Partitioning Models for Scaling Parallel Sparse Matrix-Matrix Multiplication

KADIR AKBUDAK, OGUZ SELVITOPI, and CEVDET AYKANAT, Bilkent University

We investigate outer-product–parallel, inner-product–parallel, and row-by-row-product–parallel formulations of sparse matrix-matrix multiplication (SpGEMM) on distributed memory architectures. For each of these three formulations, we propose a hypergraph model and a bipartite graph model for distributing SpGEMM computations based on one-dimensional (1D) partitioning of input matrices. We also propose a communication hypergraph model for each formulation for distributing communication operations. The computational graph and hypergraph models adopted in the first phase aim at minimizing the total message volume and balancing the computational loads of processors, whereas the communication hypergraph models adopted in the second phase aim at minimizing the total message count and balancing the message volume loads of processors. That is, the computational partitioning models reduce the bandwidth cost and the communication hypergraph models reduce the latency cost. Our extensive parallel experiments on up to 2048 processors for a wide range of realistic SpGEMM instances show that although the outer-product–parallel formulation scales better, the row-by-row-product–parallel formulation is more viable due to its significantly lower partitioning overhead and competitive scalability. For computational partitioning models, our experimental findings indicate that the proposed bipartite graph models are attractive alternatives to their hypergraph counterparts because of their lower partitioning overhead. Finally, we show that by reducing the latency cost besides the bandwidth cost through using the communication hypergraph models, the parallel SpGEMM time can be further improved up to 32%.

CCS Concepts: • **Mathematics of computing → Hypergraphs**; **Graph algorithms**; **Computations on matrices**; • **Theory of computation → Parallel computing models**; • **Computing methodologies → Parallel algorithms**; **Distributed algorithms**;

Additional Key Words and Phrases: Sparse matrix-matrix multiplication, SpGEMM, hypergraph partitioning, graph partitioning, communication cost, bandwidth, latency

ACM Reference format:
Kadir Akbudak, Oguz Selvitopi, and Cevdet Aykanat. 2018. Partitioning Models for Scaling Parallel Sparse Matrix-Matrix Multiplication. *ACM Trans. Parallel Comput.* 4, 3, Article 13 (January 2018), 34 pages.
https://doi.org/10.1145/3155292

1 INTRODUCTION

We consider the parallelization of sparse matrix-matrix multiplication (SpGEMM) of the form $C = AB$ in a distributed setting. Based on one-dimensional (1D) partitioning of the input matrices A and B, four parallel algorithms can be devised:

This work is partially supported by the Scientific and Technological Research Council of Turkey (TUBITAK) under project EEEAG-115E512. This article is based on work from ICT COST Action IC1406 (cHiPSet).

Authors' addresses: K. Akbudak, O. Selvitopi, and C. Aykanat, Computer Engineering Department, Bilkent University, Ankara 06800, Turkey; emails: {kadir, reha, aykanat}@cs.bilkent.edu.tr.

https://doi.org/10.1145/3155292

Table 1. Partitioning Dimensions and Data Access Requirements of Parallel SpGEMM Algorithms

		Partitioning Dimension				Data Access Requirement		
	Parallel SpGEMM Algorithm	*A*	*B*	*C*		*A*	*B*	*C*
OP	Outer-Product Parallel	colwise	rowwise	nz-based		single	single	**multiple**
IP	Inner-Product Parallel	rowwise	colwise	rowwise	*A*-resident	single	**multiple**	single
				colwise	*B*-resident	**multiple**	single	single
RRP	Row-by-Row-Product Parallel	rowwise	rowwise	rowwise		single	**multiple**	single
CCP	Column-by-Column-Product Parallel	colwise	colwise	colwise		**multiple**	single	single

colwise: columnwise; nz-based: nonzero based.

—Columnwise partitioning of *A* and row-wise partitioning of *B*, which induce an outer-product–parallel algorithm (OP)
—Rowwise partitioning of *A* and column-wise partitioning of *B*, which induce an inner-product–parallel algorithm (IP),
—Rowwise partitioning of both *A* and *B*, which induces a row-by-row-product–parallel algorithm (RRP)
—Columnwise partitioning of both *A* and *B*, which induces a column-by-column-product–parallel algorithm (CCP)

OP is based on conformable columnwise partitioning of *A* and rowwise partitioning of *B*. A processor is held responsible for computing the *outer product* of a column slice of *A* with the respective row slice of *B*. In this scheme, the elements of *A* and *B* are accessed once and the elements of the output matrix *C* are accessed multiple times as the partial results produced for the same nonzeros of *C* need to be accumulated.

IP is based on rowwise partitioning of *A* and columnwise partitioning of *B*. A processor is held responsible for computing the *inner product* of a row slice of *A* with a column slice of *B*. IP has two variants: *A*-resident and *B*-resident. In the former, the elements of *A* and *C* are accessed once and the elements of *B* are accessed multiple times, whereas in the latter, the elements of *B* and *C* are accessed once and the elements of *A* are accessed multiple times. Since these two variants are dual, we only consider the former.

RRP is based on rowwise partitioning of both *A* and *B*. A processor is held responsible for computing the *premultiply* of a row slice of *A* with *B*. In this scheme, the elements of *A* and *C* are accessed once and the elements of *B* are accessed multiple times.

CCP is based on columnwise partitioning of both *A* and *B*. A processor is held responsible for computing the *postmultiply* of *A* with a column slice of *B*. In this scheme, the elements of *B* and *C* are accessed once and the elements of *A* are accessed multiple times. Since RRP and CCP display similar performance in most of our test cases, we only consider RRP.

Whenever we refer to OP, IP, or RRP, we refer to either the partitioning scheme or the parallel SpGEMM algorithm induced by this partitioning, which should be clear from the context. We assume the owner computes rule; i.e., the computations related to a portion of the matrix in a distributed setting are assigned to the processor that owns that portion. Hence, the matrix partitions also determine the ownership of the computations.

Table 1 compares the described algorithms in terms of their partitioning dimensions and data access requirements. Observe that all algorithms necessitate multiple accesses to the elements of a single matrix, whereas the elements of the other two matrices are accessed only once. In a distributed setting, single accesses do not necessitate communication as the elements of the respective matrix are processed by a single processor. Multiple accesses, on the other hand, necessitate communication on the elements of the respective matrix if these accesses are performed by more than one processor. The partitioning of the input matrices A and B may or may not induce a natural partitioning of the output matrix C. We describe how to obtain a partition of C later in detail.

Our main goal in this work is to efficiently parallelize the SpGEMM kernel for large-scale distributed systems. For this purpose, we propose partitioning models that encode communication-related objectives. Our contributions are twofold:

— We propose and compare graph and hypergraph partitioning models for OP, IP, and RRP, a total of six models. These are *computational partitioning models* as they obtain a partitioning of the computations on matrices. The aim of all these models is to reduce the total message volume while maintaining the computational load balance. The hypergraph models correctly encode the message volume incurred in parallel SpGEMM, while the graph models approximate it. However, the graph models prove themselves to be worthy alternatives to the hypergraph models due to their significantly lower partitioning overhead. Among the computational partitioning models, the hypergraph model for OP is previously investigated in a distributed setting in Akbudak and Aykanat (2014). Also, the hypergraph and bipartite graph models for RRP are proposed and utilized in a shared-memory setting (Akbudak and Aykanat 2017). Nonetheless, we describe them as they are evaluated in our experiments. The remaining three models are new and belong to this work.

— We further address the communication overheads with the proposed communication hypergraph models for the SpGEMM algorithms. These are different from the computational partitioning models as they obtain a distribution of the communication operations among processors. The aim is to reduce the total message count while maintaining a balance on the message volume loads of processors. The computational partitioning models aim at reducing the message volume, i.e., the bandwidth cost, while the communication hypergraph models aim at reducing the latency cost. By using the communication hypergraph models after the computational partitioning models, we are able to address both the bandwidth and the latency cost, both of which are important for scalability.

We conduct a thorough comparison of the aforementioned models (a total of 12 models) for three realistic SpGEMM categories of the forms $C = AA^T$ (Bisseling et al. 1993; Boman et al. 2005; Karypis et al. 1994), $C = AA$ (Borštnik et al. 2014; VandeVondele et al. 2012), and $C = AB$ (Linden et al. 2003) and perform realistic experiments on a large-scale system up to 2048 processors with the instances in these categories. Considering only the computational partitioning models (six models), compared to the recent work that uses a hypergraph model for OP (Akbudak and Aykanat 2014), we improve the parallel SpGEMM time up to 16% on average for the SpGEMM of the form $C = AA$. By using the graph models for the SpGEMM algorithms, we decrease the partitioning overhead about 15× to 35× compared to Akbudak and Aykanat (2014) while achieving close parallel SpGEMM performance with the hypergraph models. With the further utilization of the communication hypergraph models (six models), the parallel SpGEMM time is improved by up to 32%, 13%, and 6% for the $C = AA^T$, $C = AA$, and $C = AB$ categories, respectively.

The rest of the article is organized as follows. Section 2 gives the related work on parallelization of the SpGEMM kernel. The computational graph and hypergraph partitioning models are described in Section 3. Section 4 describes the communication hypergraph models. The experiments are presented in Section 5. Section 6 concludes.

2 RELATED WORK

SpGEMM is a kernel operation in many applications such as molecular dynamics (Challacombe 2000; VandeVondele et al. 2012; Challacombe 1999; Itoh et al. 1995; Schlegel et al. 2001; Li et al. 1993;

Millam and Scuseria 1997; Daniels et al. 1997; CP2K 2016), linear programming (LP) (Karypis et al. 1994; Bisseling et al. 1993; Boman et al. 2005), domain decomposition-based finite element simulations (Total-FETI 2016; Hapla et al. 2013), multigrid interpolation and restriction (Briggs et al. 2000), breadth-first search from multiple source vertices (Buluç and Gilbert 2011), triangle counting in graphs (Azad et al. 2015), data summarization (Ordonez et al. 2016), similarity join (Ordonez 2010), and item-to-item collaborative filtering in recommendation systems (Linden et al. 2003), all of which benefit from parallel processing to reduce execution times.

Parallelization of SpGEMM is well studied for shared–memory architectures (MKL 2015; Patwary et al. 2015), GPUs (Gremse et al. 2015; Bell et al. 2012; Dalton et al. 2013; Liu and Vinter 2014), and distributed memory architectures. Among the works on parallelization for distributed memory architectures, there are publicly available libraries such as Trilinos (Heroux et al. 2003) and Combinatorial BLAS (CombBLAS) (Buluç and Gilbert 2011).

The SpGEMM algorithm in the Tpetra (Nusbaum 2011) package of Trilinos uses 1D rowwise partitioning of the input and output matrices. It uses the A-resident algorithm so that only the rows of B are replicated via shift operations on a virtual ring of processors in K stages, K being the number of processors. CombBLAS (Buluç and Gilbert 2012) uses the SUMMA algorithm (van de Geijn and Watts 1997) for parallelization and an algorithm based on Doubly Compressed Sparse Column format (Buluç and Gilbert 2008) as a sequential kernel. In Akbudak and Aykanat (2014), three hypergraph models are proposed for outer-product–parallel SpGEMM in order to reduce the message volume and balance the computational loads of processors. The Distributed Block-Compressed Sparse Row library (Borštnik et al. 2014), which is developed for linear-scaling quantum simulations performed by CP2K (2016) and VandeVondele et al. (2012) use Cannon's algorithm (Cannon 1969) and tune SpGEMM kernels for dense blocks.

Theoretical lower bounds on the expected cost of communication in multiplication of sparse random matrices are studied by Ballard et al. (2013). In order to match these lower bounds, they propose 3D algorithms that are adaptations of existing dense algorithms (Demmel et al. 2013; Solomonik et al. 2011). Recently, a fine-grained hypergraph model for SpGEMM was proposed (Ballard et al. 2015). This model encodes the data requirements of each scalar multiplication operation, which makes the size of the hypergraph impractical to partition with the state-of-the-art hypergraph partitioners in case of big SpGEMM instances.

These works except Ballard et al. (2015) do not utilize the sparsity structure of the matrices in order to reduce the parallelization overheads. The main motivation of the models proposed in this work is to reduce the parallelization overheads via exploiting the sparsity structures of the matrices in the SpGEMM kernel. We investigate both graph and hypergraph models in our work to serve this purpose.

All of the applications mentioned at the beginning of this section may easily benefit from the proposed partitioning models. However, since our models necessitate partitioning as a preprocessing step, the applications repeatedly performing the SpGEMM operation are better suited for the proposed models so as to amortize the preprocessing overhead. There are several different applications that perform SpGEMM in a repeated manner in which the sparsity patterns of the matrices in the SpGEMM remain the same throughout the iterations while their numerical values are updated in each iteration. Examples of such applications include similarity join (Ordonez 2010) and collaborative filtering (Linden et al. 2003), both of which may be expressed as SpGEMM of the forms $C = AWA$ or $C = AWB$. In similarity join, matrix W is used for relative ranking of the features, whereas in item-to-item collaborative filtering, it is used for adjusting the importance of items in the filtering. Repeated SpGEMM also occurs in numerical algebra in the solution of LP problems (Karypis et al. 1994; Bisseling et al. 1993; Boman et al. 2005) through interior point methods, in which the positive-definite linear system $(AD^2A^T)x = b$ is solved in each iteration. Here, A

is the constraint matrix and D is a positive diagonal matrix that keeps changing. In each iteration, the coefficient matrix is formed with SpGEMM operation $C = AB$, with $B = D^2A^T$, which changes the numerical values of the matrices while keeping their sparsity patterns unchanged.

3 PARTITIONING MODELS FOR REDUCING BANDWIDTH COST

We first describe the notation used to represent the hypergraph and bipartite graph models. In the vertex, net, or edge sets, the superscripts "A," "B," and "C" show the association between the vertex/net/edge sets and the matrices, whereas the subscripts "r," "c," and "z" respectively imply that these sets represent rows, columns, and nonzeros of the matrices in the superscripts. For example, the vertices in the vertex set $\mathcal{V}_z^C$ represent the nonzeros of matrix C; i.e., $\mathcal{V}_z^C$ contains a vertex for each nonzero of C. Two matrix names in a superscript indicate a conformable representation of rows and/or columns of the respective matrices. That is, for example, $\mathcal{V}_{cr}^{AB}$ contains a vertex v_i for each column i of A and row i of B.

The row i and column i of a matrix, say, A, are respectively denoted with $a_{i,*}$ and $a_{*,i}$. The function $cols(\cdot)$ is used to denote the column indices of nonzeros in a row and the function $rows(\cdot)$ is used to denote the row indices of nonzeros in a column. For example, $cols(a_{i,*})$ denotes the column indices of the nonzeros in row i of A. The function $nnz(\cdot)$ is used to denote the number of nonzeros in a row, column, or matrix. The functions $nrows(\cdot)$ and $ncols(\cdot)$ are used to denote the number of rows and columns in a matrix, respectively. To indicate a nonzero element, we use $a_{i,j} \in A$.

The inner product of two vectors is denoted with "$\cdot$" (e.g., $a_{i,*} \cdot b_{*,j}$) and the outer product of two vectors is denoted with "$\otimes$" (e.g., $a_{*,i} \otimes b_{i,*}$). We do not use any symbol for scalar multiplication (e.g., multiplying a vector with a scalar, $a_{i,j}b_{j,*}$), vector-matrix multiply (e.g., premultiplying a matrix with a vector, $a_{i,*}B$), and matrix-matrix multiply (e.g., AB). Note that the scalar multiplication $a_{i,j}b_{j,*}$ refers to multiplying the scalar $a_{i,j}$ by the row-vector $b_{j,*}$.

We assume there are K processors in the parallel system. The following sections use the concept of an *atomic task*, which is defined to be the largest computation that cannot further be divided among processors; i.e., an atomic task can be executed by only one processor. The partitioning models assume that the reader is familiar with the notation for graph and hypergraph partitioning. For details, see Appendix A.

3.1 Outer-Product–Parallel (OP) SpGEMM

In OP, there are two types of atomic tasks: the outer product $a_{*,x} \otimes b_{x,*}$ and the reduction of partial results for nonzero $c_{i,j} \in C$. Here, A is partitioned columnwise and B is partitioned rowwise:

$$\hat{A} = AQ = [\,A_1 \ \ldots \ A_K\,] \quad \text{and} \quad \hat{B} = QB = \begin{bmatrix} B_1 \\ \vdots \\ B_K \end{bmatrix},$$

where Q is the permutation matrix obtained via partitioning. Each processor P_k owns the kth column slice of A and the respective kth row slice of B. A conformable partition of A and B is desired in order to avoid redundant communication in local outer products. For this reason, the processor responsible for column x of A is held responsible for row x of B as well.

The partition of A and B does not yield a natural partition of C. Partitioning C corresponds to determining the processor that will be responsible for accumulating the partial results for each nonzero $c_{i,j}$, where $c_{i,j} = \sum_k c_{i,j}^{(k)}$. Here, $c_{i,j}^{(k)}$ is the partial result produced by P_k for $c_{i,j}$. We only focus on obtaining a two-dimensional (2D) partition of C, as it is shown to be more efficient than the 1D partitions of C (Akbudak and Aykanat 2014).

The multiplication phase containing the outer products can be performed without any communication. On the other hand, the computation of $c_{i,j}$ requires partial results $c_{i,j}^{(k)}$ and may incur communication since the partial result produced by each such P_k must be sent to the processor responsible for computing the final value of $c_{i,j}$. Hence, the computational load balance in the outer products and reduction operations and the communication costs in communicating nonzeros of C are two main performance issues that should be taken into account for scalability of OP.

Note that the hypergraph model described in Section 3.1.1 is already proposed in Akbudak and Aykanat (2014), while the bipartite graph model in Section 3.1.2 is new and proposed in this work.

3.1.1 Hypergraph Model. The outer-product–parallel SpGEMM is modeled with the hypergraph $\mathcal{H}_{OP} = \{\mathcal{V}_{cr}^{AB} \cup \mathcal{V}_z^C, \mathcal{N}_z^C\}$. There are $(ncols(A) = nrows(B)) + nnz(C)$ vertices and $nnz(C)$ nets in $\mathcal{H}_{OP}$. $\mathcal{V}_{cr}^{AB}$ contains a vertex v_x for each outer product $a_{*,x} \otimes b_{x,*}$ and $\mathcal{V}_z^C$ contains a vertex $v_{i,j}$ for the reduction of each $c_{i,j}$. Vertex v_x also represents column x of A and row x of B, and $v_{i,j}$ also represents $c_{i,j}$. $\mathcal{N}_z^C$ contains a net $n_{i,j}$ for each $c_{i,j} \in C$, where $n_{i,j}$ captures the dependency of $c_{i,j}$ to the outer products that produce a partial result for $c_{i,j}$. Hence, the vertices connected by $n_{i,j}$ are given by

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

The weight of v_x is the computational load of the respective outer product, i.e., $nnz(a_{*,x}) \times nnz(b_{x,*})$. The weight of $v_{i,j}$ is the computational load of the respective reduction operation, i.e., the number of partial results produced for $c_{i,j}$. With a two-constraint weight formulation used to capture the computational loads of the outer products and reduction operations, the vertex weights are assigned as

$$\begin{array}{llll} w_1(v_x) & = nnz(a_{*,x}) \times nnz(b_{x,*}) & w_2(v_x) & = 0 \\ w_1(v_{i,j}) & = 0 & w_2(v_{i,j}) & = |Pins(n_{i,j})| - 1. \end{array}$$

The nets are assigned unit costs:

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

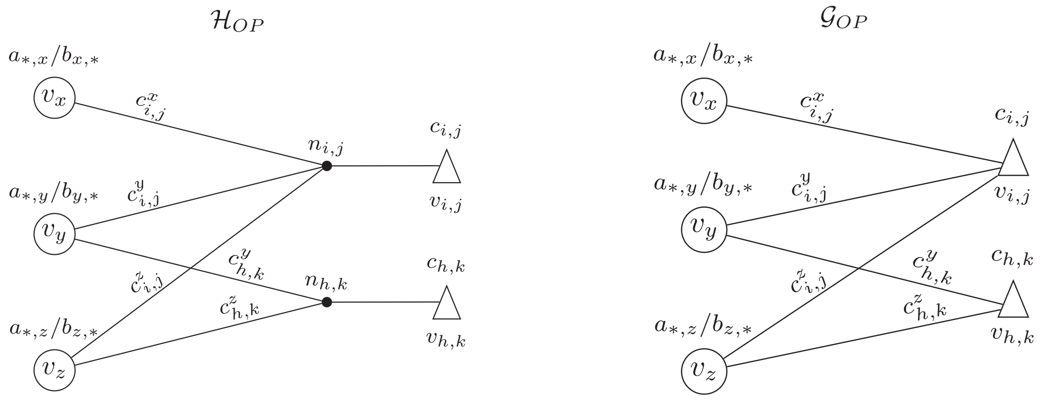
The left of Figure 1 illustrates how a hypergraph models three outer products contributing to two nonzeros.

3.1.2 Bipartite Graph Model. The outer-product–parallel SpGEMM is also modeled with the bipartite graph $\mathcal{G}_{OP} = \{\mathcal{V}_{cr}^{AB} \cup \mathcal{V}_z^C, \mathcal{E}_{z'}^C\}$. There are $(ncols(A) = nrows(B)) + nnz(C)$ vertices in $\mathcal{G}_{OP}$. The semantics of the vertices in the vertex sets $\mathcal{V}_{cr}^{AB}$ and $\mathcal{V}_z^C$ are the same with those in $\mathcal{H}_{OP}$. The dependency of $c_{i,j}$ to the outer products, however, is captured with edges instead of a net. $\mathcal{E}_{z'}^C$ contains an edge between $v_x \in \mathcal{V}_{cr}^{AB}$ and $v_{i,j} \in \mathcal{V}_z^C$ if the outer product $a_{*,x} \otimes b_{x,*}$ produces a partial result for $c_{i,j}$. Formally, $(v_x, v_{i,j}) \in \mathcal{E}_{z'}^C$ if $a_{i,x} \in A$ and $b_{x,j} \in B$. Thus, an edge represents a partial result (hence the subscript z' rather than z). The number of edges in $\mathcal{G}_{OP}$ is equal to the number of all partial results. The adjacency list of v_x is given by the vertices corresponding to the $c_{i,j}$ values for which the outer product represented by v_x produces a partial result, whereas the adjacency list of $v_{i,j}$ is given by the vertices corresponding to the outer products that produce a partial result for $c_{i,j}$:

$$\begin{aligned} Adj(v_x) & = \{v_{i,j} : i \in rows(a_{*,x}), j \in cols(b_{x,*})\}, \\ Adj(v_{i,j}) & = \{v_x : x \in cols(a_{i,*}) \wedge x \in rows(b_{*,j})\}. \end{aligned}$$

In weighting the vertices, the same multiconstraint formulation in $\mathcal{H}_{OP}$ is used. The edges are assigned unit costs as they represent a single partial result:

$$c((v_x, v_{i,j})) = 1.$$



$$c_{i,j} \leftarrow c^x_{i,j} + c^y_{i,j} + c^z_{i,j}$$
$$c_{h,k} \leftarrow c^y_{h,k} + c^z_{h,k}$$

Fig. 1. Hypergraph model (left) (Akbudak and Aykanat 2014) and bipartite graph model (right) for outer-product–parallel SpGEMM with three outer products contributing to two nonzeros. $c^x_{i,j}$ reads as the partial result produced by the outer product $a_{*,x} \otimes b_{x,*}$ for $c_{i,j}$.

The right of Figure 1 illustrates how a bipartite graph models three outer products contributing to two nonzeros.

3.2 Inner-Product–Parallel (IP) SpGEMM

In IP, an atomic task is defined as the multiplication of row x of A with each column j of B such that the result of $a_{x,*} \cdot b_{*,j}$ is nonzero, i.e., $c_{x,j} \in C$. The inner products that involve row x of A are given by the set $\{a_{x,*} \cdot b_{*,j} : j \in cols(c_{x,*})\}$, which we denote with the vector-matrix multiply $a_{x,*}B$. Defining each individual inner product as an atomic task would result in more degrees of freedom due to finer granularity, which may at first seem to lead to more scalable partitioning. Doing so, however, requires multiple accesses to elements of both A and B; hence, in a distributed setting, the elements of both A and B would need to be communicated. For this reason, we prefer the former, which results in multiple accesses to the elements of only B. In this scheme, A and C are partitioned rowwise and B is partitioned columnwise:

$$\hat{A} = PA = \begin{bmatrix} A_1 \\ \vdots \\ A_K \end{bmatrix}, \quad \hat{B} = BQ = [B_1 \cdots B_K] \text{ and } \quad \hat{C} = PCQ = \begin{bmatrix} C_1 \\ \vdots \\ C_K \end{bmatrix},$$

where P and Q are the permutation matrices obtained via partitioning. Each processor P_k hence owns the kth row slice of A and C, and the kth column slice of B.

The rowwise partition of A naturally yields a rowwise partition of C since no task other than $a_{x,*}B$ contributes to $c_{x,*}$ and $a_{x,*}B$ contributes only to $c_{x,*}$. In other words, $c_{x,*} = a_{x,*}B$. For this reason, the processor responsible for row x of A naturally becomes responsible for row x of C as well.

For P_k to perform $a_{x,*}B$, it needs to receive the nonzeros in respective columns of B via communication. Specifically, for a nonzero $a_{x,i}$ in row x of A, P_k needs to receive $b_{i,j}$ from the processor that owns column j of B. After processors receive specific nonzeros needed for their inner products, the computation of C can be performed without any communication. Hence, the computational load balance in the inner products and the communication costs in communicating nonzeros of B are two main performance issues that should be taken into account for scalability of IP.

3.2.1 Hypergraph Model. The inner-product–parallel SpGEMM is modeled with the hypergraph $\mathcal{H}_{IP} = \{\mathcal{V}_{rr}^{AC} \cup \mathcal{V}_{c}^{B}, \mathcal{N}_{z}^{B}\}$. There are $(nrows(A) = nrows(C)) + ncols(B)$ vertices and $nnz(B)$ nets in $\mathcal{H}_{IP}$. $\mathcal{V}_{rr}^{AC}$ contains a vertex v_x for each vector-matrix multiply $c_{x,*} = a_{x,*}B$. Vertex v_x also represents both row x of A and row x of C. $\mathcal{V}_{c}^{B}$ contains a vertex v_j for each column of B. This vertex does not signify computation and its sole purpose is to enable the columnwise partitioning of B. $\mathcal{N}_{z}^{B}$ contains a net $n_{i,j}$ for each $b_{i,j} \in B$, where $n_{i,j}$ captures the dependency of $a_{x,*}B$ computations to $b_{i,j}$. Hence, the vertices connected by $n_{i,j}$ are given by

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

The weight of v_x is the computational load of the respective vector-matrix multiply $a_{x,*}B$:

$$w(v_x) = \sum_{i \in cols(a_{x,*})} nnz(b_{i,*}),$$

whereas the weight of v_j is zero as it does not signify computation. The nets are assigned unit costs since they indicate the dependency on a single nonzero:

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

The left of Figure 2 illustrates how a hypergraph models the relations for three vector-matrix multiplies needing a total of three nonzeros from two columns of B.

3.2.2 Bipartite Graph Model. The inner-product–parallel SpGEMM is also modeled with the bipartite graph $\mathcal{G}_{IP} = \{\mathcal{V}_{rr}^{AC} \cup \mathcal{V}_{c}^{B}, \mathcal{E}_{z}^{C}\}$. There are $(nrows(A) = nrows(C)) + ncols(B)$ vertices and $nnz(C)$ edges in $\mathcal{G}_{IP}$. The semantics of the vertices in the vertex sets $\mathcal{V}_{rr}^{AC}$ and $\mathcal{V}_{c}^{B}$ are the same as those of $\mathcal{H}_{IP}$. $\mathcal{E}_{z}^{C}$ contains an edge between $v_x \in \mathcal{V}_{rr}^{AC}$ and $v_j \in \mathcal{V}_{c}^{B}$ if the vector-matrix multiply $a_{x,*}B$ needs at least one nonzero in column j of B, or in short, if $c_{x,j} \in C$. Formally, $\mathcal{E}_{z}^{C} = \{(v_x, v_j) : c_{x,j} \in C\}$. The adjacency list of v_x is given by the vertices corresponding to the columns of B that contain at least one nonzero required for the multiplication represented by v_x, whereas the adjacency list of v_j is given by the vertices corresponding to the multiplications that need at least one nonzero in the column represented by v_j:

$$Adj(v_x) = \{v_j : j \in cols(c_{x,*})\},$$
$$Adj(v_j) = \{v_x : x \in rows(c_{*,j})\}.$$

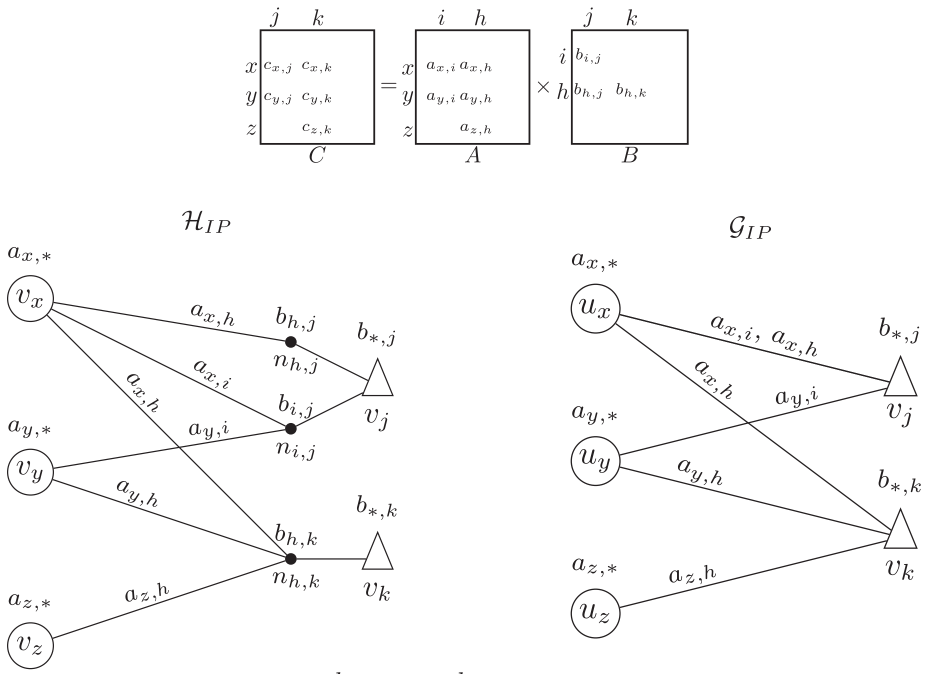
The weights of the vertices in $\mathcal{G}_{IP}$ are the same as those of $\mathcal{H}_{IP}$. The edge costs are assigned the number of nonzeros needed by a vector-matrix multiply:

$$c((v_x, v_j)) = |\{i : i \in cols(a_{x,*}) \wedge i \in rows(b_{*,j})\}|.$$

The right of Figure 2 illustrates how a bipartite graph models the relations for three vector-matrix multiplies needing a total of three nonzeros from two columns of B.

3.3 Row-by-Row-Product–Parallel (RRP) SpGEMM

In RRP, an atomic task is defined as the multiplication of row x of A with each row i of B, where a nonzero $a_{x,i}$ is multiplied with $b_{i,*}$. This atomic task is denoted with $a_{x,*}B$. Although the atomic task notation is the same as the one used for IP, this is a different atomic task, as the rows, instead of columns, of B are multiplied with the rows of A. The set of scalar multiplications necessitated



$$c_{x,j} \leftarrow a_{x,i}\ b_{i,j} + a_{x,h}\ b_{h,j}$$
$$c_{x,k} \leftarrow a_{x,h}\ b_{h,k}$$
$$c_{y,j} \leftarrow a_{y,i}\ b_{i,j}$$
$$c_{y,k} \leftarrow a_{y,h}\ b_{h,k}$$
$$c_{z,k} \leftarrow a_{z,h}\ b_{h,k}$$

Fig. 2. The matrices at top illustrate an SpGEMM instance of the form $C = AB$. Hypergraph model (left) and bipartite graph model (right) for inner-product–parallel SpGEMM with three vector-matrix multiplies needing a total of three nonzeros from two columns of B. Note that the computation of $c_{z,k}$ is not shown for the sake of clarity of the model drawings.

by row x of A is given by $\{a_{x,i}b_{i,*} : i \in cols(a_{x,*})\}$. In this scheme, A, B, and C are all partitioned rowwise:

$$\hat{A} = PAQ = \begin{bmatrix} A_1 \\ \vdots \\ A_K \end{bmatrix}, \quad \hat{B} = QB = \begin{bmatrix} B_1 \\ \vdots \\ B_K \end{bmatrix} \quad \text{and} \quad \hat{C} = PC = \begin{bmatrix} C_1 \\ \vdots \\ C_K \end{bmatrix},$$

where P and Q are the permutation matrices obtained via partitioning. Each processor P_k hence owns the kth row slice of A, B, and C.

The partition of A yields a natural partition of C since no task other than $a_{x,*}B$ contributes to $c_{x,*}$ and $a_{x,*}B$ contributes only to $c_{x,*}$. In other words, $c_{x,*} = a_{x,*}B$. For this reason, the processor responsible for row x of A naturally becomes responsible for row x of C as well.

For P_k to perform $a_{x,*}B$, it needs to receive the respective rows of B via communication. Specifically, for a nonzero $a_{x,i}$ in row x of A, P_k needs to receive row i of B from the processor that owns it. After processors receive needed rows for their scalar multiplications, the computation of C can be performed without any communication. Hence, the computational load balance in the scalar multiplications and the communication costs in communicating rows of B are two main performance issues that should be taken into account for scalability of RRP.

Note that the models in Sections 3.3.1 and 3.3.2 are utilized in Akbudak and Aykanat (2017) for improving the performance of SpGEMM on many-core architectures. This work evaluates them in a distributed setting. The main difference is that partitioning methods proposed in Akbudak and Aykanat (2017) aim at exploiting cache locality via using a larger number of partitions and looser allowed imbalance thresholds.

3.3.1 Hypergraph Model. The row-by-row-product–parallel SpGEMM is modeled with the hypergraph $\mathcal{H}_{RRP} = \{\mathcal{V}_{rr}^{AC}, \mathcal{N}_r^B\}$. There are $nrows(A) = nrows(C)$ vertices and $nrows(B)$ nets in $\mathcal{H}_{RRP}$. $\mathcal{V}_{rr}^{AC}$ contains a vertex v_x for each vector-matrix multiply $c_{x,*} = a_{x,*}B$. Vertex v_x also represents both row x of A and row x of C. $\mathcal{N}_r^B$ contains a net n_i for each row i of B, where n_i captures the dependency of $a_{x,*}B$ computations to $b_{i,*}$. Hence, the vertices connected by n_i are given by

$$Pins(n_i) = \{v_x : x \in rows(a_{*,i})\}.$$

The weight of v_x is the computational load of the respective vector-matrix multiply:

$$w(v_x) = \sum_{i \in cols(a_{x,*})} nnz(b_{i,*}).$$

Note that a single weight is enough here since there exists a single computational phase and the rows are needed by tasks as a whole (not specific nonzeros as in IP). The nets are assigned the costs of number of nonzeros in the respective rows of B in order to indicate the dependency to rows as a whole:

$$c(n_i) = nnz(b_{i,*}).$$

The left of Figure 3 illustrates how a hypergraph models the relations for three vector-matrix multiplies needing two rows of B.

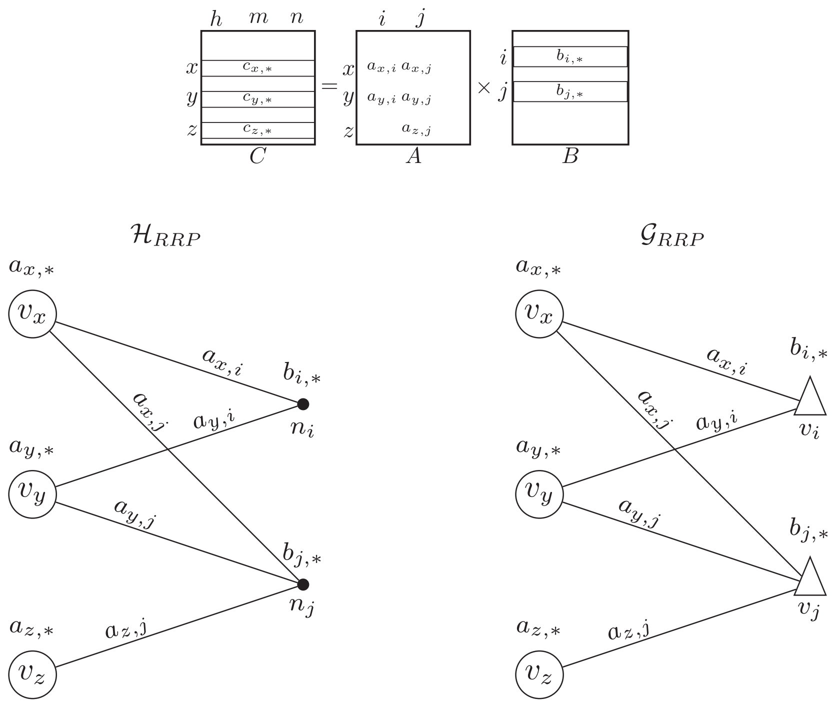
3.3.2 Bipartite Graph Model. The row-by-row-product–parallel SpGEMM is also modeled with the bipartite graph $\mathcal{G}_{RRP} = \{\mathcal{V}_{rr}^{AC} \cup \mathcal{V}_r^B, \mathcal{E}_z^A\}$. There are $(nrows(A) = nrows(C)) + nrows(B)$ vertices and $nnz(A)$ edges in $\mathcal{G}_{RRP}$. The semantics of the vertex set $\mathcal{V}_{rr}^{AC}$ in both $\mathcal{H}_{RRP}$ and $\mathcal{G}_{RRP}$ are the same. However, there is an additional vertex set $\mathcal{V}_r^B$ in the bipartite graph model, where $\mathcal{V}_r^B$ contains a vertex v_i for each row of B. These vertices do not signify computation; rather, they exist to help capture computational dependencies and partitioning of B (in $\mathcal{H}_{RRP}$ the nets are used for this purpose). $\mathcal{E}_z^A$ contains an edge between $v_x \in \mathcal{V}_{rr}^{AC}$ and $v_i \in \mathcal{V}_r^B$ if the vector-matrix multiplies $a_{x,*}B$ needs row i of B for the computation of $c_{x,*}$. Formally, $(v_x, v_i) \in \mathcal{E}_z^A$ if $i \in cols(a_{x,*})$. Row i of B is actually needed for each nonzero in $a_{*,i}$; hence, there are $nnz(A)$ number of edges. The adjacency list of v_x is given by the vertices corresponding to the rows of B that are required by the multiplication represented by v_x, whereas the adjacency list of v_i is given by the vertices corresponding to multiplications that need the row represented by v_i:

$$Adj(v_x) = \{v_i : i \in cols(a_{x,*})\},$$
$$Adj(v_i) = \{v_x : x \in rows(a_{*,i})\}.$$

The weight of v_x is the same as that of in $\mathcal{H}_{RRP}$, whereas the weight of v_i is zero as it does not signify computation. The edge costs are assigned the number of nonzeros in the respective rows of B whose corresponding vertices they are adjacent to:

$$c((v_x, v_i)) = nnz(b_{i,*}).$$

The right of Figure 3 illustrates how a bipartite graph models the relations for three vector-matrix multiplies needing two rows of B.



$$c_{x,*} \leftarrow a_{x,i}\ b_{i,*} + a_{x,j}\ b_{j,*}$$
$$c_{y,*} \leftarrow a_{y,i}\ b_{i,*} + a_{y,j}\ b_{j,*}$$
$$c_{z,*} \leftarrow a_{z,j}\ b_{j,*}$$

Fig. 3. The matrices at top illustrate an SpGEMM instance of the form $C = AB$. Hypergraph model (left) and bipartite graph model (right) for row-by-row-product–parallel SpGEMM with three vector-matrix multiplies needing two rows of B.

3.4 Decoding Partitions

We now describe how to decode the partitions for OP, IP, and RRP in order to obtain a distribution of the matrices and the computations on them. Without loss of generality, we assume that processor P_k is associated with the computational tasks corresponding to the vertices in part $\mathcal{V}_k$ of the partition obtained.

OP. Consider a K-way vertex partition $\Pi_{OP} = \{\mathcal{V}_1, \ldots, \mathcal{V}_K\}$ on $\mathcal{H}_{OP}$ or $\mathcal{G}_{OP}$. We use the same partition notation for both $\mathcal{H}_{OP}$ and $\mathcal{G}_{OP}$ as they have the same vertex sets. A part $\mathcal{V}_k$ may contain vertices from both $\mathcal{V}_{cr}^{AB}$ (e.g., v_x) and $\mathcal{V}_z^{C}$ (e.g., $v_{i,j}$). A vertex $v_x \in \mathcal{V}_k$ is decoded by assigning column x of A, row x of B, and the outer product $a_{*,x} \otimes b_{x,*}$ to P_k. Similarly, a vertex $v_{i,j} \in \mathcal{V}_k$ is decoded by assigning $c_{i,j}$, the reduction of partial results for $c_{i,j}$, and the possible communication operation on $c_{i,j}$ to P_k.

IP. Consider a K-way vertex partition $\Pi_{IP} = \{\mathcal{V}_1, \ldots, \mathcal{V}_K\}$ on $\mathcal{H}_{IP}$ or $\mathcal{G}_{IP}$. Again, we use the same partition notation for both as their vertex sets are the same. A part $\mathcal{V}_k$ may contain vertices from both $\mathcal{V}_{rr}^{AC}$ (e.g., v_x) and $\mathcal{V}_c^{B}$ (e.g., v_j). A vertex $v_x \in \mathcal{V}_k$ is decoded by assigning row x of A, row x of C, and the vector-matrix multiply $c_{x,*} = a_{x,*}B$ to P_k. Similarly, a vertex $v_j \in \mathcal{V}_k$ is decoded by assigning column j of B and the possible communication operation on this column to P_k.

Table 2. Comparison of Partitioning Models

	Requires Symbolic Multiplication	Hypergraph			Bipartite Graph	
		Number of			Number of	
		Vertices	Nets	Pins	Vertices	Edges
OP	Yes	$ncols(A) + nnz(C)$	$nnz(C)$	$\#flops/2 + nnz(C)$	$ncols(A) + nnz(C)$	$\#flops/2$
IP	Yes	$nrows(A) + ncols(B)$	$nnz(B)$	$\#flops/2 + ncols(B)$	$nrows(A) + ncols(B)$	$nnz(C)$
RRP	No	$nrows(A)$	$nrows(B)$	$nnz(A)$	$nrows(A) + nrows(B)$	$nnz(A)$

RRP. We consider the partitions on the hypergraph and bipartite graph models separately as their vertex sets are different. Consider a K-way partition $\Pi_{RRP} = \{\mathcal{V}_1, \ldots, \mathcal{V}_K\}$ on $\mathcal{H}_{RRP}$. A vertex $v_x \in \mathcal{V}_k$ is decoded by assigning row x of A, row x of C, and the vector-matrix multiply $c_{x,*} = a_{x,*}B$ to P_k. Observe that this partition does not directly induce a partition of the rows of B. However, B can easily be partitioned by associating row i of B corresponding to net n_i with one of the processors corresponding to one of the parts in the connectivity set of this net. Doing otherwise, i.e., assigning row i to a part that is not in the connectivity set of the respective net, incurs extra communication.

A K-way vertex partition $\Pi_{RRP} = \{\mathcal{V}_1, \ldots, \mathcal{V}_K\}$ on $\mathcal{G}_{RRP}$ is decoded in a similar manner. A part $\mathcal{V}_k$ may contain vertices from both $\mathcal{V}_{rr}^{AC}$ (e.g., v_x) and $\mathcal{V}_r^B$ (e.g., v_i) in the bipartite graph model. A vertex $v_x \in \mathcal{V}_k$ is decoded in the same way as is done in $\mathcal{H}_{RRP}$. A partition of the bipartite graph, however, also contains the partitioning information of B within as the rows of B are represented by the vertices in $\mathcal{V}_r^B$. Simply, a vertex $v_i \in \mathcal{V}_k$ is decoded by assigning row i of B to P_k.

Partitioning Constraint and Objective. The partitioning constraint of balancing part weights in both $\mathcal{H}_{OP}$ and $\mathcal{G}_{OP}$ corresponds to maintaining computational load balance in outer product computations and reduction of partial results for nonzeros of C, while in $\mathcal{H}_{IP}$, $\mathcal{G}_{IP}$, $\mathcal{H}_{RRP}$, and $\mathcal{G}_{RRP}$, it solely corresponds to maintaining computational load balance in the respective vector-matrix multiply. The partitioning objective of minimizing cutsize in $\mathcal{H}_{OP}$, $\mathcal{H}_{IP}$, $\mathcal{H}_{RRP}$ and in $\mathcal{G}_{OP}$, $\mathcal{G}_{IP}$, $\mathcal{G}_{RRP}$ differs as the hypergraph models correctly encapsulate the message volume incurred during parallel SpGEMM, while the bipartite graph models encapsulate an approximation of the same metric. In O, the bipartite graph model encodes the data dependencies as if a processor P_k will send multiple partial results (say $c_{i,j}^{k_1}, c_{i,j}^{k_2}, c_{i,j}^{k_3}$, all produced by P_k) for the same nonzero $c_{i,j}$ to P_l that is responsible for reducing this nonzero (i.e., $c_{i,j} = c_{i,j}^{k_1} + c_{i,j}^{k_2} + c_{i,j}^{k_3}$) without first summing them itself. This overestimates the message volume incurred in parallel SpGEMM (here, the bipartite graph model encodes it as three elements being sent; however, it is only one as P_k first sums them). In a similar manner, in IP or RRP, the bipartite graph model encodes the data dependencies as if a processor P_k expands (sends) the same column $b_{*,j}$ or row $b_{i,*}$ to P_l multiple times, where in reality it will be sent only once as they are the same values, which again overestimates the message volume. Note that the bandwidth requirements of the bipartite graph models are equal to $\#flops/2$ in the worst case, which occurs when all edges are on the cut. The respective hypergraph models refrain from these issues by correctly encoding the multiway directed relations with nets instead of edges.

3.5 Comparison of Partitioning Models

We compare the models described so far in Table 2. The models are compared with respect to their sizes and symbolic multiplication requirements. In the table, $\#flops$ refers to the number of multiply-and-add operations performed for $C = AB$ under the assumption that each scalar multiplication requires an addition. Hence, $\#flops/2 \geq nnz(C)$, and in general $\#flops \gg nnz(C)$. Note that $\#flops/2$ is equal to the number of partial results produced for nonzeros of C.

A symbolic multiplication for an SpGEMM algorithm is required when the computation pattern of the output matrix C is needed to determine the topology of the respective model. OP and IP require a symbolic multiplication since the partitioning models for both require full access to the actual computation that forms C. RRP, on the other hand, does not require a symbolic multiplication as the topology can be directly obtained from the sparsity patterns of A and B. This can be apparently seen in Table 2 as the number of pins in the hypergraph models and number of edges in the bipartite graph models for OP and IP involve $nnz(C)$ and/or $\#flops$, both of which can only be determined by performing a symbolic multiplication. In this regard, it can be said that RRP has an inherent advantage over OP and IP.

When the hypergraph models are compared among themselves, it can be said that the hypergraph model for OP has the highest number of vertices, whereas the hypergraph model for RRP has the smallest. The hypergraph model for RRP among them again has the smallest number of nets and pins. Hence, it is expected that the hypergraph model for RRP will have a lower partitioning overhead compared to the other two. Among the bipartite graph models, the one for OP has the highest number of vertices and edges, and hence it is expected to have the highest partitioning overhead among the models. When a hypergraph model and a bipartite graph model are compared for a specific SpGEMM algorithm, although their sizes seem comparable, in practice the bipartite graph model is likely to have a considerably lower partitioning overhead as the graph partitioners are usually faster than the hypergraph partitioners due to the inherent complexity of dealing with hypergraphs (Çatalyürek and Aykanat 1999a).

4 PARTITIONING MODELS FOR REDUCING LATENCY COST

In this section, we propose new models to reduce the latency cost of parallel SpGEMM. All models described up to this section aim at reducing the total message volume. In order to address the latency cost, we make use of a model called communication hypergraph. This model is successfully used to improve the performance of 1D- and 2D-parallel sparse matrix-vector multiplication on distributed systems (Uçar and Aykanat 2004). Here, we describe three such novel models for OP, IP, and RRP.

4.1 Basics

The main goal of the communication hypergraph model is to obtain a distribution of communication operations among processors. The hypergraph and bipartite graph models described in Section 3 are computational partitioning models as the vertices of these models represent computational tasks. In the communication hypergraph model, however, the vertices represent communication operations and the nets represent the processors in the system. A communication operation in an SpGEMM algorithm is determined according to the adopted partitioning and can be of two types: (i) sending matrix elements possibly to multiple processors or (ii) receiving matrix elements possibly from multiple processors. The former is referred to as an *expand* type of operation and the latter is referred to as a *reduce* type of operation.

If a processor participates in a communication operation, the net corresponding to that processor connects the vertex representing the respective communication operation. Since the communication operations are to be distributed among K processors, a K-way partitioning is performed, and as a result of this partitioning, the communication operations corresponding to the vertices in the kth part are, without loss of generality, associated with processor P_k. The partitioning objective of minimizing cutsize (2) minimizes the total message count, while the partitioning constraint of maintaining balance on part weights (1) preserves a balance on the message volume loads of processors. For more details, see Uçar and Aykanat (2004) and Selvitopi and Aykanat (2016).

To denote the communication operations in parallel SpGEMM, we use the sets $\mathcal{X}$ and $\mathcal{R}$ for parallelizations that contain the e$\mathcal{X}$pand type and $\mathcal{R}$educe type of communication operations, respectively. Each element of these sets is a two-tuple in which the first entry of the tuple is the data being communicated and the second entry is the set of processors that communicate this data. The elements of $\mathcal{X}$ and $\mathcal{R}$ are used to form the communication hypergraphs.

The communication hypergraph models rely on the partitioning information obtained with the computational partitioning models and they differ in their formation with respect to the computational model utilized. In order to avoid confusion between the computational partitioning models and the communication hypergraph models, we respectively use "nodes" and "processor nets" to refer to the vertices and the nets of the communication hypergraphs.

4.2 Outer-Product–Parallel (OP) SpGEMM

The responsibility of a communication operation on $c_{i,j}$ in OP is originally assigned to processor P_k if the vertex representing $c_{i,j}$ is in part $\mathcal{V}_k$ as the result of partitioning the computational model $\mathcal{H}_{OP}$ or $\mathcal{G}_{OP}$ (Section 3.4). The proposed communication hypergraph model presents an alternative way for the assignment of communication operations on nonzeros of C with the reduction of the latency cost being the primary objective.

In OP, the communication operations are denoted with $\mathcal{R}_{OP}$ and they are reduce-type operations that are performed on nonzeros of C. Hence, $|\mathcal{R}_{OP}| \leq nnz(C)$, as not all nonzeros of C may necessitate communication. An element in $\mathcal{R}_{OP}$ is given by the tuple $(c_{i,j}, \mathcal{P}_{i,j})$, where $\mathcal{P}_{i,j}$ is the set of processors that participate in communicating $c_{i,j}$, and $|\mathcal{P}_{i,j}| > 1$ since otherwise no communication is needed.

In the computational hypergraph model for OP, the set of communication operations is determined from the set of *external nets*; hence, $|\mathcal{R}_{OP}|$ is equal to the number of external nets in a partition of $\mathcal{H}_{OP}$. Utilizing the partition $\Pi_{OP} = \{\mathcal{V}_1, \ldots, \mathcal{V}_K\}$ of $\mathcal{H}_{OP} = \{\mathcal{V}_{cr}^{AB} \cup \mathcal{V}_z^C, \mathcal{N}_z^C\}$, the communication operations and the processors that participate in reducing $c_{i,j}$ are formed as follows:

$$\mathcal{P}_{i,j} = \left\{P_k : v_x \in Pins(n_{i,j}) \wedge v_x \in \mathcal{V}_{cr}^{AB} \wedge v_x \in \mathcal{V}_k\right\}.$$

Recall that v_x represents column x of A and row x of B and the outer product of them, $a_{*,x} \otimes b_{x,*}$, and $n_{i,j}$ is the net that captures the dependency on $c_{i,j}$.

In the computational bipartite graph model, the set of communication operations is given by the set of *boundary vertices* belonging to $\mathcal{V}_z^C$; hence, $|\mathcal{R}_{OP}|$ is equal to the number of boundary vertices of this set in a partition of $\mathcal{G}_{OP}$. For the partition Π_{OP} on $\mathcal{G}_{OP} = \{\mathcal{V}_{cr}^{AB} \cup \mathcal{V}_z^C, \mathcal{E}_{z'}^C\}$,

$$\mathcal{P}_{i,j} = \left\{P_k : v_x \in Adj(v_{i,j}) \wedge v_x \in \mathcal{V}_{cr}^{AB} \wedge v_x \in \mathcal{V}_k\right\}.$$

That is, to compute the final value of $c_{i,j}$, the processor responsible for $c_{i,j}$ receives a partial result from each $P_k \in \mathcal{P}_{i,j}$.

$\mathcal{R}_{OP}$ is then used to form the communication hypergraph $\mathcal{H}_{OP}^{COM} = \{\mathcal{U}, \mathcal{N}\}$ for the outer-product–parallel SpGEMM. $\mathcal{U}$ contains a node $u_{i,j}$ for each $(c_{i,j}, \mathcal{P}_{i,j}) \in \mathcal{R}_{OP}$ and $\mathcal{N}$ contains a processor net p_k for each processor P_k. p_k connects $u_{i,j}$ if P_k produces a partial result for $c_{i,j}$:

$$Pins(p_k) = \{u_{i,j} : u_{i,j} \in \mathcal{U} \wedge P_k \in \mathcal{P}_{i,j}\} \cup \{u_{f_k}\}.$$

This processor net also connects another node u_{f_k}, which is referred to as a *fixed node*. Fixed nodes are included to later decode the assignment of communication operations to processors and u_{f_k} is fixed to part $\mathcal{U}_k$ in the partitioning, for $k = 1, \ldots, K$. All nodes have unit weights, which is the preferred case when the communication operations are of reduce type (Uçar and Aykanat 2004).

Note that if we had utilized nonunit weights, we would have balanced the load on received matrix elements, not sent, which would not be very useful. Processor nets are assigned unit costs.

4.3 Inner-Product–Parallel (IP) SpGEMM

The responsibility of a communication operation on $b_{*,j}$ in IP is originally assigned to processor P_k if the vertex representing $b_{*,j}$ is in part $\mathcal{V}_k$ as the result of partitioning the computational models $\mathcal{H}_{IP}$ or $\mathcal{G}_{IP}$ (Section 3.4). As in OP, the purpose of the proposed communication hypergraph model for IP is to reduce the latency cost.

In IP, the communication operations are denoted with $\mathcal{X}_{IP}$ and they are expand type of operations that are performed on columns of B (specific nonzeros of these columns). Hence, $|\mathcal{X}_{IP}| \leq ncols(B)$, as not all columns of B may necessitate communication. An element in $\mathcal{X}_{IP}$ is given by the tuple $(b_{*,j}, \mathcal{P}_j)$, where $\mathcal{P}_j$ is the set of processors that participate in communicating nonzeros of $b_{*,j}$, and $|\mathcal{P}_j| > 1$.

In the computational hypergraph model for IP, $|\mathcal{X}_{IP}|$ is smaller than or equal to the number of external nets in a partition of $\mathcal{H}_{IP}$ as the nets in $\mathcal{H}_{IP}$ represent the nonzeros of B and the communication operations are defined on columns of B. Utilizing the partition $\Pi_{IP} = \{\mathcal{V}_1, \ldots, \mathcal{V}_K\}$ of $\mathcal{H}_{IP} = \{\mathcal{V}_{rr}^{AC} \cup \mathcal{V}_c^B, \mathcal{N}_z^B\}$, the communication operations and the processors that participate in expanding nonzeros of $b_{*,j}$ are formed as follows:

$$\mathcal{P}_j = \left\{P_k : n_{i,j} \in Nets(v_j), v_x \in Pins(n_{i,j}) \wedge v_x \in \mathcal{V}_{rr}^{AC} \wedge v_x \in \mathcal{V}_k\right\}.$$

Recall that v_x represents row x of A and its multiplication with B, $a_{x,*}B$, v_j represents column j of B, and $n_{i,j}$ is the net that captures the dependency on $b_{i,j}$.

In the computational bipartite graph model, $|\mathcal{X}_{IP}|$ is equal to the number of boundary vertices belonging to $\mathcal{V}_c^B$ in a partition of $\mathcal{G}_{IP}$. For the partition Π_{IP} of $\mathcal{G}_{IP} = \{\mathcal{V}_{rr}^{AC} \cup \mathcal{V}_c^B, \mathcal{E}_z^C\}$,

$$\mathcal{P}_j = \left\{P_k : v_x \in Adj(v_j) \wedge v_x \in \mathcal{V}_{rr}^{AC} \wedge v_x \in \mathcal{V}_k\right\}.$$

That is, the processor responsible for $b_{*,j}$ sends certain or all nonzeros of this column to each $P_k \in \mathcal{P}_j$ for the inner-product computations.

$\mathcal{X}_{IP}$ is then used to form the communication hypergraph $\mathcal{H}_{IP}^{COM} = \{\mathcal{U}, \mathcal{N}\}$ for the inner-product–parallel SpGEMM. $\mathcal{U}$ contains a node u_j for each $(b_{*,j}, \mathcal{P}_j) \in \mathcal{X}_{IP}$ and $\mathcal{N}$ contains a processor net p_k for each processor P_k. p_k connects u_j if P_k needs at least one nonzero from $b_{*,j}$:

$$Pins(p_k) = \{u_j : u_j \in \mathcal{U} \wedge P_k \in \mathcal{P}_j\} \cup \{u_{f_k}\}.$$

This processor net also connects another node u_{f_k} (fixed node), which is included to later decode the assignment of communication operations and is fixed to part $\mathcal{U}_k$ in the partitioning. The weight of u_j is equal to the volume incurred in communicating $b_{*,j}$ and it is given from the partition on $\mathcal{H}_{IP}$,

$$w(u_j) = \sum_{n_{i,j} \in Nets(v_j)} \lambda(n_{i,j}) - 1.$$

It is not possible to directly form the vertex weights using the partition on $\mathcal{G}_{IP}$. For this reason, the matrices in SpGEMM are used to form the vertex weights. Processor nets are assigned unit costs.

4.4 Row-by-Row-Product–Parallel (RRP) SpGEMM

Originally, for the computational model $\mathcal{H}_{RRP}$, the responsibility of a communication operation on $b_{i,*}$ is assigned to a processor corresponding to one of the parts connected by n_i (note that n_i represents $b_{i,*}$), i.e., a processor corresponding to one of the parts in $\Lambda(n_i)$ (Section 3.4). For the computational model $\mathcal{G}_{RRP}$, the responsibility is assigned to processor P_k if the vertex representing

$b_{i,*}$ is in part $\mathcal{V}_k$ (Section 3.4). As in the two previous communication hypergraph models, the purpose of the proposed communication hypergraph model for RRP is also to reduce the latency cost.

In RRP, the communication operations are denoted with $\mathcal{X}_{RRP}$ and they are expand-type operations that are performed on rows of B. Hence, $|\mathcal{X}_{RRP}| \leq nrows(B)$, as not all rows of B may necessitate communication. An element in $\mathcal{X}_{RRP}$ is given by the tuple $(b_{i,*}, \mathcal{P}_i)$, where $\mathcal{P}_i$ is the set of processors that participate in communicating $b_{i,*}$, and $|\mathcal{P}_i| > 1$.

In the computational hypergraph model for RRP, $|\mathcal{X}_{RRP}|$ is equal to the number of external nets in a partition of $\mathcal{H}_{RRP}$. Utilizing the partition $\Pi_{RRP} = \{\mathcal{V}_1, \ldots, \mathcal{V}_K\}$ of $\mathcal{H}_{RRP} = \{\mathcal{V}_{rr}^{AC}, \mathcal{N}_r^B\}$, the communication operations and the processors that participate in expanding $b_{i,*}$ are formed as follows:

$$\mathcal{P}_i = \{P_k : v_x \in Pins(n_i) \wedge v_x \in \mathcal{V}_k\}.$$

Recall that v_x represents row x of A and its multiplication with B, $a_{x,*}B$, and n_i is the net that captures the dependency on row i of B.

In the computational bipartite graph model, $|\mathcal{X}_{RRP}|$ is equal to the number of boundary vertices belonging to $\mathcal{V}_r^B$ in a partition of $\mathcal{G}_{RRP}$. For the partition Π_{RRP} of $\mathcal{G}_{RRP} = \{\mathcal{V}_{rr}^{AC} \cup \mathcal{V}_r^B, \mathcal{E}_z^A\}$,

$$\mathcal{P}_i = \{P_k : v_x \in Adj(v_i) \wedge v_x \in \mathcal{V}_k\}.$$

That is, the processor responsible for $b_{i,*}$ sends this row to each $P_k \in \mathcal{P}_i$ to be multiplied with a nonzero in a specific row of A.

$\mathcal{X}_{RRP}$ is then used to form the communication hypergraph $\mathcal{H}_{RRP}^{COM} = \{\mathcal{U}, \mathcal{N}\}$ for the row-by-row-product–parallel SpGEMM. $\mathcal{U}$ contains a node u_i for each tuple $(b_{i,*}, \mathcal{P}_i) \in \mathcal{X}_{RRP}$ and $\mathcal{N}$ contains a processor net p_k for each processor P_k. p_k connects u_i if P_k needs row $b_{i,*}$:

$$Pins(p_k) = \{u_i : u_i \in \mathcal{U} \wedge P_k \in \mathcal{P}_i\} \cup \{u_{f_k}\}.$$

This processor net also connects another node u_{f_k} (fixed node), which is included to later decode the assignment of communication operations and is fixed to part $\mathcal{U}_k$ in the partitioning. The weight of u_i is equal to the volume incurred in communicating $b_{i,*}$ and it is determined from the partitions on $\mathcal{H}_{RRP}$ and $\mathcal{G}_{RRP}$ as

$$w(u_i) = c(n_i)(\lambda(n_i) - 1) \quad \text{and} \quad w(u_i) = nnz(b_{i,*}) \times (|\{P_k : v_x \in Adj(v_i) \wedge v_x \in \mathcal{V}_k\}| - 1),$$

respectively. Processor nets are assigned unit costs.

4.5 Decoding Partitions

We now describe how to decode the partitions obtained as a result of partitioning the communication hypergraphs for OP, IP, and RRP in order to determine the assignment of communication operations.

OP. Obtaining a K-way partition $\Pi_{OP}^{COM} = \{\mathcal{U}_1, \ldots, \mathcal{U}_K\}$ of $\mathcal{H}_{OP}^{COM}$ induces a distribution of communication operations, where the responsibilities of reduce operations corresponding to the nodes in $\mathcal{U}_k$ are assigned to processor P_k. A processor net p_k signifies that P_k receives a message that contains partial results for nonzeros of C from the processors corresponding to the parts in $\Lambda(p_k) - \{\mathcal{U}_k\}$. Note that $\mathcal{U}_k \in \Lambda(p_k)$ because of the fixed node u_{f_k} included in the partitioning.

IP and RRP. Obtaining a K-way partition $\Pi_{IP}^{COM} = \{\mathcal{U}_1, \ldots, \mathcal{U}_K\}$ of $\mathcal{H}_{IP}^{COM}$ and $\Pi_{RRP}^{COM} = \{\mathcal{U}_1, \ldots, \mathcal{U}_K\}$ of $\mathcal{H}_{RRP}^{COM}$ induces a distribution of communication operations, where the responsibilities of expand operations corresponding to the nodes in $\mathcal{U}_k$ are assigned to processor P_k in both schemes. In IP, a processor net p_k signifies that P_k sends a message that contains nonzeros of columns of B to the processors corresponding to the parts in $\Lambda(p_k) - \{\mathcal{U}_k\}$. In RRP, it is the same except this message contains the rows of B. Again, note that $\mathcal{U}_k \in \Lambda(p_k)$ because of the fixed node u_{f_k} included in the partitioning.

Partitioning Constraint and Objective. In partitioning all three communication hypergraph models, the partitioning objective of minimizing cutsize corresponds to minimizing the total message count, whereas the partitioning constraint of maintaining balance relates to balancing the message volume loads of processors.

5 EXPERIMENTS

5.1 Setup

The hypergraph models described in Sections 3.1.1, 3.2.1, and 3.3.1 are partitioned using PaToH (Çatalyürek and Aykanat 1999b) and the bipartite graph models described in Sections 3.1.2, 3.2.2, and 3.3.2 are partitioned using MeTiS (Karypis and Kumar 1999). We also used parallel graph partitioner ParMeTiS (Karypis and Kumar 1998) to further reduce the partitioning overhead of the bipartite graph models. The maximum allowed imbalance threshold for all partitioners is set to 10%. Since the partitioners contain randomization, we partition the graphs and hypergraphs three times with different seeds and report the averages.

The communication hypergraphs described in Section 4 are partitioned using the direct K-way hypergraph partitioner kPaToH (Aykanat et al. 2008). We preferred kPaToH instead of PaToH for partitioning the communication hypergraphs as these hypergraphs contain fixed vertices and kPaToH utilizes a matching algorithm for assigning fixed nodes to parts in the initial partitioning phase, while PaToH performs the same task in a random manner.

All parallel SpGEMM algorithms are implemented in C and they utilize MPI for communication. Local SpGEMM computations are implemented using Gustavson's SpGEMM algorithm (Gustavson 1978). The sequential SpGEMM implementation uses Gustavson's algorithm as well. The sequential times are used to obtain the speedups of the parallel algorithms. We used our own sequential implementation of SpGEMM rather than the sequential implementation of CSparse (Davis 2006) since we found ours to be faster. The runtimes of SpGEMM algorithms are the averages of 10 runs performed after a warmup phase of three runs.

The experiments are performed on a BlueGene/Q system. A node in this system consists of 16 PowerPC A2 cores and 16GB RAM. Cores are clocked at 1.6GHz. The nodes are connected with a 5D torus network with a bandwidth capacity of 40GBps. BlueGene/Q's MPI implementation is based on MPICH2.

5.2 Datasets

We evaluate three categories of SpGEMM: $C = AA^T$, $C = AA$, and $C = AB$. Table 3 displays the properties of the input and output matrices in these categories.

For $C = AA^T$, we test 10 LP constraint matrices from the UFL sparse matrix collection (Davis and Hu 2011). For $C = AA$, we test 25 instances, 23 of which are again from the UFL sparse matrix collection. The remaining two instances `cp2k-h2o-e6` and `cp2k-h2o-.5e7` are obtained from H_2O simulations performed by CP2K (2016), which involve parallel SpGEMM in order to calculate the sign of a given sparse matrix.

For $C = AB$, we test 10 instances from the UFL sparse matrix collection. Two instances involving `amazon0302` and `amazon0312` matrices are used for item-to-item collaborative filtering in recommendation systems (Linden et al. 2003). Here, A represents the similarity between items and B represents the users' preferences. To generate B, we utilize a Zipf distribution (with exponent set to 3.0) to determine the item preferences and a uniform distribution to determine the users that prefer a specific item. The multiplication of these two matrices gives the candidate items to be

Table 3. Properties of Input and Output Matrices

	Input Matrices							Output Matrix
	Number of			Nnz in Row		Nnz in Column		
Matrix	Rows	Columns	Nonzeros	Avg	Max	Avg	Max	Nnz
$C = AA^T$								
cont11_l	1,468,599	1,961,394	5,382,999	4	5	3	7	18,064,261
fome13	48,568	97,840	285,056	6	228	3	14	658,136
fome21	67,748	216,350	465,294	7	96	2	3	640,240
fxm3_16	41,340	85,575	392,252	9	57	5	36	765,526
fxm4_6	22,400	47,185	265,442	12	57	6	24	526,536
pds-30	49,944	158,489	340,635	7	96	2	3	468,266
pds-40	66,844	217,531	466,800	7	96	2	3	637,867
sgpf5y6	246,077	312,540	831,976	3	61	3	12	2,776,645
watson_1	201,155	386,992	1,055,093	5	93	3	9	1,937,163
watson_2	352,013	677,224	1,846,391	5	93	3	15	3,390,279
$C = AA$								
2cubes_sphere	101,492	101,492	1,647,264	16	31	16	31	8,974,526
598a	110,971	110,971	1,483,868	13	26	13	26	7,104,683
bcsstk32	44,609	44,609	2,014,701	45	216	45	216	6,819,653
bfly	49,152	49,152	196,608	4	4	4	4	540,672
brack2	62,631	62,631	733,118	12	32	12	32	3,944,481
cca	49,152	49,152	139,264	3	3	3	3	311,296
cp2k-h2o-.5e7	279,936	279,936	3,816,315	14	24	14	27	17,052,039
cp2k-h2o-e6	279,936	279,936	2,349,567	8	20	8	20	7,846,956
cvxbqp1	50,000	50,000	349,968	7	9	7	9	1,099,432
fe_rotor	99,617	99,617	1,324,862	13	125	13	125	7,175,441
fe_tooth	78,136	78,136	905,182	12	39	12	39	4,914,718
finance256	37,376	37,376	298,496	8	55	8	55	2,297,728
majorbasis	160,000	160,000	1,750,416	11	11	11	18	8,243,392
mario002	389,874	389,874	2,101,242	5	7	5	7	6,449,598
mark3jac140	64,089	64,089	399,735	6	44	6	47	1,817,705
oilpan	73,752	73,752	3,597,188	49	70	49	70	11,609,864
onera_dual	85,567	85,567	419,201	5	5	5	5	1,279,793
pkustk03	63,336	63,336	3,130,416	49	90	49	90	8,924,832
poisson3Da	13,514	13,514	352,762	26	110	26	110	2,957,530
raefsky3	21,200	21,200	1,488,768	70	80	70	80	4,053,376
srb1	54,924	54,924	2,962,152	54	270	54	270	8,388,936
tandem_dual	94,069	94,069	460,493	5	5	5	5	1,420,681
tmt_sym	726,713	726,713	5,080,961	7	9	7	9	14,503,181
torso2	115,967	115,967	1,033,473	9	10	9	10	2,858,293
wave	156,317	156,317	2,118,662	14	44	14	44	10,973,239
$C = AB$								
amazon0302 (*A*)	262,111	262,111	1,234,877	5	5	5	420	2,717,029
amazon0302-user (*B*)	262,111	50,000	576,413	2	302	12	27	
amazon0312 (*A*)	400,727	400,727	3,200,440	8	10	8	2,747	7,031,743
amazon0312-user (*B*)	400,727	50,000	882,813	2	1,675	18	38	

(Continued)

Table 3. Continued

boneS01 (A)	127,224	127,224	6,715,152	53	81	53	81	1,161,045
boneS01.P (B)	127,224	2,394	470,235	4	10	196	513	
cfd2 (A)	123,440	123,440	3,087,898	25	30	25	30	1,374,012
cfd2.P (B)	123,440	4,825	528,769	4	10	110	181	
denormal (A)	89,400	89,400	1,156,224	13	13	13	13	560,020
denormal.P (B)	89,400	6,000	278,565	3	4	46	55	
finance256 (A)	37,376	37,376	298,496	8	55	8	55	487,583
finance256.P (B)	37,376	2,432	120,831	3	20	50	128	
offshore (A)	259,789	259,789	4,242,673	16	31	16	31	3,558,234
offshore.P (B)	259,789	9,893	1,159,999	4	13	117	221	
s3dkq4m2 (A)	90,449	90,449	4,820,891	53	54	53	54	486,853
s3dkq4m2.P (B)	90,449	1,734	249,749	3	4	144	150	
shipsec5 (A)	179,860	179,860	10,113,096	56	126	56	126	1,273,553
shipsec5.P (B)	179,860	2,959	541,099	3	13	183	456	
thermomech_dK (A)	204,316	204,316	2,846,228	14	20	14	20	7,874,148
thermomech_dM (B)	204,316	204,316	1,423,116	7	10	7	10	

Nnz: number of nonzeros.

recommended to each user. Another application that utilizes SpGEMM form $C = AB$ is the setup phase of Algebraic Multigrid (AMG) (Bell et al. 2012). The Galerkin product *RAP* in the setup phase is a costly operation that necessitates two SpGEMM sof type $C = AB$. For our experiments, we only consider the parallelization of interpolation, i.e., *AP*. Using the tool[†] provided by the authors of that work, we generated the interpolation operators for seven matrices (`boneS01`, `cfd2`, `denormal`, `finance256`, `offshore`, `s3dkq4m2`, `shipsec5`). A suffix ".P" in the table indicates the operator matrix. The last instance in this category contains `thermomech_dK` and `thermomech_dM`, which are conformable for multiplication.

5.3 Performance Comparison of Parallel SpGEMM Algorithms

In Table 4, we compare the performance of parallel SpGEMM algorithms OP, IP, and RRP in terms of communication cost metrics and obtained speedups for $K = 512$ and 1024. The two measured cost metrics are total message volume in terms of kilo words and average number of messages sent by a processor (or average message count). The results are grouped separately for three categories of SpGEMM. We present the detailed results for each matrix as well as the averages (geometric means) over the three categories. A bold value indicates the best value attained in the respective performance metric for a given matrix and K value. The results in the table are obtained with the hypergraph models. The average values obtained by the algorithms on 1024 processors are also illustrated with bar charts in Figure 4 to provide a visual comparison. We compare the performance of bipartite graph and hypergraph models in the following section as the focus of this section is the comparison of the parallel SpGEMM algorithms among themselves.

In the $C = AA^T$ category, OP attains significantly less message volume, achieving 76% to 77% less message volume than IP and RRP on average for all K. This can be attributed to the fact that fat and short LP constraint matrices are amenable to better partitioning along the longer dimension, which is the case for OP. In terms of average message count, OP and RRP achieve close performance, while IP incurs 37% to 49% more messages than these two on average. In this category, OP obtains the highest speedups in all test instances due to its significantly lower message volume. However,

[†]https://github.com/pyamg/pyamg.

Table 4. Performance Comparison of OP, IP, and RRP Using Hypergraph Models

	K=512									K=1024								
	Msg. vol. (10^3)			Avg. msg. count			Speedup			Msg. vol. (10^3)			Avg. msg. count			Speedup		
Matrix	OP	IP	RRP	OP	IP	RRP	OP	IP	RRP	OP	IP	RRP	OP	IP	RRP	OP	IP	RRP
$C = AA^T$																		
cont11_l	**247**	414	412	5.8	6.4	**5.6**	**436**	409	410	**350**	578	586	5.8	6.4	**5.6**	**813**	735	749
fome13	**59**	261	264	**27.8**	44.2	32.1	**196**	135	140	**73**	315	316	**27.6**	52.2	31.5	**210**	140	175
fome21	**38**	152	152	**14.3**	26.7	17.3	**217**	123	136	**50**	203	201	**14.0**	26.6	16.3	**220**	148	191
fxm3_16	**61**	208	200	6.2	7.9	**4.7**	**179**	100	139	**179**	406	383	6.1	7.6	**4.7**	**206**	116	177
fxm4_6	**86**	221	227	**4.6**	8.0	5.2	**140**	100	111	**190**	416	416	**5.2**	9.1	5.8	**172**	106	141
pds-30	**31**	126	128	**14.8**	27.4	17.7	**177**	107	118	**41**	169	168	**13.6**	25.5	15.8	**182**	107	154
pds-40	**39**	153	153	**15.6**	27.6	18.8	**204**	132	134	**50**	205	204	**14.4**	26.6	16.5	**236**	143	181
sgpf5y6	**27**	341	330	9.0	14.3	**7.7**	**228**	113	150	**43**	521	518	8.9	17.0	**8.5**	**279**	103	170
watson_1	**28**	214	220	4.3	2.8	**2.6**	**311**	236	224	**43**	325	327	5.1	3.8	**3.3**	**395**	312	298
watson_2	**32**	174	192	3.9	3.6	**3.3**	**386**	282	253	**51**	424	417	4.6	4.0	**3.6**	**532**	397	375
Average	**49**	212	214	8.6	11.8	**8.4**	**232**	153	166	**78**	330	328	8.8	12.7	**8.5**	**284**	181	225
$C = AA$																		
2cubes_sphere	2,580	**2,223**	2,234	16.1	20.5	**15.6**	**380**	336	355	3,353	**2,964**	2,970	17.1	23.5	**16.5**	345	536	**588**
598a	1,996	**1,498**	1,510	12.6	14.7	**11.9**	335	348	**358**	2,675	**2,082**	2,089	13.9	16.9	**12.8**	572	573	**598**
bcsstk32	3,676	**3,673**	3,681	8.4	12.1	**7.1**	254	307	**313**	5,973	5,795	**5,745**	9.3	14.7	**7.6**	270	487	**509**
bfly	**28**	111	112	**31.5**	51.7	31.9	105	86	**109**	**32**	137	137	**21.0**	38.6	21.2	130	108	**138**
brack2	1,096	**844**	847	11.9	15.8	**10.1**	208	206	**235**	1,552	**1,223**	1,232	13.7	18.0	**11.2**	251	231	**297**
cca	**18**	55	55	**19.8**	32.4	19.9	81	74	**88**	**24**	70	70	**14.1**	24.5	14.4	88	79	**108**
cp2k-h2o-.5e7	**1,607**	2,282	2,237	14.3	17.1	**14.3**	**374**	355	362	**2,092**	2,954	2,903	**13.5**	17.4	13.6	410	622	**648**
cp2k-h2o-e6	**367**	704	695	15.5	15.0	**12.4**	**373**	357	360	**478**	919	919	14.4	14.5	**11.6**	545	588	**606**
cvxbqp1	**176**	197	198	9.2	11.4	**8.1**	**172**	157	172	**249**	273	277	8.4	10.6	**7.2**	**235**	206	223
fe_rotor	1,871	**1,470**	1,478	15.3	20.7	**13.6**	286	257	**290**	2,566	**2,084**	2,098	16.6	23.8	**14.5**	**441**	349	433
fe_tooth	1,267	**987**	992	12.6	15.9	**11.1**	226	221	**250**	1,764	**1,391**	1,406	14.4	18.3	**11.9**	297	296	**346**
finance256	**425**	474	475	15.2	12.8	**9.9**	153	178	**190**	**541**	674	678	17.7	20.7	**13.0**	174	189	**221**
majorbasis	881	572	**465**	5.9	6.2	**3.6**	335	343	**348**	1,283	868	**733**	6.2	6.8	**3.9**	554	536	**567**
mario002	**187**	266	264	**4.7**	5.7	5.1	**409**	396	391	**269**	380	379	**4.7**	5.8	5.3	**727**	680	672
mark3jac140	**185**	398	348	21.3	30.4	**19.5**	153	130	**157**	**262**	525	461	20.9	35.6	**19.8**	**202**	156	185
oilpan	**3,596**	4,393	4,307	6.1	8.1	**5.6**	**382**	362	374	**5,514**	6,670	6,515	6.7	9.9	**5.9**	**695**	666	688
onera_dual	**101**	204	205	10.5	12.1	**9.9**	**193**	173	184	**136**	274	275	10.5	12.6	**9.9**	**232**	215	225
pkustk03	**3,912**	4,328	4,321	7.5	10.7	**6.4**	**365**	351	**365**	**6,128**	6,654	6,607	8.4	12.6	**6.6**	642	608	**652**
poisson3Da	2,452	1,774	**1,744**	23.0	39.9	**20.5**	**232**	160	230	3,304	2,606	**2,514**	24.5	42.0	**21.7**	274	180	**292**
raefsky3	**4,233**	4,694	4,587	7.5	10.2	**6.1**	306	323	**338**	**7,251**	7,782	7,564	9.4	12.4	**6.9**	488	470	**526**
srb1	**4,182**	4,463	4,400	7.0	8.8	**5.6**	249	360	**370**	**6,216**	6,744	6,655	7.6	10.7	**5.8**	256	636	**666**
tandem_dual	**104**	220	220	10.7	12.1	**9.8**	**200**	179	190	**138**	293	295	10.7	12.7	**9.9**	**262**	225	246
tmt_sym	**548**	559	557	5.4	5.7	**5.0**	**445**	442	443	**782**	802	798	5.6	5.9	**5.0**	**827**	809	803
torso2	342	**300**	318	5.4	5.6	**4.7**	299	313	**327**	501	**447**	465	5.6	6.0	**5.0**	**487**	479	482
wave	2,830	**2,112**	2,120	14.9	18.9	**14.1**	290	271	**299**	3,719	**2,870**	2,881	15.8	20.4	**14.6**	354	417	**476**
Average	**696**	829	818	11.0	13.9	**9.7**	252	243	**264**	**971**	1,172	1,156	11.3	15.0	**9.8**	340	351	**394**
$C = AB$																		
amazon0302	**0**	812	216	**0.0**	461.0	70.9	**359**	30	133	**0**	841	250	**0.0**	517.4	55.5	**684**	19	213
amazon0312	**0**	1564	662	**0.0**	488.9	124.9	**310**	63	128	**0**	1,681	784	**0.0**	692.5	114.3	**668**	26	149
boneS01	**562**	639	581	**8.9**	12.4	9.1	**365**	352	354	921	951	**865**	10.1	13.9	**9.6**	562	548	**589**
cfd2	643	645	**552**	10.6	12.3	**10.4**	**301**	296	299	891	926	**795**	**11.7**	14.6	11.9	**447**	412	427
denormal	**131**	168	149	5.4	6.1	**5.3**	252	**276**	272	**192**	246	216	5.7	6.5	**5.5**	**384**	376	362
finance256	250	211	**184**	13.7	14.0	**10.6**	104	99	**119**	330	295	**259**	18.0	20.1	**13.3**	112	87	**117**
offshore	1,152	1,147	**1,013**	14.2	19.9	**13.8**	**462**	374	386	1,544	1,571	**1,376**	15.6	22.3	**14.6**	**755**	584	651
s3dkq4m2	**210**	280	239	**3.9**	6.7	6.0	360	**366**	357	**327**	437	374	**4.0**	7.7	6.5	**576**	569	574
shipsec5	**474**	583	498	7.8	10.0	**7.6**	**382**	373	377	**732**	851	739	8.2	11.2	**7.8**	616	651	**674**
thermomech_dK	550	**404**	406	**5.3**	5.7	5.3	347	**420**	415	806	**591**	595	5.5	5.9	**5.4**	654	**756**	750
Average	406	519	**379**	**8.0**	21.6	13.2	**305**	202	257	594	707	**529**	**8.7**	25.3	13.6	**494**	239	381

***bold value**: the best value attained in the respective performance metric for a given matrix and a K value.*

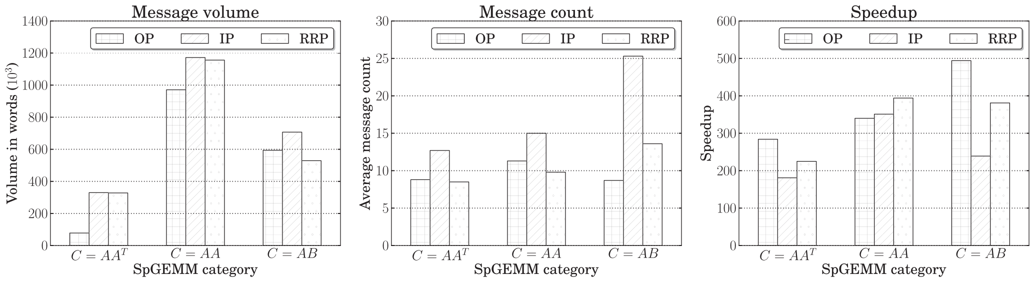


Fig. 4. Comparison of parallel SpGEMM algorithms for 1024 processors. The graphs accompany the values in Table 4. For message volume and count, the lower the better, whereas for speedup, the higher the better.

with increasing K, the speedup performance of RRP gets closer to that of OP due to the increased importance of latency. For example, at $K = 512$, OP achieves 40% better speedup than RRP on average, while at $K = 1024$, this performance gap reduces to 26%. For a visual comparison of OP, IP, and RRP on 1024 processors in these metrics of interest, see Figure 4.

In the $C=AA$ category, IP and RRP obtain very close total message volumes, where OP performs better than these two by obtaining 15% to 17% less message volume on average. RRP achieves the lowest message count, obtaining 12% to 13% and 30% to 35% fewer messages on average than OP and IP, respectively. In this category, RRP obtains the highest speedups, which is closely trailed by OP: out of 50 test instances, RRP obtains the highest speedups in 30 of them and OP in 20 of them, while IP in none of them. The better performance of OP and RRP in this category can be attributed to their lower message counts compared to IP. Again, the gap in speedup performances increases in favor of RRP when K is increased from 512 to 1024. Observe that the message volumes of the SpGEMM algorithms in the $C=AA$ category are significantly higher than those in the $C=AA^T$ category. This can partially be attributed to the fact that the matrices in the $C=AA^T$ category have relatively fewer nonzeros than the matrices in the $C=AA$ category as seen in Table 3.

In the $C=AB$ category, in both amazon instances, OP has the best performance in both communication cost metrics, whereas IP has the worst. The inferior performance of IP is because there are more rows than columns in B (see Table 3), causing the columns of B to be denser compared to the rows of B, and thus making the partitioning process more difficult for IP. This consequently incurs a high message count. OP incurs no communication in these two instances (zero message volume and count) as the large number of rows in B have a very small number of nonzeros, which reduces the probability of multiple processors contributing to the same nonzero of C. The better performance of OP in these two metrics is reflected in the speedups for amazon instances. In seven AMG instances, OP and RRP perform close in terms of parallel SpGEMM time and IP performs the worst. For the thermomech instance, IP and RRP obtain better speedups than OP due to their relatively lower message volume. Overall, the best speedup values are obtained by OP, followed by RRP. If the extraordinary performance of OP in amazon instances is put aside, it can be said that OP and RRP are equally preferable to IP in this category.

5.4 Performance Comparison of Hypergraph and Bipartite Graph Models

We compare the hypergraph and bipartite graph models in terms of communication cost metrics, parallel SpGEMM times obtained using these models, and partitioning overhead. The point of this section is to justify the claim that although the bipartite graph models may perform slightly worse in communication cost metrics compared to their hypergraph counterparts, they achieve comparable speedup performance with a significantly lower partitioning overhead. The results

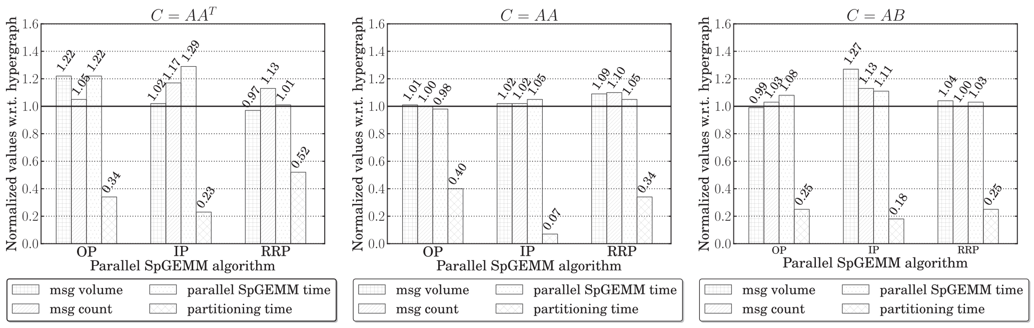


Fig. 5. The communication statistics, parallel SpGEMM times, and partitioning times of the bipartite graph models normalized with respect to those of the hypergraph models, for K=1024.

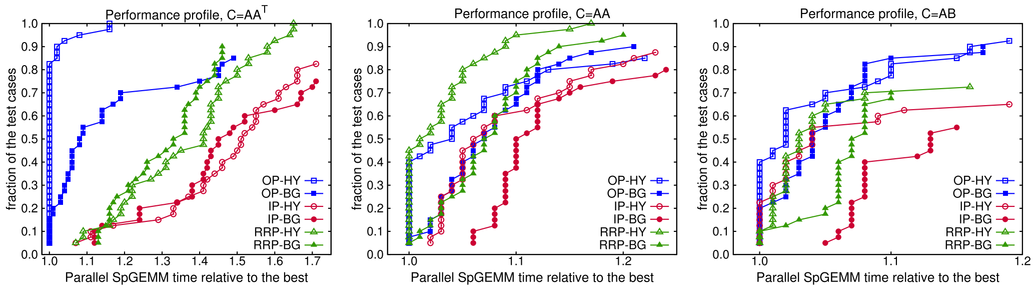


Fig. 6. Performance profiles for the parallel SpGEMM times obtained by the hypergraph (indicated by "HY") and the bipartite graph (indicated by "BG") models for OP, IP, and RRP.

obtained by the bipartite graph models are normalized with respect to those by the hypergraph models, and they are displayed in Figure 5 for categories $C = AA^T$, $C = AA$, and $C = AB$. We present the results for only K=1024 as the results for other K values are similar. In each bar chart in Figure 5, there is a separate bar group for each of OP, IP, and RRP. The four bars in each bar group respectively represent the total message volume, total message count, parallel SpGEMM time, and partitioning time of the respective bipartite graph model, all of which are normalized with respect to those of the hypergraph model; i.e., the values obtained by the bipartite graph model for OP are normalized with respect to those of the hypergraph model for OP, and so forth.

For the instances in all categories, the bipartite graph models usually yield a slightly higher message volume than their hypergraph counterparts (see first bar of each bar group), the exceptions being RRP in the $C = AA^T$ category and OP in the $C = AB$ category. The bipartite graph models usually perform worse in this metric since the hypergraph models correctly encapsulate the partitioning objective of minimizing the total message volume. The bipartite graph models also obtain higher message counts compared to the hypergraph models: 0% to 5%, 2% to 17%, and 0% to 13% higher in OP, IP, and RRP, respectively, on average. Figure 5 shows that although generally performing slightly worse in both cost metrics, the bipartite graph models often attain comparable speedup performance (especially for the instances in the $C=AA$ and $C=AB$ category) with respect to their hypergraph counterparts in significantly less partitioning time. The bipartite graph models obtain partitions in 60% to 75%, 77% to 93%, and 48% to 75% less time than the hypergraph models for OP, IP, and RRP, respectively. A more detailed analysis of the parallel SpGEMM times follows.

In Figure 6, we present the performance profiles for the parallel SpGEMM times obtained by the hypergraph and bipartite graph models for OP, IP, and RRP to provide a better comparison.

The performance profiles are proposed in Dolan and Moré (2002) and they are especially useful when the number of compared schemes and/or test instances is high. A point (x, y) in the figure reads as the respective scheme being within the x factor of the best results in y fraction of the test cases. In other words, the closer the performance profile of a scheme to the y-axis, the better it is. In the figure, "HY" stands for the hypergraph model and "BG" stands for the bipartite graph model. A test instance is the parallel SpGEMM time obtained by a partitioning model for a given matrix and a K value. Considering three values of $K = 256, 512$, and 1024, there are a total of 30, 75, and 30 instances in the $C = AA^T$, $C = AA$, and $C = AB$ categories, respectively. These profiles are in agreement with the arguments in Section 5.3. For the $C = AA^T$ category, the hypergraph model for OP performs the best, followed by the bipartite graph model for OP. For the $C = AA$ category, the hypergraph model for RRP performs the best, followed by the hypergraph model for OP. For the $C = AB$ category, the hypergraph model for OP performs the best.

Figures 5 and 6 show that the bipartite graph models are viable alternatives to their hypergraph counterparts, staying generally within 10% of the hypergraph models' parallel SpGEMM times for the $C = AA$ and $C = AB$ categories, while this value is higher in the $C = AA^T$ category. They are further justified with their lower partitioning overhead.

5.5 Effect of Matrix Density in Partitioning

In this section, we investigate the partitioning performance of bipartite graph and hypergraph models with the matrices that are denser than the ones in Table 3. We only consider the SpGEMM of the form $C = AA$ as this category contains more matrices than the others. Two metrics are of interest: sparseness of A and sparseness of C. Regarding the 25 matrices in Table 3 in category $C = AA$, the average sparseness of A is 0.014% and the average sparseness of C is 0.055%. The sparseness ratios of the tested 20 denser matrices are given in Table 5. The average sparseness of A and C in these denser matrices are respectively 0.235% and 0.852%, amounting to an increase of $15\times$ to $16\times$ in matrix density compared to the matrices in Table 3. Table 5 presents the message volumes obtained by the partitioning models (the target metric that these models aim to reduce) for three SpGEMM algorithms and $K = 1024$ parts.

The results indicate that the quality of the partitions obtained by the bipartite graph models worsens compared to the hypergraph models for SpGEMM algorithms IP and RRP, while it does not change for OP. For the matrices in Table 3, BG respectively obtains 1%, 2%, and 9% higher volume than HY for OP, IP, and RRP (see Figure 5), whereas for denser matrices in Table 5, these values are 0%, 22%, and 24%. The degradations in RRP and IP are explained by the fact that the flaws of the graph models compared to the hypergraph models increase with increasing granularity of the communicated elements. In RRP, whole B matrix rows are communicated; in IP, the subcolumns of B matrix are communicated; and in OP, only individual partial results for the C matrix elements are communicated. In other words, the highest communication granularity belongs to RRP, followed by IP, and then OP—which is unit. This also explains why the difference between graph and hypergraph models for OP does not change with changing sparseness of the matrices.

5.6 Partitioning Overhead and Amortization

In this section, we first compare the partitioning overheads of SpGEMM algorithms with both hypergraph and bipartite graph models. This comparison is performed on a local system with sequential partitioning. Then, we analyze the amortization of the partitioning overhead. This analysis is performed on the BlueGene/Q system with parallel partitioning and parallel SpGEMM.

Table 6 compares the partitioning times of the hypergraph and bipartite graph models for different numbers of partitions and SpGEMM algorithms. The partitioning is performed sequentially on a local system. We used PaToH for partitioning hypergraphs and MeTiS for partitioning

Table 5. Message Volume Obtained by the Bipartite Graph (BG) and Hypergraph (HY) Models on Denser Matrices for $C = AA$ and 1024 Parts

	Sparsity (%)		Message Volume (10^3)					
			OP		IP		RRP	
Matrix	A	C	HY	BG	HY	BG	HY	BG
bcsstk29	0.316	1.004	3,556	3,507	3,770	4,729	3,672	5,288
bcsstk35	0.159	0.501	5,122	4,505	5,051	5,243	5,008	5,599
bcsstk36	0.215	0.684	4,828	4,343	4,909	4,920	4,797	5,246
crplat2	0.296	0.839	4,420	4,201	4,806	5,183	4,661	5,941
crystk02	0.497	2.013	10,946	11,441	10,111	17,148	9,988	18,193
FEM_3D_thermal1	0.135	0.565	2,422	2,469	1,929	1,982	1,894	1,973
gyro_m	0.113	0.639	1,984	1,977	1,399	1,542	1,372	1,512
igbt3	0.196	0.534	766	703	872	910	816	986
inlet	0.239	0.980	1,890	1,933	1,523	1,964	1,458	1,636
k3plates	0.307	0.843	1,696	1,466	1,826	2,219	1,759	2,333
lhr10	0.204	1.097	1,804	1,795	1,644	2,265	1,551	1,621
lhr14	0.151	0.813	1,797	1,835	1,715	2,637	1,580	1,684
msc23052	0.217	0.686	4,934	4,415	4,938	4,914	4,866	5,230
nmos3	0.112	0.309	770	735	965	837	883	828
olafu	0.389	1.300	7,396	7,855	7,107	10,707	7,032	10,871
pkustk01	0.202	0.769	5,342	5,264	5,495	6,834	5,313	8,492
pkustk02	0.694	2.138	12,673	21,463	10,931	16,960	10,730	15,204
rim	0.199	0.810	6,603	6,339	5,806	9,119	5,690	9,034
ted_AB	0.464	2.491	10,023	10,609	7,401	11,011	7,278	10,339
tube1	0.194	0.543	2,688	2,492	3,090	3,016	2,971	3,098
Average	0.235	0.852	3,444	3,431	3,294	4,020	3,186	3,961
BG/HY	–	–	–	1.00	–	1.22	–	1.24

Table 6. Sequential Partitioning Overheads of the Parallel Algorithms in Seconds

	OP		IP		RRP	
K	HY	BG	HY	BG	HY	BG
256	42.50	9.52	33.97	1.60	7.40	1.22
512	46.80	12.01	38.47	2.67	8.65	1.99
1024	49.90	17.28	42.67	4.84	9.94	3.50

The hypergraph models are indicated by "HY" and the bipartite graph models are indicated by "BG."

bipartite graphs. The obtained times are averaged over all matrices, regardless of the category. Among the compared models, the hypergraph model for OP is the most expensive one, costing around 42 to 50 seconds. Note that this hypergraph model was proposed in Akbudak and Aykanat (2014). The bipartite graph model proposed in this work for OP improves this partitioning time by 65% to 78%. The hypergraph and bipartite graph models for IP and RRP further improve the partitioning time drastically compared to those for OP. Especially the bipartite graph models for IP and RRP are noteworthy, which respectively cost 1.6 to 4.8 and 1.2 to 3.5 seconds. This is about a 15× to 35× improvement in the partitioning time over the recent work by Akbudak and Aykanat

Table 7. Amortization of Parallel Bipartite Graph Partitioning Overheads with Respect to CombBLAS in Terms of Number SpGEMMs

K	OP	IP	RRP
256	322.7	7.4	7.7
1024	397.1	7.3	8.0

(2014). Partitioning the hypergraph model for RRP is faster than partitioning the hypergraph models for OP and IP since the number of nets in RRP is $nrows(B)$, while it is $nnz(C)$ in OP and $nnz(B)$ in IP. Partitioning the bipartite graph model for RRP is faster than partitioning the bipartite graph models for OP and IP as well since the number of edges in RRP is $nnz(A)$, while it is $\#flops/2$ in OP and $nnz(C)$ in IP. RRP is slightly faster than IP since A is generally more sparse than C.

Table 7 displays the number of SpGEMMs required to amortize the partitioning overhead by comparing the bipartite graph models to CombBLAS (Buluç and Gilbert 2011). Note that CombBLAS does not rely on an intelligent partitioning model based on sparsity patterns of matrices. Instead, it uses 2D block distribution of matrices for parallelization. We used ParMeTiS to partition the bipartite graph models in parallel on BlueGene/Q. Then we compared the parallel SpGEMM times obtained by using the partitions produced by the bipartite graph models and those obtained by CombBLAS. The SpGEMM counts in the table are computed according to the formula $T_{\text{ParMeTiS}}/(T_{\text{CombBLAS}} - T_{\text{scheme}})$, where T_{ParMeTiS} is the parallel partitioning time, T_{CombBLAS} is the parallel SpGEMM time attained by CombBLAS, and T_{scheme} is the parallel SpGEMM time attained by using one of the OP, IP, or RRP schemes. CombBLAS works for processor counts that are perfect squares, so there are no results for 512 processors regarding amortization. The results are averaged over all matrices. On 256 processors, OP, IP, and RRP respectively necessitate 322.7, 7.4, and 7.7 SpGEMMs to amortize the cost of partitioning, while on 1024 processors these values are 397.1, 7.3, and 8.0. These values show that parallel SpGEMM can greatly benefit from the proposed partitioning models.

5.7 Effect of Reducing Latency Cost

Table 8 presents the communication statistics and parallel SpGEMM times obtained by further reducing the latency costs of the models via utilizing the communication hypergraphs. Recall that the communication hypergraphs utilize the partitions obtained with the computational partitioning models. In other words, after obtaining a partition with a computational model, we utilize the communication hypergraph on this partition to further reduce the latency cost. The communication statistics include three metrics: total message volume, message volume imbalance on sent matrix elements, and average message count. The values in the table are the normalized values and the normalization is performed as follows: for each partitioning model and the SpGEMM algorithm, the results obtained by further applying the communication hypergraph are normalized with respect to the results obtained by its baseline counterpart in which only the message volume is reduced. In the table, the average normalized values are presented separately for three different categories, for $K = 256$, 512, and 1024. In the $C = AB$ category for OP, we did not utilize the communication hypergraphs for amazon matrices as these matrices had already very low communication overhead when partitioned with the computational hypergraph and bipartite graph models (see Table 4).

Table 8. Communication Statistics for the Communication Hypergraphs

			$C = AA^T$			$C = AA$			$C = AB$		
			K=256	512	1024	256	512	1024	256	512	1024
OP	Total message volume	HY	1.44	1.42	1.38	1.62	1.59	1.52	1.55	1.53	1.48
		BG	1.48	1.45	1.41	1.62	1.59	1.56	1.54	1.52	1.48
	Message volume imbalance	HY	0.94	0.95	0.91	0.91	0.88	0.74	0.84	0.80	0.80
		BG	1.03	1.01	0.99	0.98	0.94	0.90	1.00	0.96	0.89
	Average message count	HY	0.70	0.69	0.70	0.75	0.74	0.74	0.85	0.82	0.81
		BG	0.64	0.66	0.70	0.76	0.75	0.74	0.80	0.78	0.76
	Parallel SpGEMM time	HY	0.93	0.89	0.87	0.98	0.95	0.87	1.01	0.99	0.97
		BG	0.87	0.78	0.84	0.98	0.96	0.91	1.02	1.00	0.97
IP	Total message volume	HY	2.16	2.05	1.98	2.05	1.87	1.71	1.86	1.63	1.45
		BG	2.93	2.46	2.15	2.03	1.87	1.72	1.83	1.63	1.47
	Message volume imbalance	HY	0.90	0.92	0.93	0.78	0.71	0.58	0.72	0.72	0.66
		BG	0.78	0.86	0.89	0.83	0.80	0.70	0.78	0.81	0.85
	Average message count	HY	0.72	0.66	0.65	0.76	0.78	0.82	0.78	0.79	0.84
		BG	0.65	0.64	0.65	0.77	0.77	0.81	0.82	0.82	0.87
	Parallel SpGEMM time	HY	1.01	0.89	0.87	1.05	1.00	0.94	1.03	0.99	0.95
		BG	0.88	0.75	0.68	1.00	0.96	0.92	1.01	0.98	0.95
RRP	Total message volume	HY	1.10	1.24	1.30	1.52	1.46	1.39	1.48	1.43	1.35
		BG	1.53	1.50	1.48	1.47	1.42	1.36	1.47	1.41	1.34
	Message volume imbalance	HY	0.46	0.44	0.39	0.55	0.48	0.38	0.62	0.52	0.46
		BG	0.57	0.58	0.54	0.79	0.73	0.66	0.79	0.75	0.70
	Average message count	HY	0.71	0.68	0.67	0.70	0.71	0.72	0.76	0.75	0.73
		BG	0.67	0.65	0.67	0.70	0.70	0.70	0.76	0.74	0.73
	Parallel SpGEMM time	HY	0.93	0.83	0.82	0.99	0.97	0.89	1.02	1.00	0.94
		BG	0.92	0.87	0.81	0.97	0.94	0.91	1.01	0.99	0.95

The results obtained by utilizing the communication hypergraph are normalized with respect to the results for the respective hypergraph/bipartite graph model.

The two communication cost metrics considered in the communication hypergraphs are the total message count and the message volume imbalance, in which the former is reduced and a constraint on the latter is enforced. Accordingly, using the communication hypergraphs after the hypergraph and bipartite graph models leads to improvements in these two metrics. For the computational hypergraph models, the total message count is reduced by 28% to 35%, 18% to 30%, and 10% to 43% for the $C = AA^T$, $C = AA$, and $C = AB$ categories, respectively, by using the communication hypergraphs. These improvements are 30% to 36%, 19% to 30%, and 15% to 29% for the computational bipartite graph models. The communication hypergraphs often improve the message volume imbalance as well. The best improvements in this metric are usually obtained by RRP, followed by IP, and then OP. Notice that the improvements in OP are limited. This is because of the utilization of unit weights for the vertices in the communication hypergraph for OP, which does not capture, but approximates the send volume. The communication hypergraphs offer a

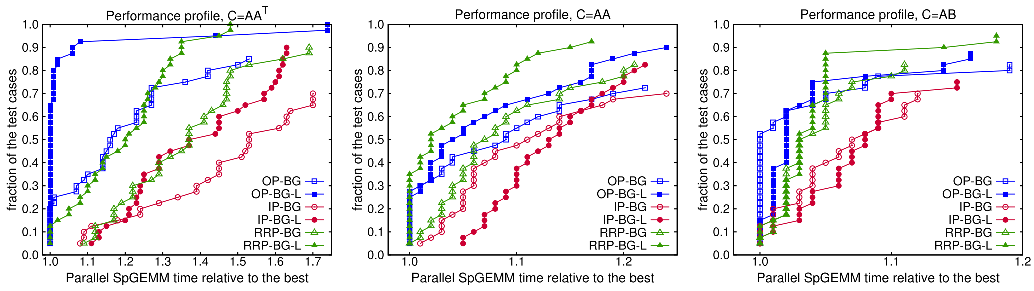


Fig. 7. Performance profiles for the parallel SpGEMM times obtained by the computational bipartite graph models (indicated by "BG") and further using the communication hypergraphs (indicated by "L") for OP, IP, and RRP.

tradeoff between the message count and the message volume, favoring the former at the expense of the latter (Uçar and Aykanat 2004). The volume is usually increased by the communication hypergraphs since, in order to reduce latency, they may assign the communication tasks to the processors that do not depend on those tasks. In other words, the responsibility of a communicated entity may be given to a processor even though that processor does not need that entity in its computations. This is seen in the table as the message volumes of all SpGEMM algorithms are increased when the communication hypergraphs are utilized. The degradations in the message volume are relatively higher in IP for the $C=AA^T$ category due to the existence of coarser vertices in the respective communication hypergraph.

The matrices in the $C=AA^T$ category benefit from reducing the latency cost as the parallel SpGEMM time is reduced by up to 13% for $K=256$, 25% for $K=512$, and 32% for $K=1024$. The improvements in the parallel SpGEMM runtimes for the matrices in the $C=AA$ category are lower. The reason for this is that the message volumes of the matrices in this category are higher than those of the matrices in the $C=AA^T$ category (see Table 4), which makes the latency cost relatively less critical for the parallel performance, so reducing it does not pay off as much as it does in the $C=AA^T$ category. In Table 8, a common trend observed in all categories is that with increasing K, the parallel SpGEMM times get better with the utilization of the communication hypergraphs. This is because the latency cost becomes more important at higher processor counts (as the message count usually increases more sharply than the message volume in the case of strong scaling). For this reason, it can be said that using communication hypergraphs is beneficial for improving the scalability of any of the SpGEMM algorithms.

In Figure 7, for a more detailed analysis, we present the performance profiles for the parallel SpGEMM times obtained by the bipartite graph models for OP, IP, and RRP and the communication hypergraphs further utilized to improve the latency cost of their baseline counterparts. We do not present the profiles for the hypergraph models as they resemble those for the bipartite graph models. In the figure, "BG" indicates the bipartite graph model and "L" indicates the models that further utilize the respective communication hypergraph. For the $C=AA^T$ and $C=AA$ categories, further reducing the latency cost usually pays off as the communication hypergraphs for OP, IP, and RRP improve the parallel SpGEMM time (compare OP-BG with OP-BG-L, IP-BG with IP-BG-L, etc.). In the $C = AA^T$ category, OP-BG-L clearly attains the best performance, while in the $C = AA$ category, RRP-BG-L attains the best performance. Both of these schemes make use of the communication hypergraph. In the $C=AB$ category, while OP-BG is better than OP-BG-L, RRP-BG and RRP-BG-L, as well as IP-BG and IP-BG-L, exhibit close performance.

As a final comparison, we present the performance profiles of the partitioning models that utilize the communication hypergraphs in Figure 8 separately for the $C=AA^T$, $C=AA$, and $C=AB$

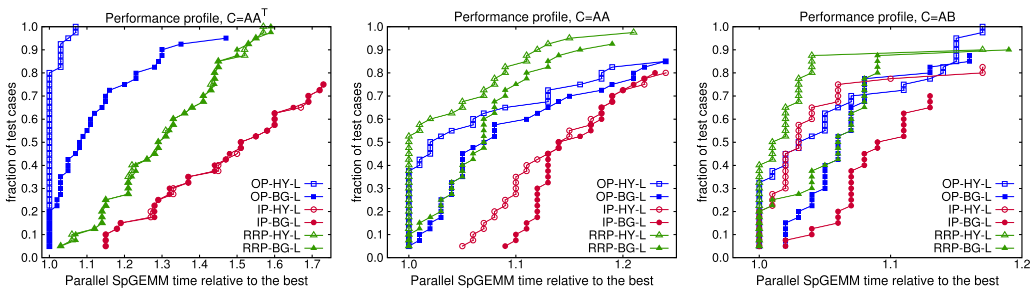


Fig. 8. Performance profiles for the parallel SpGEMM times obtained by the schemes that utilize the communication hypergraphs. The hypergraph and bipartite graph models are respectively indicated by "HY" and "BG." "L" indicates that the communication hypergraph is utilized in the respective scheme.

categories. This comparison determines the best partitioning model, as the models that utilize the communication hypergraphs are usually better than their counterparts that do not.

In the $C = AA^T$ category, the best-performing models clearly belong to OP, followed by RRP, and IP performs the worst. The two best-performing models are OP-HY-L and OP-BG-L. For a specific SpGEMM algorithm, the performance of the bipartite graph model is usually close to that of the hypergraph model (compare RRP-HY-L with RRP-BG-L, IP-HY-L with IP-BG-L, etc.), OP being the exception. Note that the arguments of Sections 5.3 and 5.4 are in agreement with the performances of the partitioning models in the figure.

In the $C = AA$ category, the best-performing models can be said to belong to RRP, closely followed by OP. The best-performing models are RRP-HY-L, RRP-BG-L, and OP-HY-L, where the former two exhibit more stable performance. Again, the performance of a bipartite graph model for a specific SpGEMM algorithm is usually close to the performance of its hypergraph counterpart.

In the $C = AB$ category, the best-performing model is clearly RRP-HY-L, followed by IP-HY-L and OP-HY-L.

5.8 Scalability Analysis

In Figure 9, we compare OP, IP, and RRP in terms of their strong scaling performance for $K = 256$, 512, 1024, and 2048. For a specific SpGEMM algorithm, the best-performing partitioning model among four alternatives at $K = 2048$ is selected for comparison. For example, for OP, the best partitioning model among the hypergraph model, the bipartite graph model, and the two respective communication hypergraphs is selected for comparison (e.g., the best of OP-HY, OP-BG, OP-HY-L, and OP-BG-L, etc.). We include five matrices for the $C = AA^T$ category and seven matrices for the $C = AA$ category.

As seen in Figure 9, for the matrices in the $C = AA^T$ category, OP usually exhibits the best scalability, followed by RRP. Yet in matrices such as `fome21`, `fxm4_6`, and `sgpf5y6`, RRP slowly closes the performance gap with OP as K increases. For the matrices in the $C = AA$ category, RRP scales better than OP and IP. Again, observe that RRP's performance gets better with increasing K, where the gap between RRP and the other schemes gets wider for most of the matrices in this category.

In Figure 10, we investigate the effect of reducing latency cost on scalability. The dashed lines in the figure indicate the models in which only the bandwidth cost is reduced (i.e., without the communication hypergraph), while the solid lines indicate the models in which both the bandwidth and latency costs are reduced (i.e., with the communication hypergraph). For a specific SpGEMM algorithm, again, the best-performing model (either hypergraph or bipartite graph) at $K = 2048$ is

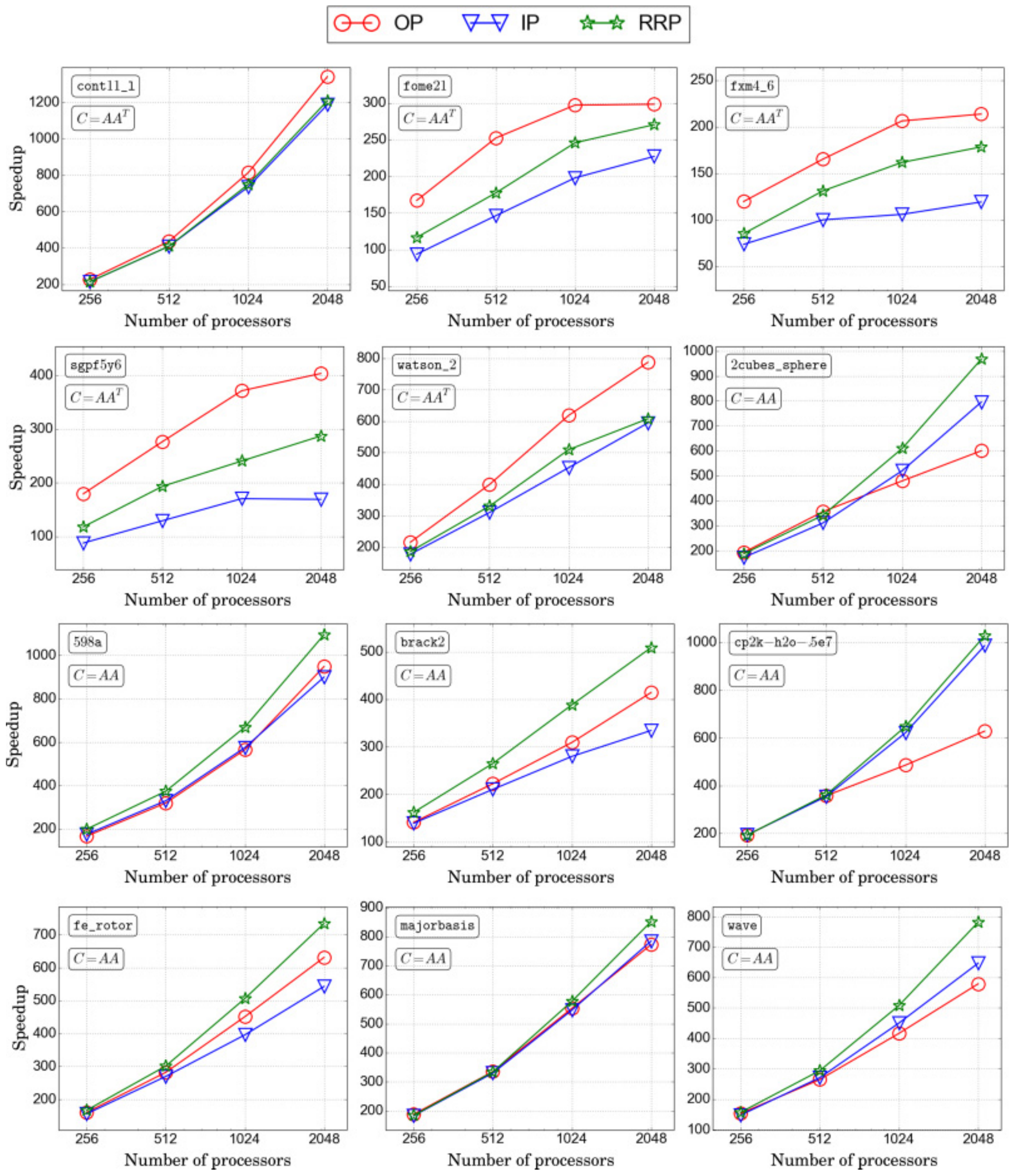


Fig. 9. Speedup curves comparing the SpGEMM algorithms.

selected for display. For example, for OP, the best of the hypergraph and bipartite graph models (e.g., the best of OP-HY and OP-BG) is compared with the best of the respective communication hypergraphs for these two models (e.g., the best of OP-HY-L and OP-BG-L).

Figure 10 shows that reducing latency cost often pays off as better scalability due to the reasons discussed in Section 5.7. In general, the performance gap increases in favor of the models that utilize the communication hypergraphs with increasing K.

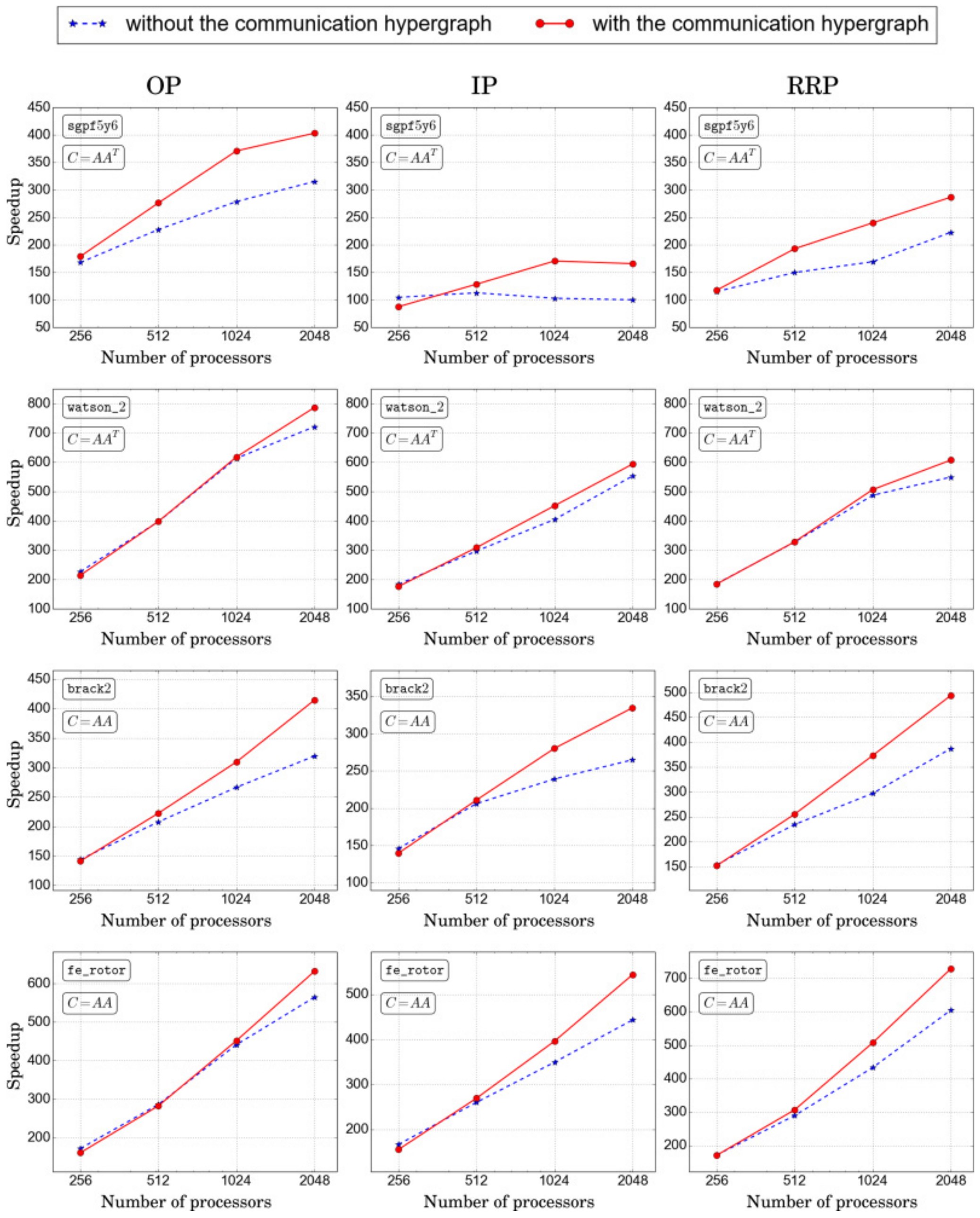


Fig. 10. Speedup curves showing merits of using the communication hypergraphs.

5.9 Overall Assessment

Among all partitioning models, it can be said that the partitioning models for RRP that further utilize the communication hypergraphs (i.e., RRP-HY-L and RRP-BG-L) are the most appealing models because:

(1) They perform the best in the $C = AA$ category. Although not performing the best in the $C = AA^T$ category, they still exhibit average performance, ranking second after OP. In the $C = AB$ category, RRP-HY-L leads other schemes.
(2) They perform better with increasing K, meaning they exhibit better scalability (Section 5.8).
(3) Partitioning the graphs/hypergraphs for RRP is faster than partitioning them for OP and IP; for example, partitioning the bipartite graph model for RRP is $15\times$ to $35\times$ faster than partitioning the hypergraph model for OP, where this factor is $5\times$ to $8\times$ for the bipartite graph model for OP (Section 5.4).
(4) Finally, RRP does not require a symbolic multiplication, whereas the other two schemes require it in the formation of the models (Section 3.5).
(5) We can go further and prefer RRP-BG-L over RRP-HY-L due to faster partitioning of graphs, as partitioning the graphs for RRP is $3\times$ to $6\times$ faster than partitioning the hypergraphs for RRP.

To sum up, although the partitioning models based on OP show stronger speedup performance (especially in the $C = AA^T$ category), they suffer from high partitioning overhead and symbolic multiplication requirements, thus leaving RRP as a better alternative to OP. Another important finding is that the performance of the bipartite graph models for RRP and IP in message volume is negatively affected with increasing density of matrices.

6 CONCLUSION

We proposed bipartite graph and hypergraph partitioning models for efficient parallelization of the SpGEMM kernel on distributed memory architectures. These models enable different 1D partitionings of the input matrices in the kernel. Our models consider both the bandwidth and the latency components of the communication costs in a two-phase methodology in order to improve scalability. The extensive experiments on different categories of SpGEMM operations show that the 1D rowwise partitioning of both input matrices is the best alternative due to its good parallel performance, better scalability, and very low partitioning overhead. The experiments also show that although the bipartite graph models perform slightly worse than the hypergraph models in parallel performance, their significantly low partitioning overhead makes them very attractive.

A APPENDIX: HYPERGRAPH AND BIPARTITE GRAPH PARTITIONING

A hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ is defined as a set of vertices $\mathcal{V}$ and a set of nets (hyperedges) $\mathcal{N}$. Every net $n \in \mathcal{N}$ connects a subset of vertices. The vertices connected by a net n are called its *pins* and denoted as $Pins(n)$. The nets that connect a vertex v are called its nets and denoted as $Nets(v)$. The size of a given hypergraph is defined in terms of three attributes: the number of vertices $|\mathcal{V}|$, the number of nets $|\mathcal{N}|$, and the number of pins, which is equal to $\sum_{n \in \mathcal{N}} |Pins(n)| = \sum_{v \in \mathcal{V}} |Nets(v)|$. Each net n is associated with cost $c(n)$. In case of multiconstraint partitioning, a vertex v is associated with T weights, where T is the number of constraints.

A bipartite graph $\mathcal{G} = (\mathcal{V}^A \cup \mathcal{V}^B, \mathcal{E})$ is defined as two disjoint sets of vertices $\mathcal{V}^A$ and $\mathcal{V}^B$, and a set of edges $\mathcal{E}$. Each edge (v, u) connects a vertex $v \in \mathcal{V}^A$ and another vertex $u \in \mathcal{V}^B$. $Adj(v)$ is used to denote the set of vertices adjacent to vertex $v \in \mathcal{G}$. The size of a bipartite graph is defined in terms of two attributes: the number of vertices $|\mathcal{V}^A \cup \mathcal{V}^B|$ and the number of edges $|\mathcal{E}|$. An edge (v, u) has a cost $c((v, u))$. In case of multiconstraint partitioning, a vertex v is associated with T weights.

Given a hypergraph $\mathcal{H}$ or a bipartite graph $\mathcal{G}$, $\Pi(\mathcal{V}) = \{\mathcal{V}_1, \ldots, \mathcal{V}_K\}$ is called a K-way vertex partition of $\mathcal{H}$ or $\mathcal{G}$ if the K parts are mutually exclusive and exhaustive. A K-way vertex partition of $\mathcal{H}$ or $\mathcal{G}$ is said to satisfy the partitioning constraint if

$$W_t(\mathcal{V}_k) \leq W_t^{avg}(1+\varepsilon), \quad \text{for } k = 1, \ldots, K; \text{ and for } t = 1, \ldots, T. \tag{1}$$

Here, for constraint t, the weight $W_t(\mathcal{V}_k)$ of a part $\mathcal{V}_k$ is defined as the sum of the weights $w_t(v)$ of the vertices in that part (i.e., $W_t(\mathcal{V}_k) = \sum_{v\in\mathcal{V}_k} w_t(v)$), W_t^{avg} is the average part weight (i.e., $W_t^{avg} = (\sum_{v\in\mathcal{V}} w_t(v))/K$), and ε is the predetermined, maximum allowable imbalance ratio.

In a partition $\Pi(\mathcal{V})$ of $\mathcal{H}$, a net that has at least one pin (vertex) in a part is said to *connect* that part. *Connectivity set* $\Lambda(n)$ of a net n is defined as the set of parts connected by n. *Connectivity* $\lambda(n) = |\Lambda(n)|$ of a net n denotes the number of parts connected by n. A net n is said to be *external* if it connects more than one part (i.e., $\lambda(n) > 1$), and *internal* otherwise (i.e., $\lambda(n) = 1$). The set of cut nets in a partition is denoted as $\mathcal{N}_{\text{cut}}$. The partitioning objective is to minimize the cutsize defined over the cut nets. There are various cutsize definitions. The relevant one utilized in this work is Çatalyürek and Aykanat (1999a):

$$cutsize(\Pi(\mathcal{V})) = \sum_{n\in\mathcal{N}_{\text{cut}}} c(n)(\lambda(n)-1). \tag{2}$$

Here, each cut net n incurs a cost of $c(n)(\lambda(n)-1)$ to the cutsize. The hypergraph partitioning problem is known to be NP-hard (Lengauer 1990).

In a partition $\Pi(\mathcal{V})$ of $\mathcal{G}$, an edge is said to be *cut* if it is adjacent to two vertices that reside in different parts and *uncut* otherwise. The set of cut edges in a partition is denoted as $\mathcal{E}_{\text{cut}}$. A vertex v is said to be a *boundary* vertex if it is connected by at least one cut edge. Otherwise, v is said to be an *internal* vertex. The partitioning objective is to minimize the cutsize defined over the cut edges. There are various cutsize definitions. The relevant one utilized in this work is

$$cutsize(\Pi(\mathcal{V})) = \sum_{(v,u)\in\mathcal{E}_{\text{cut}}} c((v,u)). \tag{3}$$

Here, each cut edge (v,u) incurs a cost of $c((v,u))$ to the cutsize.

ACKNOWLEDGMENTS

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

REFERENCES

Kadir Akbudak and Cevdet Aykanat. 2014. Simultaneous input and output matrix partitioning for outer-product–parallel sparse matrix-matrix multiplication. *SIAM Journal on Scientific Computing* 36, 5 (2014), C568–C590. DOI: https://doi.org/10.1137/13092589X arXiv:http://dx.doi.org/10.1137/13092589X

Kadir Akbudak and Cevdet Aykanat. 2017. Exploiting locality in sparse matrix-matrix multiplication on many-core architectures. *IEEE Transactions on Parallel and Distributed Systems* 28, 8 (2017), 2258–2271. DOI: https://doi.org/10.1109/TPDS.2017.2656893

Cevdet Aykanat, B. Barla Cambazoglu, and Bora Uçar. 2008. Multi-level direct K-way hypergraph partitioning with multiple constraints and fixed vertices. *Journal of Parallel and Distributed Computing* 68, 5 (May 2008), 609–625.

Ariful Azad, Aydın Buluç, and John R. Gilbert. 2015. Parallel triangle counting and enumeration using matrix algebra. In *Proceedings of the IPDPSW, Workshop on Graph Algorithm Building Blocks (GABB'15)*. 804–811. DOI: https://doi.org/10.1109/IPDPSW.2015.75

Grey Ballard, Aydin Buluc, James Demmel, Laura Grigori, Benjamin Lipshitz, Oded Schwartz, and Sivan Toledo. 2013. Communication optimal parallel multiplication of sparse random matrices. In *Proceedings of the 25th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'13)*. ACM, New York, 222–231. DOI: https://doi.org/10.1145/2486159.2486196

Grey Ballard, Alex Druinsky, Nicholas Knight, and Oded Schwartz. 2015. Brief announcement: Hypergraph partitioning for parallel sparse matrix-matrix multiplication. In *Proceedings of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures (SPAA'15)*. ACM, New York, 86–88. DOI: https://doi.org/10.1145/2755573.2755613

Nathan Bell, Steven Dalton, and Luke N. Olson. 2012. Exposing fine-grained parallelism in algebraic multigrid methods. *SIAM Journal on Scientific Computing* 34, 4 (2012), C123–C152.

Rob H. Bisseling, Timothy M. Doup, and L. Daniel J. C. Loyens. 1993. A parallel interior point algorithm for linear programming on a network of transputers. *Annals of Operations Research* 43 (1993), 51–86.

Erik G. Boman, Ojas Parekh, and Cynthia Phillips. 2005. *LDRD Final Report on Massively-Parallel Linear Programming: The parPCx System.* Technical Report. SAND2004-6440, Sandia National Laboratories.

Urban Borštnik, Joost VandeVondele, Valéry Weber, and Jürg Hutter. 2014. Sparse matrix multiplication: The distributed block-compressed sparse row library. *Parallel Computing* 40, 5 (2014), 47–58.

William L. Briggs, Van Emden Henson, and Steve F. McCormick. 2000. *A Multigrid Tutorial.* 2nd ed. Siam, Philadelphia.

Aydın Buluç and John R. Gilbert. 2008. On the representation and multiplication of hypersparse matrices. In *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing (IPDPS'08).* 1–11. DOI: https://doi.org/10.1109/IPDPS.2008.4536313

Aydin Buluç and John R. Gilbert. 2011. The combinatorial BLAS: Design, implementation, and applications. *International Journal of High Performance Computing Applications* 25, 4 (2011), 496–509.

Aydın Buluç and John R. Gilbert. 2012. Parallel sparse matrix-matrix multiplication and indexing: Implementation and experiments. *SIAM Journal of Scientific Computing (SISC)* 34, 4 (2012), 170–191. DOI: https://doi.org/10.1137/110848244

Lynn Cannon. 1969. *A Cellular Computer to Implement the Kalman Filter Algorithm.* Ph.D. Dissertation. Montana State University, Bozeman, MN.

Ümit V. Çatalyürek and Cevdet Aykanat. 1999a. Hypergraph-partitioning based decomposition for parallel sparse-matrix vector multiplication. *IEEE Transactions on Parallel Distributed Systems* 10, 7 (1999), 673–693.

Ümit V. Çatalyürek and Cevdet Aykanat. 1999b. *PaToH: A Multilevel Hypergraph Partitioning Tool, Version 3.0.* Computer Engineering Department, Bilkent University, Ankara, Turkey.

Matt Challacombe. 1999. A simplified density matrix minimization for linear scaling self-consistent field theory. *Journal of Chemical Physics* 110, 5 (1999), 2332–2342. DOI: https://doi.org/10.1063/1.477969

Matt Challacombe. 2000. A general parallel sparse-blocked matrix multiply for linear scaling SCF theory. *Computer Physics Communications* 128, 12 (2000), 93–107. DOI: https://doi.org/10.1016/S0010-4655(00)00074-6

CP2K. 2016. CP2K home page. Retrieved from http://www.cp2k.org/.

Steven Dalton, Nathan Bell, and Luke Olson. 2013. *Optimizing Sparse Matrix-Matrix Multiplication for the GPU.* Technical report. NVIDIA.

Andrew D. Daniels, John M. Millam, and Gustavo E. Scuseria. 1997. Semiempirical methods with conjugate gradient density matrix search to replace diagonalization for molecular systems containing thousands of atoms. *Journal of Chemical Physics* 107, 2 (1997), 425–431. DOI: https://doi.org/10.1063/1.474404

Timothy A. Davis. 2006. *Direct Methods for Sparse Linear Systems.* Society for Industrial and Applied Mathematics. DOI: https://doi.org/10.1137/1.9780898718881 .arXiv:http://epubs.siam.org/doi/pdf/10.1137/1.9780898718881

Timothy A. Davis and Yifan Hu. 2011. The university of florida sparse matrix collection. *ACM Transactions on Mathematical Software* 38, 1 (2011), 1.

James Demmel, David Eliahu, Armando Fox, Shoaib Kamil, Benjamin Lipshitz, Oded Schwartz, and Omer Spillinger. 2013. Communication-optimal parallel recursive rectangular matrix multiplication. In *Proceedings of 27th International Parallel Distributed Processing Symposium.* IEEE, 261–272. DOI: https://doi.org/10.1109/IPDPS.2013.80

Elizabeth D. Dolan and Jorge J. Moré. 2002. Benchmarking optimization software with performance profiles. *Mathematical Programming* 91, 2 (2002), 201–213.

Felix Gremse, Andreas Hofter, Lars Ole Schwen, Fabian Kiessling, and Uwe Naumann. 2015. GPU-accelerated sparse matrix-matrix multiplication by iterative row merging. *SIAM Journal on Scientific Computing* 37, 1 (2015), C54–C71.

Fred G. Gustavson. 1978. Two fast algorithms for sparse matrices: Multiplication and permuted transposition. *ACM Transactions on Mathematical Software* 4, 3 (1978), 250–269.

Václav Hapla, David Horák, and Michal Merta. 2013. Use of direct solvers in TFETI massively parallel implementation. In *Applied Parallel and Scientific Computing*, Pekka Manninen and Per Ster (Eds.). Springer, Berlin, 192–205. DOI: https://doi.org/10.1007/978-3-642-36803-5_14

Michael Heroux, Roscoe Bartlett, Vicki Howle, Robert Hoekstra, Jonathan Hu, Tamara Kolda, Richard Lehoucq, Kevin Long, Roger Pawlowski, Eric Phipps, Andrew Salinger, Heidi Thornquist, Ray Tuminaro, James Willenbring, and Alan Williams. 2003. *An Overview of Trilinos.* Technical Report SAND2003-2927. Sandia National Laboratories, Albuquerque, NM.

Satoshi Itoh, Pablo Ordejn, and Richard M. Martin. 1995. Order-N tight-binding molecular dynamics on parallel computers. *Computer Physics Communications* 88, 2–3 (1995), 173–185. DOI: https://doi.org/DOI: 10.1016/0010-4655(95)00031-A

George Karypis, Anshul Gupta, and Vipin Kumar. 1994. A parallel formulation of interior point algorithms. In *Supercomputing'94.*

George Karypis and Vipin Kumar. 1998. A parallel algorithm for multilevel graph partitioning and sparse matrix ordering. *Journal of Parallel and Distributed Computing* 48, 1 (1998), 71–95.

George Karypis and Vipin Kumar. 1999. A fast and highly quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing* 20, 1 (1999), 359–392. Also available at http://www.cs.umn.edu/~karypis. A short version appears in International Conference on Parallel Processing 1995.

Thomas Lengauer. 1990. *Combinatorial Algorithms for Integrated Circuit Layout*. Willey–Teubner, Chichester, UK.

X.-P. Li, R. W. Nunes, and David Vanderbilt. 1993. Density-matrix electronic-structure method with linear system-size scaling. *Physical Review B* 47, 16 (Apr. 1993), 10891–10894. DOI:https://doi.org/10.1103/PhysRevB.47.10891

Greg Linden, Brent Smith, and Jeremy York. 2003. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing* 7, 1 (2003), 76–80.

Weifeng Liu and Brian Vinter. 2014. An efficient GPU general sparse matrix-matrix multiplication for irregular data. In *Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium*. IEEE, 370–381.

John M. Millam and Gustavo E. Scuseria. 1997. Linear scaling conjugate gradient density matrix search as an alternative to diagonalization for first principles electronic structure calculations. *Journal of Chemical Physics* 106, 13 (1997), 5569–5577. DOI:https://doi.org/10.1063/1.473579

Intel MKL. 2015. Math Kernel Library (MKL). Retrieved from http://software.intel.com/en-us/articles/intel-mkl/.

Kurtis L. Nusbaum. 2011. *Optimizing Tpetra's Sparse Matrix-Matrix Multiplication Routine*. Technical Report. SAND2011-6036, Sandia National Laboratories.

Carlos Ordonez. 2010. Optimization of linear recursive queries in SQL. *IEEE Transactions on Knowledge and Data Engineering* 22, 2 (2010), 264–277.

Carlos Ordonez, Yiqun Zhang, and Wellington Cabrera. 2016. The gamma matrix to summarize dense and sparse data sets for big data analytics. *IEEE Transactions on Knowledge and Data Engineering* 28, 7 (2016), 1905–1918.

Md Mostofa Ali Patwary, Nadathur Rajagopalan Satish, Narayanan Sundaram, Jongsoo Park, Michael J. Anderson, Satya Gautam Vadlamudi, Dipankar Das, Sergey G. Pudov, Vadim O. Pirogov, and Pradeep Dubey. 2015. Parallel efficient sparse matrix-matrix multiplication on multicore platforms. In *High Performance Computing*. Springer, 48–57.

H. Bernhard Schlegel, John M. Millam, Srinivasan S. Iyengar, Gregory A. Voth, Andrew D. Daniels, Gustavo E. Scuseria, and Michael J. Frisch. 2001. Ab initio molecular dynamics: Propagating the density matrix with Gaussian orbitals. *Journal of Chemical Physics* 114, 22 (2001), 9758–9763. DOI:https://doi.org/10.1063/1.1372182

Oguz Selvitopi and Cevdet Aykanat. 2016. Reducing latency cost in 2D sparse matrix partitioning models. *Parallel Computing* 57, Supplement C (2016), 1–24. DOI:https://doi.org/10.1016/j.parco.2016.04.004

Edgar Solomonik, Abhinav Bhatele, and James Demmel. 2011. Improving communication performance in dense linear algebra via topology aware collectives. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC'11)*. ACM, New York, Article 77, 11 pages. DOI:https://doi.org/10.1145/2063384.2063487

Total-FETI. 2016. Total-FETI Massively Parallel Implementation Research Group. Retrieved from http://spomech.vsb.cz/feti/.

Bora Uçar and Cevdet Aykanat. 2004. Encapsulating multiple communication-cost metrics in partitioning sparse rectangular matrices for parallel matrix-vector multiplies. *SIAM Journal on Scientific Computing* 26, 6 (2004), 1837–1859.

Robert A. van de Geijn and Jerrell Watts. 1997. SUMMA: Scalable universal matrix multiplication algorithm. *Concurrency - Practice and Experience* 9, 4 (1997), 255–274.

Joost VandeVondele, Urban Borstnik, and Jurg Hutter. 2012. Linear scaling self-consistent field calculations with millions of atoms in the condensed phase. *Journal of Chemical Theory and Computation* 8, 10 (2012), 3565–3573.

Received February 2016; revised February 2017; accepted August 2017