This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2016.2577024, IEEE Transactions on Parallel and Distributed Systems

A Recursive Hypergraph Bipartitioning Framework for Reducing Bandwidth and Latency Costs Simultaneously

Oguz Selvitopi, Seher Acer, and Cevdet Aykanat

Abstract—Intelligent partitioning models are commonly used for efficient parallelization of irregular applications on distributed systems. These models usually aim to minimize a single communication cost metric, which is either related to communication volume or message count. However, both volume- and message-related metrics should be taken into account during partitioning for a more efficient parallelization. There are only a few works that consider both of them and they usually address each in separate phases of a two-phase approach. In this work, we propose a recursive hypergraph bipartitioning framework that reduces the total volume and total message count in a single phase. In this framework, the standard hypergraph models, nets of which already capture the bandwidth cost, are augmented with *message nets*. The message nets encode the message count so that minimizing conventional cutsize captures the minimization of bandwidth and latency costs together. Our model provides a more accurate representation of the overall communication cost by incorporating both the bandwidth and the latency components into the partitioning objective. The use of the widely-adopted successful recursive bipartitioning framework provides the flexibility of using any existing hypergraph partitioner. The experiments on instances from different domains show that our model on the average achieves up to 52% reduction in total message count and hence results in 29% reduction in parallel running time compared to the model that considers only the total volume.

Index Terms—Communication cost, bandwidth, latency, partitioning, hypergraph, recursive bipartitioning, load balancing, sparse matrix vector multiplication, combinatorial scientific computing.

1 Introduction

For irregular applications in the scientific computing domain and several other domains, the intelligent partitioning methods are commonly employed to reduce the communication overhead for efficient parallelization in a distributed setting. Graph and hypergraph partitioning models are ubiquitously utilized in this regard.

1.1 Motivation and Related Work

A common cost model for representing the communication requirements of parallel applications consists of the bandwidth and latency components. The bandwidth component is proportional to the amount (volume) of data transferred and the latency component is proportional to the number of messages communicated. In order to capture the communication requirements of parallel applications more accurately, both components should be taken into account in the partitioning models.

Although graph/hypergraph partitioning models that address the bandwidth component are abundant in the literature [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], there exist only a few works that also address the latency component. A relatively early work by Uçar and Aykanat [12] proposes a two-phase approach in which the bandwidth and latency components are respectively addressed in the first and second phases by reducing total communication volume in the former and total message count in the latter. They propose the communication hypergraph model for the second phase to capture the messages and the processors involved. Their method is used for partitioning sparse matrices in the context of iterative solvers for nonsymmetric linear systems and exploits the flexibility of using nonconformal partitions for the vectors in the solver. A recent study by Deveci et al. [13] addresses multiple communication cost metrics via hypergraph partitioning in a single phase. These metrics involve the bandwidth-related metrics such as total volume, maximum send/receive volume, etc. as well as the latency-related metrics such as total message count and maximum send message count. All metrics are addressed in the refinement stage of the partitioning. Their approach introduces an additional cost of $O(VK^2)$ to each refinement pass for handling multiple metrics, where V and K denote the number of tasks in the application and the number of processors, respectively. Another work that is reported to reduce the latency cost in an indirect manner uses the $\lambda(\lambda-1)$ metric in order to correctly encapsulate the total communication volume in the target application [14].

There are studies that address the latency overhead via providing an upper bound on the number of messages communicated [8], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25]. These works usually assume that K processors are organized as a $\sqrt{K} \times \sqrt{K}$ mesh and restrict the communication along the rows and the columns of the processor

• *Oguz Selvitopi, Seher Acer and Cevdet Aykanat are with the Department of Computer Engineering, Bilkent University, Turkey, 06800. {reha, acer, aykanat}@cs.bilkent.edu.tr*

This work is partially supported by the Scientific and Technological Research Council of Turkey (TUBITAK) under project EEEAG-114E545. This project also supports COST Network by taking part in COST action CA15109, COSTNET.

mesh, which results in $O(\sqrt{K})$ messages for each processor. Most of the works bounding the latency component do not explicitly reduce the bandwidth component. The target applications in these works are usually centered around parallelizing sparse matrix computations.

There are a few studies that also aim at reducing volume besides bounding the message count. Çatalyürek and Aykanat [8] propose a two-phase method that makes use of hypergraph partitioning to achieve a Cartesian distribution of sparse matrices, namely 2D checkerboard partitioning. In the first phase, they obtain a rowwise $\sqrt{K}$-way partition and in the second phase, they use multiple vertex weights determined from the partition information of the first phase and obtain a columnwise $\sqrt{K}$-way partition. In both phases, the objective is to minimize the total volume. Boman et al. [23] achieves a similar feat with a faster method for scale-free graphs, again in two phases. In the first phase, their approach can make use of any available graph/hypergraph partitioner to obtain a 1D vertex partition. In the second phase, they use an effective algorithm to redistribute the nonzeros in the off-diagonal blocks to guarantee the $O(\sqrt{K})$ upper bound. These two methods are proposed for efficient parallelization of sparse matrix vector multiplication.

1.2 Contributions

Most of the existing graph/hypergraph partitioning models in the literature address only the bandwidth component while ignoring the latency component. In this work, we propose an augmentation to the existing models in order to minimize the bandwidth and the latency components simultaneously in a *single* phase. Our approach relies on the commonly adopted recursive bipartitioning (RB) framework [1], [5], [26], [27], [28]. The RB framework recursively partitions a given domain of computational tasks and data items into two until desired number subdomains is obtained. Consider a subdomain to be bipartitioned and the set of data items in this subdomain that are required by the tasks in some other subdomain. Keeping these items together in the bipartitioning ensures only one of the new subdomains to send a message to that other subdomain, avoiding an increase in the total number of messages. In order to encourage keeping these items together, we introduce *message nets* to the standard hypergraph model so that dividing these items is penalized with a cost equal to startup latency. The nets of the standard hypergraph model are referred to as the *volume nets* and with the addition of the message nets, this augmented hypergraph now contains both the volume and message nets. Partitioning this hypergraph presents a more accurate picture of the communication cost model as the objective of minimizing the cutsize in the partitioning encapsulates the reduction of both the total volume and the total message count.

Our approach is tailored for the parallel applications in which there exists a single communication phase, that is either preceded or succeeded by a computational phase. The parallel application is also assumed to be performed iteratively and a conformal partition on input and output data is required, where the input of the next iteration is obtained from the output of the current iteration. These common assumptions are suited well to the needs of several applications from various domains. Compared to the standard hypergraph partitioning model in which only the bandwidth component is minimized [1], our approach introduces an additional cost of $O(p \lg_2 K)$ due to the addition of the message nets, where p is the number of pins in the hypergraph. The proposed model does not depend on a specific hypergraph partitioning tool implementation, hence it can make use of any hypergraph partitioner such as PaToH [1], [29], hMetis [26] or Mondriaan [7]. In our experiments, we consider 1D parallel sparse matrix vector multiplication (SpMV) as an example application. Our approach is shown to be effective at reducing the latency component as it attains an 18%-52% reduction in the total number of messages at the expense of an 8%-70% increase in the total volume compared to the standard model. The experiments validate the necessity of addressing both communication components as the proposed model reduces the parallel running time of SpMV up to 29% for 2048 processors on the average.

The rest of the paper is organized as follows. Section 2 describes the properties of the target applications and how to model them with hypergraphs for parallelization. The proposed hypergraph partitioning model and its extensions are given in Section 3. Section 4 evaluates the proposed model in terms of both the communication statistics and the parallel running time of SpMV. Section 5 concludes.

2 Background

2.1 Hypergraph Partitioning Problem

A hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ is defined as a set of n vertices $\mathcal{V} = \{v_1, \ldots, v_n\}$ and a set of m nets $\mathcal{N} = \{n_1, \ldots, n_m\}$. Each net $n_j \in \mathcal{N}$ connects a subset of vertices, which is denoted by $Pins(n_j) \subseteq \mathcal{V}$. The set of nets that connect v_i is denoted by $Nets(v_i) \subseteq \mathcal{N}$. Each vertex v_i has an associated weight $w(v_i)$ and each net n_j has an associated cost $c(n_j)$.

$\Pi = \{\mathcal{V}_1, \ldots, \mathcal{V}_K\}$ is said to be a K-way vertex partition of $\mathcal{H}$ if parts are mutually disjoint and exhaustive. In Π, net n_j is said to connect part $\mathcal{V}_k$ if it connects at least one vertex in $\mathcal{V}_k$, i.e., $Pins(n_j) \cap \mathcal{V}_k \neq \emptyset$. The connectivity set $\Lambda(n_j)$ of n_j is defined as the set of parts connected by n_j. The connectivity of n_j, $\lambda(n_j)$, denotes the number of parts connected by n_j. n_j is said to be cut if it connects more than one part, i.e., $\lambda(n_j) > 1$, and uncut otherwise.

The hypergraph partitioning problem is defined as finding a K-way vertex partition Π with the objective of minimizing cutsize, which is defined as

$$cut(\Pi) = \sum_{n_j \in \mathcal{N}^{\text{cut}}} c(n_j)(\lambda(n_j) - 1), \tag{1}$$

subject to the balance constraint

$$W(\mathcal{V}_k) \leq (1 + \epsilon) W_{avg}, \tag{2}$$

where $\mathcal{N}^{\text{cut}}$ denotes the set of cut nets, $W(\mathcal{V}_k) = \sum_{v_i \in \mathcal{V}_k} w(v_i)$ denotes the weight of $\mathcal{V}_k$, $W_{avg} = \sum_k W(\mathcal{V}_k)/K$ denotes the average part weight and ϵ denotes the maximum allowed imbalance ratio. The hypergraph partitioning problem is NP-hard [30].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2016.2577024, IEEE Transactions on Parallel and Distributed Systems

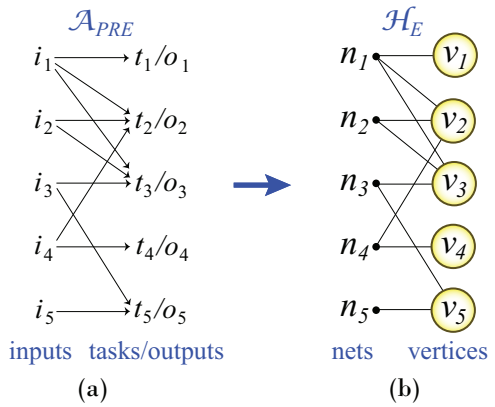

Fig. 1: (a) An example for A_{PRE}. (b) The hypergraph $\mathcal{H}_E$ that represents A_{PRE}.

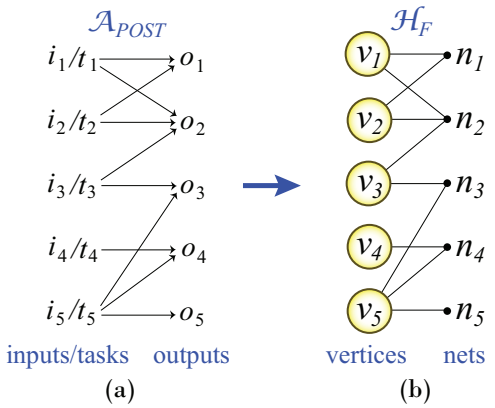

Fig. 2: (a) An example for A_{POST}. (b) The hypergraph $\mathcal{H}_F$ that represents A_{POST}.

2.2 Recursive Hypergraph Bipartitioning

Our work relies on recursive bipartitioning (RB), hence we give the relevant notation. In RB, a given hypergraph $\mathcal{H}$ is recursively partitioned into two parts until K parts are obtained. Obtaining a K-way partition of $\mathcal{H}$ through RB induces a binary tree with $\lceil \log_2 K \rceil$ levels, which is referred to as an RB tree. For the sake of simplicity, we assume K is a power of two. The ℓth level of the RB tree contains 2^ℓ hypergraphs, denoted with $\mathcal{H}_0^\ell, \ldots, \mathcal{H}_{2^\ell-1}^\ell$ from left to right, $0 \leq \ell \leq \log_2 K$. A bipartition $\Pi = \{\mathcal{V}_L, \mathcal{V}_R\}$ on hypergraph $\mathcal{H}_k^\ell$ in the ℓth level forms two new vertex-induced hypergraphs $\mathcal{H}_{2k}^{\ell+1} = (\mathcal{V}_L, \mathcal{N}_L)$ and $\mathcal{H}_{2k+1}^{\ell+1} = (\mathcal{V}_R, \mathcal{N}_R)$, both in level $\ell+1$. Here, $\mathcal{V}_L$ and $\mathcal{V}_R$ are respectively used to refer to the left and right part of the bipartition without loss of generality. A single bipartitioning is also referred to as an RB step.

The net sets of the newly formed hypergraphs in an RB step are constructed via cut-net splitting method [1] in order to capture the cutsize (1). In this method, a cut-net n_j in $\Pi = \{\mathcal{V}_L, \mathcal{V}_R\}$ is split into two new nets $n_j^L \in \mathcal{N}_L$ and $n_j^R \in \mathcal{N}_R$, where $Pins(n_j^L) = Pins(n_j) \cap \mathcal{V}_L$ and $Pins(n_j^R) = Pins(n_j) \cap \mathcal{V}_R$. Internal nets of $\mathcal{V}_L$ and $\mathcal{V}_R$ are respectively included in $\mathcal{N}_L$ and $\mathcal{N}_R$.

2.3 Parallelizing Applications

2.3.1 Target Application Properties

Consider an application $\mathcal{A} = (\mathcal{I}, \mathcal{T}, \mathcal{O})$ to be parallelized, where $\mathcal{I} = \{i_1, \ldots, i_I\}$ is the set of input data items, $\mathcal{T} = \{t_1, \ldots, t_T\}$ is the set of tasks and $\mathcal{O} = \{o_1, \ldots, o_O\}$ is the set of output data items. I, T and O respectively denote the sizes of $\mathcal{I}$, $\mathcal{T}$ and $\mathcal{O}$. The items and tasks of this application constitute a domain to be partitioned for parallelization. The tasks operate on input items and produce output items. There is no dependency among tasks, however there is interaction among the tasks that need the same input as well as the tasks that contribute to the same output. Input $i_j \in \mathcal{I}$ is required by a subset of tasks, denoted by $tasks(i_j) \subseteq \mathcal{T}$. A subset of tasks produce intermediate results for output $o_j \in \mathcal{O}$, again denoted by $tasks(o_j) \subseteq \mathcal{T}$. $size(t_i)$ denotes the amount of time required to complete task t_i and $size(i_j)$ ($size(o_j)$) denotes the storage size of item i_j (o_j). The tasks are atomic, i.e., each task is processed exactly by one processor. In a parallel setting, tasks and items are distributed among a number of processors.

We focus on applications in which either the intermediate results for o_j are produced by a single task for each $o_j \in \mathcal{O}$ ($|tasks(o_j)| = 1$) or i_j is required by a single task for each $i_j \in \mathcal{I}$ ($|tasks(i_j)| = 1$). In a distributed setting, there is only a single communication phase in both cases, in which either only the inputs or only the intermediate results of the outputs are communicated. The applications that exhibit the properties in the former and the latter cases are respectively denoted with A_{PRE} and A_{POST}. In A_{PRE}, the communications are performed in a so-called *pre-communication* phase, whereas in A_{POST}, the communications are performed in a so-called *post-communication* phase.

In A_{PRE}, the processor responsible for task t_j is also held responsible for storing output o_j. First, for each input i_k, the processor that stores i_k sends it to each processor which is responsible for at least one task in $tasks(i_k)$. This communication operation on i_k is referred to as an *expand* operation. Then, the processor responsible for t_j executes it by operating on inputs $\{i_k : t_j \in tasks(i_k)\}$ to compute the result for o_j in a communication-free manner. An example for A_{PRE} is illustrated in Fig. 1(a).

In A_{POST}, the processor responsible for task t_j is also held responsible for storing input i_j. First, the processor responsible for t_j executes it on i_j in a communication-free manner and produces intermediate results for the outputs $\{o_k : t_j \in tasks(o_k)\}$. Then, the processor responsible for o_k receives corresponding intermediate results from each processor which is responsible for at least one task in $tasks(o_k)$ and reduces them through an associative and/or commutative operator. This communication operation on o_k is referred to as a *fold* operation. An example for A_{POST} is illustrated in Fig. 2(a).

We assume that A_{PRE} and A_{POST} accommodate the following common properties: (i) they are performed repeatedly, (ii) the number of input and output items are equal, and (iii) there exists a one-to-one dependency between input and output items through successive iterations, i.e., output o_j of the current iteration is used to obtain input i_j of the next iteration. Note that if this one-to-one dependency is not respected in assigning items to processors, redundant communication is incurred. For this reason, a *conformal partition* on input and output items should be adopted in

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2016.2577024, IEEE Transactions on Parallel and Distributed Systems

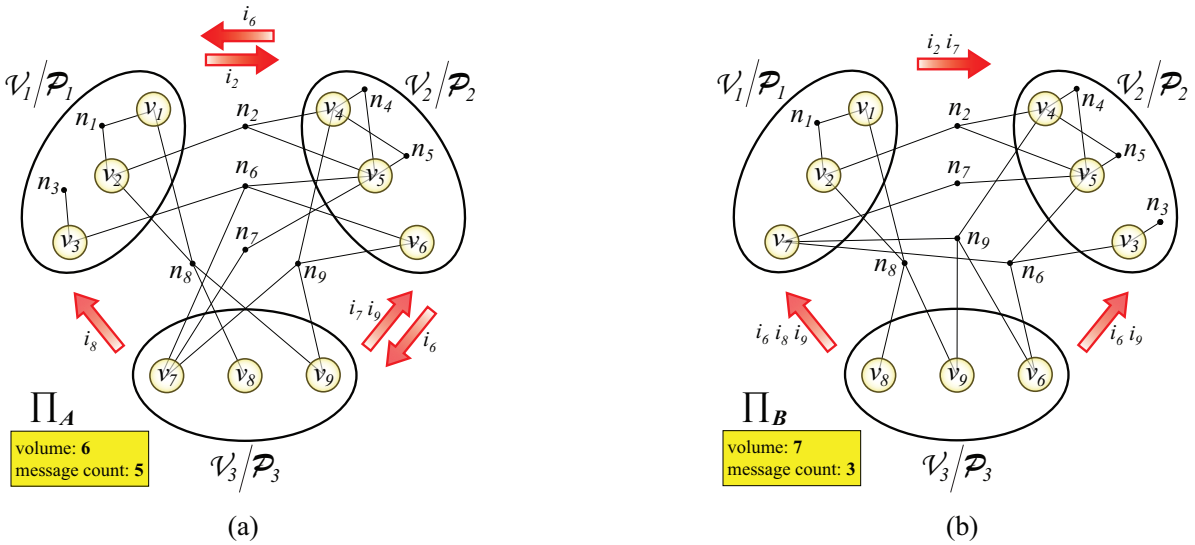

Fig. 3: Two 3-way partitionings of the same hypergraph A_{PRE}. Only the parts of v_3, v_6 and v_7 differ in Π_A and Π_B. Π_A incurs less volume but more messages while Π_B incurs more volume but less messages. Note that v_j represents both task t_j and input i_j.

which i_j and o_j are assigned to the same processor.

2.3.2 Hypergraph Models for A_{PRE} and A_{POST}

We use hypergraphs $\mathcal{H}_E = (\mathcal{V}_E, \mathcal{N}_E)$ and $\mathcal{H}_F = (\mathcal{V}_F, \mathcal{N}_F)$ to represent A_{PRE} and A_{POST}, respectively. Subscripts "E" and "F" are used to denote the fact that A_{PRE} and A_{POST} contain "Expand" and "Fold" operations, respectively. In both $\mathcal{H}_E$ and $\mathcal{H}_F$, the vertices represent tasks and items, i.e., $\mathcal{V}_E = \mathcal{V}_F = \{v_1, \ldots, v_T\}$, where v_i represents task t_i together with possibly multiple input-output pairs (i_j, o_j) such that $tasks(o_j) = \{t_i\}$ for A_{PRE} and $tasks(i_j) = \{t_i\}$ for A_{POST}. The weight of a vertex $w(v_i)$ in both $\mathcal{H}_E$ and $\mathcal{H}_F$ is assigned the amount of time required to execute t_i, i.e., $w(v_i) = size(t_i)$. Both net sets $\mathcal{N}_E$ and $\mathcal{N}_F$ consist of $I = O$ nets, $\mathcal{N}_E = \mathcal{N}_F = \{n_1, \ldots, n_{I=O}\}$. The nets in $\mathcal{N}_E$ capture the interactions between tasks and inputs: For each input i_j, there exists an expand net n_j to represent the expand operation on i_j with $Pins(n_j) = \{v_i : t_i \in tasks(i_j)\}$. The nets in $\mathcal{N}_F$ capture the interactions between tasks and outputs: For each output o_j, there exists a fold net n_j to represent the fold operation on o_j with $Pins(n_j) = \{v_i : t_i \in tasks(o_j)\}$. The cost of an expand net $n_j \in \mathcal{N}_E$ is assigned the size of the respective input i_j multiplied with t_w, i.e., $c(n_j) = size(i_j)\ t_w$. In a similar manner, the cost of a fold net $n_j \in \mathcal{N}_F$ is assigned the size of the respective output o_j multiplied with t_w, i.e., $c(n_j) = size(o_j)\ t_w$. Here, t_w is the time required to transfer a single unit of data item. Figures 1(b) and 2(b) display the hypergraphs $\mathcal{H}_E$ and $\mathcal{H}_F$ that respectively represent the example applications in Figures 1(a) and 2(a).

A K-way partition of $\mathcal{H}_E/\mathcal{H}_F$ is decoded to obtain a distribution of tasks and data items among K processors. The responsibility of executing tasks and storing items in the subdomain represented by part $\mathcal{V}_k$ is, without loss of generality, assigned to processor P_k. A cut-net n_j in the partition of $\mathcal{H}_E$ necessitates an expand operation on input i_j from the processor that stores i_j to $\lambda(n_j) - 1$ processors, whereas a cut-net n_j in the partition of $\mathcal{H}_F$ necessitates a fold operation on the intermediate results for output o_j from $\lambda(n_j) - 1$ processors to the processor that stores o_j. These operations respectively amount to $size(i_j)(\lambda(n_j) - 1)$ and $size(o_j)(\lambda(n_j) - 1)$ volume of communication units. The partitioning constraint of maintaining balance on part weights (2) in both $\mathcal{H}_E$ and $\mathcal{H}_F$ corresponds to maintaining balance on the processors' expected execution time. The partitioning objective of minimizing cutsize (1) in both $\mathcal{H}_E$ and $\mathcal{H}_F$ corresponds to minimizing the total communication volume incurred in pre-/post-communication phases.

2.3.3 Examples for A_{PRE} and A_{POST}

We consider parallel sparse matrix vector multiplication (SpMV) $y \leftarrow Ax$ performed in a repeated manner (such as in iterative solvers) which is a common kernel in scientific computing. Here, A is an $n \times n$ matrix, and x and y are vectors of size n. In SpMV, the inputs are x-vector elements, i.e., $\mathcal{I} = \{x_1, \ldots, x_n\}$, and the outputs are y-vector elements, i.e., $\mathcal{O} = \{y_1, \ldots, y_n\}$, where x_j and y_i respectively denote the jth x-vector element and ith y-vector element. A conformal partition on input and output vectors is usually preferred in order to avoid redundant communication.

1D row-parallel SpMV and 1D column-parallel SpMV are examples for A_{PRE} and A_{POST}. In 1D row-parallel SpMV, task t_j stands for the inner product $\langle a_{j*}, x \rangle$, while in 1D column-parallel SpMV, t_j stands for the scalar multiplication $x_j a_{*j}$, where a_{j*} and a_{*j} respectively denote the jth row and jth column of A, for $1 \leq j \leq n$. The size of t_j is equal to the number of nonzeros in jth row and jth column of A, respectively in A_{PRE} and A_{POST}, i.e., the number of multiply-and-add operations in $\langle a_{j*}, x \rangle$ and $x_j a_{*j}$. In both, there exist a total of n inputs, n tasks and n outputs. In 1D row-parallel SpMV, input x_j is required by each inner product $\langle a_{i*}, x \rangle$ such that a_{i*} contains a nonzero in jth column, that is, $tasks(x_j) = \{t_i : a_{ij} \neq 0\}$. The intermediate results for each output y_j are produced only by the task $\langle a_{j*}, x \rangle$, i.e., $tasks(y_j) = \{t_j\}$. In 1D column-parallel SpMV, input x_j is required only by the task

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2016.2577024, IEEE Transactions on Parallel and Distributed Systems

$x_j a_{*j}$, i.e., $tasks(x_j) = \{t_j\}$. Each task $x_i a_{*i}$ produces an intermediate result for output y_j such that a_{*i} contains a nonzero in its jth row, that is, $tasks(y_j) = \{t_i : a_{ji} \neq 0\}$. We represent 1D row-parallel and 1D column-parallel SpMVs respectively with $\mathcal{H}_E = (\mathcal{V}_E, \mathcal{N}_E)$ and $\mathcal{H}_F = (\mathcal{V}_F, \mathcal{N}_F)$. The cost of net n_j in both $\mathcal{H}_E$ and $\mathcal{H}_F$ is assigned t_w since it incurs the communication of a single item if it is cut. $\mathcal{H}_E$ and $\mathcal{H}_F$ are respectively called the column-net and row-net hypergraph models and they are proposed in [1].

The objective of partitioning $\mathcal{H}_E/\mathcal{H}_F$ correctly captures the total volume while disregarding the message count (Section 2.3.2). To illustrate the aspects of different partitions on these two metrics, we present a motivating example in Fig. 3, where the same $\mathcal{H}_E$ is 3-way partitioned in two different ways. The arrows represent the messages between processors that are associated with the parts in the figure. For example, in Fig. 3(a), i_2 $(= v_2)$ needs to be sent from $\mathcal{P}_1$ to $\mathcal{P}_2$ since it is stored by $\mathcal{P}_1$ and the tasks t_4 $(= v_4)$ and t_5 $(= v_5)$ in $\mathcal{P}_2$ need it (n_2 captures this dependency). The contents of the messages are indicated next to the arrows. In the first partition Π_A in Fig. 3(a), there are five messages and six communicated items, making up a total of $5t_s + 6t_w$ communication cost. In the second partition Π_B in Fig. 3(b), there are three messages and seven communicated items, making up a total of $3t_s + 7t_w$ communication cost. Partitioning $\mathcal{H}_E$ is more likely to produce Π_A since total volume is lower in Π_A as nets of $\mathcal{H}_E$ encode volume. However, Π_B is more desirable since $3t_s + 7t_w$ is less than $5t_s + 6t_w$ as t_s is usually much larger than t_w.

3 Simultaneous Reduction of Bandwidth and Latency Costs

We consider parallelizations of A_{PRE} and A_{POST} via K-way partitions on $\mathcal{H}_E$ and $\mathcal{H}_F$. We describe our model first for A_{PRE} in detail (Section 3.1) and then show how to apply it to A_{POST} (Section 3.2.1), as they are dual of each other. Hereinafter, we refer to $\mathcal{H}_E$ as $\mathcal{H}$ and Π_E as Π. We first assume that $I = O = T$ and describe the model for this case and then extend it to the more general case $I = O \geq T$ (Section 3.2.2).

3.1 Encoding Messages in Recursive Hypergraph Bipartitioning

Consider a recursive bipartitioning (RB) tree being produced in a breadth-first manner to obtain a K-way partition of $\mathcal{H} = (\mathcal{V}, \mathcal{N})$. Let the RB process be currently at the ℓth level, prior to bipartitioning hypergraph $\mathcal{H}_i^\ell$ in this level. There are currently $2^\ell + i$ hypergraphs, enumerated from left to right, $\mathcal{H}_i^\ell, \ldots, \mathcal{H}_{2^\ell-1}^\ell, \mathcal{H}_0^{\ell+1}, \ldots, \mathcal{H}_{2i-1}^{\ell+1}$, at the leaf nodes of the RB tree: $2^\ell - i$ of them at level ℓ and $2i$ of them at level $\ell+1$. The vertex sets of these hypergraphs induce a $(2^\ell + i)$-way vertex partition

$$\Pi_{\text{cur}}(\mathcal{H}) = \{\mathcal{V}_i^\ell, \ldots, \mathcal{V}_{2^\ell-1}^\ell, \mathcal{V}_0^{\ell+1}, \ldots, \mathcal{V}_{2i-1}^{\ell+1}\}.$$

This vertex partition is also assumed to induce a $(2^\ell + i)$-way processor partition $\mathbb{P}_{\text{cur}} = \{\mathcal{P}_i^\ell, \ldots, \mathcal{P}_{2^\ell-1}^\ell, \mathcal{P}_0^{\ell+1}, \ldots, \mathcal{P}_{2i-1}^{\ell+1}\}$, where processor group $\mathcal{P}_i^\ell$ is held responsible for the items/tasks that are in the subdomain represented by $\mathcal{V}_i^\ell$. An example Π_{cur} is seen in the upper RB tree in Fig. 4.

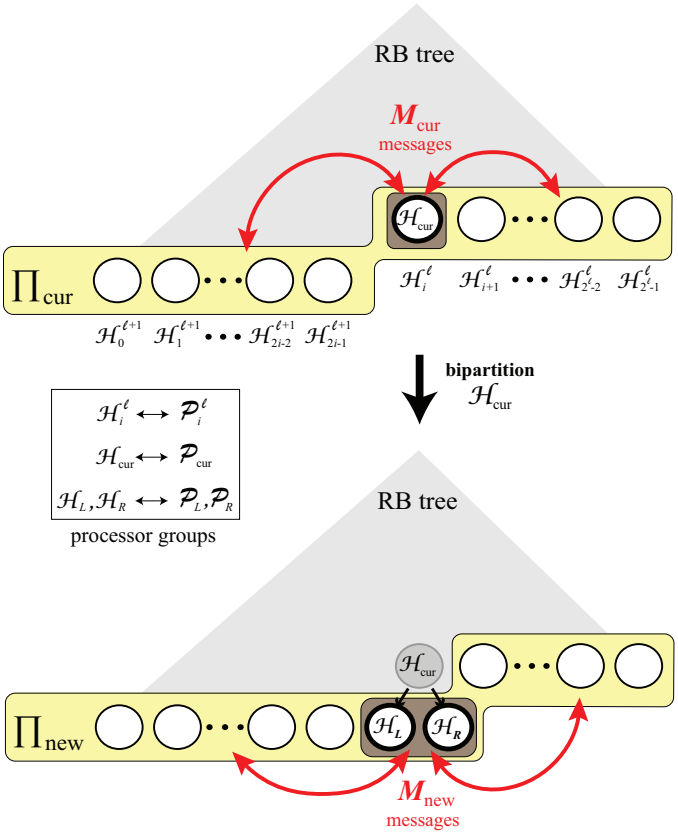

Fig. 4: The state of the RB tree and the number of messages from/to $\mathcal{P}_{\text{cur}}$ and $\{\mathcal{P}_L, \mathcal{P}_R\}$ to/from the other processor groups before and after bipartitioning $\mathcal{H}_{\text{cur}}$. The processor groups corresponding to the vertex sets of the hypergraphs are shown in the box.

We refer to the current hypergraph to be bipartitioned $\mathcal{H}_i^\ell$ as $\mathcal{H}_{\text{cur}} = (\mathcal{V}_{\text{cur}} = \mathcal{V}_i^\ell, \mathcal{N}_{\text{cur}} = \mathcal{N}_i^\ell)$. This bipartitioning generates $\Pi(\mathcal{H}_{\text{cur}}) = \{\mathcal{V}_L, \mathcal{V}_R\}$ and forms two new hypergraphs $\mathcal{H}_L = (\mathcal{V}_L, \mathcal{N}_L)$ and $\mathcal{H}_R = (\mathcal{V}_R, \mathcal{N}_R)$. Note that $\mathcal{H}_L = \mathcal{H}_{2i}^{\ell+1}$ and $\mathcal{H}_R = \mathcal{H}_{2i+1}^{\ell+1}$. After bipartitioning, there now exist $2^\ell + i + 1$ hypergraphs at the leaf nodes and their vertex sets induce a $(2^\ell + i + 1)$-way partition:

$$\Pi_{\text{new}}(\mathcal{H}) = \Pi_{\text{cur}}(\mathcal{H}) - \{\mathcal{V}_{\text{cur}}\} \cup \{\mathcal{V}_L, \mathcal{V}_R\}.$$

Bipartitioning $\mathcal{V}_{\text{cur}}$ into $\mathcal{V}_L$ and $\mathcal{V}_R$ is assumed to also bipartition the processor group $\mathcal{P}_{\text{cur}}$ into two processor groups $\mathcal{P}_L$ and $\mathcal{P}_R$. In accordance, $\mathbb{P}_{\text{new}} = \mathbb{P}_{\text{cur}} - \{\mathcal{P}_{\text{cur}}\} \cup \{\mathcal{P}_L, \mathcal{P}_R\}$.

Fig. 4 displays the two states of the RB tree before and after bipartitioning $\mathcal{H}_{\text{cur}}$ and highlights the messages communicated. Let M_{cur} be the number of messages between $\mathcal{P}_{\text{cur}}$ and $\mathbb{P}_{\text{cur}} - \{\mathcal{P}_{\text{cur}}\}$ and $M_{\text{new}} \geq M_{\text{cur}}$ be the number of messages between $\{\mathcal{P}_L, \mathcal{P}_R\}$ and $\mathbb{P}_{\text{new}} - \{\mathcal{P}_L, \mathcal{P}_R\}$. M_{new} can be at most $2M_{\text{cur}}$ which occurs when both $\mathcal{P}_L$ and $\mathcal{P}_R$ communicate with every $\mathcal{P}_k$ that $\mathcal{P}_{\text{cur}}$ communicates with. A new message is incurred when items/tasks that necessitate a message between $\mathcal{P}_{\text{cur}}$ and $\mathcal{P}_k$ get scattered across $\mathcal{P}_L$ and $\mathcal{P}_R$. Consequently, after bipartitioning, both $\mathcal{P}_L$ and $\mathcal{P}_R$ communicate with $\mathcal{P}_k$. Here, the idea is to find a way for items/tasks which as a whole necessitate a message between $\mathcal{P}_{\text{cur}}$ and $\mathcal{P}_k$ to be assigned together to either $\mathcal{P}_L$ or $\mathcal{P}_R$ so that only one of them communicates with $\mathcal{P}_k$. By doing so, the goal is to keep the number of messages

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2016.2577024, IEEE Transactions on Parallel and Distributed Systems

between $\{\mathcal{P}_L, \mathcal{P}_R\}$ and the remaining processor groups in $\mathbb{P}_{\text{new}}$ as small as possible.

To this end, we define new nets, referred to as *message nets*, to keep the vertices corresponding to items/tasks that necessitate messages altogether. We extend $\mathcal{H}_{\text{cur}} = (\mathcal{V}_{\text{cur}}, \mathcal{N}_{\text{cur}})$ to $\mathcal{H}^M_{\text{cur}} = (\mathcal{V}_{\text{cur}}, \mathcal{N}^M_{\text{cur}})$ by adding message nets and keeping the expand nets as is, referred to as *volume nets*. We include both volume and message nets in $\mathcal{H}^M_{\text{cur}}$ in order to reduce the total volume and the total message count simultaneously. The following sections define message nets and present an algorithm for forming them.

3.1.1 Message Nets

Recall that a vertex v_j in $\mathcal{V}$ represents input i_j besides task t_j and output o_j, and the processor that stores i_j is also held responsible for the possible expand operation on i_j. Since net n_j represents this expand operation, for convenience, we define a function $src : \mathcal{N} \rightarrow \mathcal{V}$, that maps each original net $n_j \in \mathcal{N}$ to its *source* vertex $src(n_j) \in \mathcal{V}$, where $src(n_j) = v_j$.

To aid the discussions in this section, we present an example RB tree in Fig. 5 that currently consists of four leaf hypergraphs $\mathcal{H}_{\text{cur}}$, $\mathcal{H}_a$, $\mathcal{H}_b$ and $\mathcal{H}_c$, whose vertex sets form 4-way partition Π_{cur}. We refer to the nets in given $\mathcal{H} = \mathcal{H}^0_0$ as *original nets* and use these nets in describing the algorithm for forming the message nets. An original net may split several times during RB or it may not split by being uncut in the bipartitionings it takes part in. For example in Fig. 5, the original net n_3 has split three times, producing n'_3, n''_3 and n'''_3 in $\mathcal{H}_{\text{cur}}$, $\mathcal{H}_a$ and $\mathcal{H}_b$, respectively. Observe that the vertices connected by n_3 in $\mathcal{H}$ are equal to the union of the vertices connected by n'_3, n''_3 and n'''_3 in the hypergraphs at the leaf nodes. On the other hand, the original net n_5 is never split and currently in $\mathcal{H}_{\text{cur}}$. In the figure, without loss of generality, a split net with a single prime in the superscript (e.g., n'_3) connects the source vertex of the respective original net (n_3), while split nets with two or more primes (e.g., n''_3, n'''_3) do not.

In the formation of the message nets, we make use of the most recent $(2^\ell + i)$-way partition information Π_{cur}. The message nets are categorized into two as send message nets and receive message nets, or simply *send nets* and *receive nets*.

We form a send net s_k for each $\mathcal{P}_k \neq \mathcal{P}_{\text{cur}}$ to which $\mathcal{P}_{\text{cur}}$ sends a message. s_k connects the vertices corresponding to the items sent to $\mathcal{P}_k$:

$$Pins(s_k) = \{v_j \in \mathcal{V}_{\text{cur}} : src(n_j) = v_j \text{ and } Pins(n_j) \cap \mathcal{V}_{k \neq \text{cur}} \neq \emptyset\}. \quad (3)$$

In other words, s_k connects source vertex v_j of each original net n_j that represents the expand operation which necessitates sending i_j to $\mathcal{P}_k$. Algorithm 1 shows the formation of the set of send nets $\mathcal{N}_{\text{SND}}$, which is initially empty (line 1). For each vertex $v_j \in \mathcal{V}_{\text{cur}}$, we first retrieve net n_j such that $src(n_j) = v_j$ (line 4). Then, the vertices which are connected by n_j and not in $\mathcal{V}_{\text{cur}}$ are traversed (line 5). Let v be such a vertex, currently in part $\mathcal{V}_k$. Since $\mathcal{P}_k$ needs i_j due to the task represented by v, i_j is sent from $\mathcal{P}_{\text{cur}}$ to $\mathcal{P}_k$. Hence, the vertices connected by send net s_k representing this message are updated (lines 7-12): If s_k is processed for the first time, $Pins(s_k)$ is initialized with $\{v_j\}$ and s_k is added to $\mathcal{N}_{\text{SND}}$ (lines 8 and 10), otherwise, $Pins(s_k)$ is updated to include v_j (line 12). Since $\mathcal{P}_{\text{cur}}$ can send at most one message to each of $2^\ell + i - 1$ processor groups, the number of send nets included in $\mathcal{H}^M_{\text{cur}}$ is at most $2^\ell + i - 1$, i.e., $0 \leq |\mathcal{N}_{\text{SND}}| \leq 2^\ell + i - 1$. In Fig. 5, the send nets s_a and s_b are formed and included in $\mathcal{H}^M_{\text{cur}}$ to represent the messages from $\mathcal{P}_{\text{cur}}$ to $\mathcal{P}_a$ and $\mathcal{P}_b$. s_a connects the respective source vertices v_1, v_2 and v_3 of the original nets n_1, n_2 and n_3 since $\mathcal{H}_a$ contains vertices that are also connected by these nets (indicated by the split nets n''_1, n''_2 and n''_3). Similarly, s_b connects v_3 and v_4 due to the vertices connected by n_3 and n_4 in $\mathcal{H}_b$.

Algorithm 1 FORM-MESSAGE-NETS

Require: $\mathcal{H} = (\mathcal{V}, \mathcal{N}), \mathcal{V}_{\text{cur}}, t_s$

1: $\mathcal{N}_{\text{SND}} = \emptyset$ ▷ The set of send nets
2: $\mathcal{N}_{\text{RCV}} = \emptyset$ ▷ The set of receive nets
3: **for** $v_j \in \mathcal{V}_{\text{cur}}$ **do**
4: Let $src(n_j) = v_j$
▷ *Add send nets*
5: **for** $v \in Pins(n_j)$ **and** $v \notin \mathcal{V}_{\text{cur}}$ **do**
6: Let $\mathcal{V}_k$ be the part v is currently in
7: **if** $s_k \notin \mathcal{N}_{\text{SND}}$ **then**
8: $Pins(s_k) = \{v_j\}$
9: $c(s_k) = t_s$
10: $\mathcal{N}_{\text{SND}} = \mathcal{N}_{\text{SND}} \cup \{s_k\}$
11: **else**
12: $Pins(s_k) = Pins(s_k) \cup \{v_j\}$
▷ *Add receive nets*
13: **for** $n \in Nets(v_j)$ **and** $n \neq n_j$ **and** $src(n) \notin \mathcal{V}_{\text{cur}}$ **do**
14: $v = src(n)$
15: Let $\mathcal{V}_k$ be the part v is currently in
16: **if** $r_k \notin \mathcal{N}_{\text{RCV}}$ **then**
17: $Pins(r_k) = \{v_j\}$
18: $c(r_k) = t_s$
19: $\mathcal{N}_{\text{RCV}} = \mathcal{N}_{\text{RCV}} \cup \{r_k\}$
20: **else**
21: $Pins(r_k) = Pins(r_k) \cup \{v_j\}$
22: **return** $\mathcal{N}_{\text{SND}}, \mathcal{N}_{\text{RCV}}$

We form a receive net r_k for each $\mathcal{P}_k \neq \mathcal{P}_{\text{cur}}$ from which $\mathcal{P}_{\text{cur}}$ receives a message. r_k connects the vertices corresponding to the tasks that need items from $\mathcal{P}_k$:

$$Pins(r_k) = \{v_j \in \mathcal{V}_{\text{cur}} : v_j \in Pins(n) \text{ and } src(n) \in \mathcal{V}_{k \neq \text{cur}}\}. \quad (4)$$

In other words, r_k connects vertex v_j which is connected by each original net n representing the expand operation that necessitates to receive the respective item from $\mathcal{P}_k$. Algorithm 1 shows the formation of the set of receive nets $\mathcal{N}_{\text{RCV}}$, which is initially empty (line 2). For each vertex $v_j \in \mathcal{V}_{\text{cur}}$, we traverse the nets that connect v_j other than n_j whose source vertices are not in $\mathcal{V}_{\text{cur}}$ (line 13). Let n be such a net and v be its source vertex, currently in $\mathcal{V}_k$ (lines 14-15). Since $\mathcal{P}_{\text{cur}}$ needs the item corresponding to v due to task t_j represented by v_j, this item is received by $\mathcal{P}_{\text{cur}}$ from $\mathcal{P}_k$. Hence, the vertices connected by receive net r_k representing this message are updated (lines 16-21): If r_k is processed for the first time, $Pins(r_k)$ is initialized with

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2016.2577024, IEEE Transactions on Parallel and Distributed Systems

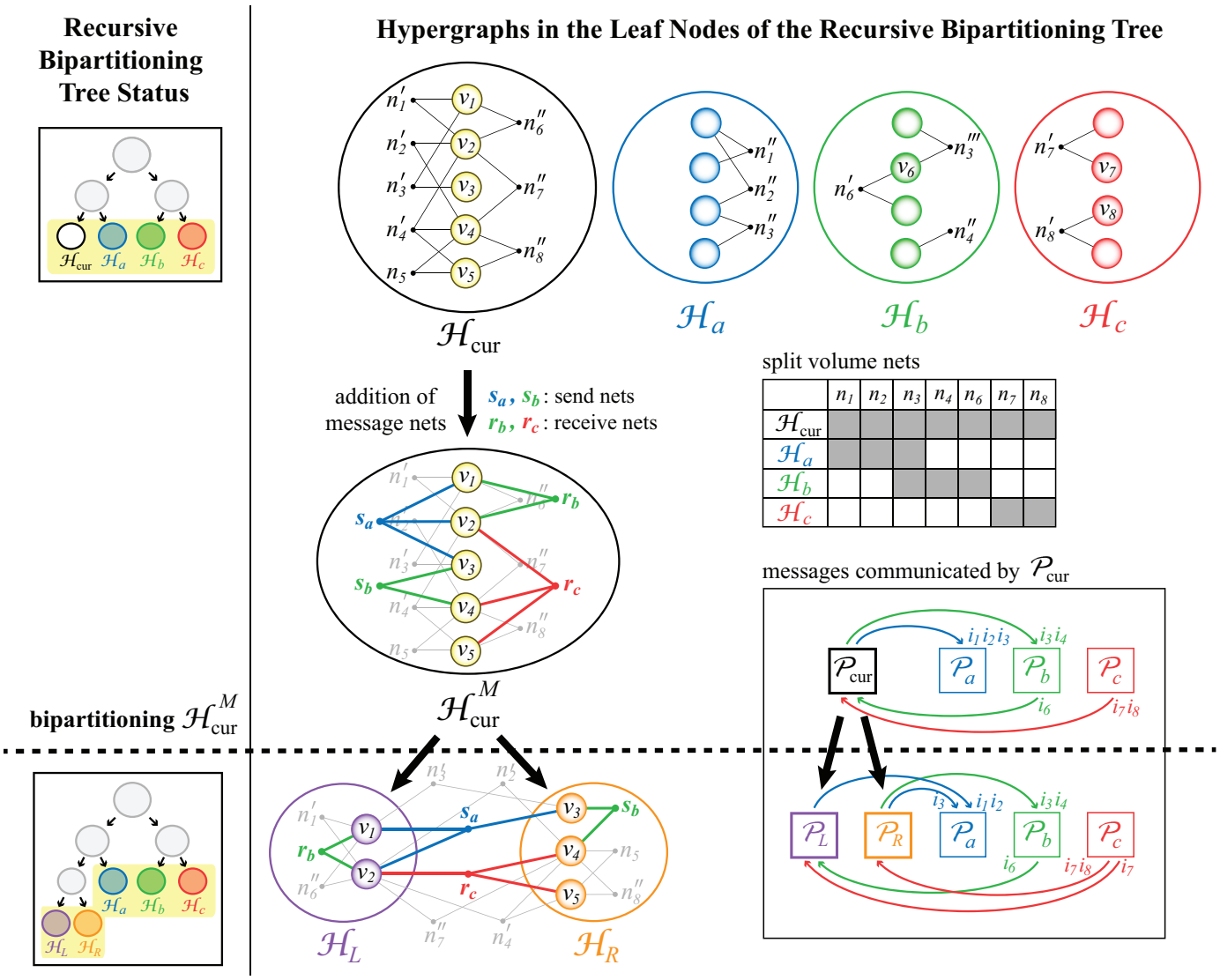

Fig. 5: The illustration of formation and addition of message nets to $\mathcal{H}_{\text{cur}}$ and bipartitioning of $\mathcal{H}^M_{\text{cur}}$. Initially, there are four hypergraphs in the leaf nodes of the RB tree: $\mathcal{H}_{\text{cur}}$, $\mathcal{H}_a$, $\mathcal{H}_b$ and $\mathcal{H}_c$. Two send (s_a and s_b) and two receive (r_b and r_c) message nets are added to form $\mathcal{H}^M_{\text{cur}}$. Then, $\mathcal{H}^M_{\text{cur}}$ is bipartitioned to obtain $\mathcal{H}_L$ and $\mathcal{H}_R$. The split nets are shown in a table where a shaded entry indicates the existence of the split net in the respective hypergraph. The messages communicated among the respective processor groups are illustrated in the frame at the right bottom corner, where $\mathcal{P}_{\text{cur}}$, $\mathcal{P}_a$, $\mathcal{P}_b$ and $\mathcal{P}_c$ are respectively associated with $\mathcal{H}_{\text{cur}}$, $\mathcal{H}_a$, $\mathcal{H}_b$ and $\mathcal{H}_c$. The colors of the message nets indicate the processor groups that the respective messages are sent to or received from. The dashed line separates the hypergraphs/processor groups before and after bipartitioning. The volume nets in $\mathcal{H}^M_{\text{cur}}$, $\mathcal{H}_L$ and $\mathcal{H}_R$ are faded out to attract the focus on the message nets.

$\{v_j\}$ and r_k is added to $\mathcal{N}_{\text{RCV}}$ (lines 17 and 19), otherwise, $Pins(r_k)$ is updated to include v_j (line 21). Since $\mathcal{P}_{\text{cur}}$ can receive at most one message from each of the $2^\ell + i - 1$ processor groups, the number of receive nets included in $\mathcal{H}^M_{\text{cur}}$ is at most $2^\ell + i - 1$, i.e., $0 \leq |\mathcal{N}_{\text{RCV}}| \leq 2^\ell + i - 1$. In Fig. 5, the receive nets r_b and r_c are formed and included in $\mathcal{H}^M_{\text{cur}}$ to represent the messages from $\mathcal{P}_b$ and $\mathcal{P}_c$ to $\mathcal{P}_{\text{cur}}$. r_b connects v_1 and v_2, which are connected by n_6 (indicated by the split net n''_6), since $\mathcal{H}_b$ contains the source vertex of n_6. Similarly, r_c connects v_2, v_4 and v_5 due to the nets n_7 and n_8, both of which have their source vertices in $\mathcal{H}_c$.

Recall that the cost of a volume net in $\mathcal{H}_{\text{cur}}$ is $size(i_j)\ t_w$ and captures the bandwidth cost. To capture the latency cost via message nets, the costs of these nets are assigned the startup latency:

$$c(s_k) = c(r_k) = t_s, \quad \text{for } s_k \in \mathcal{N}_{\text{SND}} \text{ and } r_k \in \mathcal{N}_{\text{RCV}}, \quad (5)$$

in lines 9 and 18 of Algorithm 1. Note that the cost of a volume net is the size of the corresponding item in terms of t_w, whereas the cost of a message net is unit in terms of t_s since it encapsulates exactly one message. The message nets have a higher cost than the volume nets since the startup time of a message is significantly higher than the time required to transmit a word. Finally, the message nets in $\mathcal{N}_{\text{SND}}$ and $\mathcal{N}_{\text{RCV}}$ are returned (line 22).

3.1.2 Partitioning and Correctness

The newly formed hypergraph $\mathcal{H}^M_{\text{cur}}$ is then bipartitioned to obtain $\Pi(\mathcal{H}^M_{\text{cur}}) = \{\mathcal{V}_L, \mathcal{V}_R\}$. Maintaining balance in

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2016.2577024, IEEE Transactions on Parallel and Distributed Systems

partitioning $\mathcal{H}^M_{\text{cur}}$ is the same with that of $\mathcal{H}_{\text{cur}}$ since vertices and their weights are the same in both hypergraphs. With the newly introduced message nets, the cutsize of Π is given by

$$\begin{aligned} cut(\Pi) &= \sum_{n_j \in \mathcal{N}^{\text{cut}}_{\text{cur}}} c(n_j) + \sum_{s_k \in \mathcal{N}^{\text{cut}}_{\text{SND}}} c(s_k) + \sum_{r_k \in \mathcal{N}^{\text{cut}}_{\text{RCV}}} c(r_k) \\ &= \sum_{n_j \in \mathcal{N}^{\text{cut}}_{\text{cur}}} size(i_j)\, t_w + |\mathcal{N}^{\text{cut}}_{\text{SND}}|\, t_s + |\mathcal{N}^{\text{cut}}_{\text{RCV}}|\, t_s \quad (6) \end{aligned}$$

where $\mathcal{N}^{\text{cut}}_{\text{cur}}$, $\mathcal{N}^{\text{cut}}_{\text{SND}}$ and $\mathcal{N}^{\text{cut}}_{\text{RCV}}$ respectively denote the sets of cut volume nets, cut send nets and cut receive nets in $\Pi(\mathcal{H}^M_{\text{cur}})$.

Let $msg(\mathbb{P})$ denote the total number of messages communicated among the processor groups in $\mathbb{P}$.

Theorem 1. *Consider an RB tree prior to bipartitioning the ith hypergraph $\mathcal{H}^M_{\text{cur}} = (\mathcal{V}_{\text{cur}}, \mathcal{N}^M_{\text{cur}})$ in the ℓth level with message nets added. The vertex sets of the leaf hypergraphs of the RB tree are assumed to induce a $(2^\ell + i)$-way processor partition $\mathbb{P}_{\text{cur}}$. Suppose that the bipartition $\Pi(\mathcal{H}^M_{\text{cur}}) = \{\mathcal{V}_L, \mathcal{V}_R\}$ generates two new leaf hypergraphs $\mathcal{H}_L$ and $\mathcal{H}_R$, where processor groups $\mathcal{P}_L$ and $\mathcal{P}_R$ are associated with $\mathcal{V}_L$ and $\mathcal{V}_R$. After bipartitioning, the vertex sets of the hypergraphs are assumed to induce a $(2^\ell + i + 1)$-way processor partition $\mathbb{P}_{\text{new}} = \mathbb{P}_{\text{cur}} - \{\mathcal{P}_{\text{cur}}\} \cup \{\mathcal{P}_L, \mathcal{P}_R\}$. Minimizing the number of cut message nets in $\Pi(\mathcal{H}^M_{\text{cur}})$ minimizes the increase ΔM in the number of messages between $\mathcal{P}_{\text{cur}}$ and $\mathbb{P}_{\text{cur}} - \{\mathcal{P}_{\text{cur}}\}$, which is given by*

$$\Delta M = msg(\mathbb{P}_{\text{new}}) - msg(\mathbb{P}_{\text{cur}}) - msg(\{\mathcal{P}_L, \mathcal{P}_R\}), \quad (7)$$

where $msg(\{\mathcal{P}_L, \mathcal{P}_R\}) \in \{0, 1, 2\}$.

Proof: A send net s_k in $\mathcal{H}^M_{\text{cur}}$ signifies a message from $\mathcal{P}_{\text{cur}}$ to $\mathcal{P}_k \in \mathbb{P}_{\text{cur}} - \{\mathcal{P}_{\text{cur}}\}$. If s_k is a cut-net in $\Pi(\mathcal{H}^M_{\text{cur}})$, then both $\mathcal{P}_L$ and $\mathcal{P}_R$ send a message to $\mathcal{P}_k$. The message from $\mathcal{P}_L$ to $\mathcal{P}_k$ and the message from P_R to P_k respectively contain the items corresponding to the vertices in $Pins(s_k) \cap \mathcal{V}_L$ and $Pins(s_k) \cap \mathcal{V}_R$. Hence, a cut send net contributes one to ΔM. If s_k is uncut being in either $\mathcal{V}_L$ or $\mathcal{V}_R$, then only the respective processor group sends a message to $\mathcal{P}_k$, whose content is exactly the same with the message from $\mathcal{P}_{\text{cur}}$ to $\mathcal{P}_k$. Hence, an uncut send net does not contribute to ΔM.

In a dual manner, a receive net r_k in $\mathcal{H}^M_{\text{cur}}$ signifies a message from $\mathcal{P}_k \in \mathbb{P}_{\text{cur}} - \{\mathcal{P}_{\text{cur}}\}$ to $\mathcal{P}_{\text{cur}}$. If r_k is a cut-net in $\Pi(\mathcal{H}^M_{\text{cur}})$, then both $\mathcal{P}_L$ and $\mathcal{P}_R$ receive a message from $\mathcal{P}_k$. The message from $\mathcal{P}_k$ to $\mathcal{P}_L$ and the message from $\mathcal{P}_k$ to $\mathcal{P}_R$ respectively contain the items required by the tasks corresponding to the vertices in $Pins(r_k) \cap \mathcal{V}_L$ and $Pins(r_k) \cap \mathcal{V}_R$. Hence, a cut receive net also contributes one to ΔM. If r_k is uncut being in either $\mathcal{V}_L$ or $\mathcal{V}_R$, then only the respective processor group receives a message from $\mathcal{P}_k$, whose content is exactly the same with the message from $\mathcal{P}_k$ to $\mathcal{P}_{\text{cur}}$. Hence, an uncut receive net does not contribute to ΔM.

The message nets are oblivious to the messages between $\mathcal{P}_L$ and $\mathcal{P}_R$ since our approach introduces these message nets for the processor groups in $\mathbb{P}_{\text{cur}}$. For this reason, $msg(\{\mathcal{P}_L, \mathcal{P}_R\})$ is not taken into account. Therefore, ΔM is equal to the number of cut message nets. □

By Theorem 1, the number of cut message nets in $\Pi(\mathcal{H}^M_{\text{cur}})$, $|\mathcal{N}^{\text{cut}}_{\text{SND}}| + |\mathcal{N}^{\text{cut}}_{\text{RCV}}|$, is equal to the increase in the message count, where the new messages between $\mathcal{P}_L$ and $\mathcal{P}_R$ are excluded. In other words, the number of cut message nets corresponds to the increase in the number of messages between $\mathcal{P}_{\text{cur}}$ and $\mathbb{P}_{\text{cur}} - \{\mathcal{P}_{\text{cur}}\}$ with $\mathcal{P}_{\text{cur}}$ bipartitioned into $\{\mathcal{P}_L, \mathcal{P}_R\}$ in $\mathbb{P}_{\text{new}}$. Observe that each cut message net contributes its associated cost $c(s_k)$ or $c(r_k)$ to the cutsize (6). For this reason as well as because of the presence of volume nets in (6), minimizing the cutsize does not exactly correspond to but relates to minimizing the number of cut message nets. Considering both the volume and the message nets, minimizing the cutsize corresponds to reducing both the total volume and the total message count.

Partitioning $\mathcal{H}^M_{\text{cur}}$ is oblivious to the messages between $\mathcal{P}_L$ and $\mathcal{P}_R$. However, $msg(\{\mathcal{P}_L, \mathcal{P}_R\})$ is negligible compared to $msg(\mathbb{P}_{\text{new}}) - msg(\mathbb{P}_{\text{cur}})$ since it is upper bounded by two. Moreover, $msg(\{\mathcal{P}_L, \mathcal{P}_R\})$ is empirically found to be almost constant as 2, being 0 or 1 in only 0.1% of 1M bipartitions. Note that $0 \leq \Delta M \leq 2(2^\ell + i - 1)$. The worst case for ΔM occurs when $\mathcal{H}^M_{\text{cur}}$ contains a send and a receive net for each other processor group in $\mathbb{P}_{\text{cur}}$ and they all become cut in $\Pi(\mathcal{H}^M_{\text{cur}})$.

In Fig. 5, there are two send nets s_a and s_b and two receive nets r_b and r_c in $\mathcal{H}^M_{\text{cur}}$. Among the send nets, s_a is a cut-net whereas s_b is an uncut net in $\mathcal{H}_R$ after bipartitioning. s_a necessitates both $\mathcal{P}_L$ and $\mathcal{P}_R$ to send a message to $\mathcal{P}_a$ due to the items $\{i_1, i_2\}$ and $\{i_3\}$, respectively. Since s_a is a cut-net, it increases the total message count by one as $\mathcal{P}_{\text{cur}}$ was sending a message to $\mathcal{P}_a$. s_b necessitates only $\mathcal{P}_R$ to send a message to $\mathcal{P}_b$ due to the items $\{i_3, i_4\}$. Since s_b is an uncut net, it does not change the total message count as $\mathcal{P}_{\text{cur}}$ was already sending a message to $\mathcal{P}_b$. Among the receive nets, r_c is a cut-net whereas r_b is an uncut net in $\mathcal{H}_L$ after bipartitioning. r_c necessitates both $\mathcal{P}_L$ and $\mathcal{P}_R$ to receive a message from $\mathcal{P}_c$ due to the tasks $\{t_2\}$ and $\{t_4, t_5\}$, respectively. Since r_c is a cut-net, it increases the total message count by one as $\mathcal{P}_{\text{cur}}$ was receiving a message from $\mathcal{P}_c$. r_b necessitates only $\mathcal{P}_L$ to receive a message from $\mathcal{P}_b$ due to the tasks $\{t_1, t_2\}$. Since r_b is an uncut net, it does not change the total message count as $\mathcal{P}_{\text{cur}}$ was already receiving a message from $\mathcal{P}_b$. Hence, two cut message nets cause an increase of two in the number of messages between $\mathcal{P}_{\text{cur}}$ and the other processor groups.

Algorithm 2 displays the overall RB process in which both the bandwidth and the latency costs are reduced. As inputs, the algorithm takes a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$, K (the number of parts to be obtained), and t_s as the cost of the message nets. The partitioning proceeds in a breadth-first manner (lines 2-3). Each hypergraph $\mathcal{H}_{\text{cur}}$ to be bipartitioned does not contain message nets initially (line 4). The sets of send and receive nets $\mathcal{N}_{\text{SND}}$ and $\mathcal{N}_{\text{RCV}}$ are formed via FORM-MESSAGE-NETS (Algorithm 1) (line 6). Then, these message nets are added to $\mathcal{N}_{\text{cur}}$ to obtain $\mathcal{N}^M_{\text{cur}}$ (line 7) and consequently $\mathcal{H}^M_{\text{cur}}$ (line 8). Note that if $\mathcal{H}_{\text{cur}}$ is the root hypergraph $\mathcal{H}^0_0$ of the RB tree, no message nets can be added since there is only a single processor group at this point (line 10). The current hypergraph is bipartitioned with the BIPARTITION function to obtain the vertex parts $\mathcal{V}_L$ and $\mathcal{V}_R$ (lines 9 and 11). The call to BIPARTITION can be realized with any hypergraph partitioning tool; it is a call to obtain only a two-way partition. New hypergraphs $\mathcal{H}_L = \mathcal{H}^{\ell+1}_{2i}$ and $\mathcal{H}_R = \mathcal{H}^{\ell+1}_{2i+1}$ are formed as described in Section 2.2 (lines

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2016.2577024, IEEE Transactions on Parallel and Distributed Systems

Algorithm 2 RB-BANDWIDTH-LATENCY

Require: $\mathcal{H} = (\mathcal{V}, \mathcal{N}), K, t_s$
1: $\mathcal{H}_0^0 = \mathcal{H}$
▷ *RB in breadth-first order*
2: **for** $\ell = 0$ **to** $\log_2 K - 1$ **do**
3: **for** $i = 0$ **to** $2^\ell - 1$ **do**
4: Let $\mathcal{H}_{\text{cur}} = (\mathcal{V}_{\text{cur}}, \mathcal{N}_{\text{cur}})$ denote $\mathcal{H}_i^\ell = (\mathcal{V}_i^\ell, \mathcal{N}_i^\ell)$
5: **if** $\ell > 0$ **then**
6: $\mathcal{N}_{\text{SND}}, \mathcal{N}_{\text{RCV}} =$ FORM-MESSAGE-NETS$(\mathcal{H}, \mathcal{V}_{\text{cur}}, t_s)$
7: $\mathcal{N}_{\text{cur}}^M = \mathcal{N}_{\text{cur}} \cup \mathcal{N}_{\text{SND}} \cup \mathcal{N}_{\text{RCV}}$
8: $\mathcal{H}_{\text{cur}}^M = (\mathcal{V}_{\text{cur}}, \mathcal{N}_{\text{cur}}^M)$
9: $\Pi =$ BIPARTITION$(\mathcal{H}_{\text{cur}}^M)$ ▷ $\Pi = \{\mathcal{V}_L, \mathcal{V}_R\}$
10: **else**
11: $\Pi =$ BIPARTITION$(\mathcal{H}_{\text{cur}})$ ▷ $\Pi = \{\mathcal{V}_L, \mathcal{V}_R\}$
▷ *Subhypergraphs $\mathcal{H}_L$ and $\mathcal{H}_R$ contain only volume nets*
12: Form $\mathcal{H}_L = \mathcal{H}_{2i}^{\ell+1} = (\mathcal{V}_L, \mathcal{N}_L)$ of $\mathcal{H}_{\text{cur}}$ induced by $\mathcal{V}_L$
13: Form $\mathcal{H}_R = \mathcal{H}_{2i+1}^{\ell+1} = (\mathcal{V}_R, \mathcal{N}_R)$ of $\mathcal{H}_{\text{cur}}$ induced by $\mathcal{V}_R$

12-13). $\mathcal{N}_L$ and $\mathcal{N}_R$ do not contain any message nets since these nets rely on the most recent partitioning information and thus need to be introduced from scratch just prior to bipartitioning $\mathcal{H}_{2i}^{\ell+1}$ and $\mathcal{H}_{2i+1}^{\ell+1}$. Notice that at any step of the RB, among all the leaf hypergraphs, only the current $\mathcal{H}_{\text{cur}}$ is subject to the addition of the message nets, whereas other hypergraphs remain intact.

3.1.3 Running Time Analysis

We consider the cost of adding message nets in the ℓth level of the RB tree produced in partitioning $\mathcal{H}$ into K parts, $0 < \ell < \log_2 K$. Recall that Algorithm 1 utilizes $Pins(\cdot)$ and $Nets(\cdot)$ functions on the *original hypergraph* $\mathcal{H} = (\mathcal{V}, \mathcal{N})$.

In the addition of the send nets, for each vertex v_j in the ℓth level, $Pins(n_j)$ is visited once, where $src(n_j) = v_j$ (line 5 in Algorithm 1). Observe that each net in $\mathcal{N}$ is visited only once since n_j is retrieved only for v_j. Updates related to a send net (lines 7-12) can be performed in $O(1)$ time. Hence, each pin of $\mathcal{H}$ is processed exactly once, making the cost of formation and addition of send nets $O(p)$ in the ℓth level, where p is the number of pins in $\mathcal{H}$.

In the addition of the receive nets, for each vertex v_j in the ℓth level, $Nets(v_j)$ is visited once (line 13 in Algorithm 1). Updates related to a receive net (lines 16-21) can also be performed in $O(1)$ time. Hence, each pin of $\mathcal{H}$ is again processed exactly once, making the cost of formation and addition of receive nets $O(p)$ in the ℓth level.

Therefore, the cost of adding message nets in a single level of RB is $O(p)$, which results in the overall cost of $O(p \log_2 K)$ for adding message nets. The solution of the partitioning problem with the addition of message nets is likely to be more expensive compared to partitioning of the original hypergraph.

3.2 Extensions

3.2.1 Encoding Messages for A_{POST}

We now describe how to apply the proposed model to A_{POST}. In $\mathcal{H}_F$, we define a function $dest : \mathcal{N} \rightarrow \mathcal{V}$ to determine the responsibility of the fold operation on each o_j, similar to the definition of src for expand operations in $\mathcal{H}_E$. In partitioning $\mathcal{H}_F$, a send net s_k connects the vertices corresponding to the tasks that produce intermediate results to be sent to $\mathcal{P}_k$:

$$Pins(s_k) = \{v_j \in \mathcal{V}_{\text{cur}} : v_j \in Pins(n) \text{ and } dest(n) \in \mathcal{V}_{k \neq \text{cur}}\}. \quad (8)$$

A receive net r_k connects the vertices corresponding to the output items for which the intermediate results need to be received from $\mathcal{P}_k$:

$$Pins(r_k) = \{v_j \in \mathcal{V}_{\text{cur}} : dest(n_j) = v_j \text{ and } Pins(n_j) \cap \mathcal{V}_{k \neq \text{cur}} \neq \emptyset\}. \quad (9)$$

Observe that the formation of a send net for $\mathcal{H}_F$ (8) is the same as that of a receive net for $\mathcal{H}_E$ (4) and the formation of a receive net for $\mathcal{H}_F$ (9) is the same as that of a send net for $\mathcal{H}_E$ (3). So, the message nets in $\mathcal{H}_E$ are the dual of the message nets in $\mathcal{H}_F$. Therefore, the correctness and complexity analysis carried out for A_{PRE} are also valid for A_{POST}.

3.2.2 Encoding Messages for $I = O \geq T$

To extend the proposed model to the case $I = O \geq T$ for A_{PRE}, we need a minor change in the formation of the message nets. In this case, a net n_j might be held responsible for multiple expand operations (see src definition). To reflect this change, line 4 of Algorithm 1 needs to be executed for each net n_j such that $src(n_j) = v_j$. The meaning of a message net does not change. The complexity of adding message nets is still $O(p \log_2 K)$ since each net n_j is again retrieved exactly once, uniquely by v_j. A similar discussion holds for A_{POST} by extending the definition of $dest$.

4 Experiments

4.1 Setup

For evaluation, we target the parallelization of an A_{PRE} application: 1D row-parallel SpMV. We model this application with hypergraph $\mathcal{H}_E$ as described in Section 2.3. We compare two schemes for the partitioning of $\mathcal{H}_E$:

- HP: The standard hypergraph partitioning model in which only the bandwidth cost is minimized (Section 2.3). In this scheme, $\mathcal{H}_E$ contains only the volume nets.
- HP-L: The proposed hypergraph partitioning model in which the bandwidth and the latency costs are reduced simultaneously (Section 3). In this scheme, $\mathcal{H}_E$ contains both the volume and the message nets.

Both HP and HP-L utilize recursive bipartitioning. We tested these schemes for $K \in \{128, 256, 512, 1024, 2048\}$ processors.

The two schemes are evaluated on 30 square matrices from the UFL Sparse Matrix Collection [31]. Table 1 displays the properties of these matrices. We consider square matrices since the proposed scheme aims at obtaining a conformal partition on the input and output items. The numbers of nonzeros in the test matrices range from 1.2M to 27.1M. This dataset contains small matrices (e.g., `bcsstk31`,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2016.2577024, IEEE Transactions on Parallel and Distributed Systems

TABLE 1: Properties of test matrices.

name	kind	#rows/cols	#nonzeros
d_pretok	2D/3D	182,730	1,641,672
turon_m	2D/3D	189,924	1,690,876
cop20k_A	2D/3D	121,192	2,624,331
torso3	2D/3D	259,156	4,429,042
mono_500Hz	acoustics	169,410	5,036,288
memchip	circuit sim.	2,707,524	14,810,202
Freescale1	circuit sim.	3,428,755	18,920,347
circuit5M_dc	circuit sim.	3,523,317	19,194,193
rajat31	circuit sim.	4,690,002	20,316,253
laminar_duct3D	comp. fluid dyn.	67,173	3,833,077
StocF-1465	comp. fluid dyn.	1,465,137	21,005,389
web-Google	directed graph	916,428	5,105,039
in-2004	directed graph	1,382,908	16,917,053
eu-2005	directed graph	862,664	19,235,140
cage14	directed graph	1,505,785	27,130,349
mac_econ_fwd500	economic	206,500	1,273,389
gsm_106857	electromagnetics	589,446	21,758,924
pre2	freq.-dom. circuit sim.	659,033	5,959,282
kkt_power	optimization	2,063,494	14,612,663
bcsstk31	structural	35,588	1,181,416
engine	structural	143,571	4,706,073
shipsec8	structural	114,919	6,653,399
Transport	structural	1,602,111	23,500,731
CO	theor./quant. chem.	221,119	7,666,057
598a	undirected graph	110,971	1,483,868
m14b	undirected graph	214,765	3,358,036
roadNet-CA	undirected graph	1,971,281	5,533,214
great-britain_osm	undirected graph	7,733,822	16,313,034
germany_osm	undirected graph	11,548,845	24,738,362
debr	undirected graph seq.	1,048,576	4,194,298

`mac_econ_fwd500`, etc.) for which the latency cost is expected to be more important.

The partitionings for both HP and HP-L are performed with the hypergraph partitioner PaToH [1]. Specifically, for each bipartitioning, we call `PaToH_Part` function of PaToH (lines 7 and 9 of Algorithm 2). The partitioning imbalance is set to 10%. Since PaToH contains randomized algorithms, we run each partitioning instance five times and report the average results of these runs.

We present the communication statistics for partitionings obtained via HP and HP-L and the corresponding parallel SpMV times. We used parallel SpMV of PETSc toolkit [32] on a Blue Gene/Q system. A node on this system consists of 16 PowerPC A2 processors with 1.6 GHz clock frequency and 16 GB memory. The nodes are connected by a 5D torus chip-to-chip network.

4.2 Message Net Costs

Recall that in our model, the volume nets are assigned the cost of t_w (transfer time of a single word) and the message nets are assigned the cost of t_s (startup time). Hereinafter, both the bandwidth and the latency costs are expressed in terms of t_w for the sake of presentation. Hence, the costs of volume nets are unit whereas the costs of message nets are t_s/t_w. The message net costs are denoted with *mnc* to simplify the notation. We conducted ping-pong experiments on BlueGene/Q with varying message sizes and the average t_s/t_w ratio was found to be around 200 for the matrices in our dataset.

Compared to HP, HP-L is expected to obtain a higher total volume since HP-L addresses two communication components simultaneously while HP solely optimizes a single component, which is determined by the total volume. We found out that the cost assignment of message nets in HP-L has a crucial impact on the parallel performance. The t_s/t_w ratio varies in practice for different message sizes and depends on the protocol used for transmitting messages as well as the characteristics of the target application which is likely to incur a higher t_w, hence a lower t_s/t_w, than that was found. For this reason, as well as to control the balance between the increase in the volume and the decrease in the message count compared to HP, we tried out different values for mnc in HP-L. The tested mnc values are 10, 50, 100 and 200. The reason for including smaller mnc values is that when the communication cost is dominated by the bandwidth component, utilizing a high mnc value has an adverse affect on the parallel performance compared to HP as the volume increase caused by HP-L is more apparent. Hence, small mnc values become more preferable in such cases.

TABLE 2: Communication statistics, PaToH partitioning times and parallel SpMV running times for HP-L normalized with respect to those for HP averaged over 30 matrices.

message net cost	K	volume		#messages		PaToH part. time	**parallel SpMV time**
		tot	max	tot	max		
10	128	1.08	1.11	0.82	0.87	1.07	**0.956**
	256	1.10	1.16	0.78	0.83	1.13	**0.904**
	512	1.12	1.22	0.75	0.83	1.13	**0.838**
	1024	1.16	1.29	0.73	0.84	1.25	**0.792**
	2048	1.20	1.37	0.71	0.88	1.28	**0.774**
50	128	1.17	1.25	0.65	0.76	1.08	**0.924**
	256	1.25	1.44	0.59	0.70	1.14	**0.846**
	512	1.33	1.57	0.56	0.69	1.21	**0.760**
	1024	1.41	1.69	0.57	0.74	1.24	**0.715**
	2048	1.48	1.85	0.59	0.80	1.33	**0.708**
100	128	1.24	1.43	0.59	0.73	1.09	**0.954**
	256	1.35	1.66	0.53	0.68	1.17	**0.858**
	512	1.45	1.86	0.51	0.68	1.19	**0.768**
	1024	1.54	1.92	0.53	0.71	1.31	**0.706**
	2048	1.61	2.06	0.57	0.80	1.41	**0.707**
200	128	1.33	1.60	0.54	0.72	1.15	**1.031**
	256	1.46	1.87	0.48	0.67	1.19	**0.872**
	512	1.57	2.02	0.49	0.67	1.25	**0.778**
	1024	1.65	2.09	0.52	0.72	1.37	**0.722**
	2048	1.70	2.17	0.57	0.79	1.48	**0.712**

4.3 Results

Table 2 presents the average communication statistics and the parallel SpMV running times of HP-L normalized with respect to those of HP for four mnc values and five K values. Each entry at a specific K value is the geometric mean of the normalized results obtained at that K value. The communication statistics are grouped under "volume" and "#messages". Under the "volume" grouping, the column "tot" denotes the total volume of communication and the column "max" denotes the maximum send volume of processors. Under the "#messages" grouping, the column "tot" denotes the total number of messages and the column "max" denotes the maximum number of messages that a processor sends. The PaToH partitioning times and parallel SpMV running times are respectively given under the columns "PaToH part. time" and "parallel SpMV time".

As seen in Table 2, HP-L achieves a significant reduction in the latency overhead. HP-L reduces the total number

of messages by 18%-29%, 35%-44%, 41%-49% and 43%-52% for mnc values of 10, 50, 100 and 200, respectively, compared to HP. This substantial improvement comes at the expense of increased volume as expected. Compared to HP, HP-L increases the total volume by 8%-20%, 17%-48%, 24%-61% and 33%-70% for mnc values of 10, 50, 100 and 200, respectively. In other words, HP-L achieves a factor of 1.22-1.41, 1.54-1.79, 1.69-1.96 and 1.75-2.08 reductions in the total number of messages while causing a factor of 1.08-1.20, 1.17-1.48, 1.24-1.61 and 1.33-1.70 increase in the total volume, over HP for mnc values of 10, 50, 100 and 200, respectively.

The proposed HP-L scheme achieves significantly lower parallel SpMV running times compared to the HP scheme. As seen in Table 2, HP-L achieves 4%-23%, 8%-29%, 5%-29% and −3%-29% lower running times for mnc values of 10, 50, 100 and 200, respectively, compared to HP. The lowest average running times for K values of 128, 256, 512, 1024 and 2048 are obtained with mnc values of 50, 50, 50, 100 and 100, respectively. Since using a low mnc (e.g., 10) does not attribute enough importance to the reduction of the latency cost, the parallel running times attained by this value are generally higher than those of other mnc values. Observe that for a specific K value, increasing the mnc leads to a decrease in the total message count and an increase in the total volume.

The performance improvement of HP-L over HP increases in terms of parallel SpMV time with increasing K for almost all mnc values. For example, for $mnc = 100$, HP-L only achieves a 5% improvement in running time over HP at $K = 128$, whereas at $K = 2048$ this improvement becomes 29%. The effect of reducing total message count becomes more apparent in parallel running time with increasing K since the latency component gets more important at high K values.

When we compare HP and HP-L in terms of partitioning times in Table 2, we see that HP-L has higher partitioning overhead as expected. HP-L incurs 7%-28%, 8%-33%, 9%-41% and 15%-48% slower partitionings for mnc values of 10, 50, 100 and 200, respectively. Although the formation of message nets is not expensive ($O(p \log_2 K)$), note that the partitioning with HP-L includes the message nets in addition to volume nets, which leads to increased bipartitioning times compared to HP. As K increases, HP-L's partitioning time also increases compared to HP since the number of message nets increases as well.

In Table 3, we present the detailed communication statistics and the parallel SpMV running times and speedups of 30 matrices for $K = 512$ and $mnc = 50$. In this table, the actual results of HP and HP-L are presented. The unit of the total volume is one kilo-item whereas the unit of the total number of messages is one kilo-message. The columns under "running time" denote the parallel SpMV running time in microseconds and the columns under "speedup" denote the speedups.

In 27 out of 30 matrices, HP-L obtains better speedup values than HP. As also observed and discussed for Table 2, reducing both the total volume and total message count significantly improves the parallel performance. For example, for `memchip`, HP-L increases the speedup by 71% (from 188 to 322) by reducing the total message count by 45% (from 3.9k to 2.2k). On the other hand, for `gsm_106857`, having a reduction of 18% in the total message count (from 3.8k to 3.1k) by HP-L does not lead to an improved parallel running time.

The reduction in the total number of messages leads to a reduction in the maximum number messages, as observed in Tables 2 and 3. In a similar manner, the increase in the total volume leads to an increase in the maximum volume. The models that provide an upper bound on the maximum message count usually have two communication phases, in each of which the maximum message count is $\sqrt{K} - 1$. Compared to these models, apart from the scale-free matrices, although our model does not provide such an upper bound, it usually obtains values below this bound, which is approximately $2(\sqrt{512} - 1) \approx 43$ for $K = 512$.

As seen in Table 3, a significant reduction in the total message count generally leads to a better performance. There are a couple of basic factors that can be argued to determine whether improving latency cost at the expense of bandwidth cost will result in a better parallel performance. For a partitioning instance, in general, if the average message size is relatively high and/or the maximum message count is relatively low, then it can be said that the bandwidth component dominates the latency component. For example, for `gsm_106857`, the average message size is high and the maximum message count is low compared to the other matrices. Hence, reducing the total message count by 18% does not pay off as the latency component is not worth exploiting.

Fig. 6 displays the speedup values of 16 matrices for parallel SpMV attained by two schemes for $K \in \{128, 256, 512, 1024, 2048\}$. We do not present any matrices of the directed graph kind in the figure since their efficiencies are very low. HP-L-10, HP-L-50, HP-L-100 and HP-L-200 respectively denote the HP-L scheme with mnc values of 10, 50, 100 and 200. For certain matrices, HP-L drastically changes the scalability by scaling up the parallel SpMV while HP scales down. This is observed for the matrices in the circuit simulation category, and the matrices `pre2` and `kkt_power`. For these matrices, reducing the latency overhead seems to be more important than reducing the bandwidth overhead. HP already exhibits good scalability for the matrices such as `m14b`, `great-britain_osm` and `Transport`. Reducing latency cost for these matrices pays off as HP-L further improves their scalability. HP and HP-L attain comparable scalability for `gsm_106857`, `StocF-1465` and `shipsec8`. The latency costs for these matrices are a minor component of their overall communication cost. Among the HP-L schemes, the scalability of HP-L-10 resembles that of HP the most since HP-L-10 attributes less importance to reducing the latency overhead compared to the other HP-L schemes. For `CO` and `mono_500Hz`, both HP and HP-L scale down after a certain number of processors. Nonetheless, HP-L still improves the parallel SpMV running time.

The HP-L schemes with the four different mnc values generally exhibit different parallel performance, as seen in Fig. 6. For any K value, increasing the mnc further decreases the message count and further increases the total volume (see Table 2). How this affects the parallel SpMV running time depends on the communication requirements

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2016.2577024, IEEE Transactions on Parallel and Distributed Systems

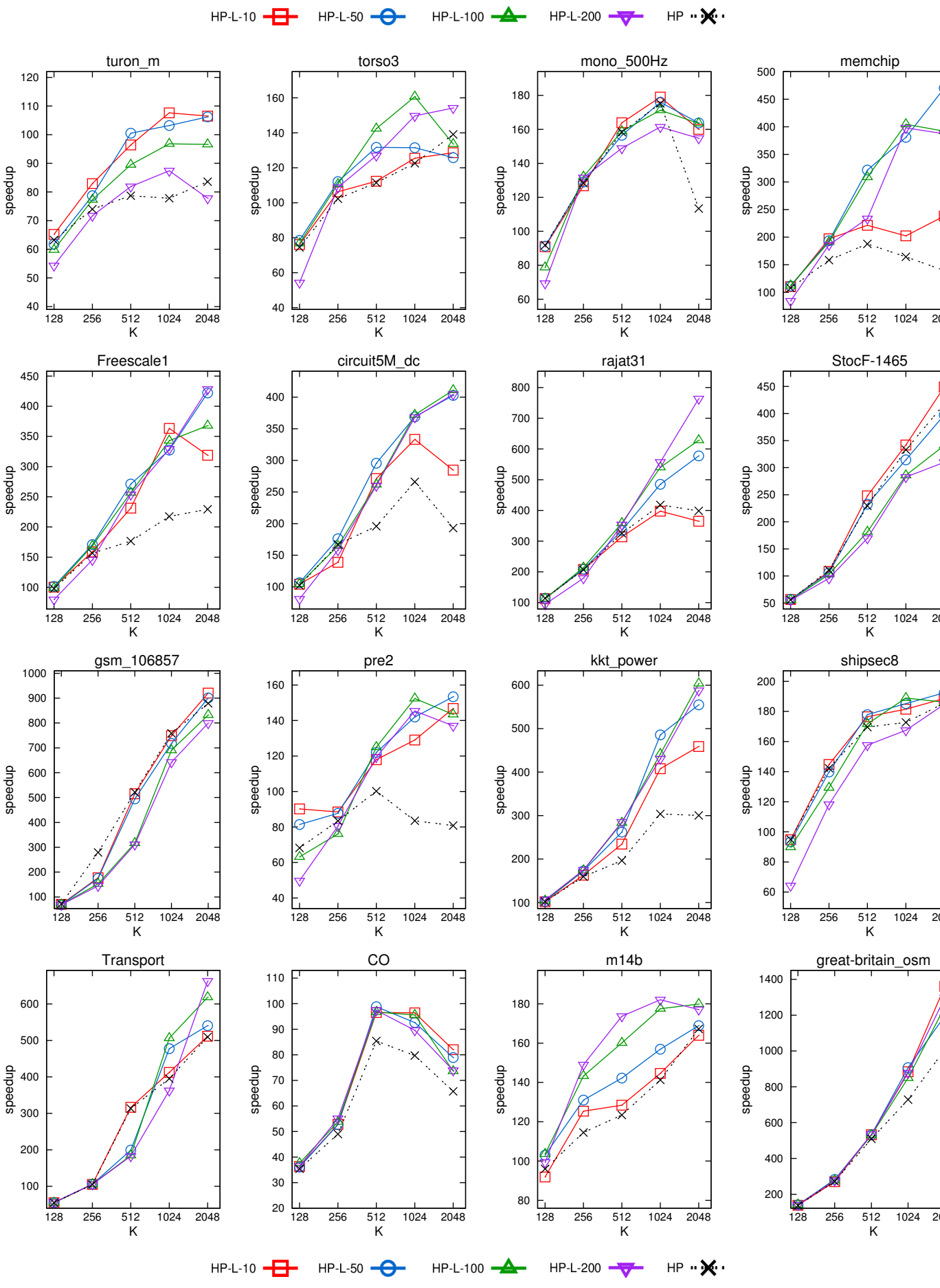

Fig. 6: Speedup values for 16 matrices.

TABLE 3: Communication statistics and parallel SpMV times/speedups for HP and HP-L for $K = 512$ and $mnc = 50$. Running time is in microseconds.

	tot vol		max vol		tot msg		max msg		running time		speedup	
name	HP	HP-L	HP	HP-L	HP	HP-L	HP	HP-L	HP	HP-L	HP	HP-L
d_pretok	108k	141k	302	451	5.3k	2.8k	19	11	312	232	66	**88**
turon_m	106k	127k	296	434	4.8k	2.5k	16	9	268	210	79	**100**
cop20k_A	142k	167k	411	482	5.2k	3.6k	20	15	542	466	82	**95**
torso3	197k	227k	550	825	4.9k	3.2k	21	15	411	348	112	**132**
mono_500Hz	226k	255k	703	812	6.0k	4.7k	26	26	492	498	**158**	157
memchip	102k	149k	679	705	3.9k	2.2k	51	16	1224	714	188	**322**
Freescale1	85k	151k	473	972	5.4k	2.4k	68	28	1697	1105	176	**271**
circuit5M_dc	84k	145k	386	939	5.5k	2.3k	64	24	1554	1030	196	**295**
rajat31	139k	184k	403	786	2.9k	2.0k	23	20	1010	983	325	**334**
laminar_duct3D	186k	211k	538	694	7.4k	5.9k	26	21	445	387	74	**85**
StocF-1465	581k	613k	1650	2109	6.8k	5.5k	31	29	1062	1048	229	**233**
web-Google	126k	266k	872	2971	37.2k	11.9k	281	231	17385	3492	12	**61**
in-2004	122k	169k	1933	2641	12.8k	6.4k	137	181	15690	4164	17	**64**
eu-2005	338k	408k	4207	6934	18.9k	9.9k	305	351	8375	4849	26	**44**
cage14	3184k	3291k	10528	10922	27.4k	19.7k	125	100	5757	4587	57	**71**
mac_econ_fwd500	124k	160k	335	490	5.6k	3.1k	16	11	252	231	80	**88**
gsm_106857	338k	371k	1161	1400	3.8k	3.1k	16	16	710	748	**521**	495
pre2	245k	261k	1143	1332	6.5k	3.8k	33	26	787	646	100	**122**
kkt_power	662k	684k	2930	3459	7.2k	4.2k	50	29	2072	1554	197	**262**
bcsstk31	62k	76k	223	297	4.3k	3.2k	19	17	283	253	45	**50**
engine	117k	144k	477	671	4.6k	3.1k	32	27	525	496	96	**102**
shipsec8	139k	159k	471	606	4.2k	3.2k	16	15	353	337	170	**178**
Transport	582k	645k	1487	1700	5.3k	3.2k	16	10	782	1230	**313**	199
CO	1044k	1070k	2710	2792	13.4k	10.1k	45	35	906	783	85	**99**
598a	104k	131k	341	529	5.6k	3.6k	29	24	435	382	103	**117**
m14b	158k	190k	596	804	5.3k	3.5k	38	31	630	547	123	**142**
roadNet-CA	35k	67k	138	496	3.1k	1.3k	15	8	848	753	178	**200**
great-britain_osm	26k	59k	168	640	4.1k	1.4k	29	11	1202	1151	509	**532**
germany_osm	33k	74k	242	772	4.2k	1.5k	31	13	1759	1719	603	**617**
debr	505k	899k	1268	2793	68.3k	16.9k	185	74	6047	1747	11	**37**

of the respective partitioning instance. It may pay off to use a high mnc value to further reduce the latency overhead (as is the case for `rajat31`) or doing so may worsen the parallel running time (as is the case for `StocF-1465`).

5 Conclusion

We have proposed a hypergraph partitioning model in order to parallelize certain types of applications with the objective of reducing the total volume and the total number of messages simultaneously. Our model exploits the recursive bipartitioning paradigm and hence provides the flexibility of using any available hypergraph partitioner. The proposed approach provides a better way for capturing the communication requirements of the target parallel applications. The experimental results showed that the better representation of the communication costs in the proposed hypergraph partitioning model led to a reduction of up to 29% in the parallel running time on the average.

As future work, we plan to develop heuristics to dynamically find the best message net cost for each bipartitioning by evaluating the relative importance of the components in the communication cost. We also wish to incorporate the other latency-based communication metrics such as the maximum number of messages communicated by a processor into our model.

Acknowledgments

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

References

[1] U. V. Çatalyürek and C. Aykanat, "Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 10, no. 7, pp. 673–693, Jul 1999.

[2] B. Hendrickson, "Graph partitioning and parallel solvers: Has the emperor no clothes? (extended abstract)," in *Proceedings of the 5th International Symposium on Solving Irregularly Structured Problems in Parallel*, ser. IRREGULAR '98. London, UK, UK: Springer-Verlag, 1998, pp. 218–225.

[3] B. Hendrickson and T. G. Kolda, "Graph partitioning models for parallel computing," *Parallel Comput.*, vol. 26, no. 12, pp. 1519–1534, Nov. 2000.

[4] ——, "Partitioning rectangular and structurally unsymmetric sparse matrices for parallel processing," *SIAM J. Sci. Comput.*, vol. 21, no. 6, pp. 2048–2072, Dec. 1999.

[5] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 359–392, Dec. 1998.

[6] K. Schloegel, G. Karypis, and V. Kumar, "Parallel multilevel algorithms for multi-constraint graph partitioning," in *Euro-Par 2000 Parallel Processing*, ser. Lecture Notes in Computer Science, A. Bode, T. Ludwig, W. Karl, and R. Wismller, Eds. Springer Berlin Heidelberg, 2000, vol. 1900, pp. 296–310.

[7] B. Vastenhouw and R. H. Bisseling, "A two-dimensional data distribution method for parallel sparse matrix-vector multiplication," *SIAM Rev.*, vol. 47, pp. 67–95, January 2005.

[8] U. Çatalyürek and C. Aykanat, "A hypergraph-partitioning approach for coarse-grain decomposition," in *Proceedings of the 2001 ACM/IEEE Conference on Supercomputing*, ser. SC '01. New York, NY, USA: ACM, 2001, pp. 28–28.

[9] B. Uçar and C. Aykanat, "Revisiting hypergraph models for sparse matrix partitioning," *SIAM Rev.*, vol. 49, pp. 595–603, November 2007.

[10] D. Pelt and R. Bisseling, "A medium-grain method for fast 2D bipartitioning of sparse matrices," in *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, May 2014, pp. 529–539.

[11] E. Kayaaslan, B. Ucar, and C. Aykanat, "Semi-two-dimensional partitioning for parallel sparse matrix-vector multiplication," in *Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International*, May 2015, pp. 1125–1134.

[12] B. Uçar and C. Aykanat, "Encapsulating multiple communication-cost metrics in partitioning sparse rectangular matrices for parallel matrix-vector multiplies," *SIAM J. Sci. Comput.*, vol. 25, no. 6, pp. 1837–1859, 2004.

[13] M. Deveci, K. Kaya, B. Uçar, and Ümit Çatalyürek, "Hypergraph partitioning for multiple communication cost metrics: Model and methods," *Journal of Parallel and Distributed Computing*, vol. 77, no. 0, pp. 69 – 83, 2015.

[14] O. Fortmeier, H. Bcker, B. F. Auer, and R. Bisseling, "A new metric enabling an exact hypergraph model for the communication volume in distributed-memory parallel applications," *Parallel Computing*, vol. 39, no. 8, pp. 319 – 335, 2013.

[15] B. Hendrickson, R. Leland, and S. Plimpton, "An efficient parallel algorithm for matrix-vector multiplication," *International Journal of High Speed Computing*, vol. 7, pp. 73–88, 1995.

[16] J. Lewis, D. Payne, and R. van de Geijn, "Matrix-vector multiplication and conjugate gradient algorithms on distributed memory computers," in *Scalable High-Performance Computing Conference, 1994., Proceedings of the*, May 1994, pp. 542–550.

[17] J. G. Lewis and R. A. van de Geijn, "Distributed memory matrix-vector multiplication and conjugate gradient algorithms," in *Proceedings of the 1993 ACM/IEEE conference on Supercomputing*, ser. Supercomputing '93. New York, NY, USA: ACM, 1993, pp. 484–492.

[18] A. Ogielski and W. Aiello, "Sparse matrix computations on parallel processor arrays," *SIAM Journal on Scientific Computing*, vol. 14, no. 3, pp. 519–530, 1993.

[19] A. Buluç and J. Gilbert, "Challenges and advances in parallel sparse matrix-matrix multiplication," in *Parallel Processing, 2008. ICPP '08. 37th International Conference on*, Sept 2008, pp. 503–510.

[20] A. Buluç and K. Madduri, "Parallel breadth-first search on distributed memory systems," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '11. New York, NY, USA: ACM, 2011, pp. 65:1–65:12.

[21] A. Yoo, A. H. Baker, R. Pearce, and 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*, ser. SC '11. New York, NY, USA: ACM, 2011, pp. 63:1–63:11.

[22] U. V. Çatalyürek, C. Aykanat, and B. Uçar, "On two-dimensional sparse matrix partitioning: Models, methods, and a recipe," *SIAM J. Sci. Comput.*, vol. 32, no. 2, pp. 656–683, Feb. 2010.

[23] E. G. Boman, K. D. Devine, and 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*, ser. SC '13. New York, NY, USA: ACM, 2013, pp. 50:1–50:12.

[24] R. O. Selvitopi, M. Ozdal, and C. Aykanat, "A novel method for scaling iterative solvers: Avoiding latency overhead of parallel sparse-matrix vector multiplies," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 26, no. 3, pp. 632–645, March 2015.

[25] O. Selvitopi and C. Aykanat, "Reducing latency cost in 2D sparse matrix partitioning models," *Parallel Computing*, pp. –, 2016.

[26] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: applications in VLSI domain," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 7, pp. 69–79, March 1999.

[27] F. Pellegrini and J. Roman, "Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs," in *High-Performance Computing and Networking*, ser. Lecture Notes in Computer Science, H. Liddell, A. Colbrook, B. Hertzberger, and P. Sloot, Eds. Springer Berlin Heidelberg, 1996, vol. 1067, pp. 493–498.

[28] M. Wang, S. Lim, J. Cong, and M. Sarrafzadeh, "Multi-way partitioning using bi-partition heuristics," in *Proceedings of the 2000 Asia and South Pacific Design Automation Conference*, ser. ASP-DAC '00. New York, NY, USA: ACM, 2000, pp. 667–.

[29] D. A. Padua, Ed., *Encyclopedia of Parallel Computing*. Springer, 2011.

[30] T. Lengauer, *Combinatorial algorithms for integrated circuit layout*. New York, NY, USA: John Wiley & Sons, Inc., 1990.

[31] T. A. Davis and Y. Hu, "The University of Florida sparse matrix collection," *ACM Trans. Math. Softw.*, vol. 38, no. 1, pp. 1:1–1:25, Dec. 2011.

[32] 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, and H. Zhang, "PETSc users manual," Argonne National Laboratory, Tech. Rep. ANL-95/11 - Revision 3.5, 2014.

Oguz Selvitopi received his B.S. degree in Computer Engineering from Marmara University (2008) and M.S. degree (2010) in Computer Engineering from Bilkent University, Turkey where he is currently a Ph.D. candidate. His research interests are parallel computing, scientific computing, and parallel and distributed systems.

Seher Acer received her B.S. (2009) and M.S. (2011) degrees in Computer Engineering from Bilkent University, Turkey, where she is currently a Ph.D. candidate. Her research interests are combinatorial scientific computing, graph and hypergraph partitioning, and parallel computing.

Cevdet Aykanat received the BS and MS degrees from Middle East Technical University, Ankara, Turkey, both in Electrical Engineering, and the Ph.D. degree from Ohio State University, Columbus, in Electrical and Computer Engineering. He worked at the Intel Supercomputer Systems Division, Beaverton, Oregon, as a research associate. Since 1989, he has been affiliated with the Department of Computer Engineering, Bilkent University, Ankara, Turkey, where he is currently a professor. His research interests mainly include parallel computing, parallel scientific computing and its combinatorial aspects. He has (co)authored about 90 technical papers published in academic journals indexed in ISI and his publications received about 1000 citations in ISI indexes. He is the recipient of the 1995 Young Investigator Award of The Scientific and Technological Research Council of Turkey and 2007 Parlar Science Award. He has served as a member of IFIP Working Group 10.3 (Concurrent System Technology) since 2004 and as an Associate Editor of IEEE Transactions of Parallel and Distributed Systems between 2008 and 2012.