SIAM J. SCI. COMPUT.
Vol. 36, No. 5, pp. C568–C590

SIMULTANEOUS INPUT AND OUTPUT MATRIX PARTITIONING FOR OUTER-PRODUCT–PARALLEL SPARSE MATRIX-MATRIX MULTIPLICATION*

KADIR AKBUDAK† AND CEVDET AYKANAT†

Abstract. For outer-product–parallel sparse matrix-matrix multiplication (SpGEMM) of the form $C = A \times B$, we propose three hypergraph models that achieve simultaneous partitioning of input and output matrices without any replication of input data. All three hypergraph models perform conformable one-dimensional (1D) columnwise and 1D rowwise partitioning of the input matrices A and B, respectively. The first hypergraph model performs two-dimensional (2D) nonzero-based partitioning of the output matrix, whereas the second and third models perform 1D rowwise and 1D columnwise partitioning of the output matrix, respectively. This partitioning scheme induces a two-phase parallel SpGEMM algorithm, where communication-free local SpGEMM computations constitute the first phase and the multiple single-node-accumulation operations on the local SpGEMM results constitute the second phase. In these models, the two partitioning constraints defined on weights of vertices encode balancing computational loads of processors during the two separate phases of the parallel SpGEMM algorithm. The partitioning objective of minimizing the cutsize defined over the cut nets encodes minimizing the total volume of communication that will occur during the second phase of the parallel SpGEMM algorithm. An MPI-based parallel SpGEMM library is developed to verify the validity of our models in practice. Parallel runs of the library for a wide range of realistic SpGEMM instances on two large-scale parallel systems JUQUEEN (an IBM BlueGene/Q system) and SuperMUC (an Intel-based cluster) show that the proposed hypergraph models attain high speedup values.

Key words. parallel computing, sparse matrices, sparse matrix-matrix multiplication, SpGEMM, matrix partitioning, hypergraph partitioning

AMS subject classifications. 65Y05, 68W10, 65F50, 05C65, 90C51

DOI. 10.1137/13092589X

1. Introduction.

Sparse matrix-matrix multiplication (SpGEMM) is a kernel operation in a wide variety of scientific applications such as finite element simulations based on domain decomposition [3, 22], molecular dynamics (MD) [15, 16, 17, 25, 28, 29, 32, 36], and linear programming (LP) [7, 8, 26], all of which utilize parallel processing technology to reduce execution times. Among these applications, below we exemplify three methods/codes from which we select realistic SpGEMM instances.

In finite element application fields, finite element tearing and interconnecting (FETI) [3, 22] type domain decomposition methods are used for numerical solution of engineering problems. In this application, the SpGEMM computation GG^T is performed, where $G = R^T B^T$, R is the block diagonal basis of the stiffness matrix, and B is the signed matrix with entries -1, 0, 1 describing the subdomain interconnectivity.

In MD application fields, CP2K program [1] performs parallel atomistic and

*Submitted to the journal's Software and High-Performance Computing section June 24, 2013; accepted for publication (in revised form) July 7, 2014; published electronically October 23, 2014. This work was supported by the PRACE-1IP project funded in part by the EU's 7th Framework Programme (FP7/2007-2013) under grant agreement RI-283493 and FP7-261557. It was also supported by PRACE, who provided Preparatory Access Call Type B (resource) awards for applications numbered 2010PA0930 and 2010PA2149. The results in this paper have been achieved using these awarded resources, JUQUEEN at the Jülich Supercomputing Centre, and SuperMUC at the Leibniz Supercomputing Center, all of which are based in Germany.
http://www.siam.org/journals/sisc/36-5/92589.html

†Computer Engineering Department, Bilkent University, Ankara 06800, Turkey (kadir@cs.bilkent.edu.tr, aykanat@cs.bilkent.edu.tr).

molecular simulations of solid state, liquid, molecular, and biological systems. In this application, SpGEMM computations of the form AA are performed during the Newton–Schulz iterations to compute the sign of a given matrix A, which is reported to take more than half of the total parallel running time [36].

Large-scale LP problems are usually solved by iterative interior point methods. These methods solve normal equations of the form $(AD^2A^T)x = b$ to find search directions at each iteration. Here, A is the original constraint matrix and D is a positive diagonal matrix which varies with each iteration. For the solution of these normal equations, direct solvers [7, 8, 26] that utilize Cholesky factorization as well as iterative solvers [8] that utilize preconditioners require explicitly forming the coefficient matrix at each iteration through the SpGEMM computation AB, where the sparsity patterns of both A and $B = D^2A^T$ remain the same throughout the iterations. It is reported in [8] that SpGEMM computation takes substantially longer than Cholesky factorization for some problems.

1.1. Related work. There exist software libraries that provide SpGEMM computation such as Intel MKL [2] for shared memory architectures, Tpetra [30] package of Trilinos [23] and Combinatorial BLAS (CombBLAS) [10] for distributed memory architectures, and CUSPARSE [31] and CUSP [6] for GPUs. SpGEMM routines of Intel MKL and CUSPARSE perform symbolic multiplication prior to numeric multiplication. Below, we briefly discuss the parallel SpGEMM algorithms for distributed memory architectures. Here and hereafter, we consider parallization of SpGEMM of the form $C = A \times B$ on a K-processor system.

Trilinos uses one-dimensional (1D) rowwise partitioning of both input matrices for inner-product–parallel SpGEMM. It consists of a sequence of K shift operations of the row blocks of the B matrix along the processor ring so that, at the end, each processor computes a distinct row block of 1D partitioned output matrix.

CombBLAS adopts the SUMMA algorithm for outer-product–parallel SpGEMM [11] and utilizes its own serial hypersparse kernels [9]. SUMMA [35] utilizes two-dimensional (2D) checkerboard partitioning of both input matrices assuming a $\sqrt{K} \times \sqrt{K}$ processor mesh. It consists of a sequence of $\sqrt{K}$ rowwise and columnwise broadcasts of the row and column blocks of A and B matrices, respectively, so that each processor incrementally computes a distinct block of the checkerboard partitioned output matrix.

Beside these libraries, recently, Ballard et. al. [5], provide tighter lower bounds on the expected communication cost of parallel SpGEMM operation and propose two new three-dimensional (3D) iterative and recursive algorithms to match the expected lower bounds. These two algorithms are adaptations of earlier two dense algorithms [19, 33].

None of the above-mentioned approaches utilizes the sparsity patterns of input or output matrices to reduce the communication overhead. The models proposed in this work utilize sparsity patterns of matrices to develop intelligent matrix partitioning schemes that aim at minimizing communication overhead while maintaining computational load balance for outer-product–parallel SpGEMM.

1.2. Communication requirements of outer-product–parallel SpGEMM. Our focus is parallelization of SpGEMM computations through conformable one-dimensional (1D) columnwise and 1D rowwise partitioning of the input matrices A and B as

$$\hat{A} = AQ = \begin{bmatrix} A_1^c & A_2^c & \dots & A_K^c \end{bmatrix} \quad \text{and} \quad \hat{B} = QB = \begin{bmatrix} B_1^r \\ B_2^r \\ \vdots \\ B_K^r \end{bmatrix}. \tag{1.1}$$

Here, Q denotes the permutation matrix induced by the partitioning. In (1.1), the use of the same permutation matrix Q for reordering columns of A and rows of B is because of the conformable columnwise and rowwise partitioning of A and B matrices. In the input partitioning given in (1.1), each processor P_k owns column stripe A_k^c and row stripe B_k^r of the permuted matrices. Hence, this conformability requirement enables each processor P_k to compute the outer product $A_k^c \times B_k^r$ without any communication. Note that the input data partitioning given in (1.1) does not involve any row/column replication.

The input data partitioning given in (1.1) leads to an outer-product–parallel SpGEMM scheme, where the output matrix C is computed as follows in terms of results of the local SpGEMM computations:

$$C = C^1 + C^2 + \cdots + C^K \quad (C^k = A_k^c \times B_k^r \text{ is performed by processor } P_k). \tag{1.2}$$

The summation of local C^k matrices incurs communication because of the multiple single-node-accumulation (SNAC) operations required for calculating the final value of each nonzero c_{ij} of C from the partial results of the local SpGEMM operations. This parallellization scheme induces a two-phase parallel SpGEMM algorithm, where communication-free local SpGEMM computations constitute the first phase and the multiple SNAC operations constitute the second phase. In the rest of the paper, the first and second phases will be referred to as multiplication and summation phases, respectively.

The input partitioning on A and B matrices does not lead to an inherent and natural output partitioning on the nonzeros of the C matrix. Output partitioning refers to determining the processor which will be responsible for accumulating partial results for each nonzero $c_{i,j}$ of C, where $c_{i,j} = \sum_{c_{i,j}^{(k)} \in C^k} c_{i,j}^{(k)}$. Here, $c_{i,j}^{(k)} \in C^k$ denotes that $c_{i,j}^{(k)}$ is a nonzero of C^k, and hence it is a partial result for $c_{i,j}$ of C. Although computational load balance is the only performance issue in the input partitioning, communication overhead is a crucial performance issue in the output partitioning.

For output partitioning, we will consider 1D rowwise and 1D columnwise partitioning as well as 2D partitioning of the output matrix C. The 2D output partitioning is a nonzero-based partitioning of C so that the tasks of computing individual nonzeros of C constitute the atomic tasks. In 1D rowwise/columnwise output matrix partitioning, the tasks of computing the nonzeros belonging to the same rows/columns constitute the atomic tasks.

In both 1D rowwise/columnwise and 2D nonzero-based output partitioning, the worst-case communication requirement is $K(K-1)$ messages and $(K-1)nnz(C)$ words, where $nnz(\cdot)$ denotes the number of nonzeros in a matrix. This worst case occurs when each local SpGEMM computation generates a partial result for each nonzero of the output matrix C.

1.3. Contributions. In this work, we first propose an elementary hypergraph model that contains a single vertex for representing the outer product of each column of A with the respective row of B in order to enforce conformable 1D columnwise and 1D rowwise partitioning of the input matrices A and B. This hypergraph also

contains a vertex for each nonzero of C to enable 2D nonzero-based partitioning of the output matrix. This hypergraph contains a hyperedge (net) for each nonzero of C to encode the total volume of communication that will occur during the multiple SNAC operations to be performed in the summation phase of the outer-product–parallel SpGEMM algorithm. Then, by utilizing this hypergraph model, we propose a two-constraint hypergraph partitioning (HP) formulation that enables simultaneous partitioning of input and output matrices in a single stage. The two partitioning constraints encode balancing computational loads of processors during the two separate phases of the parallel SpGEMM algorithm. The partitioning objective of minimizing cutsize encodes minimizing the total message volume that will be transmitted during the point-to-point communications in the summation phase of the parallel algorithm. The second and third hypergraph models are obtained by extending the elementary hypergraph model to enforce 1D rowwise and 1D columnwise partitioning of the output matrix through vertex amalgamation.

We should note here that none of the proposed models utilizes any one of the hypergraph models (e.g., row-net, column-net, and row-column-net models [12, 14]) previously proposed for partitioning sparse matrices for sparse matrix-vector multiplication (SpMV). The models proposed in this work aim directly at representing outer-product-based SpGEMM computations.

The validity of the proposed data partitioning models and formulations are tested on a wide range of large-scale SpGEMM computation instances by utilizing the state-of-the-art HP tool PaToH [13]. Experiments show that the proposed hypergraph models and HP formulations achieve successful input and output partitioning of SpGEMM computations. In order to verify that the theoretical gains obtained by the models hold in practice, a two phase outer-product–parallel SpGEMM code utilizing the MPI library is developed. Parallel SpGEMM runs on both JUQUEEN (an IBM BlueGene/Q system) and SuperMUC (an Intel-based cluster) show that the proposed partitioning models attain high speedup values.

The rest of the paper is organized as follows: Background information on hypergraphs and HP is given in section 2. In section 3, we introduce the elementary hypergraph model and show how to extend it to enable 1D rowwise/columnwise partitioning of the output matrix. The experimental results are presented in section 4. Finally, we conclude the paper in section 5.

2. Background on HP. 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 are denoted as $Pins(n)$. The degree of a net n is equal to the number of its pins, i.e., $deg(n) = |Pins(n)|$. The nets connecting a vertex v are called its nets and are denoted as $Nets(v)$. The degree of a vertex v is equal to the number of its nets, i.e., $deg(v) = |Nets(v)|$. 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 $\sum_{n\in\mathcal{N}} deg(n) = \sum_{v\in\mathcal{V}} deg(v)$. In case of multiconstraint partitioning, T weights are associated with a vertex v, where T is the number of constraints.

Given a hypergraph $\mathcal{H} = (\mathcal{V},\mathcal{N})$, $\Pi(\mathcal{V}) = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$ is called a K-way partition of the vertex set $\mathcal{V}$ if the K parts are mutually exclusive and exhaustive. A K-way vertex partition of $\mathcal{H}$ is said to satisfy the partitioning constraint if

$$W_t(\mathcal{V}_k) \leq W_t^{avg}(1+\varepsilon) \quad \text{for } k = 1, 2, \ldots, K; \text{ and for } t = 1, 2, \ldots, T. \tag{2.1}$$

Here, for the tth constraint, 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 represents 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 *cut* (*external*) if it connects more than one part (i.e., $\lambda(n) > 1$), and *uncut* (*internal*) otherwise (i.e., $\lambda(n) = 1$). The set of cut nets of a partition Π is denoted as $\mathcal{N}_{\text{cut}}$. A vertex v is said to be a *boundary* vertex if it is connected by at least one cut net. Otherwise, v is said to be an *internal* vertex. The partitioning objective is to minimize the cutsize defined over the cut nets. There are various cutsize definitions. The relevant definition is [12]

$$cutsize(\Pi(\mathcal{V})) = \sum_{n \in \mathcal{N}_{\text{cut}}} \Big(\lambda(n) - 1\Big). \tag{2.2}$$

Here, each cut net n incurs a cost of $\lambda(n) - 1$ to the cutsize. The HP problem is known to be NP-hard [27].

3. Models for simultaneous input and output matrix partitioning. All three hypergraph models proposed and discussed in this section enable 1D input partitioning by conformably partitioning A and B matrices columnwise and rowwise, respectively, for outer-product–parallel SpGEMM. The first hypergraph model achieves 2D output partitioning by performing nonzero-based partitioning of the C matrix. The second and third hypergraph models achieve 1D output partitioning by partitioning the C matrix rowwise and columnwise, respectively. The first hypergraph model will be referred to as *elementary hypergraph model* because the second and third hypergraph models can be derived from the first one, as will be discussed later.

We should note here that the construction of all hypergraph models assumes full access to the actual computation pattern that forms the output matrix C. Deriving this computation pattern from the sparsity patterns of the two input matrices requires performing symbolic SpGEMM. This symbolic multiplication requirement prior to partitioning is a major difference compared to partitioning for parallel SpMV because the computation pattern of SpMV is directly determined by the sparsity pattern of the input matrix.

3.1. Elementary hypergraph model for 2D output matrix partitioning. In the elementary hypergraph model, a given SpGEMM computation $C = A \times B$ is represented as a hypergraph $\mathcal{H}_{\text{E}}(A, B) = (\mathcal{V} = \mathcal{V}^{AB} \cup \mathcal{V}^{C}, \mathcal{N})$ for 1D conformable columnwise and rowwise partitioning of input matrices A and B, respectively, and 2D nonzero-based partitioning of the output matrix C. The vertex subsets $\mathcal{V}^{AB}$ and $\mathcal{V}^{C}$ of $\mathcal{V}$ will be referred to here as input and output vertex subsets, respectively. For each column x of A and row x of B, there exists a single input vertex v_x in the vertex subset $\mathcal{V}^{AB}$. For each nonzero $c_{i,j}$ of the output matrix C, there exist both an output vertex $v_{i,j}$ in the vertex subset $\mathcal{V}^{C}$ and a net $n_{i,j}$ in the net set $\mathcal{N}$. Net $n_{i,j}$ connects a vertex v_x if column x of A contains a nonzero at row i and row x of B contains a nonzero at column j. In other words, net $n_{i,j}$ connects a vertex v_x if the outer product of column x of A with row x of B generates a partial result for $c_{i,j}$ of C. Net $n_{i,j}$ also connects vertex $v_{i,j}$. So net $n_{i,j}$ is defined as

$$Pins(n_{i,j}) = \{v_x : a_{i,x} \in A \wedge b_{x,j} \in B\} \cup \{v_{i,j}\}. \tag{3.1}$$

As seen in (3.1), each net $n_{i,j}$ connects exactly one output vertex and at least one input vertex. Note that double subscript notation is used for identifying output vertices and nets for the sake of clarity of presentation.

The size of the proposed hypergraph model $\mathcal{H}_{\mathrm{E}}$ can be defined as follows in terms of the attributes of input matrices A and B and the output matrix C:

$$|\mathcal{V}| = cols(A) + nnz(C) = rows(B) + nnz(C), \tag{3.2}$$

$$|\mathcal{N}| = nnz(C), \tag{3.3}$$

$$\#\ of\ pins = \sum_{x=1}^{cols(A)} \Big(nnz(a_{*,x}) \cdot nnz(b_{x,*}) \Big) + nnz(C). \tag{3.4}$$

Here, $rows(\cdot)$ and $cols(\cdot)$, respectively, denote the number of rows and columns of a given matrix, $a_{*,x}$ denotes column x of A, and $b_{x,*}$ denotes row x of B. In (3.4) given for defining the number of pins of $\mathcal{H}_{\mathrm{E}}$, the summation term corresponds to the total number of scalar multiply operations to be performed in the SpGEMM computation.

In the two-constraint formulation proposed here, we assign two weights to each vertex. The first and second weights represent the computational loads of the tasks associated with the vertex for the multiplication and summation phases, respectively. Each vertex $v_x \in \mathcal{V}^{AB}$ is associated with the atomic task of computing the outer product of column x of A with row x of B. This outer product $a_{*,x} \times b_{x,*}$ incurs $nnz(a_{*,x}) \cdot nnz(b_{x,*})$ scalar multiply operations to generate $nnz(a_{*,x}) \cdot nnz(b_{x,*})$ partial results. So we assign the following two weights for vertex v_x:

$$w_1(v_x) = nnz(a_{*,x}) \cdot nnz(b_{x,*}), \qquad w_2(v_x) = 0. \tag{3.5}$$

Note that $w_1(v_x)$ also denotes the number of nets that connect input vertex v_x, i.e., $deg(v_x) = w_1(v_x)$.

Each vertex $v_{i,j} \in \mathcal{V}^{C}$ is associated with the atomic task of computing $c_{i,j}$ by accumulating the partial nonzero results obtained from the outer product computations. Each net $n_{i,j}$ represents the multiway relation for the computation of $c_{i,j}$ from the outer-product computations. That is, the vertices in $Pins(n_{i,j}) - \{v_{i,j}\}$ represent the set of outer-product results needed to accumulate $c_{i,j}$. Figure 1 illustrates the input and output dependency view of the elementary hypergraph model. As seen in this figure, net $n_{i,j}$ with $Pins(n_{i,j}) = \{v_x, v_y, v_z, v_{i,j}\}$ shows that the outer products $a_{*,x} \times b_{x,*}$, $a_{*,y} \times b_{y,*}$, and $a_{*,z} \times b_{z,*}$ yield nonzero results $c^x_{i,j}$, $c^y_{i,j}$, and $c^z_{i,j}$, respectively. Hence, vertex $v_{i,j}$ represents the task of computing the final result for $c_{i,j}$ as $c_{i,j} = c^x_{i,j} + c^y_{i,j} + c^z_{i,j}$. So we assign the following two weights for vertex $v_{i,j}$:

$$w_1(v_{i,j}) = 0, \qquad w_2(v_{i,j}) = |Pins(n_{i,j})| - 1. \tag{3.6}$$

As seen in (3.5) and (3.6), the first and second weights of vertices represent computational loads of the tasks associated with these vertices during the multiplication and summation phases of the parallel SpGEMM algorithm, respectively. So zero weights are assigned as the second weights of the input vertices (i.e., $w_2(v_x) = 0$ in (3.5)) since the tasks associated with input vertices do not involve any computation during the summation phase. In a dual manner, zero weights are assigned as the first weights of the output vertices (i.e., $w_1(v_{i,j}) = 0$ in (3.6)) since the tasks associated with output vertices do not involve any computation during the multiplication phase.

A partition $\Pi(\mathcal{V})$ on $\mathcal{V}$ automatically induces a partition $\Pi(\mathcal{V}^{AB})$ on $\mathcal{V}^{AB} \subseteq \mathcal{V}$ and a partition $\Pi(\mathcal{V}^{C})$ on $\mathcal{V}^{C} \subseteq \mathcal{V}$. Partition $\Pi(\mathcal{V}^{AB})$ is decoded as an input partition

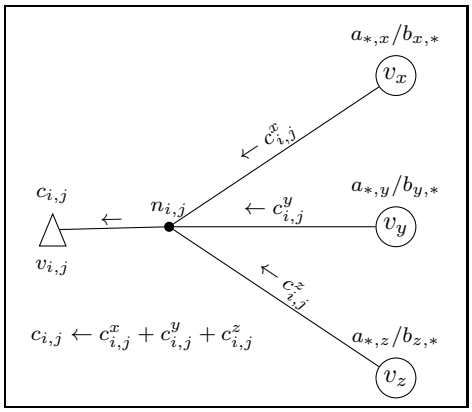

FIG. 1. *Elementary hypergraph model $\mathcal{H}_{\mathrm{E}}$ for 2D nonzero-based output partitioning.*

on the columns of A and rows of B, and partition $\Pi(\mathcal{V}^C)$ is decoded as an output partition on the nonzeros of C. That is, $v_x \in \mathcal{V}_k$ denotes that column $a_{*,x}$ of A and row $b_{x,*}$ of B are mapped to processor P_k, and P_k is held responsible for computing the outer product $a_{*,x} \times b_{x,*}$ according to the owner computes rule. $v_{i,j} \in \mathcal{V}_k$ denotes that processor P_k is held responsible for accumulating the partial results to compute the final result for $c_{i,j}$.

3.1.1. Model correctness. Here, we discuss the correctness of the proposed elementary hypergraph model by showing the following:

(a) Two constraints on part weights encode computational load balancing during the two phases.

(b) Cutsize minimization objective encodes the minimization of total communication volume during the summation phase.

For both (a) and (b), consider a partition $\Pi(\mathcal{V}) = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$ of vertices of $\mathcal{H}_{\mathrm{E}}$. Without loss of generality, we assume that part $\mathcal{V}_k$ is assigned to processor P_k for $k = 1, 2, \ldots, K$.

For (a), $\Pi(\mathcal{V})$ satisfies the two balance constraints given in (2.1) for $T = 2$. Under the first vertex weight definitions given in (3.5) and (3.6), the first constraint correctly encodes balancing the local outer-product computations to be performed by processors during the multiplication phase. Under the second vertex weight definitions given in (3.5) and (3.6), the second constraint correctly encodes balancing the number of local partial-result accumulation operations to be performed by processors during the summation phase. However, this correctness of the second constraint depends on a naive implementation in which each processor maintains its outer-product results rather than accumulating them on a single local C matrix. In an efficient implementation, each processor P_k accumulates its outer-product results on a single local output matrix on the fly after every local outer-product computation as follows: $C^k \leftarrow C^k + a_{*,x} \times b_{x,*}$, where $v_x \in \mathcal{V}_k$. This efficient implementation scheme does not disturb the correctness of the first constraint for the multiplication phase since each scalar multiply operation incurs a scalar addition operation as follows: $c^x_{i,j} \leftarrow c^x_{i,j} + a_{i,x} \cdot b_{x,j}$. However, it disturbs the correctness of the second constraint for the summation phase. Nevertheless, for this efficient implementation scheme, the second constraint can still be used to enforce balancing the local computations during the summation phase because similar errors are expected to occur for the second

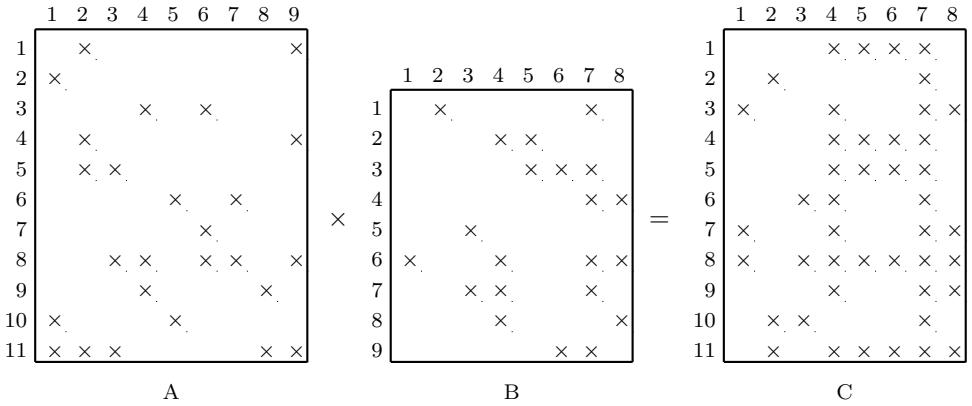

FIG. 2. *A sample SpGEMM computation* $C = A \times B$.

weights of the vertices in the different parts of a partition.

For (b), consider an output vertex $v_{i,j}$ assigned to $\mathcal{V}_k$ (i.e., $v_{i,j} \in \mathcal{V}_k$). Since each net $n_{i,j}$ connects exactly one output vertex, which is $v_{i,j} \in \mathcal{V}_k$, each part $\mathcal{V}_m \in \Lambda(n_{i,j}) - \{\mathcal{V}_k\}$ contains at least one input vertex corresponding to an outer-product computation that contributes to $c_{i,j}$. So, for each part $\mathcal{V}_m \in \Lambda(n_{i,j}) - \{\mathcal{V}_k\}$, processor P_m accumulates a partial result $c_{i,j}^{(m)} = \sum_{v_x \in \mathcal{V}_m} c_{i,j}^{x}$ from the results of its local outer-product computations and sends $c_{i,j}^{(m)}$ to processor P_k. Hence, $v_{i,j} \in \mathcal{V}_k$ means that processor P_k will receive a single partial result from each one of the $\lambda(n_{i,j}) - 1$ processors in $\Lambda(n_{i,j}) - \{\mathcal{V}_k\}$ and accumulate these partial results to compute the final result for $c_{i,j}$. As seen in (2.2), the contribution of this net to the cutsize is equal to $\lambda(n_{i,j}) - 1$. Therefore, we have the equivalence between $\lambda(n_{i,j}) - 1$ and the communication volume regarding the accumulation of $c_{i,j}$ in the summation phase. Consequently, the cutsize given in (2.2) correctly encodes the total communication volume during this summation phase.

Figure 2 shows a sample SpGEMM computation $C = A \times B$, where A and B are 11 by 9 and 9 by 8 matrices with 26 and 21 nonzeros, respectively, and C is an 11 by 8 matrix with 44 nonzeros. Figure 3 shows the hypergraph model $\mathcal{H}_{\mathrm{E}}(A, B)$ that represents the sample SpGEMM computation instance given in Figure 2. In Figure 3, circles and triangles show the input and output vertices, respectively. As seen in the figure, $\mathcal{H}_{\mathrm{E}}$ contains $9 + 44 = 53$ vertices. As also seen in the figure, $deg(v_4) = 6$ since $nnz(a_{*,4}) \cdot nnz(b_{4,*}) = 3 \cdot 2 = 6$. $\mathcal{H}_{\mathrm{E}}$ contains $\sum_{x=1}^{9} deg(v_x) = 61$ pins.

Figure 3 also shows a 3-way partition $\Pi(\mathcal{V})$ of $\mathcal{H}_{\mathrm{E}}$, and Figure 4 shows the 3-way partition of the sample input and output matrices induced by this $\Pi(\mathcal{V})$. In $\Pi(\mathcal{V})$, $W_1(\mathcal{V}_2) = w_1(v_4) + w_1(v_6) + w_1(v_8) = deg(v_4) + deg(v_6) + deg(v_8) = 6 + 12 + 4 = 22$. Similarly, $W_1(\mathcal{V}_1) = 14$ and $W_1(\mathcal{V}_3) = 25$. So $\Pi(\mathcal{V})$ incurs a percent load imbalance value of 23% on the first vertex weights. In $\Pi(\mathcal{V})$, $W_2(\mathcal{V}_1) = 13$, $W_2(\mathcal{V}_2) = 14$, and $W_2(\mathcal{V}_3) = 17$ since parts $\mathcal{V}_1$, $\mathcal{V}_2$, and $\mathcal{V}_3$ contain 13, 14, and 17 output vertices (triangles). So $\Pi(\mathcal{V})$ incurs a percent load imbalance value of 16% on the second vertex weights.

In the 3-way partition $\Pi(\mathcal{V})$ given in Figure 3, there are only four cut nets $n_{8,7}$, $n_{8,4}$, $n_{11,7}$, and $n_{11,4}$, whereas all the remaining 40 nets are internal. As seen in the figure, net $n_{8,7}$ has two pins in each vertex part, and hence $\lambda(n_{8,7}) = 3$. Consequently, cut net $n_{8,7}$ incurs a cost of $\lambda(n_{8,7}) - 1 = 3 - 1 = 2$ to the cutsize. Since $v_{8,7} \in \mathcal{V}_1$,

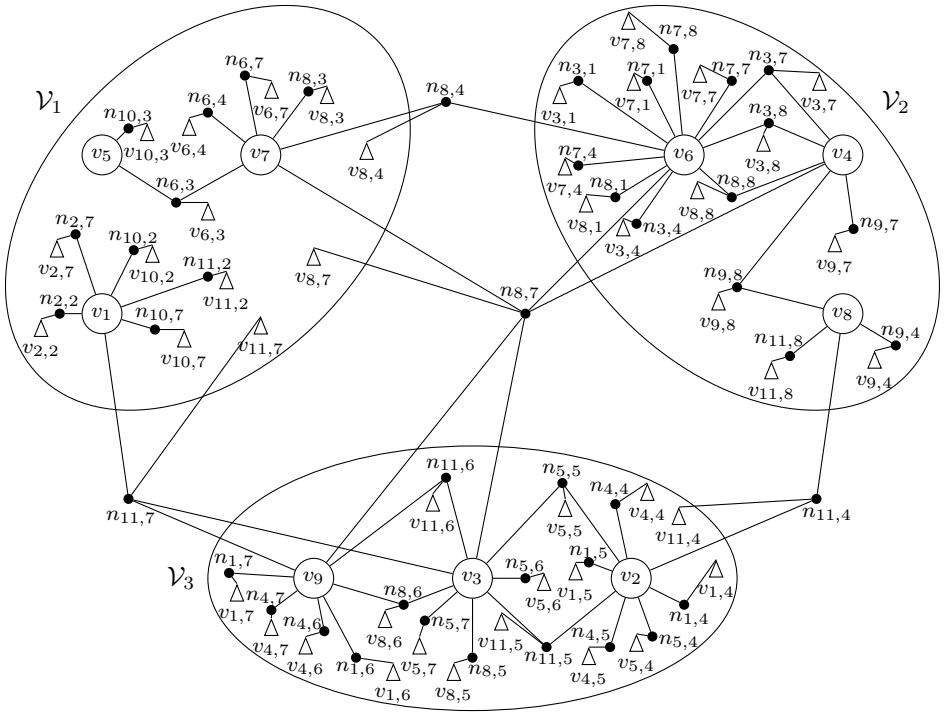

FIG. 3. *Hypergraph model $\mathcal{H}_{\mathrm{E}}(A,B)$ that represents the sample SpGEMM computation instance given in Figure 2 and its 3-way partition $\Pi(\mathcal{V})$. In the figure, each circular vertex v_x represents outer-product computation $a_{*,x}\times b_{x,*}$, and each triangle vertex $v_{i,j}$ represents the task of computing the final result for nonzero $c_{i,j}$ of matrix C. Each net $n_{i,j}$ represents the multiway relation for the computation of $c_{i,j}$ from the results of outer-product computations.*

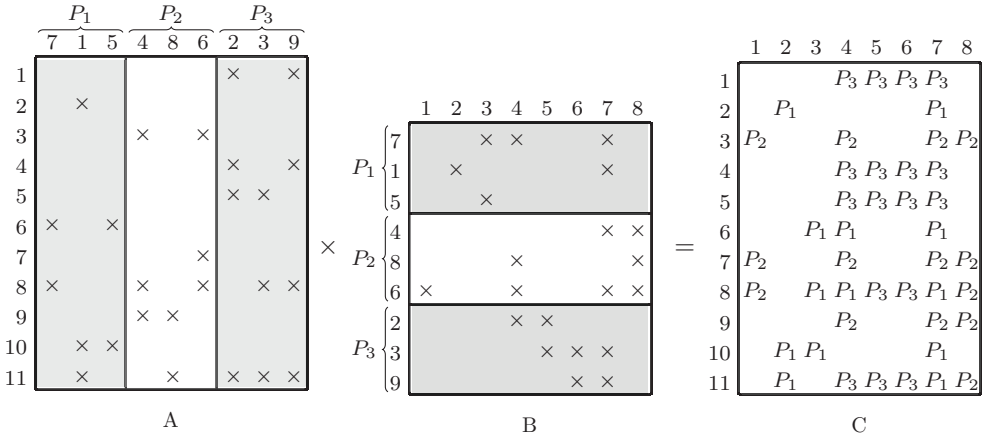

FIG. 4. *Matrices A, B, and C partitioned according to the partition $\Pi(\mathcal{V})$ of $\mathcal{H}_{\mathrm{E}}(A,B)$ given in Figure 3.*

processor P_1 is responsible for accumulating the partial nonzero results obtained from the outer-product computations. P_2 will send the partial result $c^{(2)}_{8,7} = c^4_{8,7} + c^6_{8,7}$ to P_1,

and P_3 will send the partial result $c_{8,7}^{(3)} = c_{8,7}^3 + c_{8,7}^9$ to P_1. Hence, accumulation of $c_{8,7}$ by P_1 will incur a communication cost of two words. Therefore, we have the equivalence between $\lambda(n_{8,7}) - 1$ and the communication volume regarding the accumulation of $c_{8,7}$ in the summation phase. Similarly, since $\lambda(n_{8,4}) - 1 = 1$, $\lambda(n_{11,7}) - 1 = 1$, and $\lambda(n_{11,4}) - 1 = 1$ for the other cut nets, the total cutsize is five. So the total communication volume is five words. Consequently, the cutsize given in (2.2) correctly encodes the total communication volume during this summation phase.

3.2. Extended hypergraph models for 1D output matrix partitioning. In this subsection, we describe how to extend the elementary hypergraph model $\mathcal{H}_{\mathrm{E}} = (\mathcal{V}^{AB} \cup \mathcal{V}^{C}, \mathcal{N})$ to $\mathcal{H}_{\mathrm{ext}}^{\mathrm{row}} = (\mathcal{V}^{AB} \cup \mathcal{V}_{\mathrm{row}}^{C}, \mathcal{N})$ and $\mathcal{H}_{\mathrm{ext}}^{\mathrm{col}} = (\mathcal{V}^{AB} \cup \mathcal{V}_{\mathrm{col}}^{C}, \mathcal{N})$ for 1D rowwise and 1D columnwise partitioning of the output matrix C, respectively. Both $\mathcal{H}_{\mathrm{ext}}^{\mathrm{row}}$ and $\mathcal{H}_{\mathrm{ext}}^{\mathrm{col}}$ have the same nets as $\mathcal{H}_{\mathrm{E}}$. Both $\mathcal{H}_{\mathrm{ext}}^{\mathrm{row}}$ and $\mathcal{H}_{\mathrm{ext}}^{\mathrm{col}}$ have the same input vertices as $\mathcal{H}_{\mathrm{E}}$. So the extended hypergraphs differ from the elementary hypergraph only in the output vertices. The output vertex subset $\mathcal{V}_{\mathrm{row}}^{C}$ of $\mathcal{H}_{\mathrm{ext}}^{\mathrm{row}}$ contains one vertex for each row of matrix C, and similarly, the output vertex subset $\mathcal{V}_{\mathrm{col}}^{C}$ of $\mathcal{H}_{\mathrm{ext}}^{\mathrm{col}}$ contains one vertex for each column of matrix C, whereas $\mathcal{V}^{C}$ of $\mathcal{H}_{\mathrm{E}}$ contains one vertex for each nonzero of matrix C.

$\mathcal{H}_{\mathrm{ext}}^{\mathrm{row}}$ is obtained from $\mathcal{H}_{\mathrm{E}}$ by amalgamating the output vertices of $\mathcal{V}^{C}$ that represent the nonzeros at the same row of matrix C into a single output vertex of $\mathcal{V}_{\mathrm{row}}^{C}$. The net set of the resulting composite vertex is set to the union of the nets of its constituent vertices. In other words, we amalgamate the output vertices $\bigcup_{\{j:v_{i,j} \in \mathcal{V}^C\}} v_{i,j}$ of $\mathcal{V}^{C}$ into $v_{i,*}$ of $\mathcal{V}_{\mathrm{row}}^{C}$ so that

$$Nets(v_{i,*}) = \bigcup_{\{j:v_{i,j} \in \mathcal{V}^C\}} Nets(v_{i,j}) = \{n_{i,j} : c_{i,j} \text{ is a nonzero at row } i \text{ of } C\}. \tag{3.7}$$

Note that net lists of input vertices of $\mathcal{H}_{\mathrm{ext}}^{\mathrm{row}}$ remain the same as those of $\mathcal{H}_{\mathrm{E}}$. The weights of an amalgamated vertex are set to be equal to the sum of weights of its constituent vertices, i.e.,

$$w_1(v_{i,*}) = 0, \qquad w_2(v_{i,*}) = \sum_{\{j:v_{i,j} \in \mathcal{V}^C\}} \Big(|Pins(n_{i,j})| - 1 \Big). \tag{3.8}$$

$\mathcal{H}_{\mathrm{ext}}^{\mathrm{col}}$ is obtained from $\mathcal{H}_{\mathrm{E}}$ by adopting a dual output vertex amalgamation scheme which amalgamates the output vertices that represent the nonzeros at the same column of matrix C. So the discussion about how $\mathcal{H}_{\mathrm{ext}}^{\mathrm{col}}$ can be obtained from $\mathcal{H}_{\mathrm{E}}$ directly follows from the discussion given above for obtaining $\mathcal{H}_{\mathrm{ext}}^{\mathrm{row}}$ from $\mathcal{H}_{\mathrm{E}}$.

The sizes of extended hypergraphs reduce only in the number of vertices compared to the elementary hypergraph model. This reduction can be obtained via replacing $nnz(C)$ in (3.2) with $rows(C)$ for $\mathcal{H}_{\mathrm{ext}}^{\mathrm{row}}$ and $cols(C)$ for $\mathcal{H}_{\mathrm{ext}}^{\mathrm{col}}$.

The output vertex amalgamation scheme adopted in obtaining $\mathcal{H}_{\mathrm{ext}}^{\mathrm{row}}$ from $\mathcal{H}_{\mathrm{E}}$ refers to the fact that vertex $v_{i,*}$ represents the task of computing the final results for all nonzeros in row i of C. Similarly for $\mathcal{H}_{\mathrm{ext}}^{\mathrm{col}}$, vertex $v_{*,j}$ represents the task of computing the final results for all nonzeros in column j of C. Figures 5 and 6 illustrate the input and output dependency views of the extended hypergraph models $\mathcal{H}_{\mathrm{ext}}^{\mathrm{row}}$ and $\mathcal{H}_{\mathrm{ext}}^{\mathrm{col}}$, respectively.

A partition on the input vertices of $\mathcal{H}_{\mathrm{ext}}^{\mathrm{row}}/\mathcal{H}_{\mathrm{ext}}^{\mathrm{col}}$ induces the same partition on the input vertices of $\mathcal{H}_{\mathrm{E}}$ since both $\mathcal{H}_{\mathrm{ext}}^{\mathrm{row}}$ and $\mathcal{H}_{\mathrm{ext}}^{\mathrm{col}}$ have the same input vertices as $\mathcal{H}_{\mathrm{E}}$.

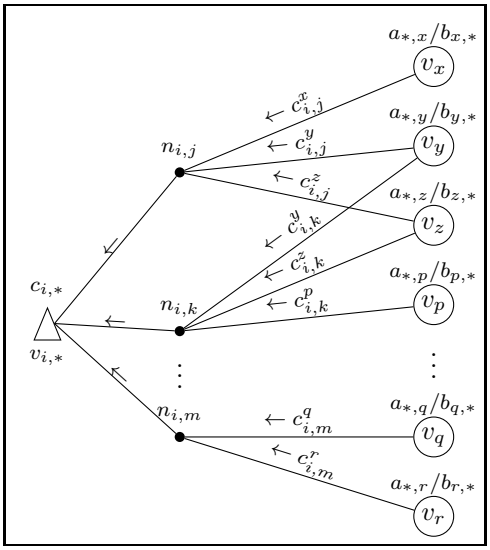

FIG. 5. *Extended hypergraph model* $\mathcal{H}^{\text{row}}_{\text{ext}}$ *for* 1*D rowwise output partitioning.*

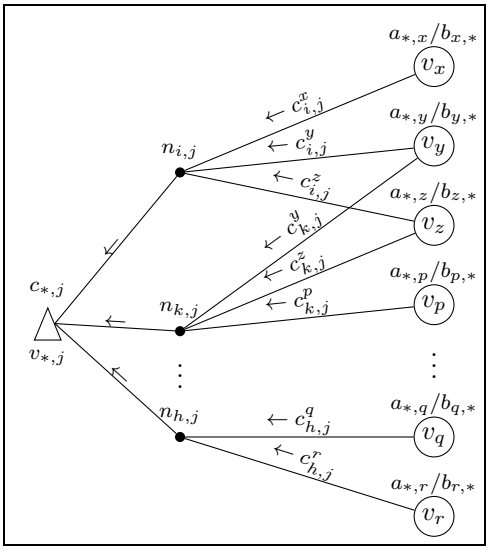

FIG. 6. *Extended hypergraph model* $\mathcal{H}^{\text{col}}_{\text{ext}}$ *for* 1*D columnwise output partitioning.*

A partition on the output vertices of $\mathcal{H}^{\text{row}}_{\text{ext}}/\mathcal{H}^{\text{col}}_{\text{ext}}$ induces a partition on the output vertices of $\mathcal{H}_{\text{E}}$, where all output vertices representing the C-matrix nonzeros in a row/column are restricted to be in the same part. Hence, the correctness of both extended hypergraph models directly follow from the correctness of the elementary hypergraph model.

4. Experiments.

4.1. Data sets. Three realistic categories as well as a synthetic category of SpGEMM instances are used for performance evaluation. For the first realistic category, we selected three G matrices from a FETI type domain decomposition application. The matrices `feti-B02` and `feti-B03` belong to car engine block simulations, whereas `feti-box512` belongs to an academic benchmark. For the second category, we selected two test matrices, which are obtained from the simulation of H_2O molecules via using CP2K's implementation of Kohn–Sham density functional theory calculations. The matrices `cp2k-h2o-e6` and `cp2k-h2o-.5e7` are obtained for cut-off values of 10^{-6} and $0.5\ 10^{-7}$, respectively. For the last realistic category, five LP constraint matrices are obtained from the University of Florida Sparse Matrix Collection [18].

The synthetic category contains five SpGEMM computation instances obtained by selecting sparse matrices from the University of Florida Sparse Matrix Collection [18]. Two SpGEMM instances are of the form AA and three SpGEMM instances are of the form AB, where A and B matrices are conformable for multiplication. The reason behind including the latter three SpGEMM instances is to show that the proposed HP formulations can handle the partitioning of two input sparse matrices with different sparsity patterns.

Tables 1 and 2, respectively, display the properties of the input and output test matrices involved in the 15 SpGEMM instances. In each category, the SpGEMM instances are listed in the order of increasing number of nonzeros ("nnz") of their output matrices. In the tables, "avg" and "max", respectively, denote the average and the maximum number of nonzeros per row/column.

TABLE 1
Properties of input matrices of SpGEMM instances.

Instance	Matrix	Number of rows	Number of cols	Number of nonzeros	nnz in row avg	nnz in row max	nnz in col. avg	nnz in col. max
FETI application ([3, 22])								
FETI1	feti-B02	612	152,826	920,433	1,504	3,044	6	15
FETI2	feti-box512	3,072	1,782,816	11,043,249	3,595	5,428	6	24
FETI3	feti-B03	6,084	472,320	3,004,692	494	1,076	6	30
CP2K application ([36])								
CP2K1	cp2k-h2o-e6	279,936	279,936	2,349,567	8	20	8	20
CP2K2	cp2k-h2o-.5e7	279,936	279,936	3,816,315	14	24	14	27
LP application ([7, 8, 26])								
LP1	fome21	67,748	216,350	465,294	7	96	2	3
LP2	pds-80	129,181	434,580	927,826	7	96	2	3
LP3	pds-100	156,243	514,577	1,096,002	7	101	2	3
LP4	sgpf5y6	246,077	312,540	831,976	3	61	3	12
LP5	cont11l	1,468,599	1,961,394	5,382,999	4	5	3	7
Synthetic application ([18])								
SYN1	darcy003	389,874	389,874	2,101,242	5	7	5	7
	mario002	389,874	389,874	2,101,242	5	7	5	7
SYN2	thermomechdK	204,316	204,316	2,846,228	14	20	14	10
	thermomechdM	204,316	204,316	1,423,116	7	10	7	10
SYN3	crashbasis	160,000	160,000	1,750,416	11	11	11	11
	majorbasis	160,000	160,000	1,750,416	11	11	11	18
SYN4	netherlandsosm	2,216,688	2,216,688	4,882,476	2	7	2	7
SYN5	tmtsym	726,713	726,713	5,080,961	7	9	7	9

TABLE 2
Properties of output matrices of SpGEMM instances.

Instance	Number of rows/cols	Number of nonzeros	nnz in row avg	nnz in row max	nnz in col. avg	nnz in col. max
FETI application ([3, 22])						
FETI1	612	19,088	31	70	31	70
FETI2	3,072	255,552	83	135	83	135
FETI3	6,084	258,816	43	140	43	140
CP2K application ([36])						
CP2K1	279,936	7,846,956	28	50	28	50
CP2K2	279,936	17,052,039	61	99	61	103
LP application ([7, 8, 26])						
LP1	67,748	640,240	9	97	9	97
LP2	129,181	1,249,074	10	97	10	97
LP3	156,243	1,470,688	9	102	9	102
LP4	246,077	2,776,645	11	367	11	367
LP5	1,468,599	18,064,261	12	23	12	23
Synthetic application ([18])						
SYN1	389,874	6,449,598	17	19	17	19
SYN2	204,316	7,874,148	39	52	39	52
SYN3	160,000	8,243,392	52	52	52	68
SYN4	2,216,688	8,755,758	4	16	4	16
SYN5	726,713	14,503,181	20	25	20	25

4.2. Experimental setup. The state-of-the-art serial HP tool PaToH [13], which supports multiple constraints, is used for partitioning the hypergraph models of the test SpGEMM instances. PaToH utilizes recursive bipartitioning paradigm

for obtaining multiway hypergraph partitions. For hypergraph bipartitioning, it utilizes a successive multilevel paradigm that contains coarsening, initial bipartitioning, and uncoarsening phases [12, 13]. PaToH is used with the `PATOH_SUGPARAM_SPEED` parameter, which is reported in the manual [13] as producing reasonably good bipartitions faster than the default parameter. This parameter establishes a trade-off between the solution quality and bipartitioning time by utilizing absorption clustering using pins in the coarsening phase that leads to a smaller number of levels and boundary FM for faster refinement in the uncoarsening phase. Since PaToH contains randomized algorithms, the results are reported by averaging the values obtained in three different runs, each randomly seeded. The allowed imbalance threshold ϵ is set to be equal to 0.10. For each test SpGEMM instance, a parallel SpGEMM instance is experimented for each value of $K = 32$, 64, 128, 256, 512, and 1024 for a parallel system with K processors.

In order to confirm the validity of locality preserving task partitioning achieved by the proposed hypergraph models, we implemented a binpacking (BP)-based method as a baseline algorithm which considers only load balancing. BP adapts the best-fit-decreasing heuristic used in solving the K-feasible BP problem [24] to balance computational and communication loads separately in two steps. For computational load balancing in the multiplication phase, outer-product tasks associated with the input matrices are assigned to K processors in decreasing multiplication cost (i.e., $w_1(v_x)$ shown in (3.5)). For communication load balancing in the summation phase, accumulation tasks associated with output matrix entries are assigned to K processors in decreasing summation cost (i.e., $w_2(v_x)$ shown in (3.6)). In task assignment for both phases, the best-fit criterion corresponds to assigning a task to the processor that currently has the minimum load in the respective phase.

For evaluating the actual performance of the proposed hypergraph models, we have developed a two-phase outer-product–parallel SpGEMM library [4] in C language. Using this SpGEMM library, we ran experiments on two different parallel systems.

The first system is IBM BlueGene/Q, called JUQUEEN, located in Germany at the Jülich Supercomputing Centre. The network of the system has five-dimensional torus topology. One node of the BlueGene/Q system consists of 16 IBM PowerPC A2 cores that run at 1.6 GHz. These 16 cores share 16 GB of RAM. We used 8 processes per node for memory considerations. We used the IBM XL compiler suite with `O2` flag and BlueGene/Q's MPI implementation, which is based on MPICH2 (version 1.5) on JUQUEEN.

The second system is SuperMUC, which is an Intel-based cluster located in Germany at the Leibniz Supercomputing Centre. The system uses Infiniband FDR10, which is based on nonblocking tree topology. One node of the system consists of two Sandy Bridge Intel Xeon E5-2680 processors, each with 8 cores that run at 2.7 GHz. Each node is equipped with 32 GB of RAM. We used 16 processes per node. We used GCC and MPICH2 (version 1.4) on SuperMUC.

Parallel timing results on both systems are measured by averaging 10 successive SpGEMM computations performed after a warm-up period of three SpGEMM computations. Speedup values are computed against run times of our implementation of Gustavson's sequential algorithm [21] on a single core of the respective system.

4.3. Performance evaluation. The performance of the proposed models is evaluated in terms of the speedup values measured on JUQUEEN and SuperMUC as well as the partitioning quality values of communication overhead and load balancing

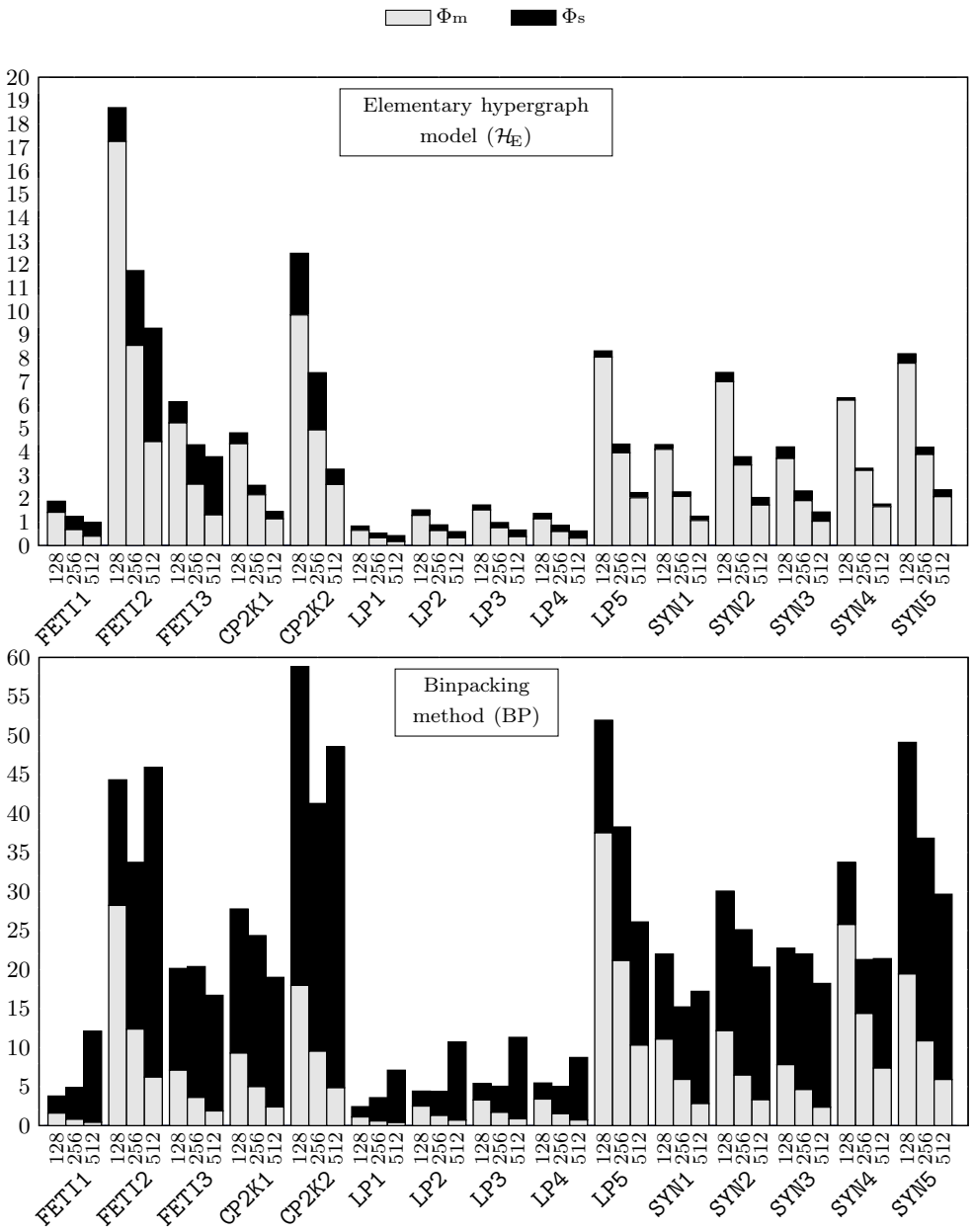

FIG. 7. *Dissection of parallel SpGEMM times on JUQUEEN (in milliseconds) into two phases* Φ_m *(multiplication) and* Φ_s *(summation) for* $K = 128, 256,$ *and* 512 *processors.*

measured only on JUQUEEN.

4.3.1. Importance of locality-preserving task partitioning. Figure 7 displays the dissection of parallel SpGEMM times on JUQUEEN into multiplication phase Φ_m and summation phase Φ_s to compare the performance of locality preserving $\mathcal{H}_E$ against the baseline algorithm BP, both of which produce 2D nonzero-based output matrix partition. Barrier synchronization is used between the two phases for

the sake of displaying these dissection results.

As seen in Figure 7, $\mathcal{H}_{\mathrm{E}}$ attains significantly smaller runtime than BP in Φ_{s} of all parallel SpGEMM instances, and this runtime difference increases with increasing number of processors in favor of $\mathcal{H}_{\mathrm{E}}$. Since communication is involved only in Φ_{s}, this significant difference shows the importance of communication in the performance of parallel SpGEMM, as well as confirming the validity of the $\mathcal{H}_{\mathrm{E}}$ model in reducing the communication overhead. As also seen in the figure, $\mathcal{H}_{\mathrm{E}}$ also attains considerably smaller runtime than BP in Φ_{m}. This is because locality-preserving partitions produced by $\mathcal{H}_{\mathrm{E}}$ achieve better temporal locality in local partial-result accumulation operations performed during Φ_{m}. Speedup curves displayed in Figures 8 and 9 show that $\mathcal{H}_{\mathrm{E}}$ attains significantly higher speedups than BP and the performance gap between $\mathcal{H}_{\mathrm{E}}$ and BP increases significantly with increasing number of processors in favor of $\mathcal{H}_{\mathrm{E}}$.

4.3.2. Load balancing. For experimental load balancing performance evaluation, we report percent load imbalance values $LI(\Phi m)$ and $LI(\Phi s)$ measured for Φ_{m} and Φ_{s}, respectively. Barrier synchronization is also used between the two phases for measuring $LI(\Phi m)$ and $LI(\Phi s)$ on JUQUEEN.

As seen in Table 3, $LI(\Phi m)$ values are below the allowed imbalance ratio of 10% for 32 out of 45 parallel SpGEMM instances, and they are slightly above 10% for the remaining 13 instances. On the average, $LI(\Phi m)$ values are 6.6%, 7.5%, and 9.0% for 128, 256, and 512 processors, respectively. This shows the validity of the first weighting scheme used in the two-constraint formulation.

As seen in Table 3, $LI(\Phi s)$ values are observed to be considerably higher than the allowed imbalance ratio of 10%. On the average, $LI(\Phi s)$ values are 49.9%, 64.9%, and 64,4% for 128, 256, and 512 processors, respectively. There are five extreme instances in which $LI(\Phi s)$ values are more than 100%. This may stem from two factors: The first factor is because of the adverse effect of the efficient implementation scheme on the correctness of the proposed models, as discussed in section 3.1.1. Recall that this efficient implementation scheme makes local accumulations on the fly during the multiplication phase, whereas the second constraint in the proposed model tries to enforce load balance on the accumulation operations assuming that all of the accumulations are to be performed naively in the summation phase. The second one is the fact that although the proposed models correctly encode the minimization of the total communication volume, they cannot encode balancing the communication loads of the processors.

For the efficient implementation scheme, in a partition of the proposed hypergraph models, only the boundary output vertices of the parts incur computation and hence communication in Φ_{s}. Computational and communication requirements of the output vertices during Φ_{s} cannot be statically determined prior to partitioning since they vary with varying states (being boundary or internal) of these vertices during partitioning. So balancing computational and communication loads of processors during Φ_{s} cannot be envisioned since current partitioners do not handle such state-dependent vertex weighting.

4.3.3. Reducing communication overhead. Two different metrics are identified for reporting communication overhead values: The first metric is the average number of floating-point words communicated per 1000 nonzeros of the respective output matrix, whereas the second metric is the average number of floating-point words communicated per Kflops. The second metric is included to give insight about the computational granularity attained through partitioning. Although the main

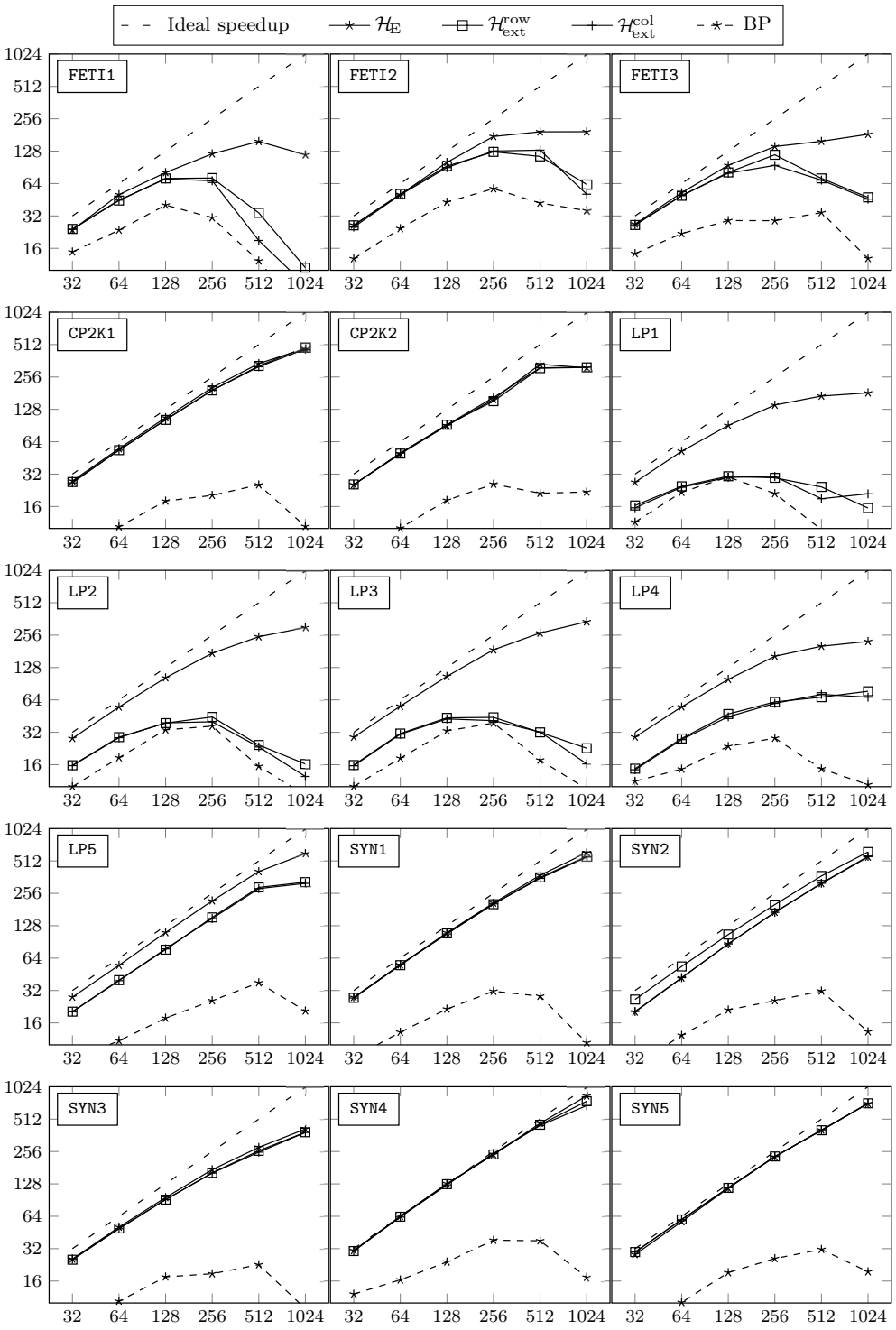

FIG. 8. *Speedup curves on JUQUEEN.*

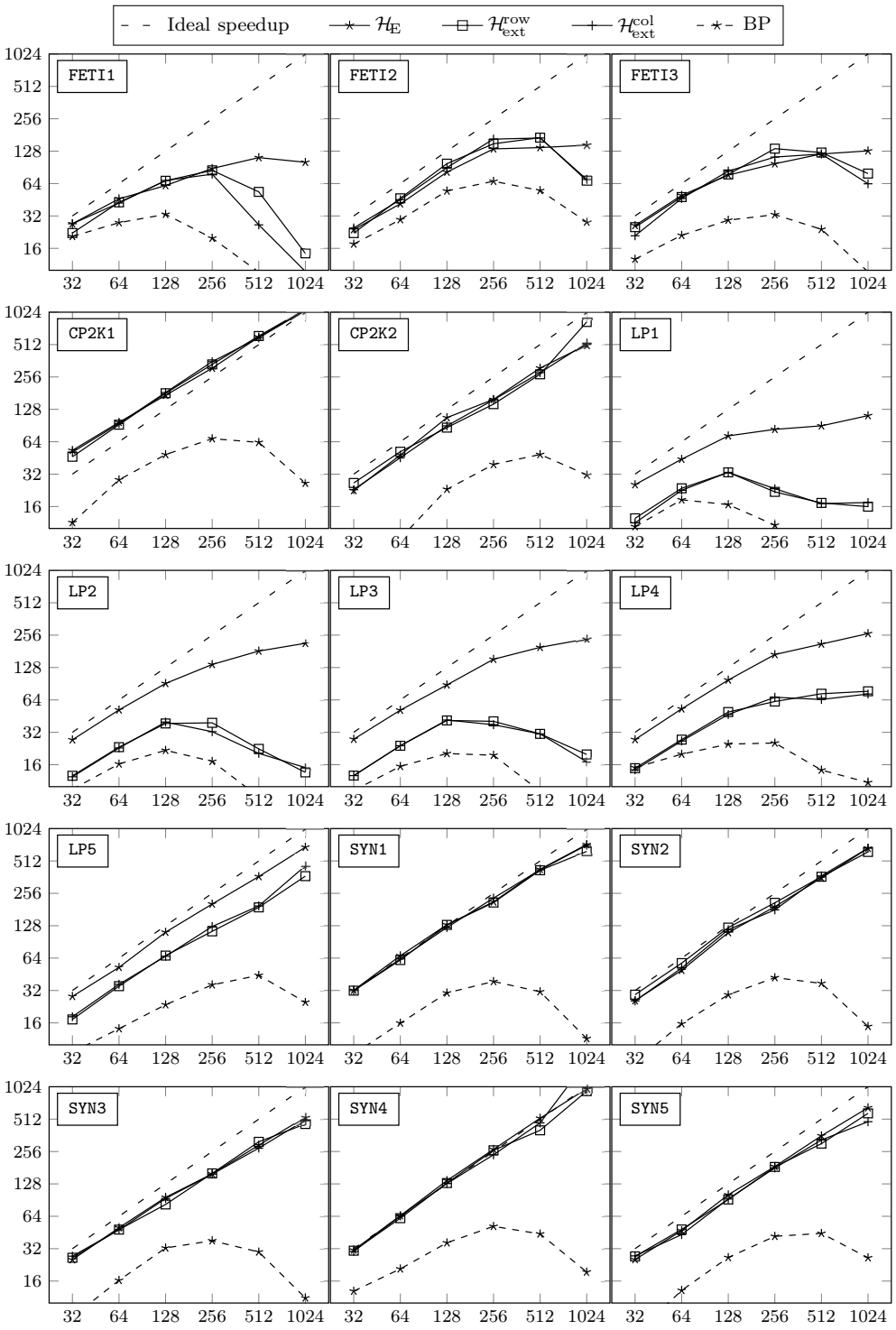

FIG. 9. *Speedup curves on SuperMUC.*

TABLE 3
Performance results obtained on JUQUEEN for the elementary hypergraph model ($\mathcal{H}_E$).

		Measured imbalance		#of mssg's per proc.		Comm. vol. per processor		Speedup Kflops	
Instance	K	$LI(\Phi m)$	$LI(\Phi s)$	avg	max	avg	max		S_K
FETI application ([3, 22])									
	128	7.4%	35.5%	80.8	115	13.58	27.45	0.27	81
FETI1	256	8.4%	83.9%	71.2	143	8.82	29.23	0.35	121
	512	23.3%	77.0%	59.8	158	6.42	23.16	0.51	157
	128	8.2%	20.5%	127.0	127	17.54	28.90	0.36	101
FETI2	256	9.8%	21.2%	247.5	255	10.26	18.95	0.42	175
	512	10.8%	33.1%	372.6	510	5.52	12.96	0.45	194
	128	5.7%	23.7%	122.1	127	6.19	11.77	0.47	95
FETI3	256	7.1%	43.9%	172.8	226	3.16	6.26	0.48	141
	512	7.5%	79.1%	167.6	278	2.14	4.71	0.65	158
CP2K application ([36])									
	128	5.7%	42.2%	18.6	37	0.26	0.36	0.54	107
CP2K1	256	4.9%	48.9%	19.1	33	0.17	0.26	0.70	204
	512	5.2%	50.5%	18.9	34	0.11	0.18	0.92	341
	128	5.0%	98.2%	19.4	33	0.54	0.79	0.95	90
CP2K2	256	5.3%	110.2%	20.7	38	0.36	0.51	1.25	159
	512	5.4%	53.3%	20.9	37	0.24	0.37	1.67	336
LP application ([7, 8, 26])									
	128	10.0%	38.3%	21.0	34	0.33	0.54	1.18	91
LP1	256	11.2%	50.3%	24.9	41	0.22	0.39	1.60	140
	512	13.7%	55.5%	25.7	50	0.15	0.27	2.13	170
	128	11.0%	35.1%	20.2	34	0.26	0.43	0.92	103
LP2	256	10.4%	39.7%	24.8	46	0.18	0.31	1.27	174
	512	12.0%	51.2%	27.4	52	0.12	0.23	1.68	247
	128	11.6%	35.6%	20.1	34	0.23	0.39	0.83	106
LP3	256	12.6%	40.6%	24.1	44	0.16	0.28	1.14	187
	512	11.8%	65.1%	26.5	51	0.11	0.19	1.53	268
	128	5.8%	136.7%	11.9	31	0.04	0.08	0.18	99
LP4	256	8.2%	157.1%	13.7	38	0.03	0.07	0.28	163
	512	8.8%	159.8%	16.2	48	0.03	0.07	0.50	202
	128	5.1%	53.5%	7.7	15	0.07	0.12	0.28	111
LP5	256	4.5%	149.7%	8.1	20	0.05	0.10	0.40	218
	512	4.4%	63.8%	8.1	18	0.04	0.06	0.56	408
Synthetic application ([18])									
	128	4.2%	62.0%	6.6	13	0.14	0.28	0.42	110
SYN1	256	6.4%	77.0%	7.1	17	0.10	0.21	0.60	208
	512	7.3%	61.2%	7.2	16	0.07	0.14	0.86	377
	128	6.3%	69.2%	6.5	14	0.33	0.62	0.76	87
SYN2	256	7.4%	84.4%	6.9	17	0.25	0.43	1.13	171
	512	8.8%	80.3%	7.4	19	0.18	0.32	1.66	316
	128	4.9%	65.5%	7.3	17	0.50	0.87	1.25	96
SYN3	256	6.2%	65.5%	8.1	20	0.37	0.64	1.87	174
	512	7.2%	62.0%	9.2	22	0.29	0.49	2.90	280
	128	8.5%	60.2%	6.6	13	0.01	0.02	0.04	129
SYN4	256	12.7%	67.4%	7.5	16	0.01	0.01	0.06	239
	512	14.4%	64.8%	7.7	17	0.00	0.01	0.09	468
	128	4.3%	70.2%	7.6	16	0.20	0.35	0.46	117
SYN5	256	4.8%	66.8%	7.8	17	0.14	0.25	0.64	228
	512	7.5%	66.8%	7.9	19	0.10	0.18	0.92	406

TABLE 4

Average performance comparison of three hypergraph models on JUQUEEN (a bold value indicates the highest speedup value attained by the respective model).

		$\mathcal{H}_{\mathrm{E}}$				$\mathcal{H}_{\mathrm{ext}}^{\mathrm{row}}$				$\mathcal{H}_{\mathrm{ext}}^{\mathrm{col}}$			
		Measured imbalance $LI(\cdot)$		Com. vol. per		Measured imbalance $LI(\cdot)$		Com. vol. per		Measured imbalance $LI(\cdot)$		Com. vol. per	
Category	K	Φ_m	Φ_s	Kflops	S_K	Φ_m	Φ_s	Kflops	S_K	Φ_m	Φ_s	Kflops	S_K
	128	7.0%	25.8%	0.35	**92**	9.9%	354.0%	0.51	81	9.1%	360.6%	0.51	80
FETI	256	8.4%	42.7%	0.41	**144**	13.9%	733.0%	0.67	102	16.7%	865.0%	0.66	94
	512	12.4%	58.6%	0.53	**169**	30.5%	2,290.4%	0.71	65	25.7%	2,878.7%	0.70	55
	128	5.3%	64.4%	0.72	**98**	3.7%	36.3%	1.69	97	4.4%	38.6%	1.68	97
CP2K	256	5.1%	73.4%	0.94	**180**	4.5%	46.0%	2.18	171	4.3%	39.5%	2.15	178
	512	5.3%	51.9%	1.24	**338**	6.8%	48.2%	2.80	316	5.9%	48.2%	2.79	315
	128	8.2%	51.1%	0.54	**102**	11.6%	88.3%	2.47	45	11.4%	84.5%	2.48	44
LP	256	8.8%	71.8%	0.77	**175**	12.4%	139.1%	3.51	56	11.5%	139.3%	3.54	54
	512	9.4%	71.6%	1.09	**248**	13.4%	171.0%	4.72	52	13.8%	158.8%	4.69	49
	128	5.4%	65.3%	0.37	107	5.4%	65.1%	0.63	**109**	5.5%	58.6%	0.58	105
SYN	256	7.1%	71.8%	0.55	202	6.7%	75.2%	0.91	**205**	6.2%	71.8%	0.87	200
	512	8.7%	66.7%	0.80	363	8.8%	75.6%	1.36	**364**	9.0%	76.1%	1.28	348
	128	6.6%	49.9%	0.46	**101**	7.5%	93.5%	1.09	76	7.5%	90.1%	1.05	74
Overall	256	7.5%	64.9%	0.62	**177**	9.0%	136.4%	1.51	113	8.8%	136.1%	1.48	109
	512	9.0%	64.4%	0.86	**272**	12.6%	184.9%	1.99	132	12.1%	189.2%	1.94	124

objective of the proposed hypergraph models is the minimization of the total communication volume, the results for the other performance metrics, such as the maximum volume, average number, and maximum number of messages handled by a single processor, are also reported. In the reported results, the number of messages handled by a processor refers to the number of messages sent by that processor.

As seen in Table 3, the total communication volume remains below one floating-point word per Kflops for 31 out of 45 instances and above two words per Kflops for only 2 out of 45 instances. On the average, the total communication volume values are 0.46, 0.62, and 0.86 words per Kflops for 128, 256, and 512 processors, respectively.

Dissection results for $\mathcal{H}_{\mathrm{E}}$ displayed in Figure 7 are in conformance with these results, as they show that processors spend much less time in communication during Φ_{s} compared to computation in Φ_{m} in most of the instances. These experimental findings show that the proposed HP formulations successfully attain coarse grain parallel SpGEMM instances.

4.3.4. Comparison of three hypergraph models. Table 4 shows the performance result averages of the three proposed hypergraph models over the four categories as well as the overall averages. The averages are computed using geometric mean. As seen in Table 4, on the average, the elementary hypergraph model $\mathcal{H}_{\mathrm{E}}$ attains slightly better load balance than both extended hypergraph models $\mathcal{H}_{\mathrm{ext}}^{\mathrm{row}}$ and $\mathcal{H}_{\mathrm{ext}}^{\mathrm{col}}$ in Φ_{m}, whereas $\mathcal{H}_{\mathrm{E}}$ attains considerably better load balance than both $\mathcal{H}_{\mathrm{ext}}^{\mathrm{row}}$ and $\mathcal{H}_{\mathrm{ext}}^{\mathrm{col}}$ in Φ_{s}. $\mathcal{H}_{\mathrm{E}}$ incurs significantly less communication volume than both $\mathcal{H}_{\mathrm{ext}}^{\mathrm{row}}$ and $\mathcal{H}_{\mathrm{ext}}^{\mathrm{col}}$ for all parallel SpGEMM instances. These experimental findings are already expected since both $\mathcal{H}_{\mathrm{ext}}^{\mathrm{row}}$ and $\mathcal{H}_{\mathrm{ext}}^{\mathrm{col}}$ have smaller search space than $\mathcal{H}_{\mathrm{E}}$.

Despite the superiority of $\mathcal{H}_{\mathrm{E}}$ compared to $\mathcal{H}_{\mathrm{ext}}^{\mathrm{row}}/\mathcal{H}_{\mathrm{ext}}^{\mathrm{col}}$ in all of the above-mentioned partitioning quality metrics, $\mathcal{H}_{\mathrm{E}}$ cannot attain higher speedups than $\mathcal{H}_{\mathrm{ext}}^{\mathrm{row}}/\mathcal{H}_{\mathrm{ext}}^{\mathrm{col}}$ in all parallel SpGEMM instances, as seen in Figure 8. $\mathcal{H}_{\mathrm{E}}$ attains the highest speedups in 69 out of 90 parallel SpGEMM instances, and in 14 of the remaining 21 instances either $\mathcal{H}_{\mathrm{ext}}^{\mathrm{row}}$ or $\mathcal{H}_{\mathrm{ext}}^{\mathrm{col}}$ or both attain very close speedups to those by $\mathcal{H}_{\mathrm{E}}$. In the remaining 7 instances, $\mathcal{H}_{\mathrm{ext}}^{\mathrm{row}}$ in 6 instances and $\mathcal{H}_{\mathrm{ext}}^{\mathrm{col}}$ in one instance attain higher speedups than $\mathcal{H}_{\mathrm{E}}$.

As seen in Figures 8 and 9, on both JUQUEEN and SuperMUC, the speedup curves attained by $\mathcal{H}_{\text{ext}}^{\text{row}}$ and $\mathcal{H}_{\text{ext}}^{\text{col}}$ are very close to those by $\mathcal{H}_{\text{E}}$ in the following six instances: `CP2K1`, `CP2K2`, `SYN1`, `SYN3`, `SYN4`, and `SYN5` out of 15 instances. On SuperMUC, the speedup curves attained by $\mathcal{H}_{\text{ext}}^{\text{row}}$ and $\mathcal{H}_{\text{ext}}^{\text{col}}$ are very close to those by $\mathcal{H}_{\text{E}}$ up to 256 processors in `FETI2` and `FETI3` instances in addition to the these six instances. These experimental findings can be attributed to the fact that, in the extended hypergraph models, enforcing all output matrix nonzeros belonging to the same row or column to be assigned to the same processor has the potential of decreasing the number of messages. However, in the remaining instances, the speedup curves attained by $\mathcal{H}_{\text{E}}$ are significantly better than those by $\mathcal{H}_{\text{ext}}^{\text{row}}$ and $\mathcal{H}_{\text{ext}}^{\text{col}}$.

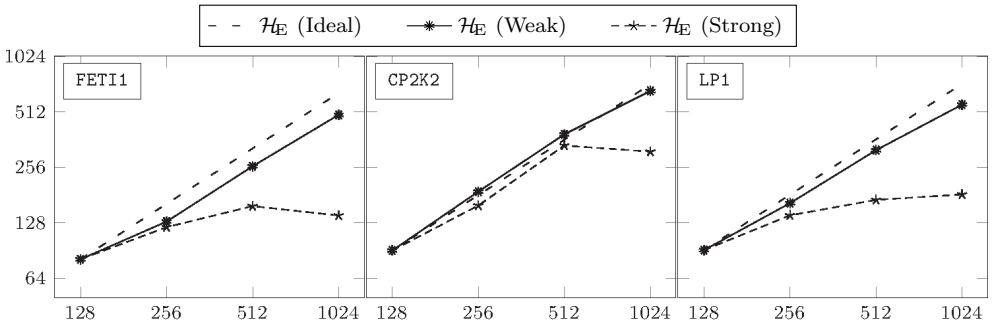

FIG. 10. *Speedup curves for weak and strong scaling on JUQUEEN.*

4.3.5. Weak scaling. Speedup curves given in Figures 8 and 9 effectively show the results of strong scaling experiments. We also conduct weak scaling experiments that keep processors' computational loads constant, as described below.

We double both the number of columns of A and the number of rows of B to obtain A' and B' matrices by replicating each column of A with a column with the same sparsity pattern and each row of B with a row with the same sparsity pattern. This replication scheme doubles the storage sizes for the input matrices and also doubles the problem size since the number of flops required to compute $C' = A' \times B'$ is exactly twice that of $C = A \times B$. This replication scheme generates A' and B' matrices whose sparsity patterns are similar to those of A and B, thus inducing SpGEMM instances $C' = A' \times B'$ and $C = A \times B$ that are expected to have similar communication patterns. Thus, we conduct linear weak scaling tests on JUQUEEN by running $C = A \times B$ on K processors and running $C' = A' \times B'$ on $K' = 2K$ processors starting from $K = 128$. Due to lack of space, we include speedup curves only for three matrices (`FETI1`, `CP2K2`, and `LP1`), one from each realistic category, in Figure 10. For the sake of comparison, Figure 10 also displays speedup curves of strong scaling tests. As seen in the figure, $\mathcal{H}_{\text{E}}$ achieves linear weak scaling from $K = 128$ to $K = 1024$ processors.

4.3.6. Amortization of partitioning overhead. We utilize the state-of-the-art parallel matrix multiplication library CombBLAS [10] for the purpose of justifying the preprocessing overhead incurred by the intelligent partitioning schemes proposed in this work. This is because CombBLAS does not utilize intelligent partitioning, neither for load balancing nor for reducing communication overhead, and thus it does not involve any preprocessing overhead.

For CombBLAS runs, we experiment with processor counts that are perfect squares, and we apply random permutation to the input matrices to balance the

TABLE 5
Average number of parallel SpGEMM computations required to amortize the partitioning overhead.

		Sequential HP			Parallel HP with 30% efficiency		
Category	K	$\mathcal{H}_{\mathrm{E}}$	$\mathcal{H}_{\mathrm{ext}}^{\mathrm{row}}$	$\mathcal{H}_{\mathrm{ext}}^{\mathrm{col}}$	$\mathcal{H}_{\mathrm{E}}$	$\mathcal{H}_{\mathrm{ext}}^{\mathrm{row}}$	$\mathcal{H}_{\mathrm{ext}}^{\mathrm{col}}$
FETI	64	57	58	58	2.95	3.03	3.04
	256	221	217	222	2.88	2.83	2.89
CP2K	64	22	18	22	1.17	0.94	1.16
	256	66	54	66	0.86	0.70	0.85
LP	64	46	40	44	2.39	2.08	2.27
	256	160	143	154	2.08	1.86	2.00
SYN	64	28	21	25	1.45	1.09	1.29
	256	96	74	86	1.25	0.96	1.12
Overall	64	37	31	35	1.92	1.63	1.82
	256	128	110	122	1.67	1.43	1.58

memory requirements and the computational loads of the processors. We use the `Mult_AnXBn_DoubleBuff` routine that uses the double buffered broadcasting scheme since it performs better than `Mult_AnXBn_Synch`.

Table 5 is given to evaluate the preprocessing overhead introduced by the HP models on SuperMUC. The values displayed in the table are computed through dividing the preprocessing time by the difference between the parallel run times of CombBLAS and our SpGEMM algorithm. The numerator of this ratio represents the preprocessing overhead due to partitioning, and the denominator represents the performance improvement obtained by using the proposed SpGEMM algorithm instead of CombBLAS for a single SpGEMM computation.

Although parallel HP tools [20, 34] exist in the literature, they do not support multiple balance constraints and thus could not be used in our proposed hypergraph models. So we adopt two approaches in computing amortization values on SuperMUC: In the first approach, we use the sequential running time of PaToH on a single core of SuperMUC. In the second approach, we use estimated parallel partitioning time by assuming an efficiency value of 0.30. The former and latter amortization values are identified as "Sequential HP" and "Parallel HP." The amortization values are displayed as averages over the four categories as well as the overall averages.

As seen in Table 5, in general, the extended hypergraph models amortize in a smaller number of SpGEMM computations compared to the elementary hypergraph model. This is due to the fact that partitioning times are considerably less in the extended hypergraph models than those of the elementary hypergraph model.

As seen in the "Parallel HP" columns of Table 5, for the `CP2K` and `SYN` instances, the use of proposed hypergraph models amortizes even for a single SpGEMM computation. For the `LP` instances, the use of proposed hypergraph models amortizes for only two interior-point method iterations. These experimental findings show that the proposed hypergraph models have the potential to have high impact in improving the performance of various parallel applications that involve SpGEMM computations.

5. Conclusion. We proposed three hypergraph models that achieve simultaneous partitioning of input and output matrices for two-phase outer-product–parallel sparse matrix-matrix multiplication (SpGEMM) of the form $C = A \times B$. All three hypergraph models contain a single vertex for representing the outer product of each column of A with the respective row of B to enforce conformable 1D columnwise and 1D rowwise partitioning of the input matrices A and B. This conformable partition-

ing enables communication-free local SpGEMM computations in the first phase. All models contain a hyperedge (net) for each nonzero of the output matrix C to encode the total message volume that will be transmitted during the accumulation of the local SpGEMM results in the second phase of the two-phase outer-product–parallel SpGEMM algorithm. The hypergraph models differ only in the definition of the output vertices. The first model contains a vertex for each nonzero of C to enable 2D nonzero-based output matrix partitioning, whereas the second and third models contain a vertex for each row/column of C to enforce 1D rowwise/columnwise output matrix partitioning. In all models, the two-constraint formulation encodes balancing computational loads of processors during the two separate phases of the parallel SpGEMM algorithm. The partitioning objective encodes minimizing the total volume of communication.

We tested the validity of the proposed data partitioning models and formulations on a wide range of large-scale SpGEMM computation instances. Experiments showed that the proposed hypergraph models and partitioning formulations achieve successful input and output partitioning of SpGEMM computations. In order to verify that the theoretical gains obtained by the models hold in practice, we developed a two-phase outer-product–parallel SpGEMM code utilizing the MPI library. Parallel SpGEMM runs on both JUQUEEN and SuperMUC showed that the proposed partitioning models attain high speedup values. The preprocessing overhead due to HP can be amortized by the parallel performance improvement over the state-of-the-art SpGEMM tool CombBLAS in a very few number of SpGEMM computations. In some practical instances, this preprocessing overhead is amortized by a single SpGEMM computation.

Acknowledgment. We would like to thank Tomas Kozubek for providing us the data sets in `FETI` category.

REFERENCES

[1] *CP2K home page*, http://www.cp2k.org/.
[2] *MKL*, http://software.intel.com/en-us/articles/intel-mkl/.
[3] *Total-FETI Massively Parallel Implementation Research Group*, http://spomech.vsb.cz/feti/.
[4] K. Akbudak and C. Aykanat, *Parallel Sparse Matrix-Matrix Multiplication Library*, Technical report BU-CE-1402, Computer Engineering Department, Bilkent University, Ankara, Turkey, 2014.
[5] G. Ballard, A. Buluc, J. Demmel, L. Grigori, B. Lipshitz, O. Schwartz, and S. Toledo, *Communication optimal parallel multiplication of sparse random matrices*, in Proceedings of the Twenty-Fifth Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'13), New York, NY, 2013, pp. 222–231.
[6] N. Bell and M. Garland, *CUSP: Generic Parallel Algorithms for Sparse Matrix and Graph Computations*, Version 0.1.0, 2010.
[7] R. H. Bisseling, T. M. Doup, and L. Loyens, *A parallel interior point algorithm for linear programming on a network of transputers*, Ann. Oper. Res., 43 (1993), pp. 51–86.
[8] E. Boman, O. Parekh, and C. Phillips, *LDRD Final Report on Massively-Parallel Linear Programming: The parPCx System*, Technical report, SAND2004-6440, Sandia National Laboratories, Albuquerque, NM, 2005.
[9] A. Buluc and J. Gilbert, *On the representation and multiplication of hypersparse matrices*, in Proceedings of the IEEE International Symposium on Parallel and Distributed Processing (IPDPS'08), 2008, pp. 1–11.
[10] A. Buluç and J. R. Gilbert, *The Combinatorial BLAS: Design, implementation, and applications*, Int. J. High Perform. Comput. Appl., 25 (2011), pp. 496–509.
[11] A. Buluç and J. R. Gilbert, *Parallel sparse matrix-matrix multiplication and indexing: Implementation and experiments*, SIAM J. Sci. Comput., 34 (2012), pp. C170–C191.
[12] Ü. V. Çatalyürek and C. Aykanat, *Hypergraph-partitioning based decomposition for paral-*

lel sparse-matrix vector multiplication, IEEE Trans. Parallel Distrib. Systems, 10 (1999), pp. 673–693.
[13] Ü. V. Çatalyürek and C. Aykanat, *PaToH: A Multilevel Hypergraph Partitioning Tool*, Version 3.0, Computer Engineering Department, Bilkent University, Ankara, Turkey, 1999.
[14] Ü. V. Çatalyürek, C. Aykanat, and B. Uçar, *On two-dimensional sparse matrix partitioning: Models, methods, and a recipe*, SIAM J. Sci. Comput., 32 (2010), pp. 656–683.
[15] M. Challacombe, *A simplified density matrix minimization for linear scaling self-consistent field theory*, J. Chem. Phys., 110 (1999), pp. 2332–2342.
[16] M. Challacombe, *A general parallel sparse-blocked matrix multiply for linear scaling SCF theory*, Comput. Phys. Comm., 128 (2000), pp. 93–107.
[17] A. D. Daniels, J. M. Millam, and G. E. Scuseria, *Semiempirical methods with conjugate gradient density matrix search to replace diagonalization for molecular systems containing thousands of atoms*, J. Chem. Phys., 107 (1997), pp. 425–431.
[18] T. A. Davis and Y. Hu, *The University of Florida sparse matrix collection*, ACM Trans. Math. Software, 38 (2011), 1.
[19] J. Demmel, D. Eliahu, A. Fox, S. Kamil, B. Lipshitz, O. Schwartz, and O. Spillinger, *Communication-optimal parallel recursive rectangular matrix multiplication*, in Proceedings of the 27th IEEE International Parallel Distributed Processing Symposium, 2013, pp. 261–272.
[20] K. Devine, E. Boman, R. Heaphy, B. Hendrickson, and C. Vaughan, *Zoltan data management services for parallel dynamic applications*, Comput. Sci. Eng., 4 (2002), pp. 90–97.
[21] F. G. Gustavson, *Two fast algorithms for sparse matrices: Multiplication and permuted transposition*, ACM Trans. Math. Software, 4 (1978), pp. 250–269.
[22] V. Hapla, D. Horák, and M. Merta, *Use of direct solvers in TFETI massively parallel implementation*, in Applied Parallel and Scientific Computing, Lecture Notes in Comput. Sci. 7782, P. Manninen and P. Öster, eds., Springer, Berlin, Heidelberg, 2013, pp. 192–205.
[23] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, et al., *An overview of the Trilinos project*, ACM Trans. Math. Software, 31 (2005), pp. 397–423.
[24] E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms*, Computer Science Press, Rockville, MD, 1978.
[25] S. Itoh, P. Ordejón, and R. M. Martin, *Order-N tight-binding molecular dynamics on parallel computers*, Comput. Phys. Comm., 88 (1995), pp. 173–185.
[26] G. Karypis, A. Gupta, and V. Kumar, *A parallel formulation of interior point algorithms*, in Proceedings of Supercomputing 94, 1994.
[27] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*, Wiley–Teubner, Chichester, UK, 1990.
[28] X.-P. Li, R. W. Nunes, and D. Vanderbilt, *Density-matrix electronic-structure method with linear system-size scaling*, Phys. Rev. B, 47 (1993), pp. 10891–10894.
[29] J. M. Millam and G. E. Scuseria, *Linear scaling conjugate gradient density matrix search as an alternative to diagonalization for first principles electronic structure calculations*, J. Chem. Phys., 106 (1997), pp. 5569–5577.
[30] K. Nusbaum, *Optimizing Tpetra's Sparse Matrix-Matrix Multiplication Routine*, Technical report, SAND2011-6036, Sandia National Laboratories, Albuquerque, NM, 2011.
[31] NVIDIA Corporation, *CUSPARSE library*, 2010.
[32] H. B. Schlegel, J. M. Millam, S. S. Iyengar, G. A. Voth, A. D. Daniels, G. E. Scuseria, and M. J. Frisch, *Ab initio molecular dynamics: Propagating the density matrix with Gaussian orbitals*, J. Chem. Phys., 114 (2001), pp. 9758–9763.
[33] E. Solomonik, A. Bhatele, and J. Demmel, *Improving communication performance in dense linear algebra via topology aware collectives*, in Proceedings of 2011 ACM International Conference for High Performance Computing, Networking, Storage and Analysis, SC '11, New York, NY, 2011, pp. 77:1–77:11.
[34] A. Trifunović and W. J. Knottenbelt, *Parallel multilevel algorithms for hypergraph partitioning*, J. Parallel Distr. Comput., 68 (2008), pp. 563–581.
[35] R. A. van de Geijn and J. Watts, *SUMMA: Scalable universal matrix multiplication algorithm*, Concurrency Pract. Ex., 9 (1997), pp. 255–274.
[36] J. VandeVondele, U. Borstnik, and J. Hutter, *Linear scaling self-consistent field calculations with millions of atoms in the condensed phase*, J. Chem. Theor. Comput., 8 (2012), pp. 3565–3573.