used to address multiple communication cost metrics regarding one- and two-dimensional sparse matrix vector multiplication successfully [26, 27, 29]. Although the main objective of this model is to reduce the latency overhead by aiming to minimize total message count, another important aspect of it is to maintain a balance on communication volume. In our case, i.e., for SpMM, the latter proves to be more crucial. Note that maintaining a balance on volume loads of processors corresponds to providing an upper bound on the same metric. We extend this model for SpMM. For this model, it is assumed a partition information is inherent, such as the one obtained in the previous section—which is also utilized for this model.

Using the partitioning information $\Pi = \{\mathscr{V}_1, \ldots, \mathscr{V}_k\}$ obtained on $\mathscr{H} = (\mathscr{V}, \mathscr{N})$, we form another hypergraph $\mathscr{H}^C = (\mathscr{V}^C, \mathscr{N}^C)$ to summarize the communication operations due to this partitioning. Recall that a net n_j necessitates communication if it is cut in Π. This communication operation is represented by a vertex in $\mathscr{H}^C$. In other words, there exists a vertex $v_j^C \in \mathscr{V}^C$ for each cut net $n_j \in \mathscr{N}$

$$\mathscr{V}^C = \{v_j^C : |\Lambda(n_j)| > 1\}.$$

There exists a net $n_k^C \in \mathscr{N}^C$ for each processor corresponding to the parts in Π. Hence, there are K nets in $\mathscr{H}^C$. v_j^C is connected to each net corresponding to the processor that participate in the communication operation represented by v_j^C. Hence,

$$Nets(v_j^C) = \{n_k^C : \mathscr{V}_k \in \Lambda(n_j)\}.$$

The vertices connected by n_k^C correspond to the communication operations that P_k participates in

$$Pins(n_k^C) = \{v_j^C : Pins(n_j) \cap \mathscr{V}_k \neq \emptyset\} \cup v_f^C.$$

Net n_k^C connects another vertex v_f^C, which is fixed to $\mathscr{V}_k^C$ in partitioning and later used to decode the assignment of communication operations to processors. Here, the important point is the assignment of vertex weights in $\mathscr{H}^C$ as they signify the volume incurred in communication operations. The volume incurred in communicating a row j of X is already described in the previous section and here the vertex weight is set accordingly to this quantity

$$w(v_j^C) = s \cdot (|\Lambda(n_j)| - 1).$$

The nets are assigned unit costs

$$c(n_j^C) = 1.$$

Now consider a partition $\Pi^C = \{\mathscr{V}_1^C, \ldots, \mathscr{V}_K^C\}$ of $\mathscr{H}^C$. This vertex partition induces a distribution of communication operations, where the responsibility of the communication operations represented by the vertices in $\mathscr{V}_k^C$ are assigned to processor P_k without loss of generality. A net n_j^C in the cut necessitates messages between the respective processors. In other words, if the fixed vertex v_f^C connected by this net is in part $\mathscr{V}_k^C$, this implies P_k will receive a message from each of the processors corresponding to the vertex parts in $\Lambda(n_j^C) - \{\mathscr{V}_k^C\}$. Hence, the number of messages incurred by this net is equal to $|\Lambda(n_j^C) - \{\mathscr{V}_k^C\}|$, which can be at most $K - 1$ as there are K vertex parts. In the partitioning, the objective of minimizing cutsize according to the connectivity metric [9] hence encodes the total message count. As a part weight indicates the amount of volume that the respective processor is responsible for, the constraint of maintaining balance corresponds to maintaining balance on the volume loads of the processors, and thus bounding the maximum volume.

With the partitioning of $\mathscr{H}$, we address total volume and with the partitioning of $\mathscr{H}^C$, we address total message count and maximum volume. By making use of respective models, we are able to address communication cost metrics that are important for SpMM.

17.4 Experiments

17.4.1 SpGEMM

17.4.1.1 Dataset

We include sparse matrices from Stanford Network Analysis Project (SNAP) [18] and the Laboratory for Web Algorithmics (LAW) [4, 5]. These matrices are downloaded from the University of Florida Sparse Matrix Collection [12] and their properties are given in Table 17.1. These matrices commonly represent the adjacency matrices of road networks or web-related graphs such as relations between products, etc. On such graphs, APSP algorithms can be used to find distances among the entities according to a predetermined metric. In this work, we only consider squaring the original adjacency matrix once, as a representative of the APSP algorithm [11].

Table 17.1 Properties of input and output matrices

Matrix	Input matrices							Output matrix
	Number of			nnz in row		nnz in column		
	rows	columns	nonzeros	avg	max	avg	max	nnz
Road networks								
belgium_osm	1,441,295	1,441,295	3,099,940	2	10	2	10	5,323,073
luxembourg_osm	114,599	114,599	239,332	2	6	2	6	393,261
netherlands_osm	2,216,688	2,216,688	4,882,476	2	7	2	7	8,755,758
roadNet-CA	1,971,281	1,971,281	5,533,214	3	12	3	12	12,908,450
roadNet-PA	1,090,920	1,090,920	3,083,796	3	9	3	9	7,238,920
roadNet-TX	1,393,383	1,393,383	3,843,320	3	12	3	12	8,903,897
Web-related graphs								
144	144,649	144,649	2,148,786	15	26	15	26	10,416,087
amazon-2008	735,323	735,323	5,158,388	7	10	7	1,076	25,366,745
amazon0505	410,236	410,236	3,356,824	8	10	8	2,760	16,148,723
amazon0601	403,394	403,394	3,387,388	8	10	8	2,751	16,258,436

nnz: number of nonzeros

17.4.1.2 Experimental Setup

In order to verify the validity of the proposed parallelization method, an SpGEMM code is implemented and run on a BlueGene/Q system, named Juqueen. For partitioning the hypergraphs, PaToH [9] is used with default parameters except the allowed imbalance ratio, which is set to be equal to 10 %.

As a baseline algorithm, we implemented a binpacking-based method which only considers computational load balancing. This method adapts the best-fit-decreasing heuristic used in K-feasible binpacking problem [16]. The outer-product tasks are assigned to one of K bins in decreasing order of the number of scalar multiplications incurred by the outer products. The best-fit criterion is assigning the task to the minimally loaded bin, whereas the capacity constraint is not used in this method.

17.4.1.3 Results

The performances of the proposed HP-based method and the baseline binpacking-based method are compared in terms of speedups in Figs. 17.1 and 17.2. As seen in these figures, in all cases, the proposed HP-based method performs substantially better than the baseline method. Thus, this improvement verifies the benefit of reducing total message volume. As seen in the figures, the parallel efficiency of HP-based method remains above 20 % in almost all instances.

17.4.2 SpMM

17.4.2.1 Dataset

We test 10 matrices for SpMM. The properties of these matrices are given in Table 17.2. These sparse matrices are also from Stanford Network Analysis Project (SNAP) [18] and the Laboratory for Web Algorithmics (LAW) [4, 5], and they are all downloaded from the University of Florida Sparse Matrix Collection [12]. A common operation that can be performed on these matrices for any kind of graph analysis is the execution of the breadth-first search from multiple sources. This corresponds to multiple SpMM iterations.

17.4.2.2 Experimental Setup

We tested our model for SpMM on SuperMUC supercomputer, which runs IBM System x iDataPlex servers. A node on this system consists of two Intel Xeon Sandy Bridge-EP processors clocked at 2.7 GHz and has 32 GB of memory. The communication interconnect is based on Infiniband FDR10.

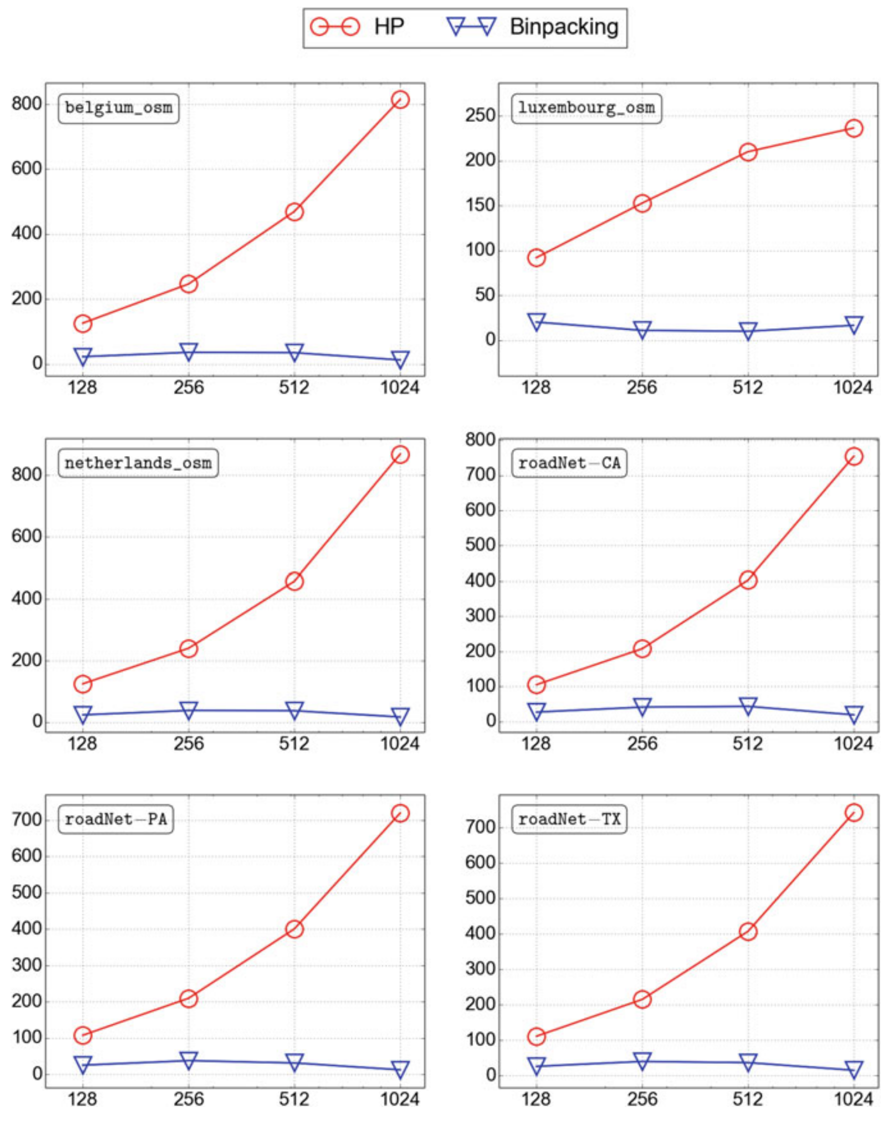

Fig. 17.1 Speedup curves comparing performances of the proposed and baseline SpGEMM algorithms on road networks

We test two models in our experiments. This first is the plain hypergraph partitioning for SpMM described in Sects. 17.3.1 and 17.3.2. This model aims at only reducing communication volume and is abbreviated with HP. The second model we test is the one that extends the HP as described in Sect. 17.3.3. This model aims at reducing communication volume and balancing communication volume in two separate phases and is abbreviated with HPVB. The dimension of the dense matrices is used as

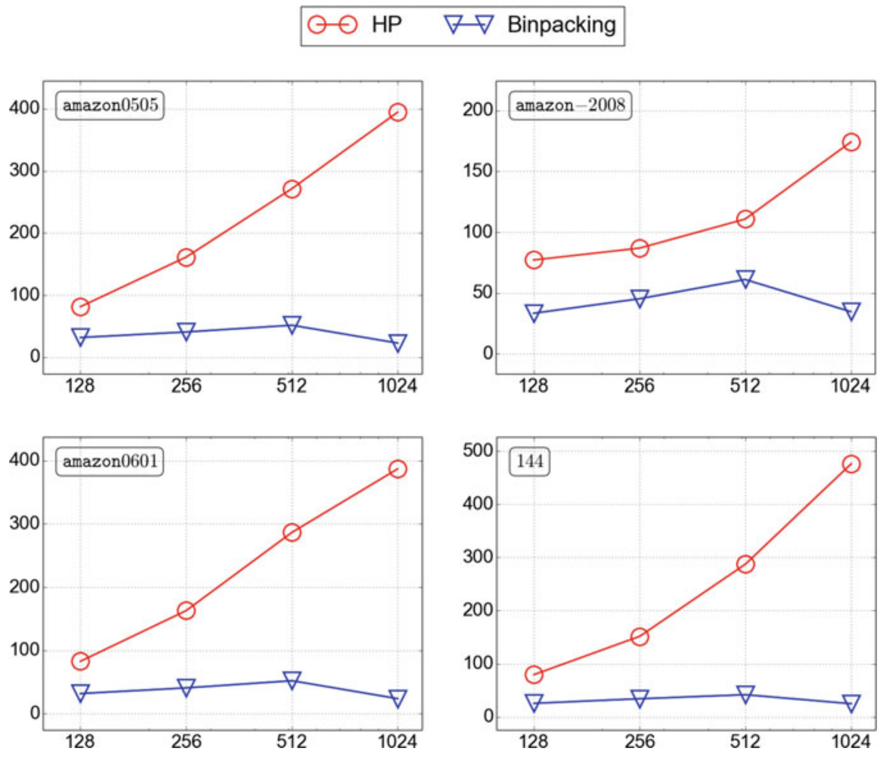

Fig. 17.2 Speedup curves comparing performances of the proposed and baseline SpGEMM algorithms on web link matrices

Table 17.2 Properties of matrices tested for SpMM

Name	Kind	#rows/cols	#nonzeros
amazon_2008	Amazon book similarity graph	735,323	5,158,388
amazon0312	Amazon product co-purchasing network	400,727	3,200,440
eu-2005	crawl of the .eu domain	862,664	19,235,140
roadNet-CA	road network	1,971,281	5,533,214
roadNet-PA	road network	1,090,920	3,083,796
roadNet-TX	road network	1,393,383	3,843,320
333SP	car mesh	3,712,815	22,217,266
asia_osm	road network	11,950,757	25,423,206
coPapersCiteseer	citation network	434,102	32,073,440
coPapersDBLP	citation network	540,486	30,491,458

$s = 5$. SpMM kernel is repeated in a loop for certain number of times and a warming up phase is included for healthier timing. We test for $K \in \{64, 128, 256, 512, 1024\}$ processors.

17.4.2.3 Partitioning and Runtime Results

We present the partitioning and runtime results in Table 17.3 for $K = 512$ and $K = 1024$. There are two communication cost metrics we consider and display in the table: total volume, indicated via "TV" column and volume load imbalance as percent, indicated via "VI (%)" column. Communication volume is in terms of communicated words and volume imbalance is on the send volumes of processors. The time of a single SpMM obtained by the two compared methods is given under the "runtime" column and it is in terms of microseconds. The lower runtimes for a specific test case are indicated via bold text.

As seen in the table, HPVB obtains significant improvements over HP in communication volume load imbalance. At $K = 512$ processors, it achieves up to $8\times$ improvement and at $K = 1024$ processors it achieves up to $9\times$ improvement. It

Table 17.3 Partitioning and runtime results for SpMM for $K = 512$ and $K = 1024$

K	Matrix	TV		VI (%)		Runtime (us)	
		HP	HPVB	HP	HPVB	HP	HPVB
512	amazon-2008	563,327	800,380	209	61	1855	**1195**
	amazon0312	324,433	447,455	155	56	1340	**948**
	eu-2005	320,741	428,054	438	57	1205	**751**
	roadNet-CA	33,764	51,524	120	37	275	**266**
	roadNet-PA	29,944	45,996	83	34	124	**120**
	roadNet-TX	28,901	44,124	120	37	181	**177**
	333SP	132,105	207,317	95	31	**713**	729
	asia_osm	10,149	15,517	451	58	**1213**	1236
	coPapersCiteseer	832,267	1,078,184	159	50	2489	**1724**
	coPapersDBLP	1,863,018	2,240,814	128	48	3860	**3082**
1024	amazon-2008	649,869	908,269	197	57	1977	**1049**
	amazon0312	383,068	518,843	168	181	1310	**1132**
	eu-2005	485,614	623,844	479	92	1325	**1174**
	roadNet-CA	52,915	81,568	111	36	**125**	128
	roadNet-PA	44,564	69,284	121	34	71	**65**
	roadNet-TX	44,288	67,460	134	40	**158**	160
	333SP	208,154	324,505	73	31	395	**374**
	asia_osm	17,971	27,222	523	58	530	**504**
	coPapersCiteseer	1,021,506	1,277,537	213	55	1797	**959**
	coPapersDBLP	2,112,593	2,522,300	233	52	3737	**1959**

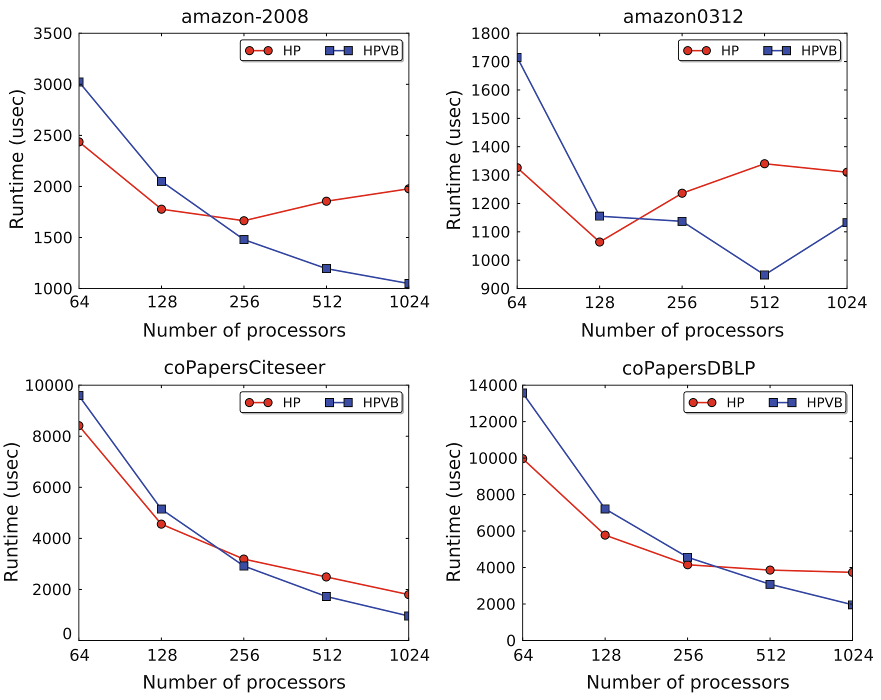

Fig. 17.3 Runtimes for SpMM

increases total volume compared to HP, causing an increase up to 57 and 56 % at 512 and 1024 processors, respectively. However, the reduction in maximum volume proves to be more vital for performance as HPVB almost always performs superior to HP as seen from the runtimes. Out of 20 instances at $K = 512$ and $K = 1024$ processors, HPVB obtains lower runtimes in 16 of them.

We present the obtained speedups in four test matrices in Fig. 17.3. As seen from the figure HPVB scales better than HP as its performance usually gets better compared to HP with increasing number of processors.

17.5 Conclusion

We described efficient parallelization of two important sparse matrix kernels for distributed systems that frequently occur in big data applications: sparse matrix–matrix multiplication and sparse matrix–dense matrix multiplication. For these two kernels, we proposed partitioning models in order to reduce the communication overheads and hence improve scalability. Our experiments show that efficient parallel performance for big data analysis on distributed systems requires careful data partitioning models and methods that are capable of exploiting certain communication cost metrics.

Acknowledgments This work was supported by The Scientific and Technological Research Council of Turkey (TUBITAK) under Grant EEEAG-115E212. This article is also based upon work from COST Action IC1406 (cHiPSet).

References

1. Intel math kernel library (2015). https://software.intel.com/en-us/intel-mkl
2. Agarwal, V., Petrini, F., Pasetto, D., Bader, D.A.: Scalable graph exploration on multicore processors. In: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, SC '10, pp. 1–11. IEEE Computer Society, Washington, DC, USA (2010). doi:10.1109/SC.2010.46
3. Akbudak, K., Aykanat, C.: Simultaneous input and output matrix partitioning for outer-product–parallel sparse matrix-matrix multiplication. SIAM J. Sci. Comput. **36**(5), C568–C590 (2014). doi:10.1137/13092589X
4. Boldi, P., Rosa, M., Santini, M., Vigna, S.: Layered label propagation: a multiresolution coordinate-free ordering for compressing social networks. In: Srinivasan S., Ramamritham K., Kumar A., Ravindra M.P., Bertino E., Kumar R. (eds.) Proceedings of the 20th International Conference on World Wide Web, pp. 587–596. ACM Press (2011)
5. Boldi, P., Vigna, S.: The WebGraph framework I: compression techniques. In: Proceedings of the Thirteenth International World Wide Web Conference (WWW 2004), pp. 595–601. ACM Press, Manhattan (2004)
6. Boman, E., Devine, K., Heaphy, R., Hendrickson, B., Heroux, M., Preis, R.: LDRD report: Parallel repartitioning for optimal solver performance. Tech. Rep. SAND2004–0365, Sandia National Laboratories, Albuquerque, NM (2004)
7. Buluç, A., Gilbert, J.R.: Parallel sparse matrix-matrix multiplication and indexing: implementation and experiments. SIAM J. Sci. Comput. (SISC) **34**(4), 170–191 (2012). doi:10.1137/110848244; http://gauss.cs.ucsb.edu/~aydin/spgemm_sisc12.pdf
8. Buluç, A., Madduri, K.: Parallel breadth-first search on distributed memory systems. In: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, SC '11, pp. 65:1–65:12. ACM, New York, NY, USA (2011). doi:10.1145/2063384.2063471; http://doi.acm.org/10.1145/2063384.2063471
9. Catalyurek, U.V., Aykanat, C.: Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. IEEE Trans. Parallel Distrib. Syst. **10**(7), 673–693 (1999)
10. CP2K: CP2K home page (Accessed at 2015). http://www.cp2k.org/
11. D'Alberto, P., Nicolau, A.: R-kleene: A high-performance divide-and-conquer algorithm for the all-pair shortest path for densely connected networks. Algorithmica **47**(2), 203–213 (2007). doi:10.1007/s00453-006-1224-z
12. Davis, T.A., Hu, Y.: The University of Florida sparse matrix collection. ACM Trans. Math. Softw. (TOMS) **38**(1), 1 (2011)
13. Dostál, Z., Horák, D., Kučera, R.: Total FETI-an easier implementable variant of the FETI method for numerical solution of elliptic PDE. Commun. Numer. Meth. Eng. **22**(12), 1155–1162 (2006)
14. Feng, Y., Owen, D., Peri, D.: A block conjugate gradient method applied to linear systems with multiple right-hand sides. Comput. Meth. Appl. Mech. Eng. **127**(14), 203–215 (1995). http://dx.doi.org/10.1016/0045-7825(95)00832-2; http://www.sciencedirect.com/science/article/pii/0045782595008322

15. 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., et al.: An overview of the Trilinos project. ACM Trans. Math. Softw. (TOMS) **31**(3), 397–423 (2005)
16. Horowitz, E., Sahni, S.: Fundamentals of Computer Algorithms. Computer Science Press (1978)
17. Kang, U., Tsourakakis, C.E., Faloutsos, C.: Pegasus: A peta-scale graph mining system implementation and observations. In: Proceedings of the 2009 Ninth IEEE International Conference on Data Mining, ICDM '09, pp. 229–238. IEEE Computer Society, Washington, DC, USA (2009). doi:10.1109/ICDM.2009.14
18. Leskovec, J., Krevl, A.: SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data (2014)
19. Marion-Poty, V., Lefer, W.: A wavelet decomposition scheme and compression method for streamline-based vector field visualizations. Comput. Graphics **26**(6), 899–906 (2002). doi:10.1016/S0097-8493(02)00178-4; http://www.sciencedirect.com/science/article/pii/S0097849302001784
20. Mattson, T., Bader, D., Berry, J., Buluc, A., Dongarra, J., Faloutsos, C., Feo, J., Gilbert, J., Gonzalez, J., Hendrickson, B., Kepner, J., Leiserson, C., Lumsdaine, A., Padua, D., Poole, S., Reinhardt, S., Stonebraker, M., Wallach, S., Yoo, A.: Standards for Graph Algorithm Primitives. ArXiv e-prints (2014)
21. NVIDIA Corporation: CUSPARSE library (2010)
22. O'Leary, D.P.: The block conjugate gradient algorithm and related methods. Linear Algebra Appl. **29**(0), 293–322 (1980). http://dx.doi.org/10.1016/0024-3795(80)90247-5; http://www.sciencedirect.com/science/article/pii/0024379580902475. Special Volume Dedicated to Alson S. Householder
23. O'Leary, D.P.: Parallel implementation of the block conjugate gradient algorithm. Parallel Comput. **5**(12), 127–139 (1987). http://dx.doi.org/10.1016/0167-8191(87)90013-5; http://www.sciencedirect.com/science/article/pii/0167819187900135. Proceedings of the International Conference on Vector and Parallel Computing-Issues in Applied Research and Development
24. Sarıyuce, A.E., Saule, E., Kaya, K., Çatalyurek, U.V.: Regularizing graph centrality computations. J. Parallel Distrib. Comput. **76**(0), 106–119 (2015). http://dx.doi.org/10.1016/j.jpdc.2014.07.006; http://www.sciencedirect.com/science/article/pii/S0743731514001282. Special Issue on Architecture and Algorithms for Irregular Applications
25. Sawyer, W., Messmer, P.: Parallel grid manipulations for general circulation models. In: Parallel Processing and Applied Mathematics. Lecture Notes in Computer Science, vol. 2328, pp. 605–608. Springer, Berlin (2006)
26. Selvitopi, O., Aykanat, C.: Reducing latency cost in 2D sparse matrix partitioning models. Parallel Comput. **57**, 1–24 (2016). http://dx.doi.org/10.1016/j.parco.2016.04.004; http://www.sciencedirect.com/science/article/pii/S0167819116300138
27. Selvitopi, R.O., Ozdal, M.M., Aykanat, C.: A novel method for scaling iterative solvers: avoiding latency overhead of parallel sparse-matrix vector multiplies. IEEE Trans. Parallel Distrib. Syst. **26**(3), 632–645 (2015). doi:10.1109/TPDS.2014.2311804
28. Shi, Z., Zhang, B.: Fast network centrality analysis using gpus. BMC Bioinf. **12**(1), 149 (2011). doi:10.1186/1471-2105-12-149
29. 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). doi:10.1137/S1064827502410463
30. Van De Geijn, R.A., Watts, J.: Summa: scalable universal matrix multiplication algorithm. Concurrency-Pract. Experience **9**(4), 255–274 (1997)