Reducing latency cost in 2D sparse matrix partitioning models ☆

Oguz Selvitopi[a], Cevdet Aykanat[a,*]

[a]*Bilkent University, Computer Engineering Department, 06800, Ankara, TURKEY*

Abstract

Sparse matrix partitioning is a common technique used for improving performance of parallel linear iterative solvers. Compared to solvers used for symmetric linear systems, solvers for nonsymmetric systems offer more potential for addressing different multiple communication metrics due to the flexibility of adopting different partitions on the input and output vectors of sparse matrix-vector multiplication operations. In this regard, there exist works based on one-dimensional (1D) and two-dimensional (2D) fine-grain partitioning models that effectively address both bandwidth and latency costs in nonsymmetric solvers. In this work, we propose two new models based on 2D checkerboard and jagged partitioning. These models aim at minimizing total message count while maintaining a balance on communication volume loads of processors; hence, they address both bandwidth and latency costs. We evaluate all partitioning models on two nonsymmetric system solvers implemented using the widely adopted PETSc toolkit and conduct extensive experiments using these solvers on a modern system (a BlueGene/Q machine) successfully scaling them up to 8K processors. Along with the proposed models, we put practical aspects of eight evaluated models (two 1D- and six 2D-based) under thorough analysis. To the best of our knowledge, this is the first work that analyzes practical performance of 2D models on this scale. Among evaluated models, the models that rely on 2D jagged partitioning obtain the most promising results by striking a balance between minimizing bandwidth and latency costs.

Keywords: Parallel iterative solvers, Nonsymmetric linear systems, Sparse matrix-vector multiplication, Sparse matrix partitioning, Latency overhead, Bandwidth overhead
2010 MSC: 00-01, 99-00

1. Introduction

Many scientific and engineering applications necessitate solving a linear system of equations. The methods used for this purpose are categorized as direct and iterative methods. When the linear system is large and sparse, iterative methods are preferred to their direct counterparts due to their speed and flexibility. Most widely used iterative methods for solving large-scale linear systems are based on Krylov subspace iterations.

A single iteration in Krylov subspace methods usually consists of one or more Sparse Matrix–Vector multiplications (SpMV), dot product(s) and vector updates. In a distributed setting, SpMV operations require regular or irregular point-to-point (P2P) communication depending on the sparsity pattern of the coefficient matrix in which each processor sends/receives messages to/from a subset of processors. On the other hand, dot products necessitate global communication that involves a reduction operation on one or a few scalars in which all processors participate. Vector updates usually do not require any communication.

A common model to capture the cost of communicating a single message of m words consists of two major components and is given by the formula $t_s + mt_w$. Here, t_s is the startup time and signifies the

☆This work was supported by The Scientific and Technological Research Council of Turkey (TÜBİTAK) under Grant EEEAG-114E545.

*Corresponding author
Email addresses: reha@cs.bilkent.edu.tr (Oguz Selvitopi), aykanat@cs.bilkent.edu.tr (Cevdet Aykanat)

costs related to preparation of the message (referred to as latency cost). The t_w component is the time required to transfer a single word between two processors and is equal to the reciprocal bandwidth (referred to as bandwidth cost). Latency cost is proportional to the number of messages whereas bandwidth cost is proportional to the number of words. Among these two components, latency costs prove to be more vital for parallel performance as they are generally harder to avoid and improve [1]. Although both costs are reduced within time, the gap between them gradually increases in favor of bandwidth costs with an approximately 20% annual improvement over latency costs [1, 2]. Furthermore, computation speeds evolve faster than communication speeds, making communication costs more critical for performance. With the latest developments in the scientific computing field, communication costs are likely to be a major factor in ranking fastest high performance computing (HPC) systems [3]. This work focuses on reducing latency costs of parallel SpMV operations in the context of nonsymmetric iterative solvers. The models and methods proposed in our work apply to matrices both with regular and irregular sparsity patterns. However, the benefits are more evident for the irregular ones, which are usually harder to exploit.

1.1. Related work

Communication requirements of iterative solvers have been of interest for more than three decades. There are numerous works on reducing communication overhead of global reduction operations in iterative solvers. Several works in this category aim at decreasing the number of global synchronization points in a single iteration of the solver by reformulating it [4, 5, 6, 7, 8, 9, 10, 11, 12]. Another important area of study is s-step Krylov subspace methods, which focus on further reducing the number of global synchronization points by a factor of s by performing only a single reduction once in every s iterations [8, 13, 14, 15, 16, 17]. The performance gain of s-step methods comes at the cost of deteriorated stability and complications related to integration of preconditioners. However, these methods recently gained popularity again and promising studies address these shortcomings [13, 16, 18, 19]. Another common technique is to overlap communication and computation with the aim of hiding global synchronization overheads [20, 21]. Especially with the introduction of nonblocking collective constructs in the MPI-3 standard, this technique is gaining attraction [22, 23, 24]. Overlapping is commonly used for SpMV operations as well. In addition, a recent work proposed hierarchical and nested Krylov methods that constrain global reductions into smaller subsets of processors where they are cheaper [25]. Another recent work uses the idea of embedding SpMV communications into global reductions to avoid latency overhead of SpMV communications [26].

The performance of iterative solvers is also addressed by minimizing communication costs related to parallel SpMV operations, which is also addressed by this work. There are studies that can handle sparse matrices that are well-structured and have predictable sparsity patterns, generally arising from 2D/3D problems [16, 27, 28, 29]. However, the studies in this field generally focus on combinatorial models that are capable of exploiting both regular and irregular patterns to obtain a good partition of the coefficient matrix. In this regard, graph and hypergraph partitioning models are widely utilized with successful partitioning tools such as MeTiS [30], PaToH [31], Scotch [32], Mondriaan [33]. These models can broadly be categorized as one-dimensional (1D) and two-dimensional (2D) partitioning models. In 1D models [30, 31, 34, 35, 36, 37, 38, 39], each processor is responsible for a row/column stripe, whereas in 2D models, each processor may be responsible for a submatrix block (generally defined by a subset of rows and columns) or as in the most general case, each processor may be responsible for an arbitrarily defined subset of nonzeros. Compared to 1D models, 2D models possess more freedom in partitioning the coefficient matrix. Some works on 2D models do not take the communication volume into account, however they provide an upper bound on the number of messages communicated [40, 41, 42, 43, 44]. On the other hand, there are 2D models that aim at reducing volume, with or without providing a bound on the maximum number of messages [33, 45, 46, 47, 48, 49, 50]. 2D partitioning models in the literature can further be categorized into three classes: *checkerboard* partitioning [47, 49, 50] (also known as coarse-grain partitioning), *jagged* partitioning [45, 49] and *fine-grain* partitioning [46, 48, 49]. Notably, a recent work [50] proposes a fast 2D partitioning for scale-free graphs via a two-phase approach. This method uses 1D partitioning to reduce volume in the first phase and an efficient heuristic in the second phase to obtain a bound on the maximum number of message. This work differs from ours as it does not explicitly minimize the message count, instead, it uses a property of the Cartesian distribution of the matrices to provide the mentioned upper bound.

1.2. Motivation and contributions

Most of the aforementioned and other existing partitioning models optimize the objective of minimizing total communication volume, which is an effort to reduce bandwidth costs. However, communication cost is a function of both bandwidth and latency, with the latter being at least as important as the former, as the current trends indicate. The need for partitioning models that also consider other cost metrics has been noted in other works [26, 34]. There are a few notable works that focus on different communication cost metrics. Balancing communication volume is one of them [33, 51, 52]. More important and overlooked work targets multiple communication metrics including latency [53], on which this study is based. Compared to [53], this study concentrates more on practical aspects.

In this work, we claim and show that attempting to minimize a single communication objective hurts parallel performance and achieving a tradeoff between bandwidth and latency costs is the key factor for achieving scalability. The basic motivation is to employ a nonsymmetric partition in the solver. Note that in parallel SpMV operations of the form $w = Ap$, one needs to partition the input vector p and the output vector w in addition to A. This can be achieved either by using a symmetric partition where the same partition is imposed on both input and output vectors, or by using a nonsymmetric partition where a distinct partition is employed for input and output vectors. The latter alternative is more appealing and it should be adopted whenever convenient since it is more flexible and allows operating in a broader search space. A nonsymmetric partition can be utilized in nonsymmetric linear system solvers such as the conjugate gradient normal equation error (CGNE) [54, 55, 56] and residual method (CGNR) [54, 55, 56, 57], and the standard quasi-minimal residual (QMR) [58] where the coefficient matrix is square and nonsymmetric. The details of how to utilize nonsymmetric partitioning without incurring communication during linear vector update operations are explained for CGNE and CGNR solvers in Appendix E. We constrain ourselves to nonsymmetric square matrices in this work, but all proposed models apply to certain iterative methods that involve rectangular matrices as well.

Our work is based on [53], which also achieves a nonsymmetric partition through a two-phase methodology with a model called *communication hypergraph*. Our contributions and differences from [53] are as follows:

i) We propose two new partitioning models for reducing latency which are based on 2D checkerboard and jagged partitioning. These models aim at reducing latency costs usually at the expense of increasing bandwidth costs. Similar models have been investigated [48, 53], but they are based on 1D and 2D fine-grain models.

ii) All proposed and investigated partitioning models are realized on two iterative methods CGNE and CGNR implemented with the widely adopted PETSc toolkit [59]. We describe how to obtain a nonsymmetric partition on the vectors utilized in these solvers using the communication hypergraph model and thoroughly evaluate partitioning requirements of them via experiments. In this manner, we differ from [53], in which the proposed methods were tested with a code developed by the authors that contains only parallel SpMV computations.

iii) We conduct extensive experiments for the mentioned iterative solvers. Although better suited to large-scale systems, the communication hypergraph model was originally tested only for 24 processors on a local cluster and only for 1D partitioning. In this work, we test and show this model's validity on a modern HPC system (a BlueGene/Q machine) successfully scaling up to 8K processors.

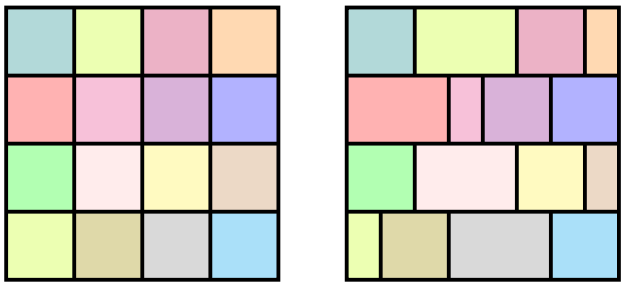
iv) We compare one 1D-based, three 2D-based models (checkerboard, jagged and fine-grain), and these four models' latency-improved versions, making a total of eight partitioning models. Among these, the 2D models are somewhat overlooked in the literature, never being tested in a realistic setting on a large-scale system. Although their theoretical merits are of no question, their practical merits are not appreciated. In our experiments, we put these methods' practical aspects into a thorough analysis. The experiments show surprising results with 2D jagged partitioning and its latency-improved version performing better in the majority of the matrices.

The rest of the paper is organized as follows. Sections 2.1 and 2.2 describe the proposed partitioning models to reduce the latency overhead of checkerboard and jagged models, respectively. These two sections

describe basic checkerboard and jagged models as well. We also briefly review the fine-grain model and its latency-improved version in Section 2.3, since they are included in our experiments. We compare communication properties of all partitioning models in Section 3. Section 4 contains the results and discussions of the extensive large-scale experimental evaluation of eight partitioning models on a BlueGene/Q system with 28 matrices. Our experiments range from 256 to 8192 processors. Final remarks are given in Section 5. The presentation of the paper relies on the assumption that the reader is already familiar with the communication hypergraph model for 1D partitioning [53]. The unfamiliar reader can find a detailed background with explanatory examples about the communication hypergraph model in Appendix D.

2. Reducing latency cost in 2D partitioning models

2D models work at a finer level of partitioning granularity compared to 1D models by allowing nonzeros of a single row/column to be assigned to more than one processor. In this manner, they possess more flexibility in partitioning since they do not constrain the search space by assigning all nonzeros of a row/column to the same processor. This leads them to exploit existing partitioning tools better. 1D models necessitate a row-parallel/column-parallel algorithm (Appendix B), whereas 2D models necessitate a row-column-parallel algorithm. The fundamental difference between them is that the former necessitates a single communication stage in parallel SpMV operations, which is either pre-communication if the matrix is partitioned rowwise, or post-communication if the matrix is partitioned columnwise, whereas the latter necessitates two distinct communication stages: one before the local SpMV computations (on the input vector in a *pre*-communication stage via *expand* communication tasks) and one after the local SpMV computations (on the output vector in a *post*-communication stage via *fold* communication tasks), For more details on implementation issues regarding 2D partitioning, see [49].



(a) Checkerboard (b) Jagged

Figure 1: A sample of 2D checkerboard and jagged partitionings on a $16 = 4 \times 4$ virtual processor mesh.

In this section, we describe how to reduce latency overhead of 2D checkerboard and jagged partitioning models. For these models, we assume a $K = P \times Q$ virtual processor mesh. A simple example depicting a matrix partitioned with these two models are given in Figure 1. Compared to their original counterparts, the proposed models are likely to increase the bandwidth costs by increasing communication volume. However, this issue is addressed by maintaining a balance on this metric. We also briefly review the 2D fine-grain model and compare it to checkerboard and jagged models as we evaluate it in our experiments. Note that a model to reduce latency overhead of fine-grain partitioning has already been investigated [48]. All proposed models are discussed on parallel $w = Ap$. However, the arguments are also valid for $z = A^T r$ as communication requirements of $w = Ap$ and $z = A^T r$ are the dual of each other and minimizing the objective function in $w = Ap$ is equivalent to minimizing the objective function in $z = A^T r$.

2.1. Checkerboard partitioning

Checkerboard partitioning is a two-phase process in which each phase utilizes a 1D partitioning model. The second phase depends on the first phase by using information obtained in the first phase to determine

multiple vertex weights utilized in the second phase. For the rest of the section, we assume a 1D rowwise partition in the first phase and a 1D columnwise partition in the second phase. For the arguments made in this section, an analogous discussion holds for the dual scheme as well.

Consider a $K = P \times Q$ processor mesh and an $n \times n$ square matrix A. In the first phase, the column-net hypergraph model $\mathcal{H}_{\mathcal{R}} = (\mathcal{V}_{\mathcal{R}}, \mathcal{N}_{\mathcal{C}})$ is used to obtain a P-way partition $\Pi_{\mathcal{R}} = \{\mathcal{V}_1, \ldots, \mathcal{V}_P\}$, which induces a P-way rowwise partition $\{\mathcal{R}_1, \ldots, \mathcal{R}_P\}$ of A. Here, $\mathcal{R}_\alpha$ denotes the set of rows that correspond to vertices in $\mathcal{V}_\alpha$, for $\alpha = 1, \ldots, P$. The rows in $\mathcal{R}_\alpha$ form a row stripe A_α whose size is $n_\alpha \times n$, with $n_\alpha = |\mathcal{V}_\alpha|$. At the end of the first phase, the assignment of rows of A is determined by associating row stripe A_α with the Q processors in row α of the processor mesh, $P_{\alpha,*}$.

In the second phase, the row-net hypergraph model $\mathcal{H}_{\mathcal{C}} = (\mathcal{V}_{\mathcal{C}}, \mathcal{N}_{\mathcal{R}})$ is used to obtain a Q-way partition $\Pi_{\mathcal{C}} = \{\mathcal{V}_1, \ldots \mathcal{V}_Q\}$, which induces a Q-way columnwise partition $\{\mathcal{C}_1, \ldots, \mathcal{C}_Q\}$ of A. Here, $\mathcal{C}_\beta$ denotes the set of columns that correspond to vertices in $\mathcal{V}_\beta$, for $\beta = 1, \ldots, Q$. The columns in $\mathcal{C}_\beta$ form column stripe A_β whose size is $n \times n_\beta$, with $n_\beta = |\mathcal{V}_\beta|$. At the end of the second phase, we complete the assignment of columns of A and actually obtain a Q-way columnwise partition of each row stripe A_α, forming Q submatrix blocks $A_{\alpha,1}, \ldots, A_{\alpha,Q}$. Hence, $(\Pi_{\mathcal{R}}, \Pi_{\mathcal{C}})$ defines an assignment for rows and columns of A where processor $P_{\alpha,\beta}$ is responsible for the set of rows in $\mathcal{R}_\alpha$ and the set of columns in $\mathcal{C}_\beta$. In other words, nonzero a_{ij} is assigned to $P_{\alpha,\beta}$ if $r_i \in \mathcal{R}_\alpha$ and $c_j \in \mathcal{C}_\beta$.

This two-phase process aims at minimizing total communication volume for the pre- and post-communication stages in the first and second phases, respectively, while maintaining computational load balance [47, 49]. A notable property of checkerboard partitioning is that it confines the communication in expand and fold operations to the processors in the same column and row of the processor mesh, respectively. It achieves a Cartesian distribution of the matrix, in which each processor owns an intersection of a subset of rows and a subset of columns. A row (column) is said to be coherent if the nonzeros of this row (column) generate partial results for (require) the same w-vector (x-vector) element. Consider a row r_i that is assigned to $\mathcal{R}_\alpha$ at the end of the first phase. The coherency of this row is preserved at this point as it is modeled by v_i in $\mathcal{H}_{\mathcal{R}}$. In the second phase, the nonzeros of this row can be distributed among Q processors in row α of the processor mesh, which is also the case for all other rows in $\mathcal{R}_\alpha$. Hence, row coherency is respected in a *coarse* level by assigning nonzeros of rows in $\mathcal{R}_\alpha$ to the processors in the same row of the processor mesh, $P_{\alpha,*}$. A coarse level here implies that the nonzeros belonging to a subset of rows are distributed among the same subset of processors (in this case among P processors in a specific column of the processor mesh). This provides the upper bound $Q-1$ on the number of messages communicated in the post-communication stage as there are Q processors in row α of the processor mesh. With a similar argument, column coherency is also respected in a coarse level by assigning nonzeros of columns in $\mathcal{C}_\beta$ to the processors in the same column of the processor mesh, $P_{*,\beta}$. This provides the upper bound $P-1$ on the number of messages communicated in the pre-communication stage as there are P processors in column β of the processor mesh. Hence, the maximum number of messages handled by a single processor is bounded by $P + Q - 2$. In checkerboard partitioning, the second phase is performed with P-way multi-constraint [60, 61] partitioning to balance computational loads.

2.1.1. Communication matrices

The expand communication tasks in the checkerboard model are bound to distinct columns of the processor mesh. For this reason, to summarize the communication requirements of expand tasks in the pre-communication stage, we form Q distinct communication matrices. Let p_{C_β} denote the vector elements that necessitate communication in column β of the processor mesh, for $1 \leq \beta \leq Q$. Note that at most P processors can participate in communicating p_{C_β}, confined to the set of processors in $P_{*,\beta}$. We summarize the communication operations in column β of the mesh with the $P \times |p_{C_\beta}|$ communication matrix M_β. Rows of M_β correspond to processors in $P_{*,\beta}$ and columns of M_β correspond to expand tasks on p_{C_β}. There exists a nonzero $m_{\alpha j} \in M_\beta$ if and only if there is a non-empty column segment in submatrix $A_{\alpha,\beta}$ at the respective column. The nonzeros in column j of M_β represent the set of processors that participate in communicating $p_{C_\beta}[j]$, which is a subset of processors in $P_{*,\beta}$. The nonzeros in row α of M_β represent the expand tasks processor $P_{\alpha,\beta}$ participates in. Note that vector elements corresponding to internal columns (those which have a single non-empty column segment in P row stripes) do not incur communication and they are not

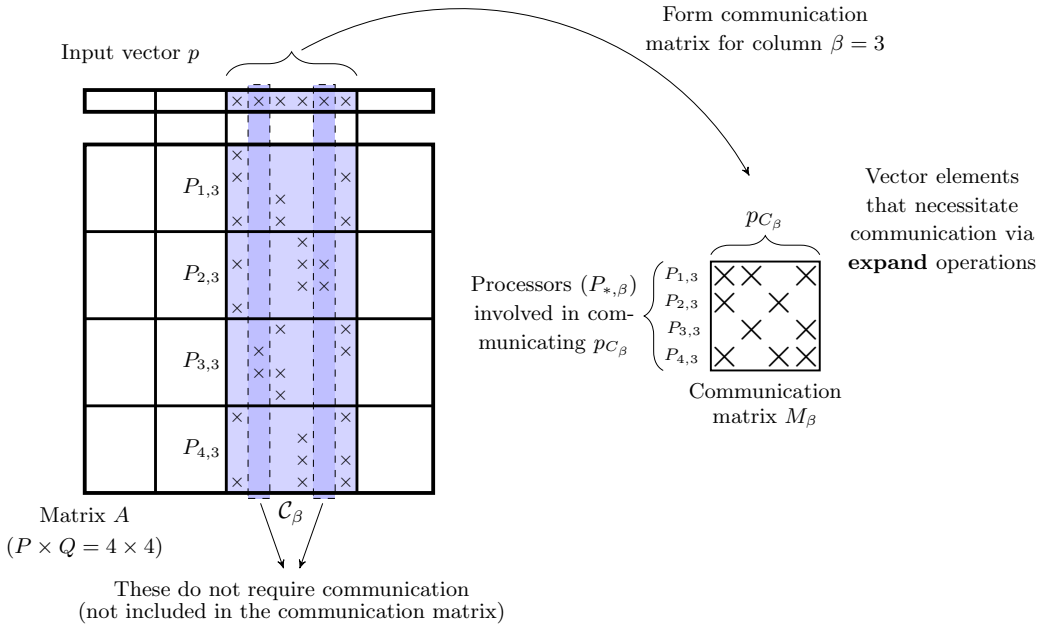


Figure 2: Formation of the communication matrix for the third column of the processor mesh ($\beta = 3$) to summarize expand operations in the pre-communication stage.

included in M_β. These vector elements should be assigned to the respective processors to avoid unnecessary communication. An example in Figure 2 is presented to illustrate the formation of communication matrix M_β for the third column of the processor mesh ($\beta = 3$) to summarize the expand tasks. There are four input vector elements that necessitate communication (denoted by p_{C_β}) and they form columns of M_β. For instance, the first column of M_β has nonzeros corresponding to processors $P_{1,3}, P_{2,3}$ and $P_{4,3}$ since in matrix A, there exist nonzero column segments in the respective submatrix blocks. Two vector elements–second and fifth–corresponding to internal columns do not incur communication and they are not included in M_β; these elements should be assigned to $P_{3,3}$ and $P_{2,3}$, respectively, to avoid unnecessary communication.

The fold communication tasks in checkerboard model are bound to distinct rows of the processor mesh. Following a similar approach, we form P distinct communication matrices to summarize the communication requirements of fold tasks in the post-communication stage. Let w_{C_α} denote the vector elements that necessitate communication in row α of the processor mesh, for $1 \leq \alpha \leq P$. We summarize the communication operations in row α of the mesh with the $|w_{C_\alpha}| \times Q$ communication matrix M_α, where there exists a nonzero $m_{i\beta} \in M_\alpha$ if and only if there is a non-empty row segment in submatrix $A_{\alpha,\beta}$ at the respective row. An example is presented in Figure 3 to illustrate the formation of communication matrix M_α in the third row of the processor mesh ($\alpha = 3$) to summarize the fold tasks. The dual of the discussions made for M_β in Figure 2 follows also for M_α.

We form a total of $P+Q$ communication matrices to summarize communication requirements of checkerboard partitioning. We can address the communication requirements of both pre- and post-communication stages independently since communication operations in these stages are bound to distinct columns and rows of the processor mesh, respectively. Formation of these communication matrices is illustrated in Figure 4.

2.1.2. Formation of communication hypergraphs

We form Q hypergraphs from Q communication matrices for the pre-communication stage. For each M_β, a communication hypergraph $\mathcal{H}_\beta^{CM}$ is formed using the row-net hypergraph model, for $1 \leq \beta \leq Q$. The net set of $\mathcal{H}_\beta^{CM}$ represents the processors in column β $(P_{*,\beta})$ of the processor mesh and the vertex set of $\mathcal{H}_\beta^{CM}$ represents the expand tasks on p_{C_β}. Hence, there are P nets and $|p_{C_\beta}|$ vertices in $\mathcal{H}_\beta^{CM}$. A vertex v_j in $\mathcal{H}_\beta^{CM}$ is connected by the set of nets corresponding to processors that communicate the respective vector element

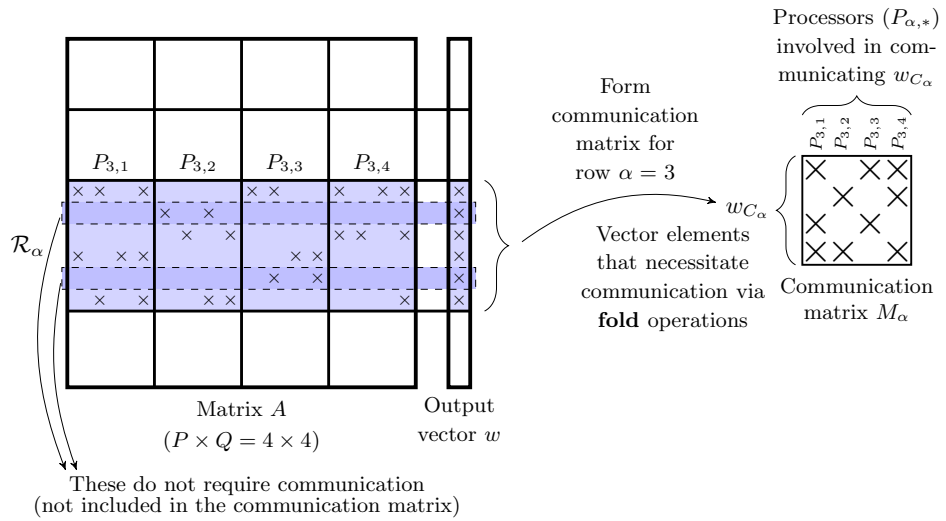


Figure 3: Formation of the communication matrix for the third row of the processor mesh ($\alpha = 3$) to summarize fold operations in the post-communication stage.

$p_{C_\beta}[j]$. In all Q communication hypergraphs, the total number of vertices is equal to $|p_C| = \sum_{\beta=1}^{Q} |p_{C_\beta}|$ and the total number of nets is equal to $Q \times P = K$.

In a similar manner, we form P hypergraphs from P communication matrices for the post-communication stage. For each M_α, a communication hypergraph $\mathcal{H}_\alpha^{CM}$ is formed using the column-net hypergraph model, for $1 \leq \alpha \leq P$. The net set of $\mathcal{H}_\alpha^{CM}$ represents the processors in row α ($P_{\alpha,*}$) of the processor mesh and the vertex set of $\mathcal{H}_\alpha^{CM}$ represents the fold tasks on w_{C_α}. Hence, there are Q nets and $|w_{C_\alpha}|$ vertices in $\mathcal{H}_\alpha^{CM}$. A vertex v_i in $\mathcal{H}_\alpha^{CM}$ is connected by the set of nets corresponding to the processors that communicate the respective vector element $w_{C_\alpha}[i]$. In all P communication hypergraphs, the total number of vertices is equal to $|w_C| = \sum_{\alpha=1}^{P} |w_{C_\alpha}|$ and the total number of nets is equal to $P \times Q = K$.

In total, we form $P + Q$ communication hypergraphs from $P + Q$ communication matrices. This process is illustrated in Figure 4.

2.1.3. Partitioning of the communication hypergraphs

We partition $\mathcal{H}_\beta^{CM}$ to get a P-way partition $\Pi_\beta = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_P\}$ and obtain a distribution of expand tasks among P processors in column β of the processor mesh for the pre-communication stage, for $1 \leq \beta \leq Q$. The responsibility of the expand tasks represented by the vertices in $\mathcal{V}_\alpha \in \Pi_\beta$ is assigned to processor $P_{\alpha,\beta}$. Consider a net $n_{\alpha,\beta}$ in $\mathcal{H}_\beta^{CM}$ that represents $P_{\alpha,\beta}$. The connectivity set of this net contains the parts that correspond to the processors each of which send a message to $P_{\alpha,\beta}$. The size of this set can be at most P since $\mathcal{H}_\beta^{CM}$ is partitioned into P, bounding the number of messages sent/received by a single processor by $P-1$ in the pre-communication stage. Hence, this feature of original checkerboard partitioning is still respected. In partitioning $\mathcal{H}_\beta^{CM}$, the partitioning objective of minimizing cutsize corresponds to *minimizing the number of messages* communicated in column β of the processor mesh in the pre-communication stage, and the partitioning constraint of maintaining balance among part weights corresponds to obtaining a balance on the communication volume loads of these processors.

Similarly, we partition $\mathcal{H}_\alpha^{CM}$ to get a Q-way partition $\Pi_\alpha = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_Q\}$ and obtain a distribution of fold tasks among Q processors in row α of the processor mesh for the post-communication stage, for $1 \leq \alpha \leq P$. $\mathcal{H}_\alpha^{CM}$ is partitioned into Q, bounding the number of messages sent/received by a single processor by $Q - 1$ in the post-communication stage. Hence, this feature of original checkerboard partitioning is also respected. The partitioning objective and the balancing constraint are identical to those in partitioning $\mathcal{H}_\beta^{CM}$.

The formed $P + Q$ hypergraphs can be independently partitioned since they do not depend on each other in any way. The maximum number of messages handled by a single processor is still $P + Q - 2$ as

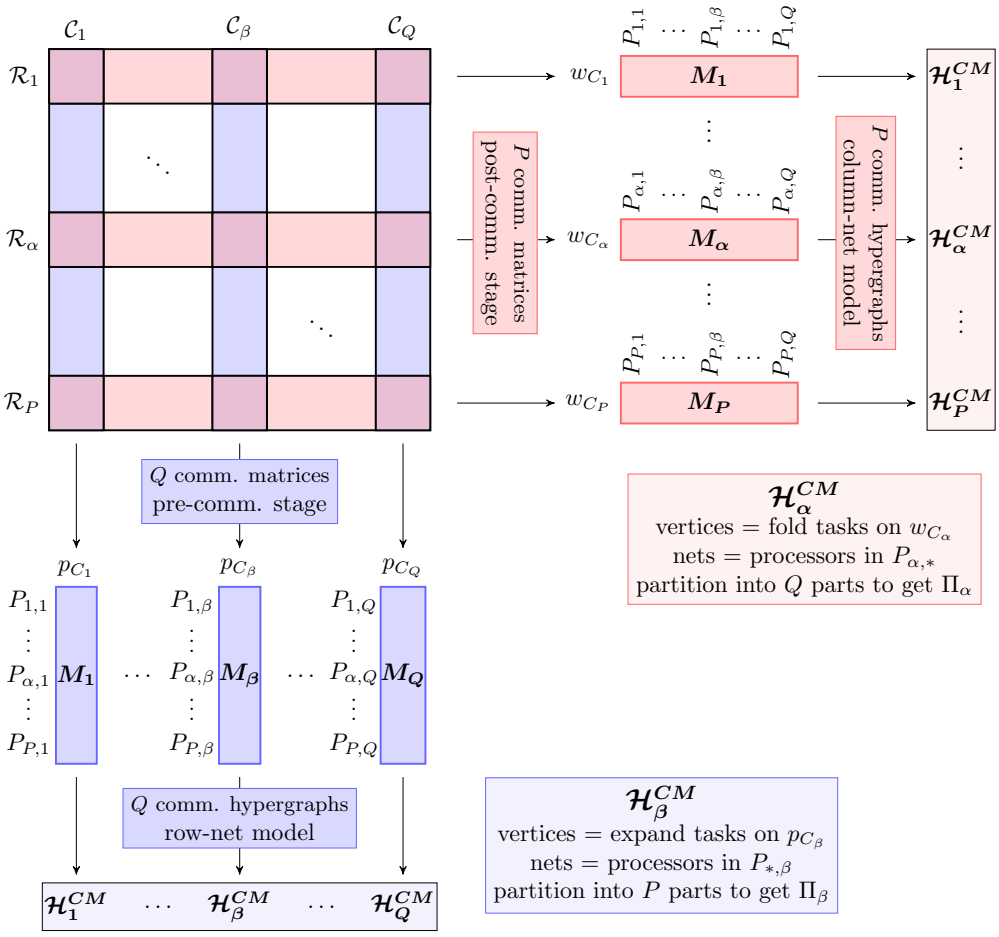


Figure 4: Minimizing latency cost in checkerboard partitioning model.

in the original checkerboard partitioning. As a result, we improve latency costs in each row/column of the processor mesh independently while respecting basic characteristics of checkerboard partitioning.

2.2. Jagged partitioning

Jagged partitioning consists of two phases. The first phase consists of a single 1D partitioning model, whereas the second phase consists of multiple, independent and same type of 1D partitioning models. The second phase depends on the first phase by using the partitioning information obtained in the first phase to determine vertex sets and vertex weights for the models formed in the second phase. For the rest of the section, we assume a 1D rowwise partition in the first phase and a 1D columnwise partition in the second phase. For the arguments made in this section, an analogous discussion holds for the dual scheme as well.

Assume a $K = P \times Q$ processor mesh and an $n \times n$ square matrix A. The first phase of jagged partitioning is exactly the same as the first phase of checkerboard partitioning: a column-net hypergraph model $\mathcal{H}_\mathcal{R} = (\mathcal{V}_\mathcal{R}, \mathcal{N}_\mathcal{C})$ is used to obtain a P-way partition $\Pi_\mathcal{R} = \{\mathcal{V}_1, \ldots, \mathcal{V}_P\}$, which induces a rowwise partition $\{\mathcal{R}_1, \ldots, \mathcal{R}_P\}$ of A. At the end of this phase, the rows in row stripe A_α are associated with the Q processors in row α of the processor mesh, $P_{\alpha,*}$.

In the second phase, we form a hypergraph $\mathcal{H}_\alpha$ for each row submatrix A_α obtained in the former phase using the row-net hypergraph model, for $1 \leq \alpha \leq P$. In total, P hypergraphs are formed. In this aspect, jagged partitioning differs from checkerboard partitioning – which forms a single hypergraph in the second phase. The net set of $\mathcal{H}_\alpha$ represents rows of A_α and the vertex set of $\mathcal{H}_\alpha$ represents columns of A that have a nonzero column segment in A_α. Hence, the same vertex can appear in multiple hypergraphs since

the corresponding column may have nonzero column segments in more than one row stripes. These P hypergraphs are independently partitioned into Q parts to obtain a Q-way partition of each row stripe. At the end of the second phase, for each row stripe A_α, we obtain Q submatrix blocks $A_{\alpha,1}, \ldots, A_{\alpha,Q}$ by partitioning vertices corresponding to columns of A_α.

The first and the second phase aim to minimize the volume of communication in pre- and post-communication stages, respectively, while maintaining computational load balance [47, 49]. In contrast to checkerboard partitioning, the objective in the second phase of the jagged partitioning is addressed independently by partitioning row stripes separately in distinct hypergraphs. The jagged model also differs from the checkerboard model as it does not lead to a Cartesian distribution of the matrix. Jagged partitioning is more flexible in this sense since it allows nonzeros of a column to be distributed among *any* processor that is in distinct rows of the processor mesh – not just among the processors that are in the *same* column of the mesh, as is the case for checkerboard partitioning. Hence, the column coherency is not preserved in the assignment of columns. This leads to improved communication volume for expand tasks in the pre-communication stage at the expense of increasing the upper bound on the number of messages handled by a single processor. In other words, the jagged model sacrifices the coarse level coherency of columns and causes the number of messages handled by a single processor in the pre-communication stage to be at most $P \times Q - Q = K - Q$. In this stage, a processor may communicate with any other processor except the processors that are in the same row of the processor mesh as this processor. On the other hand, the coherency of rows owned by a processor is respected in a coarse level as in checkerboard partitioning. This is because nonzeros of these rows are distributed among processors in the same row of processor mesh, $P_{\alpha,*}$, if the respective processor is in row α. Hence, the number of messages handled by a single processor for fold tasks in the post-communication stage is bounded by $Q - 1$ as there are Q processors in a single row of the processor mesh. Consequently, the maximum number of messages handled by a processor can be at most $(K - Q) + (Q - 1) = K - 1$ in jagged partitioning.

2.2.1. Communication matrices

The expand communication tasks in the jagged model are not bound to distinct columns of the processor mesh. For this reason, we form a single communication matrix to summarize the communication requirements of expand tasks in the pre-communication stage. Let p_C denote the vector elements that necessitate communication. We summarize the communication operations with the $(K = P \times Q) \times |p_C|$ communication matrix M_R. Rows of M_R correspond to *all* processors and columns of M_R correspond to expand tasks on p_C. Consider two vector elements owned by the same processor. Although these two elements can be communicated by at most P processors (each of which belongs to a distinct row of the processor mesh), they do not necessarily need to be confined to the same column of the processor mesh. For this reason, we include all processors in M_R. The formation of M_R essentially resembles that of 1D row-parallel $w = Ap$ (Appendix D.1): $m_{kj} \neq 0$ if and only if column c_j has a nonzero column segment in kth row stripe of A. Nonzeros in column $c_j \in M_R$ represent the set of processors that participate in communicating $p_C[j]$ and nonzeros in row $r_k \in M_R$ represent the expand tasks P_k participates in. The difference is, however, as a consequence of jagged partitioning, each column in M_R can have at most P nonzeros instead of K. As usual, vector elements corresponding to internal columns are not included in M_R since they do not necessitate communication.

In contrast to expand tasks, the fold communication tasks are bound to distinct rows of the processor mesh. The communication requirements of fold tasks in the post-communication stage are thus summarized by P distinct communication matrices, M_α, for $1 \leq \alpha \leq P$. The formation of these matrices are the same as the formation of matrices for summarizing communication requirements of fold tasks in checkerboard partitioning. The semantics of nonzeros in rows and columns of M_α are identical to those of M_α in checkerboard partitioning.

We form a total of $P + 1$ communication matrices to summarize communication requirements of jagged partitioning. We can address the communication requirements of the post-communication stage independently using P matrices. However, since communication operations in the pre-communication stage are not bound to the processors in the same column of the processor mesh, the expand tasks are represented in a single matrix with all K processors. Formation of these communication matrices is illustrated in Figure 5.

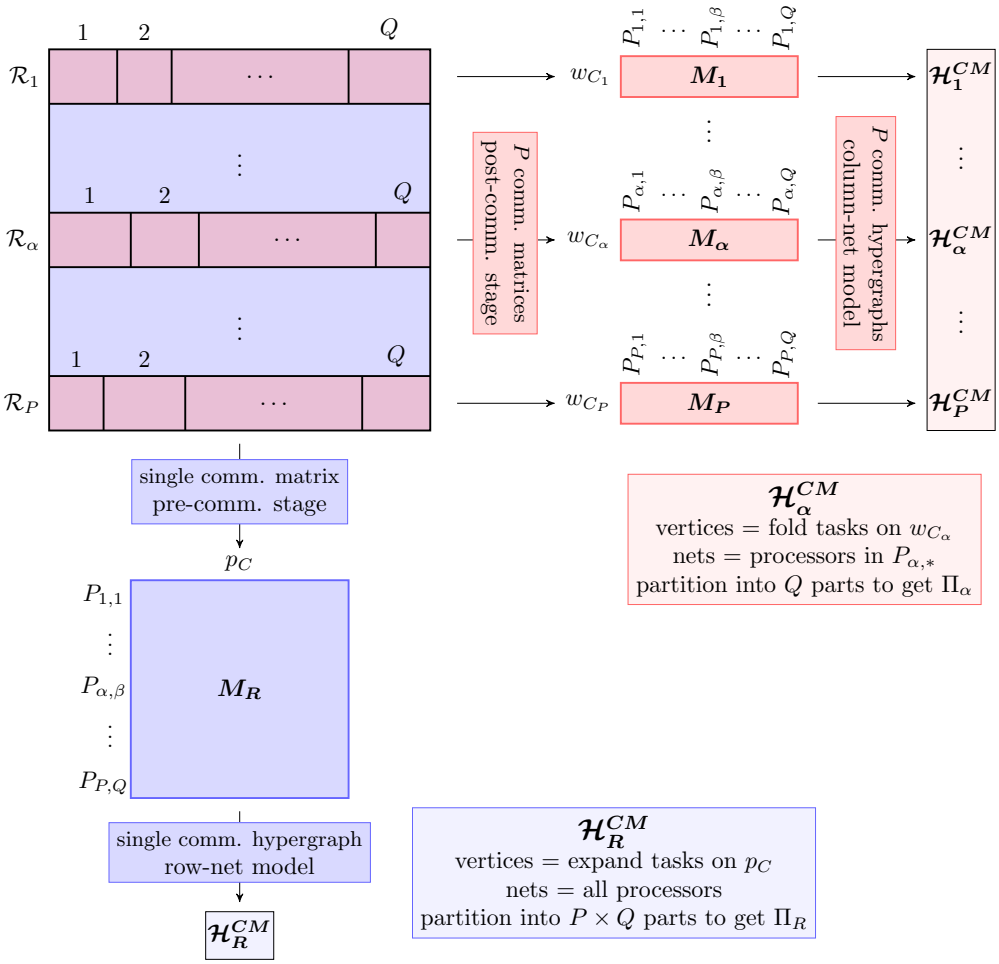


Figure 5: Minimizing latency cost in jagged partitioning model.

2.2.2. Formation of the communication hypergraphs

For the pre-communication stage, we form a single communication hypergraph $\mathcal{H}_R^{CM}$ from communication matrix M_R using the row-net hypergraph model. The net set of $\mathcal{H}_R^{CM}$ corresponds to K processors and the vertex set of $\mathcal{H}_R^{CM}$ corresponds to expand tasks on p_C. Hence, there are K nets and $|p_C|$ vertices in $\mathcal{H}_R^{CM}$. A vertex v_j in $\mathcal{H}_R^{CM}$ is connected by the set of nets corresponding to processors that communicate the respective vector element $p_C[j]$. Note that v_j can be connected by at most P nets, i.e., $d_j \leq P$.

For the post-communication stage, we form P communication hypergraphs from P communication matrices. The formation of these communication hypergraphs is actually the same as for checkerboard partitioning. A communication hypergraph $\mathcal{H}_\alpha^{CM}$ is formed using the column-net hypergraph model for matrix M_α, for $1 \leq \alpha \leq P$. The semantics of these hypergraphs are also the same: the net set of $\mathcal{H}_\alpha^{CM}$ represents the processors in row α of the processor mesh, $P_{\alpha,*}$, and the vertex set of $\mathcal{H}_\alpha^{CM}$ represents the fold tasks on w_{C_α}. Similarly, there are $|w_C|$ vertices and K nets in all communication hypergraphs.

In total, we form $P+1$ communication hypergraphs from $P+1$ communication matrices. This process is illustrated in Figure 5.

2.2.3. Partitioning of the communication hypergraphs

Communication hypergraph $\mathcal{H}_R^{CM}$ is partitioned to obtain a K-way partition $\Pi_R = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$ to induce a distribution of expand tasks in the pre-communication stage. The responsibility of the expand tasks represented by the vertices in $\mathcal{V}_k$ is assigned to processor P_k. Consider a net n_k in $\mathcal{H}_R^{CM}$ that represents

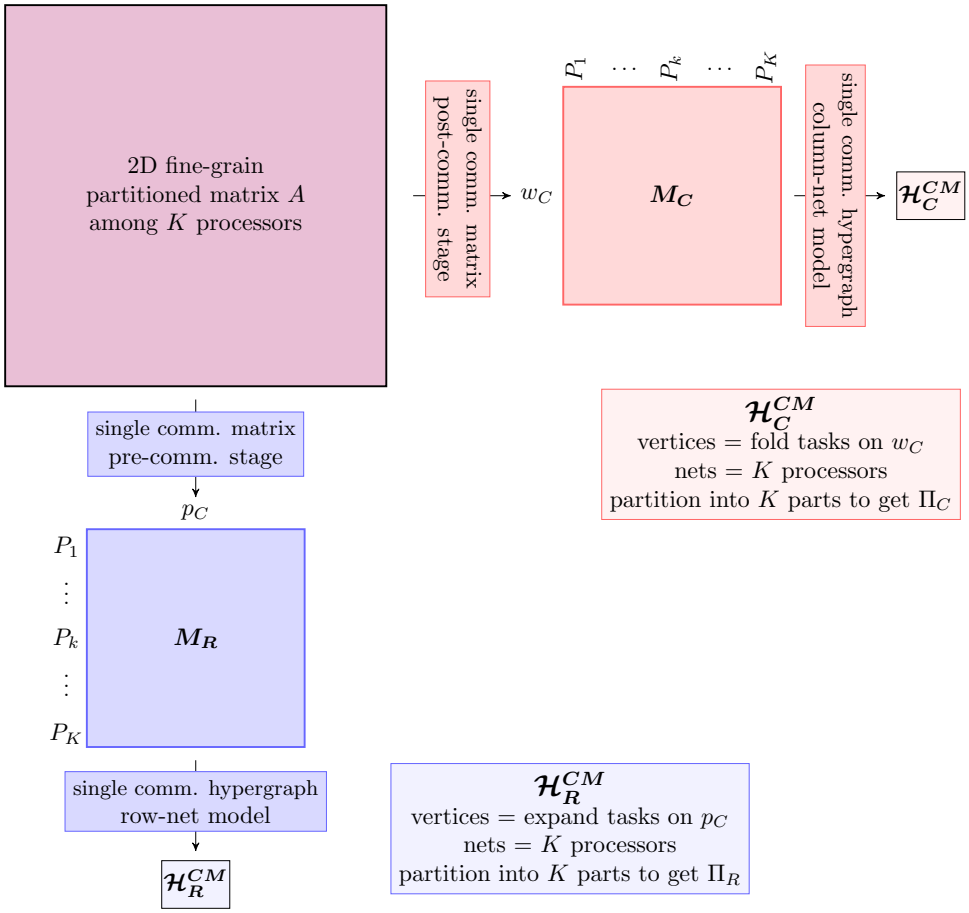


Figure 6: Minimizing latency cost in fine-grain partitioning model.

P_k. The connectivity set of this net corresponds to processors each of which send a message to P_k. Since $\mathcal{H}_R^{CM}$ is partitioned into K parts, the size of this set can be at most K. Thus, the maximum number of messages sent/received by a single processor is $K-1$ in the pre-communication stage (note that it is $K-Q$ in the original jagged partitioning). In partitioning $\mathcal{H}_R^{CM}$, the partitioning objective of minimizing cutsize corresponds to *minimizing the number of messages* communicated in the pre-communication stage, and the partitioning constraint of maintaining balance among part weights corresponds to obtaining a balance on the communication volume loads of K processors.

Communication hypergraph $\mathcal{H}_\alpha^{CM}$ is partitioned to obtain a Q-way partition $\Pi_\alpha = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_Q\}$, for $1 \leq \alpha \leq P$, to induce a distribution of fold tasks in the post-communication stage among Q processors in row α of the processor mesh. The partitioning of these communication hypergraphs have the same semantics with those in checkerboard partitioning.

The formed $P+1$ hypergraphs can be partitioned independently, since they do not depend on each other in any way. Note that the maximum number of messages handled by a single processor is slightly increased from $K-1$ to $K+Q-2$, which is caused by the partitioning for distributing tasks in the pre-communication stage. However, the cases beyond this are expected to be rare as a good partitioning tool will avoid them. As a result, we improve latency costs while respecting most characteristics of the original jagged partitioning.

2.3. Fine-grain (nonzero-based) partitioning

We briefly review fundamental properties of fine-grain partitioning. This model is first proposed in [46] and its communication costs are improved in a later work using the CHG model [48]. Both of these models are included in our experiments.

The fine-grain model forms a hypergraph in which vertices represent nonzeros of A and nets represent rows and columns of A. Nets corresponding to columns of A capture the communication volume incurred in the pre-communication stage, while nets corresponding to rows of A capture the communication volume incurred in the post-communication stage. Partitioning this hypergraph into K parts induces a distribution of nonzeros of the matrix among K processors. This leads to a completely arbitrary distribution of fine-grain computational tasks on a nonzero basis, where each vertex signifies a scalar multiplication with a single nonzero. Therefore, the fine-grain model respects neither row nor column coherence. In this aspect, it accommodates the highest level of flexibility by not restraining the computational tasks to coarser levels (i.e., nonzeros of a row and/or column) compared to checkerboard, jagged and 1D models. As a result, a processor may communicate with any other processor. Thus, the maximum number of messages handled by a single processor is $K-1$ in the pre-communication stage and is also $K-1$ in the post-communication stage, summing up to a total of $2(K-1)$ messages. The fine-grain model correctly minimizes the total volume of communication volume while maintaining computational load balance. Fore more details, see [46, 49].

The approach to improve communication requirements of the fine-grain model [48] consists of forming two communication matrices: one matrix for summarizing communication operations in the pre-communication stage and one matrix for summarizing communication operations in the post-communication stage. Compare this to the formation of communication matrices in the checkerboard and jagged models. We address the communication requirements in the checkerboard model by separately forming a total of $P+Q$ communication matrices since they are confined to distinct columns and rows of the processor mesh. Also in the jagged model, we form P communication matrices for the post-communication stage. These are not valid for the fine-grain model since the resulting partition is arbitrarily defined on the nonzeros of the matrix and any of the K processors may communicate with any other processor in both pre- and post-communication stages. For this reason, the whole set of processors and all vector elements that necessitate communication are included in two communication matrices. The process for reducing the communication costs of the fine-grain model is illustrated in Figure 6.

3. Comparison of partitioning models

We compare the basic properties of the investigated partitioning models to aid the discussions of the results in experiments. It is assumed that $P = Q = \sqrt{K}$ in $P \times Q$ processor mesh for the ease of presentation.

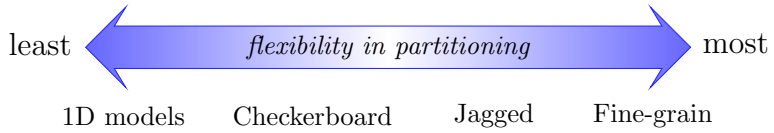


Figure 7: Comparison of models in terms of partitioning flexibility.

Figure 7 compares the partitioning models in terms of flexibility they provide during partitioning. 1D models lie at the left extreme of the spectrum since they represent each row/column of the coefficient matrix with a single distinct vertex as an atomic task. This leads to the assignment of all nonzeros of a row/column to an individual processor as a whole. Hence, 1D models respect row/column coherency at the individual processor level. The fine-grain model lies at the right extreme of the spectrum since in this model each vertex represents an atomic task corresponding to a single nonzero of the matrix. This is the most flexible and the finest level of partitioning granularity available, where neither row nor column coherency is preserved. So, in theory, nonzeros of a row/column can be distributed among K processors. Between these two extremes, the checkerboard and jagged models strive to distribute nonzeros of a row/column among a subset of processors. By doing so, they obtain a coarse-level row/column coherency at the processor mesh's row/column level. The checkerboard model leverages a coarse-level coherency in both partitioning phases whereas the jagged model leverages it in a single partitioning phase.

Among these partitioning models, 2D models are expected to achieve lower bandwidth costs compared to 1D models since they offer more flexibility in optimizing the objective of minimizing total communication volume. Among 2D models, fine-grain is expected to obtain the best results in terms of bandwidth costs,

Table 1: Comparison of partitioning models in terms of latency overhead and partitioning granularity.

number of messages	comm. stage	1D	2D partitioning models		
			Checkerboard	Jagged	Fine-grain
max	pre	$K-1$	$\sqrt{K}-1$	$K-\sqrt{K}$	$K-1$
	post	N/A	$\sqrt{K}-1$	$\sqrt{K}-1$	$K-1$
	overall	$K-1$	$2(\sqrt{K}-1)$	$K-1$	$2(K-1)$
total	**overall**	$K(K-1)$	$2K(\sqrt{K}-1)$	$K(K-1)$	$2K(K-1)$
Row/column coherency (Partitioning granularity)		Either entire row or Entire column	Coarse-level on both Rows and columns	Coarse-level on either Rows or columns	None

whereas checkerboard is likely to obtain the worst. The metrics related to latency costs (as upper bounds on the maximum number of messages) are presented in Table 1. Checkerboard has the lowest overhead with $2(\sqrt{K}-1)$ maximum messages per processor, whereas fine-grain has the highest overhead with $2(K-1)$. Although 1D and jagged models have the same upper bound $K-1$, in practice jagged partitioning is more likely to achieve better results in this metric since it restricts the number of messages in both stages of communication.

The discussions made so far in this section reflect the characteristics of the original partitioning models in which the communication hypergraph model is not used. The original models completely focus on minimizing bandwidth costs, disregarding latency-related objectives. Using the communication hypergraph model as a further step reduces latency costs at the expense of increasing bandwidth costs while respecting certain characteristics of the original models as much as possible. Our experimental evaluation shows that latency should definitely be on the table to achieve scalable performance.

4. Experiments

We evaluate two 1D-based and six 2D-based models, that is, a total of eight partitioning models. The evaluated models are based on 1D rowwise partitioning (`1D`), checkerboard partitioning (`CKBD`), jagged partitioning (`JGD`) and fine-grain partitioning (`FG`). Four of the evaluated models are the baseline models in which the communication tasks are assigned to processors using a simple heuristic that aims at balancing the communication volume loads while respecting total volume attained in the initial partitioning. This heuristic is also utilized in [53] and is an adaptation of the best-fit-decreasing heuristic used in solving the NP-hard K-feasible Bin Packing (`BP`) problem [62]. These `BP`-enhanced baseline models are referred to as `1D+BP`, `CKBD+BP`, `JGD+BP` and `FG+BP`. These four baseline models aim to reduce two important volume-related communication cost metrics total volume and maximum volume. The remaining four evaluated models are the `CHG`-enhanced versions in which the communication tasks are assigned to processors using communication hypergraph model. These `CHG`-enhanced models are referred to as `1D+CHG`, `CKBD+CHG`, `JGD+CHG` and `FG+CHG`. These aim to reduce total message count and maximum volume. Hence, we evaluate the merit of reducing a latency-related cost metric in partitioning. We use PaToH [31, 61] to partition the computational hypergraphs formed in the first phase of all models and the communication hypergraphs formed in the second phase of the `CHG`-enhanced models.

We implemented CGNE and CGNR solvers via the PETSc toolkit [59] and utilized the mentioned models for partitioning the coefficient matrix and vectors in these solvers. Since obtained runtime results for both solvers are similar, we only present speedup results corresponding to CGNR. Note that the metrics regarding partitioning models (Section 4.1) such as total volume, message count, etc. are the same for both solvers as they contain the same type of communication operations.

All models are tested with 28 matrices chosen from the UFL matrix collection [63]. The properties of these matrices are presented in Table 2. The evaluated models are tested on a BlueGene/Q system with varying number of processors $K \in \{256, 512, 1024, 2048, 4096, 8192\}$. A node on this system consists of 16 cores (single PowerPC A2 processor) with 1.6 GHz clock frequency and 16 GB memory. The nodes are connected by a 5D torus chip-to-chip network. We only consider the case of strong scaling.

Table 2: Test matrices and their properties.

	Number of		Nonzeros				
				per row		per column	
Matrix	rows/cols	nonzeros	avg	min	max	min	max
venkat01	62,424	1,717,792	27.52	16	44	16	44
mc2depi	525,825	2,100,225	3.99	2	4	2	4
poisson3Db	85,623	2,374,949	27.74	6	145	6	145
thermomech_dK	204,316	2,846,228	13.93	7	20	7	20
stomach	213,360	3,021,648	14.16	7	19	6	22
FEM_3D_thermal2	147,900	3,489,300	23.59	12	27	12	27
laminar_duct3D	67,173	3,833,077	57.06	1	89	3	89
xenon2	157,464	3,866,688	24.56	1	27	1	27
iChem_Jacobian	274,087	4,137,369	15.10	5	17	5	17
torso3	259,156	4,429,042	17.09	7	22	6	21
tmt_unsym	917,825	4,584,801	5.00	3	5	3	5
t2em	921,632	4,590,832	4.98	1	5	1	5
Hamrle3	1,447,360	5,514,242	3.81	2	6	2	9
largebasis	440,020	5,560,100	12.64	4	14	4	14
Chevron4	711,450	6,376,412	8.96	2	9	2	9
cage13	445,315	7,479,343	16.80	3	39	3	39
PR02R	161,070	8,185,136	50.82	1	92	5	88
atmosmodl	1,489,752	10,319,760	6.93	4	7	4	7
kim2	456,976	11,330,020	24.79	4	25	5	25
memchip	2,707,524	14,810,202	5.47	2	27	1	27
Freescale1	3,428,755	18,920,347	5.52	1	27	1	25
circuit5M_dc	3,523,317	19,194,193	5.45	1	27	1	25
fem_hifreq_circuit	491,100	20,239,237	41.21	12	110	12	110
rajat31	4,690,002	20,316,253	4.33	1	1252	1	1252
CoupCons3D	416,800	22,322,336	53.56	20	76	20	76
Transport	1,602,111	23,500,731	14.67	5	15	5	15
ML_Laplace	377,002	27,689,972	73.45	26	74	26	74
RM07R	381,689	37,464,962	98.16	1	295	1	245

For the pre-communication stages of `CKBD+CHG` and `JGD+CHG`, we opted not to apply the communication hypergraph model since the partitioning corresponding to this stage leads to a number of very small communication hypergraphs in which the number of communication tasks (that is, vertices) and the number of messages per processor are very low. Hence, utilizing a dedicated tool to partition these hypergraphs often does not pay off. Instead, the simple aforementioned heuristic is able to obtain comparable partition qualities in a shorter amount of time. Hence, the pre-communication stages (expand tasks) of `CKBD+BP` and `CKBD+CHG` models, and `JGD+BP` and `JGD+CHG` models have the same quantities for the statistics presented in Section 4.1. However, for the post-communication stage, the respective quantities drastically differ in these models as the benefits of using the communication hypergraph model are more apparent.

4.1. Bandwidth and latency costs of partitioning models

Table 3 displays the metrics related to latency and bandwidth costs for the evaluated models. The metrics related to latency are highlighted under "Number of messages" columns and the metrics related to bandwidth are highlighted under "Communication volume" columns. The statistics for both total and maximum metrics are presented. The columns "Expand" and "Fold" in the table indicate the results obtained in the pre- and post-communication stages of 2D models, respectively. Recall that the 1D models (`1D+BP` and `1D+CHG`) have a single communication stage, which is the pre-communication stage in our case. The values are averaged over 20 test matrices separately for each K. The communication volume statistics are in terms of words. Note that the total/maximum number of messages and maximum volume of communication of `CKBD+BP` and `CKBD+CHG`, and `JGD+BP` and `JGD+CHG` are the same since they use the same heuristic to distribute expand tasks. Total communication volume of two successive K values (512 and 1024, 2048 and 4096, etc.) are the same for the models that are based on checkerboard and jagged partitioning since there exists same number of processor columns in the corresponding processor mesh. For instance, at $K=512$ and $K=1024$,

Table 3: Average communication requirements and speedups.

		Number of messages						Communication volume						
		Total			Maximum			Total			Maximum			
K	Model	Expand	Fold	Sum	Expand	Fold	Sum	Expand	Fold	Sum	Expand	Fold	Sum	Speedup
	1D+BP	2464	-	2464	17.9	-	17.9	164470	-	164470	728	-	**728**	143
	1D+CHG	1640	-	**1640**	13.5	-	13.5	245514	-	245514	1234	-	1234	148
	CKBD+BP	414	1728	2141	4.8	6.6	11.4	38476	145554	184029	367	759	1125	142
256	CKBD+CHG	414	1423	1836	4.8	5.8	**10.5**	38476	214657	253133	367	1176	1542	138
	JGD+BP	850	1266	2117	10.0	4.9	14.9	38476	112712	**151188**	311	599	910	**158**
	JGD+CHG	850	1131	1982	10.0	5.0	15.0	38476	165081	203557	311	959	1270	153
	FG+BP	1959	1751	3710	15.5	6.8	22.3	107993	48359	156352	546	275	821	150
	FG+CHG	1513	1257	2770	12.5	4.6	17.1	173449	74672	248121	847	377	1223	150
	1D+BP	5683	-	5683	21.6	-	21.6	226127	-	226127	513	-	**513**	236
	1D+CHG	3496	-	**3496**	15.5	-	15.5	327581	-	327581	829	-	829	231
	CKBD+BP	1019	3590	4608	6.4	6.8	13.1	57434	195114	252548	264	530	794	225
512	CKBD+CHG	1019	2885	3904	6.4	6.6	**13.0**	57434	280548	337981	264	804	1067	220
	JGD+BP	2088	2604	4692	11.7	5.6	17.3	57434	147488	**204922**	220	405	626	**247**
	JGD+CHG	2088	2267	4355	11.7	4.5	16.2	57434	212002	269436	220	609	829	238
	FG+BP	4261	3784	8045	17.7	8.0	25.7	144968	70492	215460	374	205	580	228
	FG+CHG	3165	2622	5787	13.8	4.8	18.6	228987	106423	335410	566	273	839	229
	1D+BP	13201	-	13201	27.0	-	27.0	314109	-	314109	359	-	**359**	294
	1D+CHG	7451	-	**7451**	16.9	-	16.9	441340	-	441340	568	-	568	343
	CKBD+BP	1587	9624	11211	5.9	9.1	15.0	57434	288909	346343	167	399	567	320
1024	CKBD+CHG	1587	6897	8484	5.9	7.1	**13.0**	57434	404550	461984	167	534	701	320
	JGD+BP	3571	6775	10346	11.7	7.4	19.1	57434	223665	**281099**	138	314	452	**354**
	JGD+CHG	3571	5266	8837	11.7	5.2	16.9	57434	314303	371737	138	430	568	341
	FG+BP	9286	8141	17427	20.6	8.5	29.1	192165	103683	295848	252	153	405	318
	FG+CHG	6537	5420	11957	15.1	6.1	21.3	297236	152370	449606	376	201	577	327
	1D+BP	30651	-	30651	30.9	-	30.9	437009	-	437009	256	-	**256**	355
	1D+CHG	16028	-	**16028**	19.0	-	19.0	591207	-	591207	389	-	389	450
	CKBD+BP	3885	21008	24892	7.3	9.4	16.7	82863	395862	478725	116	277	393	421
2048	CKBD+CHG	3885	14275	18159	7.3	8.5	**15.8**	82863	533056	615920	116	362	478	429
	JGD+BP	8301	14621	22921	14.1	7.4	21.5	82863	297989	**380852**	99	214	313	**468**
	JGD+CHG	8301	10833	19134	14.1	5.6	19.6	82863	404389	487252	99	284	383	457
	FG+BP	20050	17669	37719	22.3	10.3	32.6	251398	154339	405737	171	116	287	412
	FG+CHG	13494	11272	24766	16.3	8.2	24.5	379611	221040	600651	243	152	395	440
	1D+BP	71313	-	71313	37.0	-	37.0	615662	-	615662	185	-	**185**	372
	1D+CHG	36563	-	**36563**	21.8	-	21.8	810012	-	810012	267	-	267	530
	CKBD+BP	6122	55186	61308	6.7	13.5	20.3	82863	579257	662121	72	209	280	511
4096	CKBD+CHG	6122	32230	38352	6.7	8.7	**15.4**	82863	747663	830526	72	252	324	556
	JGD+BP	13440	38102	51543	12.9	10.4	23.2	82863	449614	**532477**	62	163	225	584
	JGD+CHG	13440	24711	38151	12.9	6.7	19.5	82863	584442	667306	62	203	264	**586**
	FG+BP	43084	38239	81323	24.8	11.0	35.7	326646	228666	555312	114	87	201	516
	FG+CHG	28683	24606	53289	18.0	8.8	26.7	486581	323623	810204	160	117	278	554
	1D+BP	160507	-	160507	42.9	-	42.9	871090	-	871090	137	-	**137**	392
	1D+CHG	81592	-	**81592**	26.0	-	26.0	1040719	-	1040719	175	-	175	560
	CKBD+BP	14589	125660	140249	7.9	15.7	23.6	115948	814437	930384	49	149	198	593
8192	CKBD+CHG	14589	70942	85531	7.9	10.7	**18.6**	115948	1003817	1119764	49	171	220	667
	JGD+BP	28838	85171	114009	13.9	11.4	25.2	115948	618294	**734242**	43	114	157	683
	JGD+CHG	28838	52882	81720	13.9	6.5	20.3	115948	769539	885486	43	137	180	**701**
	FG+BP	90483	81319	171801	26.3	13.3	39.6	430725	327017	757741	77	65	142	591
	FG+CHG	59692	52380	112072	19.8	10.1	29.9	572597	432085	1004682	99	84	183	663

The bold values in five columns total/maximum number of messages (Sum), total/maximum communication volume (Sum) and Speedup indicate the best values obtained by the respective model at a specific K.

the processor meshes are of sizes 32×16 and 32×32, respectively. We also present the average speedup values obtained by models to give an idea about the efficiency. A more detailed and accurate discussion with performance profiles and speedup curves can be found in the next section.

The major factors that determine overall latency and bandwidth costs are the maximum number of messages and the maximum volume of communication handled by a single processor, respectively. As seen in Table 3, with increasing number of processors, the maximum number of messages increases sharply, whereas maximum volume decreases despite the increase in total volume. For instance, for 1D+BP, when K increases from 256 to 8192 processors, the maximum number of messages increases from 17.9 to 42.9, whereas maximum volume decreases from 728 to 137 words, on average. Hence, the latency overhead on average increases by a factor of 2.4, whereas the bandwidth overhead on average decreases by a factor of 5.3. Moreover, the total message count increases more sharply compared to total volume: 65.1 times versus 5.3 times. These figures imply that with increasing number of processors, latency costs steadily become more important than bandwidth costs in determining overall communication cost of parallel SpMV operations. Hence, reducing latency costs should pay off with improved scalability, as will be seen in the following section. Observe that similar arguments hold for other partitioning models as well.

If we compare the partitioning models that do not use the communication hypergraph model among themselves (i.e., 1D+BP, CKBD+BP, JGD+BP and FG+BP) in terms of total communication volume, we see from Table 3 that JGD+BP obtains the best results, whereas CKBD+BP obtains the worst results. FG+BP is expected to achieve the best results in this metric since it offers the highest flexibility by performing the partitioning on a nonzero basis – the finest granularity available. However, the reason why JGD+BP achieves slightly better results than FG+BP in this metric is related not to models themselves but to the shortcomings of recursive bisectioning used in partitioning. The shortcomings of recursive bipartitioning are well known for high partitioning values [64, 65]. For example at $K = 4096$, FG+BP directly partitions the input matrix into 4096 parts whereas JGD+BP first partitions it into 64 parts, and for each of these parts, it partitions them into 64 parts again to obtain a 4096-way partition. Hence, by using smaller partition values compared to FG+BP, JGD+BP is relatively able to mitigate the drawbacks of recursive bisection. The poor performance of CKBD+BP in this metric is due to the use of multi-constraint partitioning. This limits the search space drastically, where the higher the number of constraints, the harder it is to get good quality partitions as the search space narrows down with increasing number of constraints. However, this is a tradeoff for CKBD+BP as it often achieves good results in total message count, which are comparable to those of JGD+BP at lower processor counts. At higher processor counts, like 4096 and 8192, JGD+BP achieves better results in total message count. This is again because the high number of constraints at high values of K leads to poor total volume in CKBD+BP, which in turn affects the total message count as a side effect by causing an increase. As expected, the smallest maximum number of messages is obtained by CKBD+BP as it bounds the communication to specific rows and columns of the processor mesh in both stages of communication. FG+BP is often the worst in terms of total and maximum number of messages because it causes increases in these metrics in order to reduce total volume.

To aid the assessment of benefits of the communication hypergraph, we present Table 4. In this table, each latency-improved (CHG-enhanced) model's performance metrics are normalized with respect to those of their baselines. In other words, the results of 1D+CHG are normalized with respect to those of 1D+BP, the results of CKBD+CHG are normalized with respect to those of CKBD+BP, etc. The normalization is performed on a matrix basis and the averages of these normalized values over 28 matrices are given separately for each K. As seen from the table, CHG-enhanced models improve the total message count drastically, as minimizing this metric is one the main objectives in these models. For example at 2048 processors, 1D+CHG achieves 36% improvement over 1D+BP, CKBD+CHG achieves 23% improvement over CKBD+BP, JGD+CHG achieves 13% improvement over JGD+BP, and FG+CHG achieves 23% improvement over FG+BP. However, this comes at the cost of increased total volume. Again at 2048 processors, 1D+CHG increases the total volume by 40%, CKBD+CHG by 33%, JGD+CHG by 30%, and FG+CHG by 49%. The crucial observation, however, is that the message count improvements of partitioning models that rely on the communication hypergraph model tend to increase with increasing number of processors. For example, 1D+CHG achieves a 24% improvement over 1D+BP at $K = 256$ in total message count and this improvement becomes 37% at $K = 8192$. This improvement increases from 11% to 33%, 2% to 24%, and 14% to 24% for CKBD+CHG, JGD+CHG and FG+CHG,

Table 4: Comparison of partitioning models with communication hypergraphs normalized with respect to their baseline counterparts averaged over all matrices for each K.

		Number of messages						Communication volume					
		Total			Maximum			Total			Maximum		
K	Model	Expand	Fold	Sum	Expand	Fold	Sum	Expand	Fold	Sum	Expand	Fold	Sum
256	1D+CHG	0.76	-	0.76	0.85	-	0.85	1.51	-	1.51	1.66	-	1.66
	CKBD+CHG	1.00	0.86	0.89	1.00	1.01	0.98	1.00	1.49	1.41	1.00	1.54	1.38
	JGD+CHG	1.00	0.95	0.98	1.00	1.21	1.09	1.00	1.45	1.35	1.00	1.53	1.36
	FG+CHG	0.87	0.85	0.86	0.96	0.81	0.90	1.56	1.51	1.56	1.53	1.26	1.47
512	1D+CHG	0.72	-	0.72	0.84	-	0.84	1.48	-	1.48	1.62	-	1.62
	CKBD+CHG	1.00	0.85	0.88	1.00	1.41	1.08	1.00	1.46	1.37	1.00	1.52	1.35
	JGD+CHG	1.00	0.92	0.96	1.00	0.97	0.97	1.00	1.42	1.32	1.00	1.50	1.33
	FG+CHG	0.85	0.84	0.84	0.94	0.80	0.88	1.55	1.50	1.54	1.52	1.23	1.44
1024	1D+CHG	0.67	-	0.67	0.77	-	0.77	1.44	-	1.44	1.61	-	1.61
	CKBD+CHG	1.00	0.78	0.81	1.00	1.03	0.97	1.00	1.44	1.38	1.00	1.37	1.27
	JGD+CHG	1.00	0.86	0.91	1.00	0.88	0.94	1.00	1.41	1.34	1.00	1.42	1.29
	FG+CHG	0.81	0.81	0.80	0.86	1.37	0.87	1.53	1.48	1.51	1.51	1.24	1.43
2048	1D+CHG	0.64	-	0.64	0.74	-	0.74	1.40	-	1.40	1.57	-	1.57
	CKBD+CHG	1.00	0.74	0.77	1.00	1.02	0.99	1.00	1.39	1.33	1.00	1.33	1.23
	JGD+CHG	1.00	0.82	0.87	1.00	0.94	0.95	1.00	1.37	1.30	1.00	1.37	1.25
	FG+CHG	0.77	0.77	0.77	0.82	0.96	0.84	1.51	1.45	1.49	1.46	1.32	1.41
4096	1D+CHG	0.65	-	0.65	0.68	-	0.68	1.39	-	1.39	1.54	-	1.54
	CKBD+CHG	1.00	0.66	0.69	1.00	0.77	0.84	1.00	1.35	1.31	1.00	1.27	1.19
	JGD+CHG	1.00	0.74	0.80	1.00	0.82	0.90	1.00	1.33	1.29	1.00	1.33	1.23
	FG+CHG	0.77	0.78	0.77	0.82	0.85	0.81	1.50	1.46	1.49	1.45	1.38	1.42
8192	1D+CHG	0.63	-	0.63	0.69	-	0.69	1.35	-	1.35	1.47	-	1.47
	CKBD+CHG	1.00	0.64	0.67	1.00	0.79	0.84	1.00	1.30	1.26	1.00	1.23	1.16
	JGD+CHG	1.00	0.69	0.76	1.00	0.70	0.85	1.00	1.29	1.25	1.00	1.30	1.21
	FG+CHG	0.76	0.76	0.76	0.82	0.75	0.78	1.47	1.42	1.45	1.44	1.40	1.42

respectively. This implies that the benefits obtained using the communication hypergraph model are more prominent at higher processor counts. Note that for CKBD+CHG and JGD+CHG, normalized total message average is closer to fold message average rather than expand message average, which is also the case for normalized average volume. This is because the message count and communication volume in the post-communication stage are much higher than the pre-communication stage in these models. This is also where the communication hypergraph model is expected to perform well.

The models that use the communication hypergraph model improve the maximum number of messages as well. This is a consequence of the reduction in total message count. In Table 3, if we compare partitioning models in this metric, it can be seen that CKBD+CHG obtains the best results which is usually followed by JGD+CHG. For example, at 8192 processors on average, the maximum number of messages handled by a single processor for CKBD+CHG is only 18.6 and for JGD+CHG it is only 20.3. These values are followed by CKBD+BP with 23.6 and JGD+BP with 25.2. When we examine the other important metric the maximum volume in Table 4, it is seen that the models that rely on communication hypergraph model close the gap with their baseline counterparts with increasing K. For instance, when K increases from 256 to 8192 processors, the increase in maximum volume incurred by the use of the communication hypergraph model decreases from 66%, 38%, 36% and 47% to 47%, 16%, 21% and 42% in 1D+CHG, CKBD+CHG, JGD+CHG and FG+CHG, respectively, compared to their baseline counterparts. This is an important benefit of the communication hypergraph model since it strives for balancing volume.

The sequential partitioning times of the evaluated models are given in Table 5 averaged over all matrices and K values. The CHG times (indicated via +CHG row include only the partitioning times of communication hypergraphs formed for the respective model. As expected, the fine-grain model has the highest partitioning time as the hypergraphs formed in this model are typically larger. The partitioning times of the communi-

Table 5: Average partitioning times (sequential, in seconds).

	1D+BP	CKBD+BP	JGD+BP	FG+BP
	32.43	33.79	21.68	114.22
+CHG	1.68	0.98	1.13	1.98

cation hypergraphs are quite low compared to the respective original partitionings since they are small as they contain only the vertices that correspond to the vector elements that necessitate communication. Note that CKBD+CHG, JGD+CHG and FG+CHG form a number of communication hypergraphs that can independently be partitioned, hence the partitioning of them can easily be parallelized. A more healthy comparison of partitioning overhead for 2D models can be found in [49].

4.2. Speedup analysis

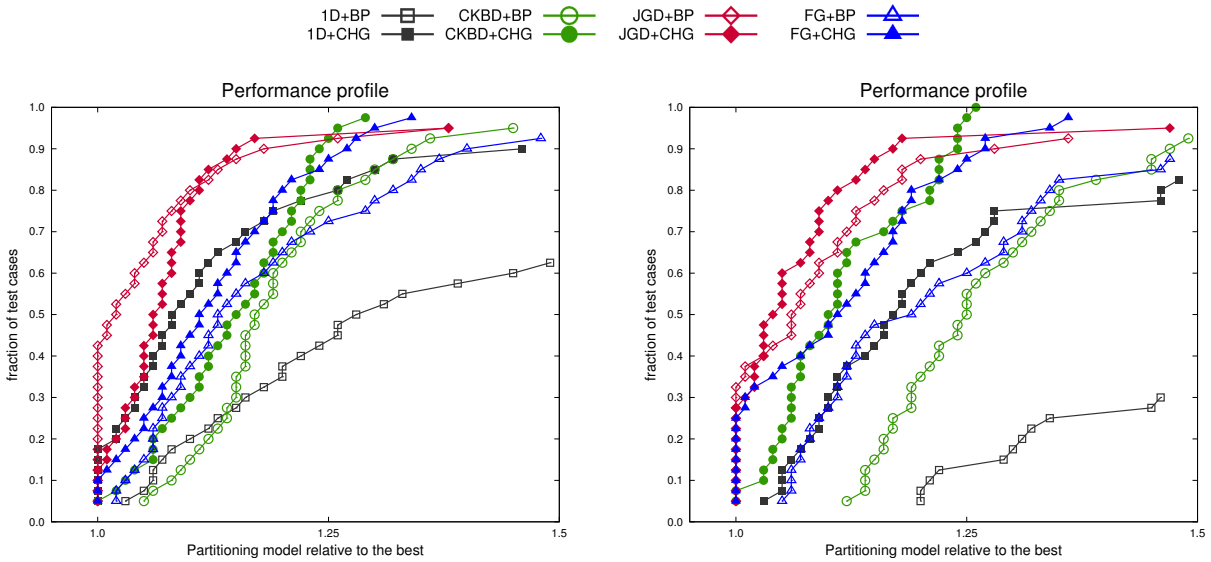
For a detailed comparison of the partitioning models in terms of parallel solver running times/speedups, we present the performance profiles in Figure 8. Performance profiles provide a better understanding of the characteristics of the compared models as they capture the relative performance of the compared models more accurately [66]. A point x, y in a profile reads as the respective model is within the x factor of the best result in y fraction of the test instances. In other words, the closer the performance profile of a scheme to the y-axis, the better it is. A test instance in our case is the parallel solver running time obtained for a specific matrix and K. We compare the performances of partitioning models for all K values in Figure 8b and for $K \in 4096, 8192$ in Figure 8c. The former contains 168 instances and the latter contains 56 instances.

When we compare the models considering all K values in Figure 8b, JGD+BP is clearly the best performing model followed by JGD+CHG. JGD+BP obtains the best results for more than 40% of the test cases and exhibits very good performance for a very large fraction of the test cases. These two models are followed by two models that use communication hypergraph: 1D+CHG and FG+CHG. Except jagged model, applying the communication hypergraph seems to improve performance of the partitioning models as 1D+CHG, CKBD+CHG and FG+CHG perform better than 1D+BP, CKBD+BP and FG+BP, respectively. 1D+BP obtains the worst results, proving itself to be not a viable partitioning model compared to the 2D models as long as communication hypergraph is not used for it.

Figure 8c is presented to better assess the benefits of using the communication hypergraph model. As discussed, latency gets more important with increasing K and it is expected that the models using the communication hypergraph model should be performing better as K increases. If we consider the performances of parittioning models at only 4096 and 8192 processors, it can be seen from the figure that the models that use communication hypergraph improve the performance much more compared to the case when all K values are considered. In other words, for example, if we compare CKBD+BP and CKBD+CHG in Figure 8b and Figure 8c the performance difference between them increases in favor of CKBD+CHG in Figure 8c. This can be observed for all parittioning models, i.e., by comparing 1D+BP and 1D+CHG, CKBD+BP and CKBD+CHG, JGD+BP and JGD+CHG, FG+BP and FG+CHG in Figure 8b and Figure 8c. This is also validated as JGD+CHG can be said to be the best performing model in Figure 8c followed by JGD+BP. These two models are again followed by two models that use the communication hypergraph: FG+CHG and CKBD+CHG. These figures show that the communication hypergraph proves to a valuable method for achieving scalability.

We present the obtained speedup values of eight evaluated models in Figure 9. Among 28 matrices, we present the speedups of 12 matrices here to keep the discussions simpler. The rest are given in Appendix G. The number of processors varies from 256 to 8192. The experiments are performed with the CGNR solver which is implemented via the PETSc toolkit.

As seen from the speedup curves, the models that adopt the communication hypergraph model often exhibit better scalability compared to their baseline counterparts. Moreover, the difference gets more prominent with increasing number of processors as latency becomes the determining factor for performance. When we compare 1D+BP with 1D+CHG, 1D+CHG achieves superior scalability in all matrices. In 2D models, applying the communication hypergraph model usually improves scalability. For example, in matrices circuit5M_dc, CoupCons3D, fem_hifreq_circuit, Freescale1, memchip, ML_Laplace, rajat31 and RM07R, the partitioning models CKBD+CHG, JGD+CHG and FG+CHG improve performance of their baseline counterparts CKBD+BP,



(b) $K \in \{256, 512, 1024, 2048, 4096, 8192\}$. (c) $K \in \{4096, 8192\}$.

Figure 8: Performance profiles of eight partitioning models.

`JGD+BP` and `FG+BP`, respectively. In the remaining matrices, latency-improved versions of 2D models either achieve close or slightly worse performance. This is mainly due to the fact that in these matrices, the latency costs obtained in the initial partitionings are already very low due to the characteristics of these matrices and further trying to improve them does not pay off since, at the other hand, the bandwidth costs are increased.

In terms of speedup values, it can be said that 2D models generally exhibit better scalability than 1D models. In most of the matrices, the best of 2D models exhibits better scalability than the best of 1D models, by obtaining lower runtime results. With increasing number of processors, this difference becomes more obvious. This can be attributed to the fact that 2D models have more flexibility in partitioning, which leads them to optimize communication objectives better.

Among all models, the performance of `JGD+BP` and `JGD+CHG` is especially worth to note. On average, these two models achieve quite good performance in terms of speedup. In matrices `atmosmodl`, `Chevron4` and `kim2`, `JGD+BP` obtains better speedup values. The communication costs of these matrices are largely determined by bandwidth costs rather than by latency costs. From this point of view, `FG+BP` might be expected to achieve the best results. However, `FG+BP` usually causes high latency costs, even in the case of these matrices which are not latency bound. `JGD+BP`, on the other hand, obtains slightly worse bandwidth costs while drastically improving latency costs compared to `FG+BP`, finding a balance between `FG+BP` and `CKBD+BP`, hence performing best in these matrices.

In the remaining nine matrices, the latency-improved versions of 2D models obtain better scalability. The `JGD+CHG` model is almost always among the two best performing models. When comparing `CKBD+CHG`, `JGD+CHG` and `FG+CHG`, although `CKBD+CHG` has the lowest maximum number of messages, it has the highest maximum volume, whereas `JGD+CHG` obtains slightly worse maximum number of messages compared to `CKBD+CHG` and has the lowest maximum volume among these three models. Hence it is able to strike a good balance between minimizing latency and bandwidth costs, which leads to better scalability. Although `FG+CHG` has low bandwidth costs, its high latency costs cause it to perform relatively poorly among these models (for example for `rajat31` matrix, the maximum number of messages of `FG+CHG` at 4096 processors is around hundreds).

A noteworthy case is seen for the `cage13` matrix. This matrix is characterized with its very high latency cost. For example, at $K = 8192$ the maximum number of messages is 345 for `1D+BP`. In such matrices, bounding *and* reducing the message count works better than by solely reducing it. As seen in Figure 9, the two models that do so, `CKBD+BP` and `CKBD+CHG`, achieve better scalability.

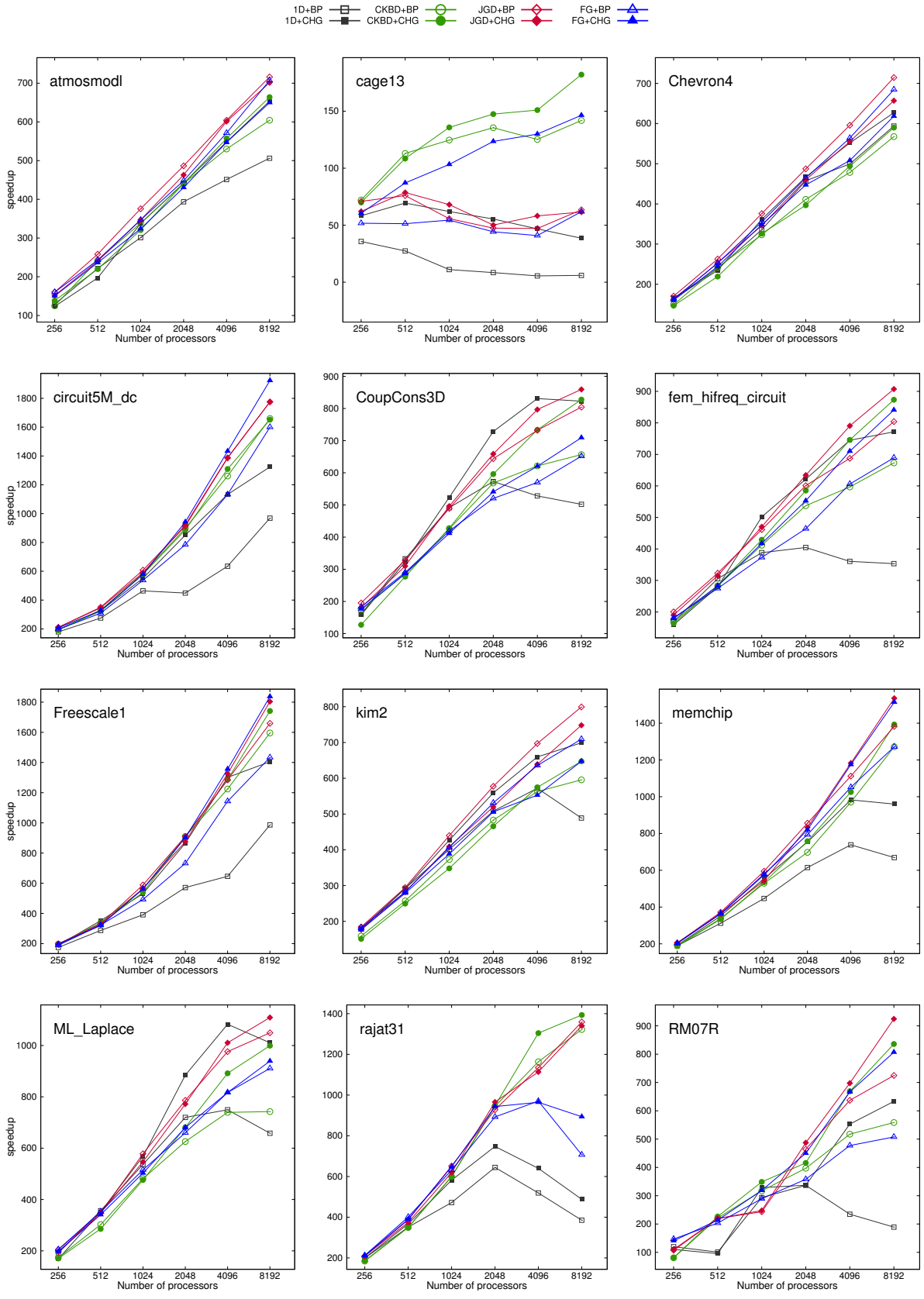


Figure 9: Speedup curves.

Judging from performance profiles and speedup curves, we can safely recommend the use of `JGD+CHG` model when latency costs prove vital in performance, and `JGD+BP` model when latency and bandwidth costs are comparable. Although other models may perform better for specific matrices in some cases, it can be said that `JGD+BP` and `JGD+CHG` will not perform too inferior even in these cases (for example, `circuit5M_dc` matrix).

5. Conclusions

This work focused on reducing latency costs of parallel sparse-matrix vector operations by proposing and utilizing several models based on 1D and 2D matrix partitioning. The latency costs are improved by using the communication hypergraph models, where the main motivation is to minimize the number of messages communicated in parallel operations. The described and tested models are realized with CGNE and CGNR solvers via PETSc toolkit on a modern HPC system. We compared a total of eight partitioning models, scaling them up to 8K processors.

The results of extensive experiments indicate that along with the bandwidth costs, latency costs should certainly be considered in order to achieve scalable performance. Solely addressing a single of them hurts scalability and leads to poor performance. Our findings indicate that among the partitioning models, the 2D jagged model and its latency-improved version obtain the most promising results. This superior performance is the result of obtaining a good balance between minimizing latency and bandwidth costs.

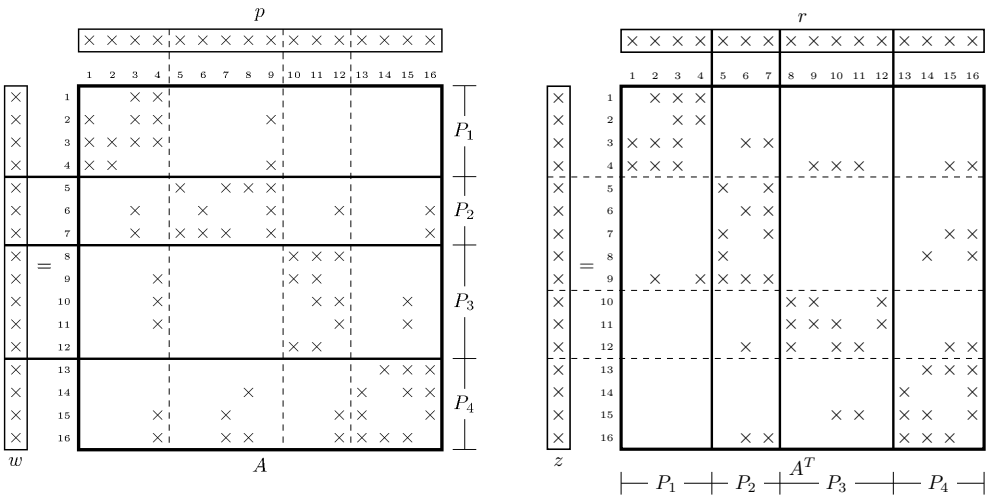
Acknowledgements

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

Appendix A. Hypergraph partitioning

A hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ consists of a set of vertices $\mathcal{V}$ and a set of nets $\mathcal{N}$ [67]. Each net $n_j \in \mathcal{N}$ connects a subset of vertices, which are referred to as pins of n_j. The set of nets that connect vertex v_i is denoted by $Nets(v_i)$. The degree of a vertex is equal to the number of nets that connect this vertex, i.e., $d_i = |Nets(v_i)|$. A weight value w_i is associated with each vertex v_i.

Given a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$, $\Pi = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$ is called a K-way partition of vertex set $\mathcal{V}$ if each part V_k is non-empty, parts are pairwise disjoint and the union of K parts is equal to $\mathcal{V}$. In Π, a net is said to connect a part if it connects at least one vertex in that part. The set of parts connected by a net n_j is called its connectivity set and is denoted by $\Lambda(n_j)$. The connectivity $\lambda(n_j) = |\Lambda(n_j)|$ of n_j is equal to the number of parts connected by this net. Net n_j is said to be an internal net if it connects only one part ($\lambda(n_j) = 1$), and an external net if it connects more than one part ($\lambda(n_j) > 1$). In Π, the weight of a part is the sum of the weights of vertices in that part. In the hypergraph partitioning (HP) problem, the objective is to minimize the *cutsize*, which is defined as $cutsize(\Pi) = \sum_{n_j \in \mathcal{N}} (\lambda(n_j) - 1)$. This objective function is known as the connectivity-1 cutsize metric and is widely used in the scientific computing community [31, 68, 69]. The partitioning *constraint* is to satisfy a balance on part weights, $(W_{max} - W_{avg})/W_{avg} \leq \epsilon$, where W_{max} and W_{avg} are the maximum and the average part weights, respectively, and ϵ is the user-defined imbalance ratio. The HP problem is known to be NP-hard [70]. Nonetheless, there exist successful HP tools such as PaToH [31], hMeTiS [60] and Mondriaan [33].



(a) Row-parallel $w = Ap$. (b) Column-parallel $z = A^T r$.

Figure B.10: Row-parallel matrix-vector and column-parallel matrix-transpose-vector multiplication.

Appendix B. Communication requirements of 1D partitioning

In 1D partitioning, $n \times n$ matrix A is partitioned either rowwise or columnwise. Assume that A is permuted into a $K \times K$ block structure as follows:

$$A_{BL} = \begin{bmatrix} A_{11} & A_{12} & \dots & A_{1K} \\ A_{21} & A_{22} & \dots & A_{2K} \\ \vdots & \vdots & \ddots & \vdots \\ A_{K1} & A_{K2} & \dots & A_{KK} \end{bmatrix},$$

where K denotes the number of processors in the parallel system and the size of block A_{kl} is $n_k \times n_l$. In rowwise partitioning, processor P_k is responsible for the kth row $[A_{k1} \dots A_{kK}]$ of size $n_k \times n$. In columnwise partitioning, processor P_k is responsible for the kth column block $[A_{1k}^T \dots A_{Kk}^T]^T$ of size $n \times n_k$. Throughout this section, without loss of generality, we assume a rowwise partition of A.

The vectors in an iterative solver should be partitioned conformally in order to avoid redundant communication during linear vector operations. For example, in the conjugate gradient solver, all vectors are partitioned conformally. In some solvers, we can utilize distinct vector partitions that separately apply to certain vectors. For example in CGNE and CGNR, it is possible to utilize two distinct partitions on the vectors. This enables utilization of a nonsymmetric partition for the coefficient matrix (see Appendix E for the details). The main motivation for adopting a nonsymmetric partition is that instead of enforcing the same partition on all vectors in the solver, we have more freedom by using a different partition on each distinct vector space – which accommodates more potential for reducing communication overheads in parallelization.

In a parallel solver, inner product operations necessitate global collective communications whereas matrix-vector or matrix-transpose-vector multiplications necessitate P2P communications. Consider parallel $w = Ap$ and $z = A^T r$ multiplies. An example for these operations is illustrated in Figure B.10 for $K = 4$ processors. Without loss of generality, assume that P_k is responsible for the kth row stripe of A, and thus the kth column stripe of A^T. Note that a rowwise partition on A induces a columnwise partition on A^T (Appendix E). Here, $w = Ap$ is performed with the row-parallel algorithm while $z = A^T r$ is performed with the column-parallel algorithm. The *row-parallel* algorithm necessitates a *pre-communication* stage in which the input vector elements are communicated. Each P_k sends the input vector elements that correspond to the nonzero *column* segments in off-diagonal blocks A_{ik}, $1 \leq i \neq k \leq K$. This is also referred

to as the *expand* operation since the same vector element can be sent to multiple processors. The vector elements that correspond to the columns which have at least one nonzero column segment in off-diagonal blocks (called coupling columns) necessitate expand operations. In Figure B.10a, eight elements of the input vector ($p[3]$, $p[4]$, $p[7]$, $p[8]$, $p[9]$, $p[12]$, $p[15]$, $p[16]$) need to be communicated. For example, P_3 sends $p[12]$ to P_2 and P_4, which need this element in their local computations. On the other hand, the *column-parallel* algorithm necessitates a *post-communication* stage in which the partial results of the output vector elements are communicated. Each P_k receives the output vector entries that correspond to the nonzero *row* segments in off-diagonal blocks A_{kj}, $1 \leq j \neq k \leq K$. This is also referred to as the *fold* operation since the partial results for the same vector element can be received from multiple processors. The vector elements that correspond to the rows which have at least one nonzero row segment in off-diagonal blocks (called coupling rows) necessitate fold operations. In Figure B.10b, eight elements of the output vector ($z[3]$, $z[4]$, $z[7]$, $z[8]$, $z[9]$, $z[12]$, $z[15]$, $z[16]$) need to be communicated. For example, P_3 receives partial results for $z[12]$ from P_2 and P_4 to compute the final value of $z[12]$. Observe that the communication of $p[12]$ in the row-parallel algorithm is the dual of the communication of $z[12]$ in the column-parallel algorithm.

It is possible to obtain 1D partitioning of sparse matrices using column-net and row-net hypergraph models. The details of these two models are given in Appendix C.

Appendix C. Two computational hypergraph models for 1D partitioning

There are several ways of obtaining a 1D rowwise/columnwise partitioning of coefficient matrix A. We briefly discuss two hypergraph models since they are central to the models proposed in this work. These models are also referred to as *computational hypergraph* models.

The column-net hypergraph model $\mathcal{H}_{\mathcal{R}} = (\mathcal{V}_{\mathcal{R}}, \mathcal{N}_{\mathcal{C}})$ can be used to obtain a rowwise partitioning of A [31]. In this model, vertex set $\mathcal{V}_{\mathcal{R}}$ represents the rows of A and net set $\mathcal{N}_{\mathcal{C}}$ represents the columns of A. There is a vertex $v_i \in \mathcal{V}_{\mathcal{R}}$ for each row r_i and there is a net $n_j \in \mathcal{N}_{\mathcal{C}}$ for each column c_j. Net n_j connects a subset of vertices that correspond to the rows that have a nonzero element in column c_j, i.e., $v_i \in n_j$ if and only if $a_{ij} \neq 0$. The weight w_i of vertex v_i is equal to the number of nonzeros in row r_i and represents the computational load associated with v_i.

The row-net hypergraph model $\mathcal{H}_{\mathcal{C}} = (\mathcal{V}_{\mathcal{C}}, \mathcal{N}_{\mathcal{R}})$ can be used to obtain a columnwise partitioning of A [31]. In this model, vertex set $\mathcal{V}_{\mathcal{C}}$ represents the columns of A and net set $\mathcal{N}_{\mathcal{R}}$ represents the rows of A. There is a vertex $v_j \in \mathcal{V}_{\mathcal{C}}$ for each column c_j and there is a net $n_i \in \mathcal{N}_{\mathcal{R}}$ for each row r_i. Net n_i connects a subset of vertices that correspond to the columns that have a nonzero element in row r_i, i.e., $v_j \in n_i$ if and only if $a_{ij} \neq 0$. The weight w_j of vertex v_j is equal to the number of nonzeros in column c_j and represents the computational load associated with v_j.

Partitioning hypergraphs $\mathcal{H}_{\mathcal{C}}$ and $\mathcal{H}_{\mathcal{R}}$ with the objective of minimizing cutsize corresponds to minimizing total communication volume incurred in parallel sparse-matrix vector multiplication while maintaining the partitioning constraint on part weights corresponds to maintaining a balance on computational loads of processors.

Appendix D. Communication hypergraph model for 1D partitioning

The communication hypergraph (CHG) model [53] is a means of distributing communication tasks among processors with the aim of minimizing latency. A communication task is defined as a subset of processors that involve in communicating a data object with a certain size. The CHG model strives to reduce the total number messages usually at the expense of increasing communication volume. However, although it increases the volume, it tries to obtain a balance on it. Reducing latency is a key factor to achieve scalability in large-scale systems as we show with our experiments. In this section, we review the CHG model for reducing latency overhead of 1D partitioned parallel $w = Ap$ and $z = A^T r$ multiplies.

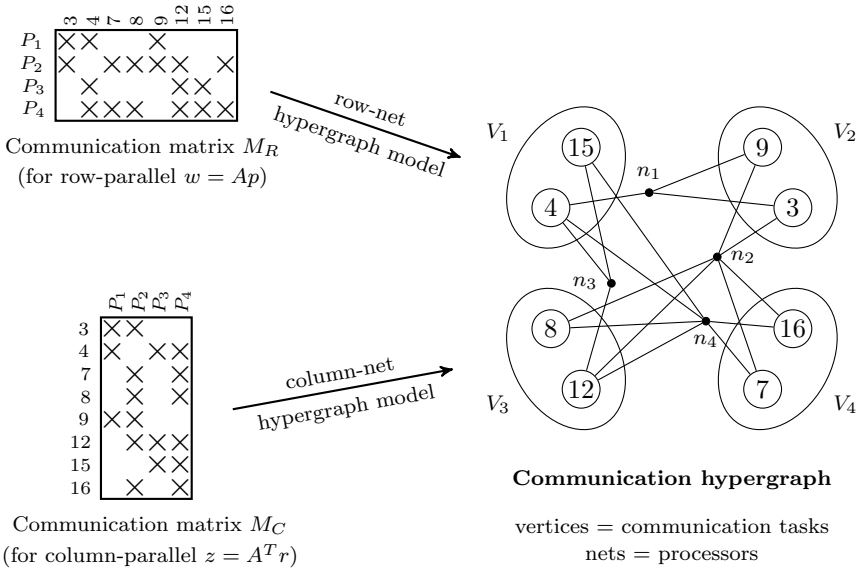


Figure D.11: Formation of the communication hypergraph from communication matrix, and a four-way partition on this hypergraph. Matrices M_R and M_C summarize the communication requirements of $w = Ap$ and $z = A^T r$ operations illustrated in Figure B.10.

Appendix D.1. Communication matrix

As the first step, we form communication matrices M_R and M_C to summarize the communication requirements of row-parallel $w = Ap$ and column-parallel $z = A^T r$, respectively. For row-parallel $w = Ap$, let p_C denote the p-vector elements that necessitate communication (via expand tasks). Communication matrix M_R is then a $K \times |p_C|$ matrix where the rows of M_R correspond to processors and the columns of M_R correspond to expand communication tasks. In M_R, $m_{kj} \neq 0$ if and only if the corresponding coupling column c_j has a nonzero column segment in the kth row stripe of A. For example in A (Figure B.10a), column 12 has a nonzero at the second row stripe, thus there exists a nonzero at the corresponding entry in M_R in Figure D.11 at the intersection of row P_2 and column 12. The nonzeros of column $c_j \in M_R$ signify the set of processors that participate in communicating $p_C[j]$. The nonzeros of row $r_k \in M_R$ signify all expand tasks that P_k takes part in. In Figure D.11, the third row in M_R has nonzero elements corresponding to columns 4, 12 and 15, indicating that P_3 is involved in communicating $p[4]$, $p[12]$ and $p[15]$. Hence, a nonzero $m_{kj} \in M_R$ actually implies that P_k participates in the communication of $p_C[j]$.

For column-parallel $z = A^T r$, let z_C denote the z-vector elements that necessitate communication (via fold tasks). Communication matrix M_C is then a $|z_C| \times K$ matrix where the rows of M_C correspond to fold communication tasks and the columns of M_C correspond to processors. In M_C, $m_{ik} \neq 0$ if and only if the corresponding coupling row r_i has a nonzero row segment in the kth column stripe of A^T. For example in A^T (Figure B.10b), row 12 has a nonzero at the second column stripe, thus there exists a nonzero at the corresponding entry in M_C in Figure D.11 at the intersection of row 12 and column P_2. The nonzeros of row $r_i \in M_C$ signify the set of processors that participate in communicating $z_C[i]$. The nonzeros of column $c_k \in M_C$ signify all fold tasks that P_k takes part in. In Figure D.11, the third column in M_C has nonzero elements corresponding to rows 4, 12 and 15, indicating that P_3 is involved in communicating $z[4]$, $z[12]$ and $z[15]$. Hence, a nonzero $m_{ik} \in M_C$ actually implies that P_k participates in the communication of $z_C[i]$.

Appendix D.2. Formation of communication hypergraph

The communication matrix is then used to form a hypergraph called *communication hypergraph*. We apply the row-net hypergraph model to communication matrix M_R (vertices = columns, nets = rows) to obtain the communication hypergraph $\mathcal{H}_R^{CM}$ and we apply the column-net hypergraph model to communication matrix M_C (vertices = rows, nets = columns) to obtain the communication hypergraph $\mathcal{H}_C^{CM}$. The vertex and net set of both hypergraphs are the same (see Figure D.11). In both hypergraphs, nets correspond to

Algorithm 1: CGNE and CGNR.

Set initial $\mathbf{x}_0$
$\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$
$\mathbf{p}_0 = \mathbf{A}^T\mathbf{r}_0$
1 **for** $i = 0, 1, \ldots$ **do**
2 $\quad \alpha_i = \langle \mathbf{r}_i, \mathbf{r}_i \rangle / \langle \mathbf{p}_i, \mathbf{p}_i \rangle$ ▷ CGNE
$\quad \alpha_i = \langle \mathbf{A}^T\mathbf{r}_i, \mathbf{A}^T\mathbf{r}_i \rangle / \langle \mathbf{A}\mathbf{p}_i, \mathbf{A}\mathbf{p}_i \rangle$ ▷ CGNR
3 $\quad \mathbf{x}_{i+1} = \mathbf{x}_i + \alpha_i \mathbf{p}_i$
4 $\quad \mathbf{r}_{i+1} = \mathbf{r}_i - \alpha_i \mathbf{A}\mathbf{p}_i$
5 $\quad \beta_i = \langle \mathbf{r}_{i+1}, \mathbf{r}_{i+1} \rangle / \langle \mathbf{r}_i, \mathbf{r}_i \rangle$ ▷ CGNE
$\quad \beta_i = \langle \mathbf{A}^T r_{i+1}, \mathbf{A}^T r_{i+1} \rangle / \langle \mathbf{A}^T\mathbf{r}_i, \mathbf{A}^T\mathbf{r}_i \rangle$ ▷ CGNR
6 $\quad \mathbf{p}_{i+1} = \mathbf{A}^T\mathbf{r}_{i+1} + \beta_i \mathbf{p}_i$

processors (there are K of them) and vertices correspond to communication tasks (there are $|p_C| = |z_C|$ of them). However, the semantics of these hypergraphs differ: the vertices in $\mathcal{H}_R^{CM}$ represent *expand* tasks in $w = Ap$, while the vertices in $\mathcal{H}_C^{CM}$ represent *fold* tasks in $z = A^T r$. A net n_k in both hypergraphs connects the set of vertices that correspond to communication tasks P_k participates in. Each vertex v_i is associated with a weight that signifies the volume of communication incurred by the corresponding expand or fold task. This value is generally equal to one less than the number of the nets v_i is connected by, i.e., $d_i - 1$.

Appendix D.3. Partitioning of the communication hypergraph

Obtaining a K-way partition $\Pi = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$ on $\mathcal{H}_R^{CM}$ or $\mathcal{H}_C^{CM}$ induces a communication task distribution for parallel matrix-vector or matrix-transpose-vector multiplies. Without loss of generality, assume that processor P_k is associated with part $\mathcal{V}_k$. Expand or fold communication tasks represented by the vertices in $\mathcal{V}_k$ are assigned to P_k by making this processor responsible for storing vector elements that necessitate these tasks. For instance in Figure D.11, since $v_{12} \in \mathcal{V}_3$, P_3 is held responsible for storing $p[12]$ and $z[12]$, and expand and fold tasks necessitated by these elements. Consider a net n_k in $\mathcal{H}_R^{CM}$ with the connectivity set $\Lambda(n_k)$. All parts except $\mathcal{V}_k$ in this set correspond to the processors that send a message to P_k, hence, $\lambda(n_k) - 1$ (or $\lambda(n_k)$ if $\mathcal{V}_k \notin \Lambda(n_k)$) is equal to the number of messages P_k *receives*. In a dual manner, consider the same net in $\mathcal{H}_C^{CM}$ again with the connectivity set $\Lambda(n_k)$. All parts except $\mathcal{V}_k$ in this set correspond to the processors that receive a message from P_k, hence, $\lambda(n_k) - 1$ (or $\lambda(n_k)$ if $\mathcal{V}_k \notin \Lambda(n_k)$) is equal to the number of messages P_k *sends*. In Figure D.11, the connectivity sets of nets are as follows: $\Lambda(n_1) = \{\mathcal{V}_1, \mathcal{V}_2\}$, $\Lambda(n_2) = \{\mathcal{V}_2, \mathcal{V}_3\, \mathcal{V}_4\}$, $\Lambda(n_3) = \{\mathcal{V}_1, \mathcal{V}_3\}$ and $\Lambda(n_4) = \{\mathcal{V}_1, \mathcal{V}_3, \mathcal{V}_4\}$, making a total of $(\lambda(n_1) - 1 = 1) + (\lambda(n_2) - 1 = 2) + (\lambda(n_3) - 1 = 1) + (\lambda(n_4) - 1 = 2) = 6$ messages. In [53], it is proven that partitioning a communication hypergraph with the aim of minimizing cutsize minimizes the total message count, while maintaining a balance among part weights preserves a balance on the communication volume.

By applying the CHG model, we obtain a different partition on rows and columns of the coefficient matrix and thus on its input and output space (for details, see Appendix F). Adopting a different partition finds its application in nonsymmetric sparse iterative solvers that allow distinct partitions on vectors and can be used to improve their scalability.

Appendix E. Partitioning vectors in CGNE and CGNR solvers

We describe why it is possible to use different partitions on the vectors used in CGNE and CGNR solvers. For other solvers, refer to [71]. We make the distinction between input and output space for the vectors in the solver. A vector is said to be in the *input* space of A if it is multiplied with A or it participates with the vectors in the input space of A through linear vector operations. On the other hand, a vector is said to be in the *output* space of A if it is obtained by multiplying A with another vector or it participates with the vectors in the output space of A through linear vector operations.

We present CGNE and CGNR algorithms in Algorithm 1. In each iteration of the solvers, there are two inner products, three SAXPY operations (for forming vectors p, r, x), one matrix-vector multiply of the form $w = Ap$ and one matrix-transpose-vector multiply of the form $z = A^T r$. In $w = Ap$, vectors p and w are in the input and output space of A, respectively. In $z = A^T r$, vectors r and z are in the input and output space of A^T, respectively. Consider a rowwise (columnwise) partition of A. This induces a columnwise (rowwise) partition on A^T. Hence, the input space of A coincides with the output space of A^T, and vice versa. This implies that the partition on vector p is conformal with the partition on vector z, and the partition on vector w is conformal with the partition on vector r. Since x is involved in linear vector operations with vector p (line 3), it should be partitioned conformally with vectors p and z to avoid unnecessary communication. As a result, we can have two distinct vector partitions in CGNE and CGNR: one on vectors p, z and x, and another one on vectors w and r.

Appendix F. Obtaining different vector partitions using communication hypergraph model

The communication hypergraph (CHG) model can be regarded as a post-processing phase to distribute the communication tasks. Although any partitioning model could be used, assume that the column-net computational hypergraph model is used in the first phase for $w = Ap$ (Appendix C). As computational tasks are represented by vertices that correspond to rows of the matrix, partitioning this hypergraph actually induces a partition of the rows of the matrix. The result of this partitioning determines the communication tasks in the second phase. Applying the CHG in the second phase induces a partition of the columns of the matrix since communication tasks represented by the vertices are expand-type tasks performed on input vector elements. Hence, in the first phase, we minimize the communication volume by obtaining a partition of the rows, and in the second phase we minimize the number of messages by obtaining a different partition of the columns. An example of these two phases can be traced from Figures B.10a and D.11. Assume that the result of the first phase is the partition obtained on A in Figure B.10a. Here, P_3 owns rows $8, 9, 10, 11$, 12 and columns 10, 11, 12. The column partition is subject to change after applying the CHG model, which assigns vertices 8 and 12 to $\mathcal{V}_3$, thus assigning columns 8 and 12 to P_3. Since columns 10 and 11 do not necessitate communication, they are not included in the CHG model and directly assigned to P_3 at the end of the first phase. As a result, P_3 owns columns 8, 10, 11, 12. Hence, we obtained a nonsymmetric partition of the rows and columns assigned to P_3.

Appendix G. Additional speedup curves

We provided 12 speedup curves in Section 4.2. In this section, we provide 16 speedup curves that belong to the remaining matrices. Note that these curves do not change the findings of the paper and they are added here for the sake of completeness.

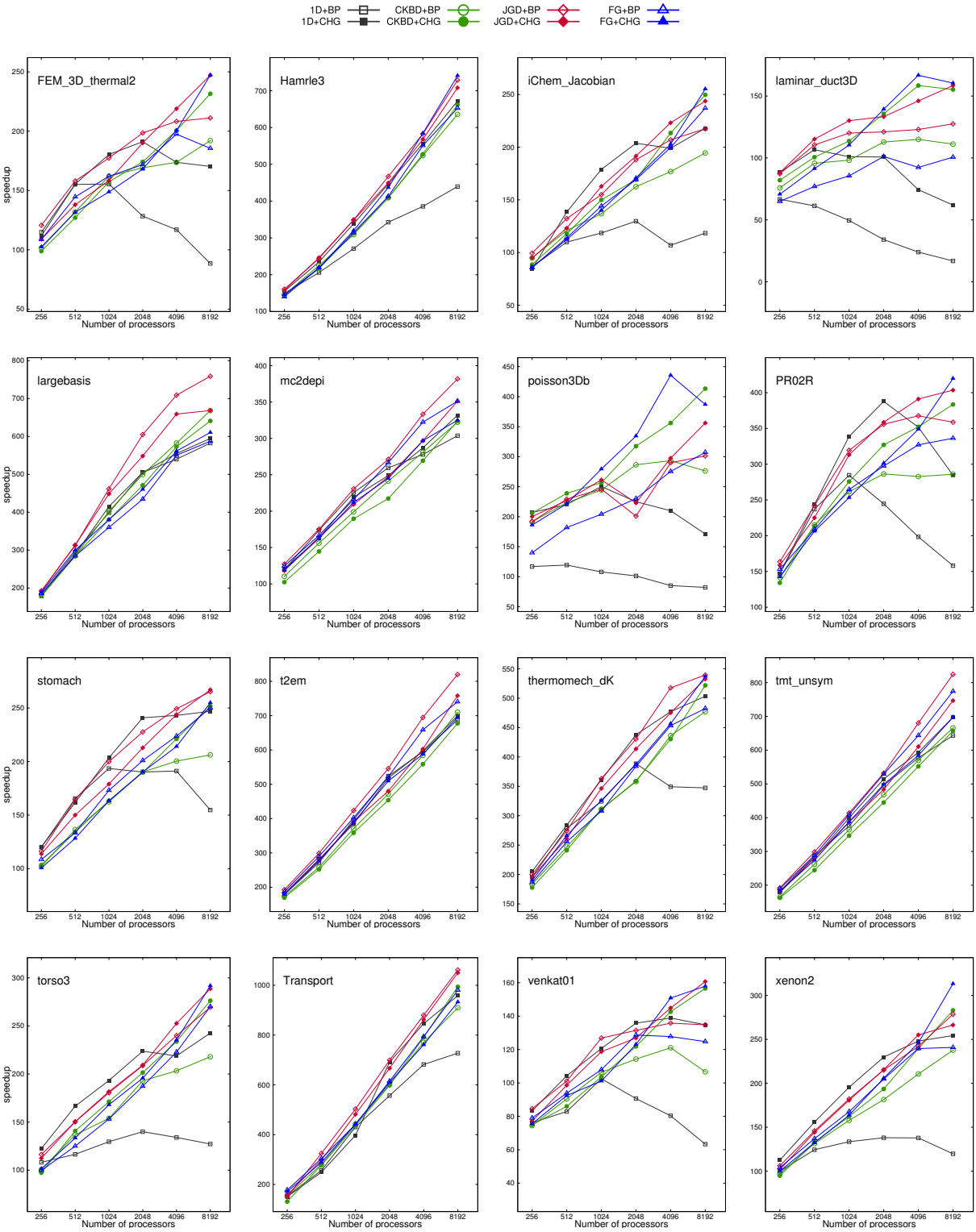


Figure F.12: Speedup curves for the remaining 16 matrices.

References

[1] D. A. Patterson, Latency lags bandwith, Commun. ACM 47 (2004) 71–75.

[2] S. L. Graham, M. Snir, C. A. Patterson (Eds.), Getting Up to Speed, The Future of Supercomputing, The National Academies Press, 2006.

[3] J. Dongarra, M. A. Heroux, Toward a New Metric for Ranking High Performance Computing Systems, Technical Report SAND2013-4744, Sandia National Laboratories, 2013.

[4] H. M. Bücker, M. Sauren, A parallel version of the unsymmetric Lanczos algorithm and its application to QMR, 1996.

[5] T.-X. Gu, X.-Y. Zuo, X.-P. Liu, P.-L. Li, An improved parallel hybrid bi-conjugate gradient method suitable for distributed parallel computing, J. Comput. Appl. Math. 226 (2009) 55–65.

[6] L. Yang, R. P. Brent, The improved BiCGStab method for large and sparse unsymmetric linear systems on parallel distributed memory architectures, in: Algorithms and Architectures for Parallel Processing, 2002. Proceedings. Fifth International Conference on, 2002, pp. 324–328. doi:10.1109/ICAPP.2002.1173595.

[7] T. P. Collignon, M. B. van Gijzen, Minimizing synchronization in IDR(s), Numerical Linear Algebra with Applications 18 (2011) 805–825.

[8] A. Chronopoulos, C. Gear, S-step iterative methods for symmetric linear systems, Journal of Computational and Applied Mathematics 25 (1989) 153 – 168.

[9] G. Meurant, Multitasking the conjugate gradient method on the CRAY X-MP/48, Parallel Computing 5 (1987) 267 – 280.

[10] E. F. D'Azevedo, V. L. Eijkhout, C. H. Romine, Conjugate Gradient Algorithms With Reduced Synchronization Overheads on Distributed Memory Processors, Technical Report 56, Lapack Working Note, 1993.

[11] Y. Saad, Practical use of polynomial preconditionings for the conjugate gradient method, SIAM Journal on Scientific and Statistical Computing 6 (1985) 865–881.

[12] L. T. Yang, R. P. Brent, The improved BiCG method for large and sparse linear systems on parallel distributed memory architectures, in: Proceedings of the 16th International Parallel and Distributed Processing Symposium, IPDPS '02, IEEE Computer Society, Washington, DC, USA, 2002, pp. 315–. URL: http://dl.acm.org/citation.cfm?id=645610.661567.

[13] Z. Bai, D. Hu, L. Reichel, A newton basis GMRES implementation, IMA Journal of Numerical Analysis 14 (1994) 563–581.

[14] A. T. Chronopoulos, S-step iterative methods for (non)symmetric (in)definite linear systems, SIAM J. Numer. Anal. 28 (1991) 1776–1789.

[15] A. Chronopoulos, C. Swanson, Parallel iterative S-step methods for unsymmetric linear systems, Parallel Computing 22 (1996) 623 – 641.

[16] M. Hoemmen, Communication-avoiding Krylov Subspace Methods, Ph.D. thesis, Berkeley, CA, USA, 2010. AAI3413388.

[17] W. D. Joubert, G. F. Carey, Parallelizable restarted iterative methods for nonsymmetric linear systems. part I: Theory, International Journal of Computer Mathematics 44 (1992) 243–267.

[18] E. Carson, N. Knight, J. Demmel, Avoiding Communication in Two-Sided Krylov Subspace Methods, Technical Report UCB/EECS-2011-93, EECS Department, University of California, Berkeley, 2011. URL: http://www.eecs.berkeley.edu/Pubs/TechRpts/2011/EECS-2011-93.html.

[19] L. Grigori, S. Moufawad, Communication Avoiding ILU0 Preconditioner, Rapport de recherche RR-8266, INRIA, 2013. URL: http://hal.inria.fr/hal-00803250.

[20] J. W. Demmel, M. T. Heath, H. A. van der Vorst, Parallel numerical linear algebra, Acta Numerica 2 (1993) 111–197.

[21] E. de Sturler, H. A. van der Vorst, Reducing the effect of global communication in GMRES(m) and CG on parallel distributed memory computers, Appl. Numer. Math. 18 (1995) 441–459.

[22] T. Hoefler, P. Gottschling, A. Lumsdaine, W. Rehm, Optimizing a conjugate gradient solver with non-blocking collective operations, Parallel Comput. 33 (2007) 624–633.

[23] P. Ghysels, T. Ashby, K. Meerbergen, W. Vanroose, Hiding global communication latency in the GMRES algorithm on massively parallel machines, SIAM Journal on Scientific Computing 35 (2013) C48–C71.

[24] P. Ghysels, W. Vanroose, Hiding global synchronization latency in the preconditioned conjugate gradient algorithm, Parallel Computing 40 (2014) 224 – 238. 7th Workshop on Parallel Matrix Algorithms and Applications.

[25] L. C. Mcinnes, B. Smith, H. Zhang, R. T. Mills, Hierarchical Krylov and nested Krylov methods for extreme-scale computing, Parallel Comput. 40 (2014) 17–31.

[26] O. Selvitopi, M. Ozdal, C. Aykanat, A novel method for scaling iterative solvers: Avoiding latency overhead of parallel sparse-matrix vector multiplies, 2014. doi:10.1109/TPDS.2014.2311804.

[27] G. Ballard, J. Demmel, O. Holtz, O. Schwartz, Minimizing communication in numerical linear algebra, SIAM Journal on Matrix Analysis and Applications 32 (2011) 866–901.

[28] J. Demmel, M. Hoemmen, M. Mohiyuddin, K. Yelick, Avoiding communication in sparse matrix computations, in: Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on, 2008, pp. 1–12. doi:10.1109/IPDPS.2008.4536305.

[29] M. Mohiyuddin, M. Hoemmen, J. Demmel, K. Yelick, Minimizing communication in sparse matrix solvers, in: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, SC '09, ACM, New York, NY, USA, 2009, pp. 36:1–36:12. URL: http://doi.acm.org/10.1145/1654059.1654096. doi:10.1145/1654059.1654096.

[30] G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, SIAM J. Sci. Comput. 20 (1998) 359–392.

[31] U. Çatalyurek, C. Aykanat, Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication, IEEE Trans. Parallel Distrib. Syst. 10 (1999) 673–693.

[32] F. Pellegrini, J. Roman, Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs, in: H. Liddell, A. Colbrook, B. Hertzberger, P. Sloot (Eds.), High-Performance Computing and Networking, volume 1067 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 1996, pp. 493–498. URL: http://dx.doi.org/10.1007/3-540-61142-8_588. doi:10.1007/3-540-61142-8_588.

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

[34] B. Hendrickson, T. G. Kolda, Graph partitioning models for parallel computing, Parallel Comput. 26 (2000) 1519–1534.

[35] W. J. Camp, S. J. Plimpton, B. A. Hendrickson, R. W. Leland, Massively parallel methods for engineering and science problems, Commun. ACM 37 (1994) 30–41.

[36] O. C. Martin, S. W. Otto, Partitioning of unstructured meshes for load balancing, Concurrency: Practice and Experience 7 (1995) 303–314.

[37] U. Çatalyurek, C. Aykanat, Decomposing irregularly sparse matrices for parallel matrix-vector multiplication, in: Proceedings of the Third International Workshop on Parallel Algorithms for Irregularly Structured Problems, IRREGULAR '96, Springer-Verlag, London, UK, 1996, pp. 75–86. URL: http://dl.acm.org/citation.cfm?id=646010.676990.

[38] C.-W. Ou, S. Ranka, Parallel incremental graph partitioning, Parallel and Distributed Systems, IEEE Transactions on 8 (1997) 884–896.

[39] A. Grama, G. Karypis, V. Kumar, A. Gupta, Introduction to Parallel Computing, 2nd ed., Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[40] A. Ogielski, W. Aiello, Sparse matrix computations on parallel processor arrays, SIAM Journal on Scientific Computing 14 (1993) 519–530.

[41] J. G. Lewis, R. A. van de Geijn, Distributed memory matrix-vector multiplication and conjugate gradient algorithms, in: Proceedings of the 1993 ACM/IEEE conference on Supercomputing, Supercomputing '93, ACM, New York, NY, USA, 1993, pp. 484–492. URL: http://doi.acm.org/10.1145/169627.169788. doi:10.1145/169627.169788.

[42] B. Hendrickson, R. Leland, S. Plimpton, An efficient parallel algorithm for matrix-vector multiplication, International Journal of High Speed Computing 7 (1995) 73–88.

[43] A. Buluç, K. Madduri, Parallel breadth-first search on distributed memory systems, in: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, SC '11, ACM, New York, NY, USA, 2011, pp. 65:1–65:12. URL: http://doi.acm.org/10.1145/2063384.2063471. doi:10.1145/2063384.2063471.

[44] A. Yoo, A. H. Baker, R. Pearce, V. E. Henson, A scalable eigensolver for large scale-free graphs using 2D graph partitioning, in: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, SC '11, ACM, New York, NY, USA, 2011, pp. 63:1–63:11. URL: http://doi.acm.org/10.1145/2063384.2063469. doi:10.1145/2063384.2063469.

[45] U. V. Çatalyürek, Hypergraph Models for Sparse Matrix Partitioning and Reordering, Ph.D. thesis, 1999.

[46] U. Çatalyürek, C. Aykanat, A fine-grain hypergraph model for 2D decomposition of sparse matrices, in: Proceedings of the 15th International Parallel & Distributed Processing Symposium, IPDPS '01, IEEE Computer Society, Washington, DC, USA, 2001, pp. 118–. URL: http://dl.acm.org/citation.cfm?id=645609.663255.

[47] U. Çatalyürek, C. Aykanat, A hypergraph-partitioning approach for coarse-grain decomposition, in: Proceedings of the 2001 ACM/IEEE Conference on Supercomputing, SC '01, ACM, New York, NY, USA, 2001, pp. 28–28. URL: http://doi.acm.org/10.1145/582034.582062. doi:10.1145/582034.582062.

[48] B. Uçar, C. Aykanat, Minimizing communication cost in fine-grain partitioning of sparse matrices, in: A. Yazıcı, C. Şener (Eds.), Computer and Information Sciences - ISCIS 2003, volume 2869 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2003, pp. 926–933. URL: http://dx.doi.org/10.1007/978-3-540-39737-3_115. doi:10.1007/978-3-540-39737-3_115.

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

[50] E. G. Boman, K. D. Devine, S. Rajamanickam, Scalable matrix computations on large scale-free graphs using 2D graph partitioning, in: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC '13, ACM, New York, NY, USA, 2013, pp. 50:1–50:12. URL: http://doi.acm.org/10.1145/2503210.2503293. doi:10.1145/2503210.2503293.

[51] R. H. Bisseling, W. Meesen, Communication balancing in parallel sparse matrix-vector multiply, Electronic Transactions on Numerical Analysis 21 (2005) 47–65.

[52] U. V. Çatalyürek, M. Deveci, K. Kaya, B. Uçar, UMPA: A Multi-objective, multi-level partitioner for communication minimization, in: D. A. Bader, H. Meyerhenke, P. Sanders, D. Wagner (Eds.), Graph Partitioning and Graph Clustering 2012, volume 588 of *Contemporary Mathematics*, AMS, 2013, pp. 53–66. URL: http://hal.inria.fr/hal-00763563. doi:10.1090/conm/588/11704.

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

[54] R. W. Freund, G. H. Golub, N. M. Nachtigal, Iterative solution of linear systems, Acta Numerica 1 (1992) 57–100.

[55] N. Nachtigal, S. Reddy, L. Trefethen, How fast are nonsymmetric matrix iterations?, SIAM Journal on Matrix Analysis and Applications 13 (1992) 778–795.

[56] Y. Saad, Iterative Methods for Sparse Linear Systems, 2nd ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2003.

[57] H. Elman, Iterative methods for large, sparse, nonsymmetric systems of linear equations., Dissertation Abstracts International Part B: Science and Engineering[DISS. ABST. INT. PT. B- SCI. & ENG.], 43 (1982) 1982 – 1982.

[58] R. Freund, N. Nachtigal, QMR: a quasi-minimal residual method for non-Hermitian linear systems, Numerische Mathe-

matik 60 (1991) 315–339.
[59] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, K. Rupp, B. F. Smith, H. Zhang, PETSc Users Manual, Technical Report ANL-95/11 - Revision 3.5, Argonne National Laboratory, 2014. URL: http://www.mcs.anl.gov/petsc.
[60] G. Karypis, R. Aggarwal, V. Kumar, S. Shekhar, Multilevel hypergraph partitioning: applications in VLSI domain, IEEE Trans. Very Large Scale Integr. Syst. 7 (1999) 69–79.
[61] C. Aykanat, B. B. Cambazoglu, B. Uçar, Multi-level direct k-way hypergraph partitioning with multiple constraints and fixed vertices, J. Parallel Distrib. Comput. 68 (2008) 609–625.
[62] E. Horowitz, S. Sahni, Fundamentals of Computer Algorithms, Computer Science Press, Rockville, MD, USA, 1978.
[63] T. A. Davis, Y. Hu, The University of Florida sparse matrix collection, ACM Trans. Math. Softw. 38 (2011) 1:1–1:25.
[64] G. Karypis, V. Kumar, Multilevel k-way hypergraph partitioning, in: Proceedings of the 36th Annual ACM/IEEE Design Automation Conference, DAC '99, ACM, New York, NY, USA, 1999, pp. 343–348. URL: http://doi.acm.org/10.1145/309847.309954. doi:10.1145/309847.309954.
[65] H. D. Simon, S.-H. Teng, How good is recursive bisection?, SIAM J. Sci. Comput. 18 (1997) 1436–1445.
[66] E. D. Dolan, J. J. Mor, Benchmarking optimization software with performance profiles, Mathematical Programming 91 (2002) 201–213.
[67] C. Berge, Graphs and Hypergraphs, Elsevier Science Ltd, 1985.
[68] C. Aykanat, A. Pinar, U. V. Çatalyürek, Permuting sparse rectangular matrices into block-diagonal form, SIAM J. Sci. Comput. 25 (2004) 1860–1879.
[69] B. Uçar, C. Aykanat, Revisiting hypergraph models for sparse matrix partitioning, SIAM Rev. 49 (2007) 595–603.
[70] T. Lengauer, Combinatorial algorithms for integrated circuit layout, John Wiley & Sons, Inc., New York, NY, USA, 1990.
[71] B. Uçar, C. Aykanat, Partitioning sparse matrices for parallel preconditioned iterative methods, SIAM J. Sci. Comput. 29 (2007) 1683–1709.