Title

Simultaneous Computational and Data Load Balancing in Distributed-Memory Setting

Permalink

Journal

SIAM Journal on Scientific Computing, 44(6)

ISSN

1064-8275

Authors

Çeliktuğ, Mestan Firat
Karsavuran, M Ozan
Acer, Seher
et al.

Publication Date

2022-12-01

DOI

10.1137/22m1485772

Copyright Information

SIMULTANEOUS COMPUTATIONAL AND DATA LOAD BALANCING IN DISTRIBUTED-MEMORY SETTING

MESTAN FIRAT ÇELIKTUĞ*, M. OZAN KARSAVURAN†, SEHER ACER‡, AND CEVDET AYKANAT†

Abstract. Several successful partitioning models and methods have been proposed and used for computational load balancing of irregularly sparse applications in distributed-memory setting. However, the literature lacks partitioning models and methods that encode both computational and data load balancing. In this article, we try to close this gap in the literature by proposing two hypergraph partitioning (HP) models which simultaneously encode computational and data load balancing. Both models utilize a two-constraint formulation where the first constraint encodes the computational load and the second constraint encodes the data load. In the first model, we introduce explicit data vertices for encoding data load and we replicate those data vertices at each recursive bipartitioning (RB) step for encoding data replication. In the second model, we introduce a data weight distribution scheme for encoding data load and we update those weights at each RB step. The nice property of both proposed models is that they do not necessitate developing a new partitioner from scratch. Both models can easily be implemented by invoking any HP tool that supports multi-constraint partitioning as a two-way partitioner at each RB step. The validity of the proposed models is tested on two widely-used irregularly sparse applications: parallel mesh simulations and parallel sparse matrix sparse matrix multiplication (SpGEMM). Both proposed models achieve significant improvement over a baseline model.

Key words. computational load balance, data load balance, distributed-memory systems, hypergraph partitioning, recursive bipartitioning, multi-constraint partitioning, general sparse matrix-matrix multiplication, mesh partitioning

AMS subject classifications. 05C85,05C65,65F50,68R10

1. Introduction.

In a distributed-memory setting, task-to-processor assignment has a significant role to attain high performance. This assignment affects multiple performance metrics such as computational load balance and communication costs which include bandwidth and latency components. Over the years, these metrics are widely studied alone or in a combination [1–3, 9–11, 14, 17, 20, 24, 36, 38, 39, 41–43]. There also exist combinatorial models and works which target minimizing these metrics under a given partial or complete data partition [6,13,18,19,21,33]. In the high performance computing community, whenever load balance is pronounced it is almost always computational load balance [11,12,14,18,25,25–27,37]. In the cloud computing community, data load balance is considered, however, in that context data load balance is usually the only objective of the partitioning [30,32,35]. That is, only data is partitioned across data centers without associated tasks. In the literature, data load is considered for data migration cost [12,19,25] and memory capacity [5,22,40]. To our knowledge, this is the first paper in which simultaneous balance on computational and data loads are considered.

The target problem consists of atomic tasks and data elements to be assigned to the processors. Tasks do not have any computational dependency, whereas a data element might be needed by multiple tasks. If such tasks are assigned to different processors, then that data element will be replicated to those processors. Tasks are

*Department of Computer Science, University of Texas at Dallas, and Department of Computer Engineering, Gazi University, Ankara, Turkey (mestanfirat.celiktug@utdallas.edu, firat.celiktug@gazi.edu.tr).

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

‡National Center for Computational Sciences, Oak Ridge National Laboratory (acers@ornl.gov).

associated with computational weights, whereas data elements are associated with sizes. Then the problem is finding the task-to-processor assignment in which maximum computational and data loads of processors are minimized simultaneously.

In this work, we propose two novel hypergraph partitioning (HP) based models which simultaneously consider computational and data loads of the processors in a distributed-memory setting. In the first model, there exist vertices representing computational tasks as well as data elements. There exist nets also representing data elements and their relations with tasks. A two-constraint formulation is utilized for simultaneous computational and data load balancing, where partitioning objective encodes minimization of the total data replication. We utilize the recursive bipartitioning (RB) scheme to increase the performance of the proposed vertex replication scheme by applying it at each level. For the RB framework, we propose a novel data vertex replication scheme to encode better data load balancing in the further RB steps.

In the second model, vertices represent computational tasks, whereas nets represent data elements as well as their relations with tasks. In this model, a similar two-constraint formulation is also utilized. In order to model data loads, in contrast to the first model which contains explicit data vertices, we propose a data weight distribution to the vertex weights. Furthermore, we also utilize the RB framework for enabling utilization of the proposed weight distribution at each level.

In two-constraint formulation utilized in both models, finding balance on the computational loads through part weights encapsulates minimizing the computational load of the maximally loaded processor, since the total computational load is fixed. This is not true for the data load, because the total data load depends on the computational task partitioning. On the other hand, both proposed models minimize the amount of the data replication with the partitioning objective. In that way, proposed two-constraint formulation also encapsulates minimizing the data load of the maximally loaded processor.

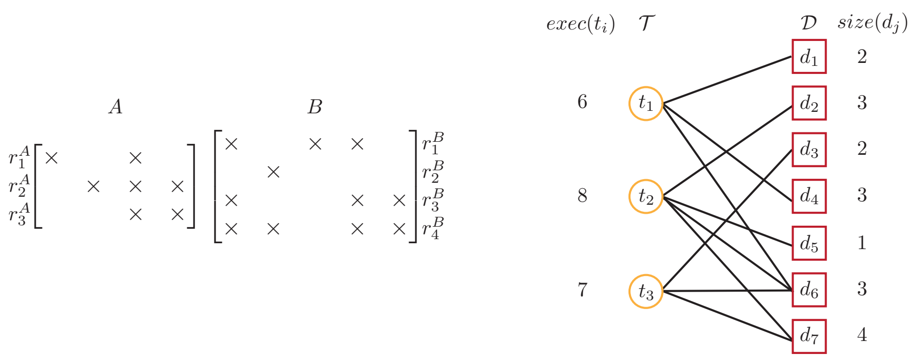
We evaluate the performance of the proposed models against a HP based baseline model which only tries to balance computational loads of the processors. We utilized two sample applications in which our target problem arises: Parallel Finite Element Method and Volume Element Method based simulations which involve partitioning irregular 2D or 3D meshes and row-row-parallel Sparse Generalized Matrix Matrix Multiplication which involves partitioning two irregularly sparse input matrices. Extensive experiments conducted for a wide range of partitioning instances on up to 1024 processors show that both proposed models achieve significantly better performance than the baseline model.

The rest of the paper is organized as follows: Section 2 describes the framework and formally defines the target problem. We give preliminary information about HP and RB framework in section 3. We propose two HP models for the target problem in section 4. In section 5 experimental results are presented and discussed. We briefly discuss related works in section 6. Finally, section 7 concludes the paper.

2. Framework and problem definition. The target application is considered as a two-tuple $\mathcal{A}=(\mathcal{T},\mathcal{D})$. Here, $\mathcal{T}=\{t_1,t_2,\cdots,t_T\}$ denotes a set of $|\mathcal{T}|=T$ independent computational tasks and $\mathcal{D}=\{d_1,d_2,\cdots,d_D\}$ denotes a set of $|\mathcal{D}|=D$ data elements. Here and hereafter $|\cdot|$ denotes the cardinality of the respective set. There exists no computational dependency between tasks. However, there exists interaction among the tasks as multiple tasks may need the same data element(s) for their executions. In a dual manner, an individual data element may be required by multiple tasks for execution. The set of data elements required by a task t_i is denoted by $Data(t_i)$, whereas the set of tasks that need/require a data element d_j is denoted by $Tasks(d_j)$. Tasks may be associated with different computational costs, as well as data elements may be associated with different memory sizes. Let $exec(t_i)$ denote the computational cost of task t_i and let $size(d_j)$ denote the memory size of data element d_j.

Row-row-parallel Sparse Generalized Matrix Matrix Multiplication (SpGEMM) of the form $C = AB$ is an example target application. That is, the pre-multiplication of individual A-matrix rows with the B-matrix constitute the tasks, whereas rows of the A and B matrices constitute the data elements. The details about the SpGEMM application is given in subsection 5.3.2. Figure 1a shows a sample SpGEMM instance with a 3×4 A-matrix and a 4×5 B-matrix.

Figure 1b shows the $\mathcal{A} = (\mathcal{T}, \mathcal{D})$ representation of the SpGEMM instance given in Figure 1a with $|\mathcal{T}| = 3$ tasks and $|\mathcal{D}| = 7$ data elements. In the figure, circles denote tasks and squares denote data elements, whereas lines denote the interaction among the tasks and data elements. Vertices d_1, d_2, and d_3 respectively denote A-matrix rows r_1^A, r_2^A, and r_3^A, whereas d_4, d_5, d_6, and d_7 respectively denote B-matrix rows r_1^B, r_2^B, r_3^B, and r_4^B. The multiplication of the second row of the A-matrix with the B-matrix is represented by t_2. This multiplication requires an A-matrix row r_2^A and three B-matrix rows r_2^B, r_3^B, and r_4^B. Therefore, $Data(t_2) = \{d_2, d_5, d_6, d_7\}$. The B-matrix row r_4^B is required by the first and second rows of the A-matrix, so $Tasks(d_7) = \{t_2, t_3\}$. $exec(t_i)$ and $size(d_j)$ values are also given for each element, for example $exec(t_2) = 8$ since it consists of $1 + 3 + 4 = 8$ multiply-add operations and $size(d_2) = 3$ since r_2^A contains 3 nonzero elements.



(a) $C = AB$ (b) $\mathcal{A} = (\mathcal{T}, \mathcal{D})$

FIG. 1. *A sample SpGEMM instance and corresponding $\mathcal{A} = (\mathcal{T}, \mathcal{D})$ representation.*

The target computing platform is a homogeneous distributed-memory parallel system consisting of K processors. The execution time of each computational task is assumed to be the same on every processor.

The target problem is to find a computational-task-to-processor partition/assignment. Let $\Pi(\mathcal{T}) = \{\mathcal{T}_1, \mathcal{T}_2, \cdots, \mathcal{T}_K\}$ denote a K-way computational-task-to-processor assignment, where $\mathcal{T}_k$ denotes the set of tasks assigned to processor p_k, for $k = 1, 2, \cdots, K$. This task-to-processor assignment $\Pi(\mathcal{T})$ incurs a data replication schema determined by the data needs of the tasks assigned to individual processors. In this schema, each data element d_j is replicated to each processor where at least one task assigned to that processor needs d_j. Let $WS(p_k)$ denote the working set of processor p_k, which corresponds to the set of data elements needed by p_k for the execution of the tasks assigned to p_k. That is,

$$WS(p_k) = \bigcup_{t_i \in \mathcal{T}_k} Data(t_i). \tag{2.1}$$

Consider a given task-to-processor assignment $\Pi(\mathcal{T})$. The computational load $CL(p_k)$ of processor p_k is computed as

$$CL(p_k) = \sum_{t_i \in \mathcal{T}_k} exec(t_i). \tag{2.2}$$

The data load $DL(p_k)$ of processor p_k is computed as

$$DL(p_k) = \sum_{d_j \in WS(p_k)} size(d_j). \tag{2.3}$$

Note that $DL(p_k)$ corresponds to the memory footprint, which refers to the amount of main memory that processor p_k references while executing the tasks assigned to itself.

In a given assignment $\Pi(\mathcal{T})$, the maximum computational load and the maximum data load of processors are respectively defined as

$$CL_{max} = \max_k \{CL(p_k)\}, \tag{2.4}$$

$$DL_{max} = \max_k \{DL(p_k)\}. \tag{2.5}$$

After describing the framework and giving the notations, we formally define the target problem as follows:

Definition 1 (Simultaneous Computation and Data Load Balancing Problem). *Given an application $\mathcal{A} = (\mathcal{T}, \mathcal{D})$ find a computation-task-to-processor assignment $\Pi(\mathcal{T}_k)$ that minimizes both CL_{max} and DL_{max} given in (2.4) and (2.5), respectively.*

3. Preliminaries.

3.1. Hypergraph partitioning.

A hypergraph $\mathcal{H} = (\mathcal{U}, \mathcal{N})$ consists of a set $\mathcal{U}$ of vertices and a set $\mathcal{N}$ of nets. Each net n_j connects a subset of vertices denoted as $Pins(n_j)$. The degree $deg(n_j)$ of a net n_j denotes the number of vertices it connects, i.e., $deg(n_j) = |Pins(n_j)|$. A cost $c(n_j)$ is associated with each net n_j. $Nets(u_i)$ denotes the set of nets that connect u_i. This easily extends to a subset of vertices $\mathcal{U}_k \subset \mathcal{U}$ so that $Nets(\mathcal{U}_k) = \bigcup_{u_i \in \mathcal{U}_k} Nets(u_i)$. Multiple weights $w^1(u_i), \cdots, w^C(u_i)$ can be associated with each vertex u_i, where $w^c(u_i)$ denotes the cth weight associated with u_i.

$\Pi(\mathcal{H}) = \{\mathcal{U}_1, \cdots, \mathcal{U}_K\}$ is called K-way partition of $\mathcal{H}$, if parts are mutually disjoint and mutually exhaustive. In $\Pi(\mathcal{H})$, the connectivity set $\Lambda(n_j)$ of net n_j consists of the parts that are connected by that net, i.e., $\Lambda(n_j) = \{\mathcal{U}_k : Pins(n_j) \cap \mathcal{U}_k \neq \emptyset\}$. The number of parts connected by n_j is denoted by $\lambda(n_j) = |\Lambda(n_j)|$. A net n_j is said to be cut if it connects more than one part, i.e., $\lambda(n_j) > 1$, and uncut otherwise. A vertex u_i in $\Pi(\mathcal{H})$ is said to be a boundary vertex if it is connected by at least one cut net. Among various cutsize definitions we focus on the connectivity metric as follows:

$$Cutsize(\Pi(\mathcal{H})) = \sum_{n_j \in \mathcal{N}} c(n_j)(\lambda(n_j) - 1). \tag{3.1}$$

In a given partition $\Pi(\mathcal{H})$, the weight $W^c(\mathcal{U}_k)$ of part $\mathcal{U}_k$ is defined as the sum of the cth weights of the vertices in $\mathcal{U}_k$. $\Pi(\mathcal{H})$ is said to be balanced if

$$W^c(\mathcal{U}_k) \le W^c_{avg}(1+\epsilon^c), \text{ for } k \in \{1,2,\cdots,K\} \text{ and } c \in \{1,2,\cdots,C\}, \tag{3.2}$$

where $W^c_{avg} = (\sum_k W^c(\mathcal{U}_k))/K$ and ϵ^c is the predetermined imbalance ratio for the cth weight.

The K-way multi-constraint HP problem [15] is then defined as finding a K-way partition such that the cutsize (3.1) is minimized while the balance constraint (3.2) is maintained. For $C=1$, this reduces to the well-studied standard partitioning problem.

3.2. Recursive bipartitioning (RB) framework. In the RB paradigm, the initial hypergraph is partitioned into two subhypergraphs. These two subhypergraphs are further bipartitioned recursively until K parts are obtained. This process forms a complete binary tree, which we refer to as an RB tree, with $\log_2 K$ levels, where K is a power of 2.

The RB-based HP tools/algorithms utilize a cut-net splitting scheme in order to correctly encode the total cutsize (3.1) at the end of the multi-way partitioning. That is, after each RB step, the cut nets of the respective vertex bipartition $\Pi_2=\{\mathcal{U}_1,\mathcal{U}_2\}$ are split into the two parts of the bipartition. Then, vertex-induced (induced by $\mathcal{U}_1$ and $\mathcal{U}_2$) subhypergraphs $\mathcal{H}_1 = (\mathcal{U}_1,\mathcal{N}_1)$ and $\mathcal{H}_2 = (\mathcal{U}_2,\mathcal{N}_2)$ of $\mathcal{H} = (\mathcal{U},\mathcal{N})$ are constructed as follows:

$$\begin{aligned}
\mathcal{H}_1 &= (\mathcal{U}_1,\mathcal{N}_1) \quad \mathcal{N}_1 = \{n' : \forall n \in \mathcal{N}, Pins(n)\cap\mathcal{U}_1 \neq \emptyset, Pins(n') = Pins(n)\cap\mathcal{U}_1\},\\
\mathcal{H}_2 &= (\mathcal{U}_2,\mathcal{N}_2) \quad \mathcal{N}_2 = \{n'' : \forall n \in \mathcal{N}, Pins(n)\cap\mathcal{U}_2 \neq \emptyset, Pins(n'') = Pins(n)\cap\mathcal{U}_2\}.
\end{aligned}$$

4. Proposed hypergraph models. In this section, we describe the two different hypergraph models proposed for solving the simultaneous computation and data load balancing problem.

4.1. Hypergraph with data vertices (DV). In this model, the application $\mathcal{A}=(\mathcal{T},\mathcal{D})$ is represented by a hypergraph $\mathcal{H}_{DV}(\mathcal{A})=(\mathcal{U}\cup\mathcal{V},\mathcal{N})$ on $|\mathcal{T}|+|\mathcal{D}|$ vertices and $|\mathcal{D}|$ nets with the number of pins equal to

$$\sum_{n\in\mathcal{N}} |Pins(n)| = \sum_{t_i\in\mathcal{T}} |Data(t_i)| + |\mathcal{D}| = \sum_{d_j\in\mathcal{D}} |Tasks(d_j)| + |\mathcal{D}|. \tag{4.1}$$

Vertex set $\mathcal{U}$ represents the computational tasks, where each computational task t_i is represented by a task vertex $u_i \in \mathcal{U}$. Vertex set $\mathcal{V}$ and net set $\mathcal{N}$ represent the data elements. That is, each data element d_j is represented by a data vertex $v_j \in \mathcal{V}$ as well as a net $n_j \in \mathcal{N}$. Each net n_j connects the set of vertices representing the tasks that require data element d_j for their execution as well as data vertex v_j. That is,

$$Pins(n_j) = \{v_j\} \cup \{u_i : t_i \in Tasks(d_j)\} = \{v_j\} \cup \{u_i : d_j \in Data(t_i)\}. \tag{4.2}$$

Therefore, each net connects one data vertex and one or more task vertices. Each net is associated with a cost which is equal to the memory size of the respective data element. That is,

$$c(n_j) = size(d_j). \tag{4.3}$$

Without loss of generality, a given vertex partition $\Pi(\mathcal{H}_{DV}) = \{\mathcal{U}_1\cup\mathcal{V}_1, \mathcal{U}_2\cup\mathcal{V}_2,\cdots,\mathcal{U}_K\cup\mathcal{V}_K\}$ is decoded as a K-way task assignment, where the tasks corresponding to the vertices in $\mathcal{U}_k$ are assigned to processor p_k for $k=1,\cdots,K$. That is $\mathcal{T}_k = \{t_i : u_i \in \mathcal{U}_k\}$. In this setting, $\Lambda(n_j)$ is interchangeably used for both the parts that net n_j connects and the respective processors. Recall that this vertex partition also incurs a data assignment/replication schema. A data element d_j is assigned/replicated to each processor p_k such that $\mathcal{U}_k \in \Lambda(n_j)$. In other words, for each net n_j that connects part $\mathcal{U}_k$, data element d_j is assigned/replicated to processor p_k. That is,

$$WS(p_k) = \{d_j : n_j \in Nets(\mathcal{U}_k)\}. \tag{4.4}$$

For a given partition $\Pi(\mathcal{H}_{DV})$, consider an internal net n_j in $\mathcal{U}_k \cup \mathcal{V}_k$. Then, all tasks which need data element d_j are assigned to the same processor p_k which already holds d_j. So, internal nets do not incur any replication.

Consider a cut net n_j and assume that v_j is assigned to $\mathcal{V}_k \in \Lambda(n_j)$. Then two cases occur as follows: Net n_j connects at least one task vertex in $\mathcal{U}_k$, net n_j does not connect any task vertex in $\mathcal{U}_k$. In the former case, each processor in $\Lambda(n_j)$ needs data element d_j, whereas in the latter case, each processor $\Lambda(n_j) \setminus \{p_k\}$ needs data element d_j. So in both cases, the data element d_j will be replicated to all processors in $\Lambda(n_j) \setminus \{p_k\}$. Hence, $c(n_j)(\lambda(n_j) - 1)$ denotes the total amount of replication because of the data element d_j. So, the partitioning objective of minimizing the cutsize (3.1) corresponds to minimizing the total data replication to be incurred by the task partition.

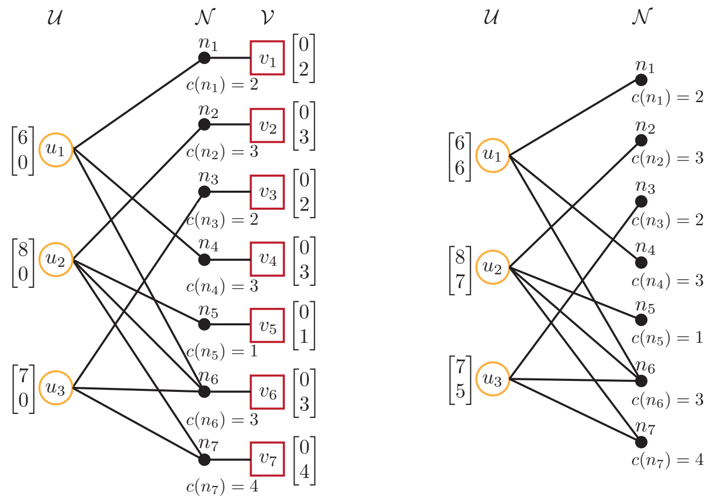
Data vertices are included in the pin lists of the respective nets (e.g., $v_j \in Pins(n_j)$ as shown in (4.2)) for encoding the data loads of the processors through a two-constraint partitioning formulation as follows: The first weight of a task vertex is set equal to the execution time of the respective task, whereas its second weight is set to zero. The first weight of a data vertex is set to zero, whereas its second weight is set equal to the memory size of the respective data element. That is,

$$w^1(u_i) = exec(t_i) \qquad w^1(v_j) = 0 \tag{4.5}$$
$$w^2(u_i) = 0 \qquad w^2(v_j) = size(d_j). \tag{4.6}$$

So, the first partitioning constraint of maintaining balance on parts' first weights encodes balancing computational loads of processors, whereas the second partitioning constraint of maintaining balance on parts' second weights relates to balancing data loads of processors.

Figure 2a shows the $\mathcal{H}_{DV}$ hypergraph for the sample application $\mathcal{A} = (\mathcal{T}, \mathcal{D})$ given in Figure 1. In the figure, circles denote task vertices, squares denote data vertices and dots denote the nets, whereas lines denote pins. For example, $Pins(n_6) = \{u_1, u_2, u_3, v_6\}$ since $Tasks(d_6) = \{t_1, t_2, t_3\}$. The array of two weights associated with each vertex is displayed next to the corresponding vertex, where the upper weight denotes $w^1(\cdot)$ and the lower weight denotes $w^2(\cdot)$. For example, $w^1(u_2) = 8$ since $exec(t_2) = 8$ and $w^2(v_6) = 3$ since $size(d_6) = 3$.

In a straightforward partitioning of $\mathcal{H}_{DV}$, the relation between second constraint and balancing processors' data loads is rather loose since each data vertex is assigned to only one part and does not encode replication of data elements according to the task partition. We enhance this two-constraint formulation with a novel boundary data vertex replication scheme utilized in an RB framework for enabling the second constraint to better encode balancing processors' data loads. In this scheme, the bipartition of the computational task vertices obtained at each RB step is utilized to determine data vertex replication in the further RB steps. The data vertex replication is performed together with the conventional cut-net splitting scheme as follows:



(a) $\mathcal{H}_{DV}(\mathcal{A})$ (b) $\mathcal{H}_{IW}(\mathcal{A})$

FIG. 2. *Hypergraph models for the sample application* $\mathcal{A}=(\mathcal{T},\mathcal{D})$ *given in Figure 1.*

Consider a bipartition $\Pi_2=\{\mathcal{U}_1\cup\mathcal{V}_1,\mathcal{U}_2\cup\mathcal{V}_2\}$ of $\mathcal{H}_{DV}$ at the end of the current RB step. Consider a cut net n_j in Π_2. Cut net n_j possibly connects task vertices in both parts, whereas it connects the respective data vertex v_j which is a boundary vertex in one of the two parts. In the conventional net splitting, net n_j will be split into both parts as n'_j and n''_j with $Pins(n'_j)=Pins(n_j)\cap(\mathcal{U}_1\cup\mathcal{V}_1)$ and $Pins(n''_j)=Pins(n_j)\cap(\mathcal{U}_2\cup\mathcal{V}_2)$, respectively. In the proposed scheme, boundary vertex v_j in one part will be replicated to the other part (as v'_j) so that both of the split nets n'_j and n''_j connect data vertex v_j or v'_j, both of which represent data element d_j. That is,

$$Pins(n'_j) = (Pins(n_j)\cap\mathcal{U}_1)\cup\{v_j\}, \tag{4.7}$$

$$Pins(n''_j) = (Pins(n_j)\cap\mathcal{U}_2)\cup\{v'_j\}. \tag{4.8}$$

So, bipartition $\Pi_2=\{\mathcal{U}_1\cup\mathcal{V}_1,\mathcal{U}_2\cup\mathcal{V}_2\}$ obtained at a particular RB step induces the hypergraphs $\mathcal{H}_{DV1}$ and $\mathcal{H}_{DV2}$ for further bipartitioning in the following RB steps:

$$\begin{aligned}
\mathcal{H}_{DV1} &= ((\mathcal{U}_1\cup\mathcal{V}_1)\cup\mathcal{V}_2^B),\mathcal{N}_1) \text{ and } \mathcal{H}_{DV2}=((\mathcal{U}_2\cup\mathcal{V}_2)\cup\mathcal{V}_1^B),\mathcal{N}_2), \text{ where}\\
\mathcal{N}_1 &= \{n' : \forall n\in\mathcal{N}, Pins(n)\cap(\mathcal{U}_1\cup\mathcal{V}_1)\neq\emptyset, Pins(n')=Pins(n)\cap(\mathcal{U}_1\cup\mathcal{V}_1\cup\mathcal{V}_2^B)\}\\
\mathcal{N}_2 &= \{n'' : \forall n\in\mathcal{N}, Pins(n)\cap(\mathcal{U}_2\cup\mathcal{V}_2)\neq\emptyset, Pins(n'')=Pins(n)\cap(\mathcal{U}_2\cup\mathcal{V}_2\cup\mathcal{V}_1^B)\}.
\end{aligned}$$

Here, $\mathcal{V}_1^B$ and $\mathcal{V}_2^B$ respectively denote boundary data vertex sets of $\mathcal{V}_1$ and $\mathcal{V}_2$. So, $((\mathcal{U}_1\cup\mathcal{V}_1)\cup\mathcal{V}_2^B)$ denotes the replication of boundary data vertex set of $\mathcal{V}_2$ to $\mathcal{H}_{DV1}$ and in a dual manner $(\mathcal{U}_2\cup\mathcal{V}_2)\cup\mathcal{V}_1^B)$ denotes the replication of the boundary data vertex set of $\mathcal{V}_1$ to $\mathcal{H}_{DV2}$. The weights of vertices remain the same after the RB step.

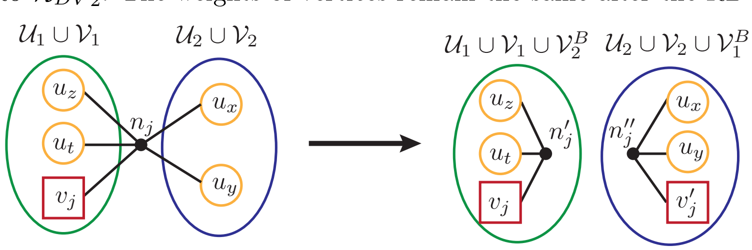


FIG. 3. *Proposed data vertex replication together with cut-net splitting.*

Figure 3 shows the usage of the RB framework through a sample bipartition in terms of a single cut net n_j which connects two task vertices and one data vertex in the left part and two task vertices in the right part. Here and hereafter part 1 and part 2 of the bipartition are respectively referred to as left and right parts for clarity of the presentation. As seen in the figure, this cut net is split as n' and n'' to the left and right parts, respectively. The boundary data vertex v_j connected by n_j in the left part is replicated to the right part as v'_j so that $Pins(n'_j) = \{u_z, u_t, v_j\}$ and $Pins(n''_j) = \{u_x, u_y, v'_j\}$.

In the conventional cut-net splitting scheme [14], after an RB step, if a cut net n_j connects only one vertex in one of the parts, then n_j is not split to that part since it will incur a single-pin net in that part and single-pin nets do not contribute to the cutsize in the further RB steps. However, in the proposed scheme, such cases should be handled differently depending on whether the only vertex connected by a cut net in one of the parts of the bipartition is a data vertex or task vertex.

Special case 1: This case occurs when the only vertex connected by a cut net n_j in one part is a data vertex v_j. The proposed scheme replicates v_j to the other part. However, vertex v_j connected by a single-pin split net n'_j corresponds to a data element assigned to a processor that is not assigned any computation task which needs the data element d_j. Hence, we move (instead of replicating) data vertex v_j to the other part so that the cut net n_j becomes internal. Note that this move operation will decrease the cutsize but has the potential of increasing the imbalance on the second part weights.

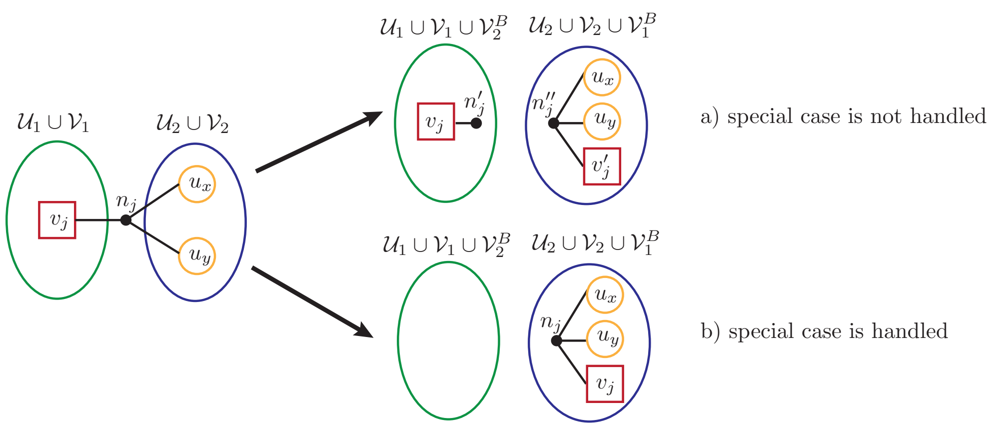


FIG. 4. *Special case 1: the only vertex connected by a cut net in one of the parts of the bipartition is a data vertex. a) special case not handled, b) special case handled*

Figure 4 shows the handling of the special case 1 through a sample bipartition in the context of a single cut net n_j. Cut net n_j connects only a data vertex v_j on the left part, whereas it connects two task vertices u_x and u_y on the right part. The upper arrow shows the splitting of n_j as well as the replication of v_j, if the special case is not handled. The lower arrow shows the proposed handling of the special case, where the data vertex v_j on the left part moved to the right part so that n_j becomes an internal net of the right part.

Special case 2: This case occurs when the only vertex connected by a cut net n_j in one part is a task vertex u_i. The proposed scheme replicates v_j in the other part to this part. However, this replication will incur a two-pin split net connecting task vertex u_i and data vertex v'_j, which refers to a data vertex needed by a single task vertex. The trivial solution for such two-pin nets is to maintain them internal in further RB steps by avoiding this net splitting together with the data vertex replication while assigning the second weight of data vertices to the second weight of the task vertices. That is, u_i will contain two nonzero weights $w^1(u_i) = exec(t_i)$ and $w^2(u_i) = w^2(u_i) + size(d_j)$, on the contrary the initial two-constraint formulation in the top-most level where each vertex has one nonzero and one zero weight.

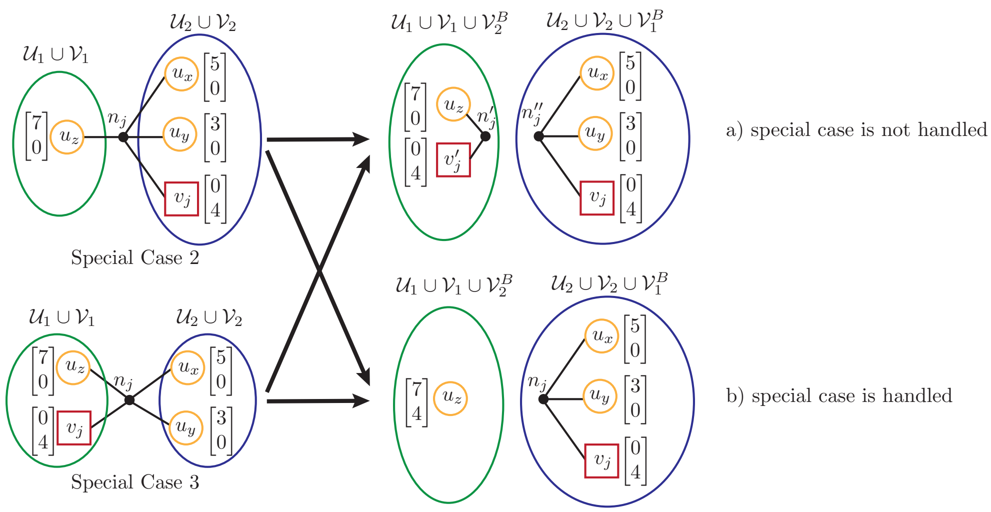


FIG. 5. *Special case 2: the only vertex connected by a cut net in one of the parts of the bipartition is a task vertex, whereas special case 3: a cut net connects only one task vertex together with the data vertex in one of the parts. a) special case not handled, b) special case handled*

Figure 5 shows the handling of special case 2 after a sample RB step through a sample bipartition in the context of a single cut net n_j. Cut net n_j connects only a task vertex on the left part, whereas it connects two task vertices and a data vertex on the right part. The upper arrow shows the splitting of n_j as well as the replication of v_j, if the special case is not handled. The lower arrow shows the proposed handling of the special case, where neither v_j is replicated to the left part nor n_j is split to the left part. Instead the second weight of data vertex v_j on the right part is added to the second weight of the task vertex on the left part.

Special case 3: This case occurs when a cut net n_j connects only one task vertex u_z together with the data vertex v_j in one of the parts. The proposed scheme will incur a two-pin split net connecting task vertex u_z and data vertex v_j in that part. So this case becomes very similar to the special case 2 and handled in the same manner by moving v_j to the other part and add its second weight to the second weight of u_z.

Figure 5 shows the handling of special case 3 which is equivalent to the special case 2 except moving v_j from the right part to the left part so that handled and unhandled split partitions will be the same.

Algorithm 4.1 shows the RB-based partitioning of $\mathcal{H}_{DV}$ utilizing the proposed data vertex replication scheme. If-statements at lines 11-14, 15-19, and 20-26 respectively show the handling of the special cases 1, 2, and 3. Statements at lines 28-30 show the replication of the data vertex to the other part. Note that data vertex replication is performed only if none of the special cases occur for the respective net.

4.2. Hypergraph model with inverse data weight (IW) distribution. In this model, the application $\mathcal{A} = (\mathcal{T}, \mathcal{D})$ is represented by a hypergraph $\mathcal{H}_{IW}(\mathcal{A}) = (\mathcal{U}, \mathcal{N})$

Algorithm 4.1 Partition $\mathcal{H}_{DV}$ with proposed data vertex replication

Input: $\mathcal{H}_{DV} = (\mathcal{U} \cup \mathcal{V}, \mathcal{N}, w^1, w^2)$, K
Output: $\Pi(\mathcal{H}_{DV})$

$\mathcal{H}_0^0 = \mathcal{H}_{DV}$
for $\ell \leftarrow 0$ **to** $\log_2 K - 1$ **do**
  for $k \leftarrow 0$ **to** $2^\ell - 1$ **do**
    $\Pi_2 \leftarrow$ BIPARTITION$(\mathcal{H}_k^\ell)$ ▷ $\Pi_2 = \{\mathcal{U}_L \cup \mathcal{V}_L, \mathcal{U}_R \cup \mathcal{V}_R\}$
    for each cut net $n_j \in \mathcal{N}_k^\ell$ **do**
      $flag \leftarrow$ **true**
      if $v_j \in \mathcal{V}_L$ **then**
        $x \leftarrow L$; $y \leftarrow R$
      else
        $x \leftarrow R$; $y \leftarrow L$
      if $Pins(n_j) \cap \mathcal{U}_x = \emptyset$ **then** ▷ special case 1
        $\mathcal{V}_y \leftarrow \mathcal{V}_y \cup \{v_j\}$
        $\mathcal{V}_x \leftarrow \mathcal{V}_x \setminus \{v_j\}$ ▷ move data vertex v_j to the other part
        $flag \leftarrow$ **false** ▷ Net n_j becomes internal
      if $|Pins(n_j) \cap \mathcal{U}_y| = 1$ **then** ▷ special case 2
        $u_i \leftarrow Pins(n_j) \cap \mathcal{U}_y$
        $w^2(u_i) \leftarrow w^2(u_i) + w^2(v_j)$
        $Pins(n_j) \leftarrow Pins(n_j) \setminus \{u_i\}$
        $flag \leftarrow$ **false** ▷ Net n_j becomes internal
      if $|Pins(n_j) \cap \mathcal{U}_x| = 1$ **then** ▷ special case 3
        $u_i \leftarrow Pins(n_j) \cap \mathcal{U}_x$
        $w^2(u_i) \leftarrow w^2(u_i) + w^2(v_j)$
        $\mathcal{V}_y \leftarrow \mathcal{V}_y \cup \{v_j\}$
        $\mathcal{V}_x \leftarrow \mathcal{V}_x \setminus \{v_j\}$ ▷ move data vertex v_j to the other part
        $Pins(n_j) \leftarrow Pins(n_j) \setminus \{u_i\}$
        $flag \leftarrow$ **false** ▷ Net n_j becomes internal
      if $flag$ **then**
        $v'_j \leftarrow v_j$
        $\mathcal{V}_y \leftarrow \mathcal{V}_y \cup \{v'_j\}$ ▷ replicate v_j to other part
        $Pins(n_j) \leftarrow Pins(n_j) \cup \{v'_j\}$
    Form $\mathcal{H}_{2k}^{\ell+1} = (\mathcal{U}_L \cup \mathcal{V}_L, \mathcal{N}_L)$ induced by $\mathcal{U}_L \cup \mathcal{V}_L$
    Form $\mathcal{H}_{2k+1}^{\ell+1} = (\mathcal{U}_R \cup \mathcal{V}_R, \mathcal{N}_R)$ induced by $\mathcal{U}_R \cup \mathcal{V}_R$

on $|\mathcal{T}|$ vertices and $|\mathcal{D}|$ nets with the number of pins equals to

$$(4.9) \qquad \sum_{n \in \mathcal{N}} |Pins(n)| = \sum_{t_i \in \mathcal{T}} |Data(t_i)| = \sum_{d_j \in \mathcal{D}} |Tasks(d_j)|.$$

Vertex set $\mathcal{U}$ represents the computational tasks. Net set $\mathcal{N}$ represents the data elements. That is, each computational task t_i is represented by a task vertex $u_i \in \mathcal{U}$ and each data element d_j is represented by a net $n_j \in \mathcal{N}$. Each net n_j connects the set of vertices representing the tasks that require the data element d_j for their execution. That is,

$$(4.10) \qquad Pins(n_j) = \{u_i : t_i \in Tasks(d_j)\} = \{u_i : d_j \in Data(t_i)\}.$$

Comparison of (4.10) and (4.2) shows that $\mathcal{H}_{DV}$ and $\mathcal{H}_{IW}$ topologically differs by vertex set $\mathcal{V}$, which corresponds to data elements of $\mathcal{H}_{DV}$. That is, $\mathcal{H}_{DV}$ becomes topologically the same with the $\mathcal{H}_{IW}$ when data vertices in $\mathcal{V}$ and corresponding pins are removed.

Each net is associated with a cost which is equal to the memory size of the respective data element. That is,

$$c(n_j) = size(d_j). \tag{4.11}$$

Without loss of generality, a given vertex partition $\Pi(\mathcal{H}_{IW})=\{\mathcal{U}_1,\mathcal{U}_2,\cdots,\mathcal{U}_K\}$ is decoded as a K-way task partition, where the tasks corresponding to the vertices in $\mathcal{U}_k$ are assigned to processor p_k for $k=1,\cdots,K$. Recall that this vertex partition also incurs a data assignment/replication schema. In other words, for each net n_j that connects part $\mathcal{U}_k$, data element d_j is assigned/replicated to processor p_k. That is,

$$WS(p_k) = \{d_j : n_j \in Nets(\mathcal{U}_k)\}. \tag{4.12}$$

For a given partition $\Pi(\mathcal{H}_{IW})$, $\lambda(n_j)$ denotes the number of processors that need the data element d_j. So $\lambda(n_j) - 1$ denotes the number of times the data element d_j needs to be replicated. Hence, $c(n_j)(\lambda(n_j)-1)$ denotes the total amount of replication because of the data element d_j. So, the partitioning objective of minimizing the cutsize according to (3.1) corresponds to minimizing the total data replication to be incurred by the task partition.

In this model, a two-constraint partitioning formulation is also used, where the first and second weights of each vertex refer to the computational and data loads of the respective task. Since $\mathcal{H}_{IW}$ does not contain data vertices, we propose a novel inverse data weight distribution model for estimating the second weights of vertices. In this model, the cost of a net, which corresponds to the size of the respective data element, is distributed evenly among the second weights of the vertices connected by that net. That is, a net n_j of cost $c(n_j)$, which represents data element d_j of $size(d_j)$, contributes $c(n_j)/deg(n_j)$ to each vertex it connects. Finally, the first weight of a task vertex is set equal to the execution time of the respective task, whereas its second weight is set to sum of the contributions from each net n_j connecting that vertex by $c(n_j)/deg(n_j)$. That is,

$$w^1(u_i) = exec(t_i) \qquad w^2(u_i) = \sum_{n_j \in Nets(u_i)} \frac{c(n_j)}{deg(n_j)} \tag{4.13}$$

So, the first partitioning constraint of maintaining balance on parts' first weights encodes balancing computational loads of processors, whereas the second partitioning constraint of maintaining balance on parts' second weights relates to balancing data loads of processors.

Figure 2b shows the $\mathcal{H}_{IW}$ hypergraph for the sample application $\mathcal{A} = (\mathcal{T}, \mathcal{D})$ given in Figure 1. For example, $Pins(n_7)=\{u_2,u_3\}$ since $Tasks(d_7)=\{t_2,t_3\}$. For example, $w^1(u_2)=8$ since $exec(t_2)=8$. Regarding the second weight of u_2; the nets n_2, n_5, n_6, and n_7, which connect u_4, respectively contribute $c(n_2)/deg(n_2)=3/1=3$, $c(n_5)/deg(n_5) = 1/1 = 1$, $c(n_6)/deg(n_6) = 3/3 = 1$, and $c(n_7)/deg(n_7) = 4/2 = 2$ to $w^2(u_2)$. That is, $w^2(u_2)=3+1+1+2=7$.

The motivation behind the proposed inverse data weight distribution model can be described as follows: In a given partition of $\mathcal{H}_{IW}$, consider an internal net n_j of $\mathcal{U}_k$. This net refers to the case where all tasks requiring data element d_j are assigned to the same processor p_k. Net n_j will contribute a total weight of $c(n_j)$ to the second weight of $\mathcal{U}_k$ which in turn will correspond to contributing $c(n_j) = size(d_j)$ to the data load $DL(p_k)$ of processor p_k. So, internal nets correctly encode the data loads of the processors to which they are internal.

However, consider a cut net n_j with connectivity set $\Lambda(n_j)$. Net n_j will contribute fractional weights to the second weights of parts/processors in $\Lambda(n_j)$. The distribution of the weight $c(n_j)$ will be proportional to the number of pins it connects in those parts. That is, for each $\mathcal{U}_k \in \Lambda(n_j)$, net n_j will contribute

$$\frac{|Pins(n_j) \cap \mathcal{U}_k|}{|Pins(n_j)|} c(n_j) \tag{4.14}$$

to the second weight $W^2(\mathcal{U}_k)$ of part $\mathcal{U}_k$. This in turn corresponds to net n_j contributing $size(d_j)|Pins(n_j) \cap \mathcal{U}_k|/deg(n_j)$ to the data load $DL(p_k)$ of processor p_k.

That is, data load of a processor is correctly encoded by internal nets in the corresponding part, whereas an error is made by the cut nets connecting that part. This is because data weight is encoded partially by the vertices assigned to that part. Note that the partitioning objective of minimizing the cutsize will minimize this error due to the cut nets. Also, errors made due to the fractional weight distribution of the cut nets can be expected to cancel each other. Consider two nets n_j and n_h of equal degree $deg(n_j) = deg(n_h) = \mathtt{deg}$ and equal cost $c(n_j) = c(n_h) = \mathtt{size}$. Assume that in the given partition, these two nets become cut and connect only the same two parts $\mathcal{U}_k$ and $\mathcal{U}_\ell$ in the partition. Also assume that n_j connects α pins in $\mathcal{U}_k$ and $\mathtt{deg}-\alpha$ pins in $\mathcal{U}_\ell$, whereas n_h connects $\mathtt{deg}-\alpha$ pins in $\mathcal{U}_k$ and α pins in $\mathcal{U}_\ell$. Despite the erroneous fractional data load contributions because of these two cut nets to the data loads of processors p_k and p_ℓ, they together contribute the same amount of data load of `size` to both processors p_k and p_ℓ. Although the actual aggregate contribution of n_j and n_h should be 2`size` to both processors, assigning the same load of `size` to both parts enables partitioner's load balancing mechanism to indirectly encode balancing data loads of processors. This discussion can be extended for the nets with different number of pins and costs.

Here we exploit the RB framework in order to improve the proposed $\mathcal{H}_{IW}$ model as follows: Recall that the proposed model distributes the cost of each net evenly among the second weights of the vertices that it connects. The degrees of the nets decrease each time they become cut during the RB process because of the cut-net splitting scheme adopted. So, updated degree information of the split nets should be used for a more accurate net cost distribution. Therefore, after each RB step, the second weights of vertices in each of the two subhypergraphs are computed from scratch by taking into account the updated degree information of the split nets. Although the contributions of the internal nets to the second vertex weights do not change since their degrees remain the same, computing the second weights from scratch seems to be more efficient.

Figure 6 shows the usage of the RB framework through a sample bipartition in terms of a single cut net n_j of degree five. As seen in the figure, n_j evenly distributes its cost 30 among the second weights of those five vertices as $30/5 = 6$ before the current bipartitioning step. After the RB step, the degrees of the split nets n' and n'' become three and two, respectively. Therefore, in the left part, n'_j contributes $30/3 = 10$ to the second weights of the vertices u_z, u_t and, u_s, whereas, in the right part, n''_j contributes $30/2 = 15$ to the second weights of the vertices u_x and u_y.

Algorithm 4.2 shows the RB-based partitioning of $\mathcal{H}_{IW}$ with the proposed inverse data weight distribution. The for loop at lines 6-9 computes the inverse data weight of each net and then distributes this weight to those vertices that it connects.

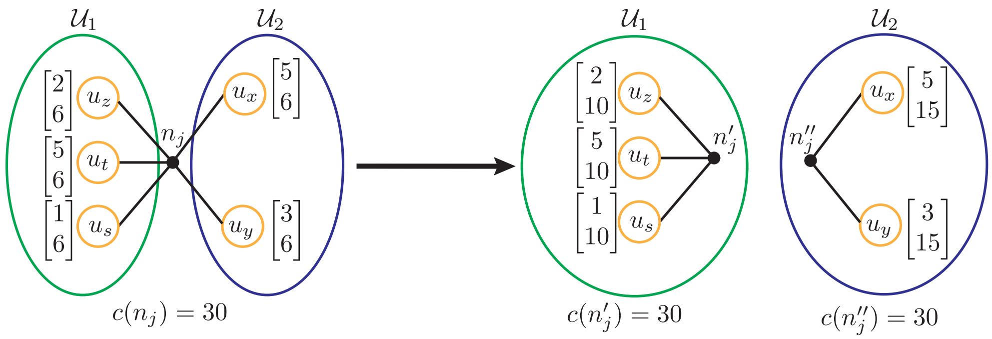


FIG. 6. *Proposed inverse data weight distribution together with cut-net splitting.*

Algorithm 4.2 Inverse Data Weight Distribution Algorithm

Input: $\mathcal{H}_{IW} = (\mathcal{U}, \mathcal{N}, w^1, c)$, K
Output: $\Pi(\mathcal{H}_{IW})$

$\mathcal{H}_0^0 = \mathcal{H}_{IW}$
for $\ell \leftarrow 0$ **to** $\log_2 K - 1$ **do**
  for $k \leftarrow 0$ **to** $2^\ell - 1$ **do**
    for each vertex $u_i \in \mathcal{U}_k^\ell$ **do**
      $w^2(u_i) \leftarrow 0$
    for each net $n_j \in \mathcal{N}_k^\ell$ **do**
      $idwContr \leftarrow c(n_j)/deg(n_j)$;
      for each $u_i \in Pins(n_j)$ **do**
        $w^2(u_i) \leftarrow w^2(u_i) + idwContr$
    $\Pi_2 \leftarrow \text{BIPARTITION}(\mathcal{H}_k^\ell)$ ▷ $\Pi_2 = \{\mathcal{V}_L, \mathcal{V}_R\}$
    Form $\mathcal{H}_L = \mathcal{H}_{2k}^{\ell+1} = (\mathcal{V}_L, \mathcal{N}_L)$ induced by $\mathcal{V}_L$
    Form $\mathcal{H}_R = \mathcal{H}_{2k+1}^{\ell+1} = (\mathcal{V}_R, \mathcal{N}_R)$ induced by $\mathcal{V}_R$

4.3. Discussion. As mentioned earlier, the two proposed models are topologically similar, where nets represent data elements. So, in both models, the partitioning objective of minimizing the cutsize encodes the minimization of the total amount of data replication via clustering the tasks that require the same data elements to the same parts under the given balancing constraints. This partitioning objective also encodes the amount of communication volume to incur for realizing the required data replication among processors. We should note here that the same data element required by multiple tasks assigned to the same processor necessitates the replication of that data element only once. Both proposed hypergraph models encapsulate this replication correctly, whereas a similar bipartite graph model would overestimate.

As discussed earlier, both models utilize a two-constraint formulation, where the first and second weights of vertices are respectively used to encode the computational and the data loads of parts/processors. Since the total amount of computation is constant, the partitioning constraint on maintaining balance on the parts' first weights encodes minimizing the computational load of the maximally loaded processor within the given computational load imbalance ratio (ϵ^1). However, the total amount of data load of processors, which is the sum of the second weights of the parts, is not constant and it depends on the quality of the task partitioning. So, a naive balancing on the seconds weights of the parts might not encode the minimization of the data load of the maximally loaded processor. For example, a very tight balance on the data loads of the processors might yield a very high data load on the maximally loaded processor if the underlying partition produces a high amount of data replication. Therefore, the second partitioning constraint on maintaining balance on parts' second weights under the partitioning objective that encodes the minimization of the total amount of data replication corresponds to minimizing the data load of the maximally loaded processor within the given data load imbalance ratio (ϵ^2).

The two hypergraph models differ in the second vertex weighting scheme they utilize for estimating processors' data loads. Both models correctly encode the processors' data loads corresponding to the data elements that are required by a single processor. Both models utilize the RB framework to increase the accuracy of their schemes utilized to estimate processors' data loads corresponding to the data elements that are required by multiple processors. After each RB step, the $\mathcal{H}_{DV}$ model corrects the topology of the subhypergraphs by augmenting the subhypergraphs with the replicated data vertices according to the induced task vertex bipartition. At the beginning of each RB step, the $\mathcal{H}_{IW}$ model utilizes the topology of the current hypergraph to predict the difference between data loads of the two parts of the bipartition. That is, the $\mathcal{H}_{DV}$ model tries to encode data loads of processors after the RB step, whereas the $\mathcal{H}_{IW}$ model tries to encode data loads of processors before the RB step.

5. Experiments.

5.1. Baseline model.

The baseline model is the conventional HP model widely used for the parallelization of irregularly sparse applications. The topology of this hypergraph model is exactly the same with that of the $\mathcal{H}_{IW}$ model described in subsection 4.2. However, this model is a single constraint model, where the vertices are weighted with the computational loads of the respective tasks they represent. In this way, the partitioning constraint of balancing part weights encodes computational balance among processors. The cost of the nets is set to be equal to the size of the data elements they represent. So, the partitioning objective of minimizing the cutsize according to connectivity metric (3.1) encodes minimizing total communication volume [15]. Note that this partitioning objective also encodes minimizing total amount of data replication. So, the baseline model differs from the proposed models in not considering data load balancing at all. Here and hereafter, we refer to this hypergraph model as $\mathcal{H}_{Base}$.

5.2. Experimental setup.

The hypergraph models proposed in subsections 4.1 and 4.2, as well as the baseline hypergraph model mentioned in subsection 5.1 are partitioned using the HP tool PaToH [14,16] for obtaining $K \in \{64, 128, 256, 512, 1024\}$-way partitions. PaToH is used with default parameters for all models except for the $\mathcal{H}_{DV}$ model described in subsection 4.1. For the $\mathcal{H}_{DV}$ model, we set PaToH's vertex visit order to the continuous/sequential vertex order (increasing vertex ID order) for the coarsening phase instead of the random vertex visit order which is the default [16]. The objective behind this is to prioritize matching of computational vertices with other computational or data vertices. In order to maintain randomness in vertex visit order, we randomly permute computational and data vertices separately before invoking PaToH. In all partitioning instances, we used maximum allowable imbalance ratio $\epsilon = 0.05$ for both computation and data weights, i.e., $\epsilon^1 = 0.05$ and $\epsilon^2 = 0.05$. All experiments have been conducted by use of random seed. As PaToH utilizes randomized algorithms, we partitioned each instance five times with different seeds and we report the geometric average of the results.

5.3. Dataset. The performance of the two proposed models are validated against the baseline model on two sample applications: Parallel Finite Element Method (FEM) and Volume Element Method (VEM) based simulations which involve partitioning irregular 2D or 3D meshes and parallel Sparse Generalized Matrix Matrix Multiplication (SpGEMM) which involves partitioning two irregularly sparse input matrices.

5.3.1. Mesh partitioning instances. In the FEM/VEM applications, computations associated with each mesh/volume element (cell) constitute an atomic computational task. These applications implement ghost layering methods which involve replicating cells according to the cell-to-cell neighborhood relation determined by the target application [22]. Hence, such FEM/VEM applications fall within the framework in section 2 since computations associated with each element are independent but they share data elements determined by the neighborhood relation.

In order to obtain mesh partitioning instances, we utilize sparse matrices which have either 2D or 3D coordinate values so that the sparsity patterns of those matrices are considered as representing the neighborhood structures of the meshes arising in FEM or VEM applications. Such matrices are selected from the SuiteSparse Matrix Collection [23]. These matrices are symmetric matrices so that rows and columns respectively represent mesh elements and data elements or vice versa. So, for a selected sparse matrix $A = (a_{ij})$, we have

$$Data(t_i) = \{d_j : a_{ij} \neq 0\}. \tag{5.1}$$

Here, t_i denotes the atomic task associated with mesh cell c_i and d_j denotes the data associated with cell c_j.

We should note that SuiteSparse Matrix Collection [23] does not contain any information about computational cost and data size distribution. Various computational cost and data size weighting schemes are utilized depending on the application nature of mesh computations [7, 22, 29, 34]. In this paper, we use the weighting scheme produced by a heuristic for generating realistic weight distributions for "Particles-in-Cells"-like simulations [7]. The computational cost of a mesh element is reported to be equal to the square of its memory size for the weighting [34] and the amount of memory needed for holding a cell is linear with the number of particles located in this cell [7]. Reasoning behind this is data size is related to the number of particles located in a cell, while computational cost associated with a cell generally increases with the square of the number of particles in this cell. That is,

$$exec(t_i) \;=\; npic^2(c_i) \qquad size(d_j) = npic(c_j), \tag{5.2}$$

where $npic(c_i)$ denotes the number of particles in cell c_i.

We did not include matrices with less than $100 \times K$ matrix rows so that each processor will be assigned at least 100 rows on average. That is, we have matrices that have at least 6400 rows for 64-way partitioning and 12800 rows for 128-way partitioning. As a result of this selection criterion, the experiments are conducted for a total of 464 partitioning instances (117, 108, 94, 82, and 63 instances for 64-, 128-, 256-, 512-, and 1024-way partitions, respectively).

5.3.2. SpGEMM partitioning instances. Consider the SpGEMM application of the form $C = AB$, where input matrices A and B are of sizes $q \times r$ and $r \times s$. In row-row-parallel SpGEMM, the atomic computational task t_i is the pre-multiplication of row i of matrix A with the whole matrix B. Then, the task set $\mathcal{T} = \{t_1, t_2, \cdots, t_q\}$

contains q tasks $\{r_1^A B, r_2^A B, \cdots, r_q^A B\}$, where r_i^A denotes row i of A matrix. The sparse-vector-matrix multiplication $r_i^A B$ requires row r_i^A as well as those B-matrix rows that correspond to the column indices ($cols$) of the nonzeros of row r_i^A. That is,

$$Data(t_i) = \{a_{i,*}\} \cup \{b_{x,*} : x \in cols(a_{i,*})\}. \tag{5.3}$$

So, the sets of A- and B-matrix rows constitute the set of $q+r$ data elements. That is, $\mathcal{D} = \{d_1, d_2, \cdots, d_{q+r}\} = \{r_1^A, r_2^A, \cdots, r_q^A, r_1^B, r_2^B, \cdots, r_r^B\}$. Hence, this row-row-parallel SpGEMM application falls within the framework in section 2, since vector-matrix multiplications are independent but they share B-matrix rows. The computational costs for atomic tasks and sizes for data elements are easily defined as follows:

$$exec(t_i) = \sum_{x \in cols(a_{i,*})} nnz(b_{x,*}), \tag{5.4}$$

$$size(d_j) = nnz(a_{j,*}) \text{ for } 1 \leq j \leq q, \tag{5.5}$$

$$size(d_j) = nnz(b_{j-q,*}) \text{ for } q+1 \leq j \leq q+r. \tag{5.6}$$

Here, we consider two types of SpGEMM instances: $C=AB$ and $C=AA$. For $C=AB$, we generate 69 instances from the SuiteSparse matrix collection [23] in a similar way to [3]. Matrices *amazon0302* and *amazon0312* are used as A matrices which represent the similarity between items and B matrices are generated utilizing a Zipf distribution (with exponent set to 3.0) to determine the item preferences and a uniform distribution to determine the users that prefer a specific item [31]. In this setting, $C = AB$ gives the candidate items to be recommended to each user. We generated 66 instances by considering the setup phase of Algebraic Multigrid methods [8] which involves the Galerkin product of the form RAP that necessitates two consecutive SpGEMM operations. 11 of these instances are of the form $C = RA$, whereas the remaining 55 instances are of the form $C = AP$. The last instance in this category contains two different matrices, namely *thermomech_ dK* and *thermomech_ dM*, which are conformable for multiplication.

For the $C=AA$ type of instances, we selected 12 matrices from the SuiteSparse Matrix Collection [23]. The number of rows/columns and number of nonzeros are in the range of 88K - 1.5M, and 2.5M - 30M, respectively. For the $C = AA$ type of instances, the fact that the B matrix is actually the A matrix can be exploited in order to reduce the memory footprint of the application. On the other hand, for the sake of computational efficiency, our parallel SpGEMM implementation does not exploit this fact. That is, we partition $C = AA$ instances as we partition $C = AB$ instances.

The same partitioning granularity principle utilized for mesh instances is also used for SpGEMM instances. So, experimental results of the $C = AB$ instances are reported for a total of 259 partitioning instances (69, 63, 54, 45, and 28 instances for 64-, 128-, 256-, 512-, and 1024-way partitions, respectively). Experimental results of the $C=AA$ instances are reported for a total of 59 partitioning instances (12, 12, 12, 12, and 11 instances for 64-, 128-, 256-, 512-, and 1024-way partitions, respectively).

5.3.3. Data size variation. Here, we compare and discuss the irregularity of the datasets in terms of coefficient of variation (CV) values on the data sizes, where CV values are computed as standard deviation divided by mean. The purpose is to observe the relation between load balancing performance of the proposed algorithms and the irregularity of the datasets defined as the size variation of the data elements.

Table 1 displays average CV values as well as the number of partitioning instances for each dataset and for each number of processors. Here higher CV values correspond to higher irregularity of the data sizes. As seen in the table, the $C=AA$ dataset has the largest average CV (0.90 on $K=64$ processors), the $C=AB$ dataset has the smallest average CV (0.32 on $K=64$ processors), and the mesh dataset has the in-between average CV (0.68 on $K=64$ processors). So, in terms of data size variation, the $C=AA$ dataset is the most irregular dataset, whereas the $C=AB$ and the mesh datasets are respectively least and in-between irregular datasets.

TABLE 1
Number of partitioning instances and average coefficient of variation (CV) values for each dataset

	mesh dataset		$C=AB$ dataset		$C=AA$ dataset	
K	# of ins.	CV	# of ins.	CV	# of ins.	CV
64	117	0.68	69	0.32	12	0.90
128	108	0.66	64	0.34	12	0.90
256	94	0.65	54	0.34	12	0.90
512	82	0.65	45	0.35	12	0.90
1024	63	0.60	28	0.36	11	0.81

5.4. Performance comparison.

5.4.1. Performance metrics. For each application, we evaluate the performance of the partitioning models in terms of maximum computational load CL_{max} and maximum data load DL_{max} handled by a processor (given in (2.4) and (2.5), respectively). For a simpler presentation, we give ratios of those metrics to their averages. That is,

$$CL^r_{max} = \frac{CL_{max}}{CL_{avg}}, \text{ where } CL_{avg} = \frac{1}{K}CL_{tot} = \frac{1}{K}\sum_{t_i \in \mathcal{T}} exec(t_i), \tag{5.7}$$

$$DL^r_{max} = \frac{DL_{max}}{DL^*_{avg}}, \text{ where } DL^*_{avg} = \frac{1}{K}DL_{tot} = \frac{1}{K}\sum_{d_j \in \mathcal{D}} size(d_j). \tag{5.8}$$

Here, CL_{avg} denotes the average computational load per processor under perfect load balance. DL^*_{avg} denotes average data load per processor without data replication under perfect load balance. So, DL^*_{avg} denotes the ideal average data load.

We also report the total data size (original total size together with replicated data size) incurred by the models as the ratio to original total data size as follows:

$$DL^r_{rep} = \frac{\sum_{k=1}^{K} DL(p_k)}{DL_{tot}} = \frac{\sum_{k=1}^{K} \sum_{d_j \in WS(p_k)} size(d_j)}{DL_{tot}}. \tag{5.9}$$

This metric also defines a lower bound for the DL^r_{max} metric. That is, under perfect data load balance $DL^r_{max} = DL^r_{rep}$. It is clear that, in each of these three metrics, a smaller value refers to a better performance.

Recall that both proposed models $\mathcal{H}_{DV}$ and $\mathcal{H}_{IW}$ utilize the RB framework to increase their effectiveness. So, we also report the performance results for these models without utilizing the RB framework in order to show the relative performance improvement attained by the use of the RB framework. These experiments are performed by directly K-way partitioning the hypergraph constructed at the very beginning. We use $_{RB}$ and $_{Dr}$ subscripts to refer to models which utilize RB and which do not utilize RB, respectively. That is, $\mathcal{H}_{DV_{RB}}$ and $\mathcal{H}_{IW_{RB}}$ refer to the models that utilize the RB framework, whereas $\mathcal{H}_{DV_{Dr}}$ and $\mathcal{H}_{IW_{Dr}}$ refer to the models that do utilize the RB framework.

5.4.2. Average performance comparison. Table 2 displays average performance comparison for the mesh partitioning instances. As seen in the table, both proposed models perform much better data load balancing than the baseline model without any performance degradation in computational load balance. The performance gap between the proposed models and the baseline model increases with increasing number of processors in favor of the proposed models. For example, performance improvement of $\mathcal{H}_{DV_{RB}}$ over $\mathcal{H}_{Base}$ in the DL^r_{max} metric is 28%, 30%, 34%, 39%, and 45% on respectively $K \in \{64, 128, 256, 512, 1024\}$ processors, .

TABLE 2
Average performance comparison for the mesh partitioning instances

K	number of instances	$\mathcal{H}_{Base}$	$\mathcal{H}_{DV_{Dr}}$	$\mathcal{H}_{DV_{RB}}$	$\mathcal{H}_{IW_{Dr}}$	$\mathcal{H}_{IW_{RB}}$
		DL^r_{max}: maximum data load ratio				
64	117	1.83	1.33	1.31	1.29	1.25
128	108	2.02	1.44	1.41	1.40	1.34
256	94	2.28	1.56	1.51	1.50	1.42
512	82	2.59	1.69	1.58	1.61	1.49
1024	63	2.88	1.71	1.59	1.66	1.50
		DL^r_{rep}: total replication ratio				
64	117	1.11	1.18	1.16	1.18	1.16
128	108	1.15	1.24	1.21	1.24	1.22
256	94	1.18	1.29	1.26	1.29	1.27
512	82	1.21	1.34	1.29	1.35	1.31
1024	63	1.18	1.33	1.26	1.33	1.28
		CL^r_{max}: maximum computational load ratio				
64	117	1.04	1.03	1.04	1.03	1.04
128	108	1.04	1.03	1.04	1.03	1.04
256	94	1.05	1.03	1.05	1.04	1.05
512	82	1.06	1.04	1.05	1.05	1.06
1024	63	1.06	1.03	1.05	1.04	1.06

As seen in Table 2, the proposed models considerably increase the total amount of data replication compared to the baseline model. This is expected since two-constraint formulation limits the search space during the partitioning. For example, on $K = 1024$ processors, $\mathcal{H}_{Base}$ incurs only 18% replication, whereas $\mathcal{H}_{DV_{RB}}$ and $\mathcal{H}_{IW_{RB}}$ incur 26% and 28% replication, respectively. On the other hand, this increase is not important since the proposed models significantly reduce the load of the maximally loaded processor via much better data load balancing. The gap between the DL^r_{max} and DL^r_{rep} metrics, which shows how close the model approaches to the lower bound, is smaller for both proposed models compared to the baseline model. For example, on $K = 1024$ processors, $\mathcal{H}_{Base}$ achieves average DL^r_{max} value of 144% (2.88 versus 1.18) above the lower bound determined by the DL^r_{rep} value, whereas $\mathcal{H}_{DV_{RB}}$ and

$\mathcal{H}_{IW_{RB}}$ respectively achieve DL^r_{max} values of only 26% (1.59 versus 1.26) and 17% (1.50 versus 1.28) above the lower bounds.

As also seen in the Table 2, the use of the RB framework leads to considerable performance improvement in both proposed models and this performance improvement increases with increasing number of processors as expected. For example, performance improvement of $\mathcal{H}_{IW_{RB}}$ over $\mathcal{H}_{IW_{Dr}}$ in the DL^r_{max} metric is 3.1%, 4.3%, 5.3%, 7.5%, and 9.6% respectively on $K \in 64, 128, 256, 512, 1024\}$ processors.

TABLE 3
Average performance comparison for the $C = AB$ SpGEMM partitioning instances

K	number of instances	$\mathcal{H}_{Base}$	$\mathcal{H}_{DV_{Dr}}$	$\mathcal{H}_{DV_{RB}}$	$\mathcal{H}_{IW_{Dr}}$	$\mathcal{H}_{IW_{RB}}$
		DL^r_{max}: maximum data load ratio				
64	69	1.19	1.12	1.10	1.12	1.09
128	64	1.23	1.16	1.13	1.15	1.11
256	54	1.25	1.19	1.14	1.19	1.13
512	45	1.28	1.24	1.17	1.22	1.15
1024	28	1.30	1.28	1.19	1.26	1.17
		DL^r_{rep}: total replication ratio				
64	69	1.05	1.07	1.06	1.07	1.06
128	64	1.06	1.09	1.07	1.09	1.07
256	54	1.07	1.11	1.08	1.11	1.09
512	45	1.09	1.13	1.10	1.13	1.10
1024	28	1.09	1.14	1.10	1.14	1.11
		CL^r_{max}: maximum computational load ratio				
64	69	1.03	1.02	1.02	1.02	1.03
128	64	1.03	1.02	1.03	1.02	1.03
256	54	1.03	1.02	1.02	1.02	1.03
512	45	1.03	1.02	1.02	1.02	1.03
1024	28	1.03	1.02	1.03	1.02	1.03

Table 3 displays average performance comparison for the $C = AB$ type of SpGEMM partitioning instances. As seen in the table, DL^r_{max} values for this dataset are much less than those for mesh partitioning instances, which is because of the considerably less data size irregularity of $C = AB$ type of SpGEMM partitioning instances compared to that of mesh partitioning instances. For example, on $K = 1024$ processors, the DL^r_{max} values attained by the different models vary between 1.30 and 1.17 on the $C = AB$ type of SpGEMM partitioning instances, whereas those values for the mesh partitioning instances vary between 2.88 and 1.50. As seen in the table, the proposed models perform about 10% better compared to the baseline model in the DL^r_{max} metric. Similar to the mesh instances, use of the RB framework increases the performance of the proposed model, as expected. For example, on 1024 processors, $\mathcal{H}_{DV_{RB}}$ performs 7.0% better than the $\mathcal{H}_{DV_{Dr}}$ model, whereas $\mathcal{H}_{IW_{RB}}$ performs 7.1% better than the $\mathcal{H}_{IW_{Dr}}$ model. On the other hand, in the DL^r_{rep} metric the proposed models do not increase the total replication considerably, in contrast to the mesh partitioning instances. Similar to the mesh partitioning instances, CL^r_{max} metric remains almost the same for all instances. Here, obtained improvement rates are significantly less than compared to the mesh partitioning instances. This can be attributed to regularity of the $C = AB$ type of SpGEMM instances.

Table 4
Average performance comparison for the $C = AA$ SpGEMM partitioning instances

K	number of instances	$\mathcal{H}_{Base}$	$\mathcal{H}_{DV_{Dr}}$	$\mathcal{H}_{DV_{RB}}$	$\mathcal{H}_{IW_{Dr}}$	$\mathcal{H}_{IW_{RB}}$
		DL^r_{max}: maximum data load ratio				
64	12	3.81	3.06	2.61	3.09	2.44
128	12	4.85	4.06	3.30	4.20	3.14
256	12	6.50	5.59	4.72	6.02	4.39
512	12	9.58	8.23	7.11	8.41	6.53
1024	11	12.51	10.88	9.07	11.24	8.91
		DL^r_{rep}: total replication ratio				
64	12	1.81	2.12	1.99	2.10	2.02
128	12	2.09	2.56	2.36	2.54	2.42
256	12	2.50	3.21	2.90	3.17	2.98
512	12	3.10	4.15	3.65	4.08	3.81
1024	11	3.86	5.20	4.59	5.12	4.83
		CL^r_{max}: maximum computational load ratio				
64	12	1.03	1.03	1.03	1.03	1.04
128	12	1.03	1.03	1.05	1.03	1.06
256	12	1.04	1.03	1.06	1.04	1.06
512	12	1.05	1.03	1.06	1.05	1.09
1024	11	1.06	1.03	1.05	1.04	1.13

Table 4 displays average performance comparison for the $C = AA$ type of SpGEMM partitioning instances. Much higher irregularity in data size distribution of this dataset incurs much higher DL^r_{max} values as seen in Tables 2 to 4. For example, on $K = 1024$, $\mathcal{H}_{Base}$ obtains DL^r_{max} values of 12.51, 1.30, and 2.88 respectively for $C = AA$, $C = AB$, and mesh partitioning instances. Such partitioning instances with high data size variation incur hard partitioning instances and justify the importance of the target optimization problem.

As seen in Table 4, both proposed models perform much better data load balancing than the baseline model with almost no performance degradation in computational load balance. The computational load balance becomes considerably worse for $\mathcal{H}_{IW_{RB}}$ model only on 512 and 1024 processors. The proposed models $\mathcal{H}_{IW_{RB}}$ and $\mathcal{H}_{DV_{RB}}$ perform 27% and 29% better than $\mathcal{H}_{Base}$ model on 1024 processors.

As seen in Table 4, the total amount of data replication increases considerably compared to the baseline model. For example, on $K = 1024$ processors, $\mathcal{H}_{Base}$ incurs 286% replication, whereas $\mathcal{H}_{DV_{RB}}$ and $\mathcal{H}_{IW_{RB}}$ incur 359% and 383% replication, respectively. The much higher difference in the DL^r_{max} and DL^r_{rep} values for this dataset (for example, 9.07 versus 4.59 for $\mathcal{H}_{DV_{RB}}$ on 1024 processors) compared to the other two datasets is because of the much higher data size variation in this dataset.

As also seen in Table 4, $\mathcal{H}_{IW}$ benefits more from the RB framework compared to $\mathcal{H}_{DV}$. For example, on $K = 1024$ processors, the RB framework increases the performance of $\mathcal{H}_{DV}$ and $\mathcal{H}_{IW}$ by 17% and 21%, respectively. This can be also observed for mesh and $C = AB$ instances. For example, on $K = 1024$ processors, for the mesh dataset, RB framework increases the performance of $\mathcal{H}_{DV}$ and $\mathcal{H}_{IW}$ by 7.0% and 9.6%, respectively. Higher sensitivity of the use of the RB framework on the $\mathcal{H}_{DV}$ model on the $C = AA$ dataset makes $\mathcal{H}_{DV_{Dr}}$ model perform better than $\mathcal{H}_{IW_{Dr}}$, although $\mathcal{H}_{IW_{RB}}$ performs better than $\mathcal{H}_{DV_{RB}}$ on average for each K value. This is because, as we discussed earlier (in subsection 4.3), the $\mathcal{H}_{DV}$ model tries to encode data loads of processors after the RB step depending on the resulting bipartition, whereas the $\mathcal{H}_{IW}$ model estimates data loads of processors before the RB step and then corrects its estimation depending on the resulting bipartition for the following RB level.

TABLE 5
Number of instances for which each model attains the best performance on DL^r_{max}

	mesh dataset			$C\!=\!AB$ dataset			$C\!=\!AA$ dataset		
K	$\mathcal{H}_{Base}$	$\mathcal{H}_{DV}$	$\mathcal{H}_{IW}$	$\mathcal{H}_{Base}$	$\mathcal{H}_{DV}$	$\mathcal{H}_{IW}$	$\mathcal{H}_{Base}$	$\mathcal{H}_{DV}$	$\mathcal{H}_{IW}$
64	2	40	75	5	22	42	1	0	11
128	6	31	71	5	12	47	1	3	8
256	6	22	66	4	11	39	1	3	8
512	6	25	51	5	11	29	1	1	10
1024	3	18	42	5	5	18	0	2	9

The relative performance comparison of the two proposed models $\mathcal{H}_{DV}$ and $\mathcal{H}_{IW}$ is as follows: As seen in Tables 2 to 4, $\mathcal{H}_{IW}$ display slightly better average performance than $\mathcal{H}_{DV}$ for each dataset and for each K value, except for the $C\!=\!AA$ dataset where $\mathcal{H}_{DV_{Dr}}$ performs slightly better than $\mathcal{H}_{IW_{Dr}}$ for each K value on average. Table 5 is presented to show the number of partitioning instances for which each model attains the best performance in the DL^r_{max} metric. As seen in the table, although $\mathcal{H}_{IW}$ attains the highest number of best DL^r_{max} values, $\mathcal{H}_{DV}$ attains considerably high number of best DL^r_{max} values especially on the mesh and the $C = AB$ datasets. So both proposed models should be considered for attaining simultaneous computational and data load balancing depending on the nature of the application and the dataset.

5.4.3. Performance profile comparison. In Figures 7 and 8, we provide performance profiles for each of the five models in terms of the DL^r_{max} and DL^r_{rep} metrics on $K \in \{64, 128, 256, 512, 1024\}$ processors. In each figure, each column corresponds to a different dataset, whereas each row corresponds to a different K value. We do not include performance profiles for the CL^r_{max} metric, since each of the five models perform similarly as discussed earlier. That is, performance profiles for the CL^r_{max} metric are almost on top of each other.

Performance profiles [28] are widely used in comparing multiple models over a large collection of test cases. In a different way from the average performance comparison given above, in the performance profiles we compare five models according to the best performing model for each partitioning instance on one of the metrics and measure in what fraction of the test cases a model performs within a factor of the best observed performance. A point (x,y) in a curve means that the respective model is within an x factor of the best result in a fraction y of the dataset. For example, consider the point $(x = 1.05,\ y = 0.70)$ on the performance curve for $\mathcal{H}_{DV_{RB}}$ for the mesh dataset on $K = 1024$ processors for the DL^r_{max} metric. This point means that for 70% of the partitioning instances $\mathcal{H}_{DV_{RB}}$ attains DL^r_{max} values at most 5 percent larger than the best DL^r_{max} values achieved. Thus, a model that is closer to the top-left corner is better.

As seen in Figures 7 and 8, the performance profiles corroborate the above discussion about the relative performance of the models. For example, comparison of profile curves for $\mathcal{H}_{DV_{RB}}$ and $\mathcal{H}_{DV_{Dr}}$ as well as for $\mathcal{H}_{IW_{RB}}$ and $\mathcal{H}_{IW_{Dr}}$ on the DL^r_{max} metric show that the performance improvement obtained from the use of the RB framework increases with increasing number of processors for the mesh and the $C=AB$ datasets, whereas this improvement remains almost the same for the $C=AA$ dataset. Furthermore, comparison of profile curves for $\mathcal{H}_{DV_{Dr}}$ and $\mathcal{H}_{IW_{Dr}}$ on DL^r_{max} metric shows that the performance obtained by the $\mathcal{H}_{DV_{Dr}}$ becomes superior than the performance obtained by $\mathcal{H}_{IW_{Dr}}$ for about more than 20% of the $C=AA$ dataset within a factor of approximately 1.20.

As seen in Figure 7, on $K=1024$ processors, $\mathcal{H}_{IW_{RB}}$ achieves the best solutions for DL^r_{max} metric for the 60% of all datasets, whereas $\mathcal{H}_{DV_{RB}}$ attains solutions within factors of 1.04, 1.02, and 1.08 for the mesh, $C=AB$ and $C=AA$ partitioning instances, respectively. As seen in Figure 8, the $\mathcal{H}_{Base}$ performs the best on the DL^r_{rep} metric, whereas $\mathcal{H}_{IW_{RB}}$ and $\mathcal{H}_{DV_{RB}}$ follows. Furthermore, this difference increases with increasing irregularity of the dataset.

6. Related work. Chevalier et al. [22] consider memory constraints during the task partitioning with the objective of balancing computational load of processors. They propose a multilevel bipartite graph partitioning algorithm as follows: In the coarsening phase, computational vertices match with other computational vertices according to their data element share. Data elements are matched with other data elements similar to the identical net detection in HP. They perform initial partitioning by greedily assigning sorted tasks to parts/processors considering the memory capacity. In the refinement phase, tasks are moved between parts/processors with the objective of decreasing the computational load. Note that this work does not balances the data load rather abides by the memory constraint.

Angel et al. [4, 5] also consider memory constraints while minimizing the computational load of the maximally loaded processor. In [4], a dynamic programming based Fixed-Parameter Tractable algorithm with respect to the path-width of the neighborhood graph is utilized. Note that this is an approximation since the problem is NP-hard. In [5], they focus on the case when the neighborhood graph has bounded tree-width. In that way tree decomposition of the graph and its traversal in a specific way which may be useful on its own allows to find a solution within a factor of epsilon.

Tzovas et al. [40] target heterogeneous distributed systems for which sparse matrices or graphs will be distributed. Also in this work, memory capacity is a constraint where the objective is maximizing utilization of each processing unit with minimum communication cost. For this purpose, they propose a two phase algorithm: In the first phase, they greedily compute block size for each processing unit. In the second phase, they feed these sizes to variety a of existing partitioning tools.

7. Conclusion. In high performance computing applications, computational load balancing is well studied in the literature, however data load balancing has become a concern recently. In that sense, to our knowledge, this is a pioneer work in which we simultaneously balance processors' computational and data loads in distributed-memory setting. We proposed two different hypergraph partitioning based models both utilizing a two-constraint formulation for load balancing. The first constraint encodes balancing the computational load, whereas the second constraint encodes balancing the data load. The partitioning objective encodes minimizing the total amount of data replication. We utilized the well known recursive bipartitioning framework for increasing the accuracy of the both proposed models. Instead of developing a new partitioner from scratch both proposed models can easily be implemented by invoking any HP tool that supports multi constraint partitioning as a two-way partitioner at each RB step. Extensive experiments show that both pro-

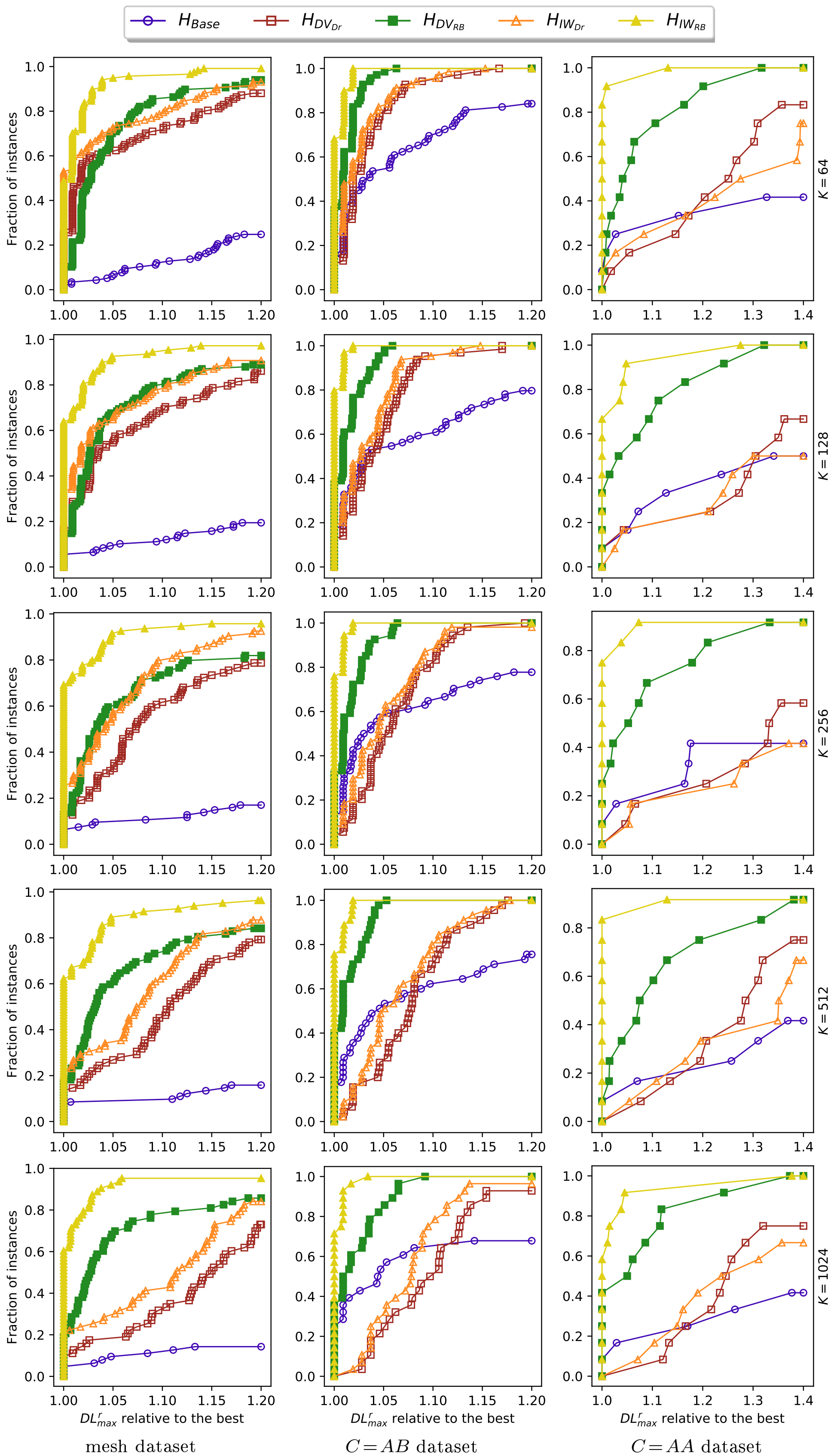


FIG. 7. *Performance profiles for the DL^r_{max} metric on $K \in \{64, 128, 256, 512, 1024\}$*

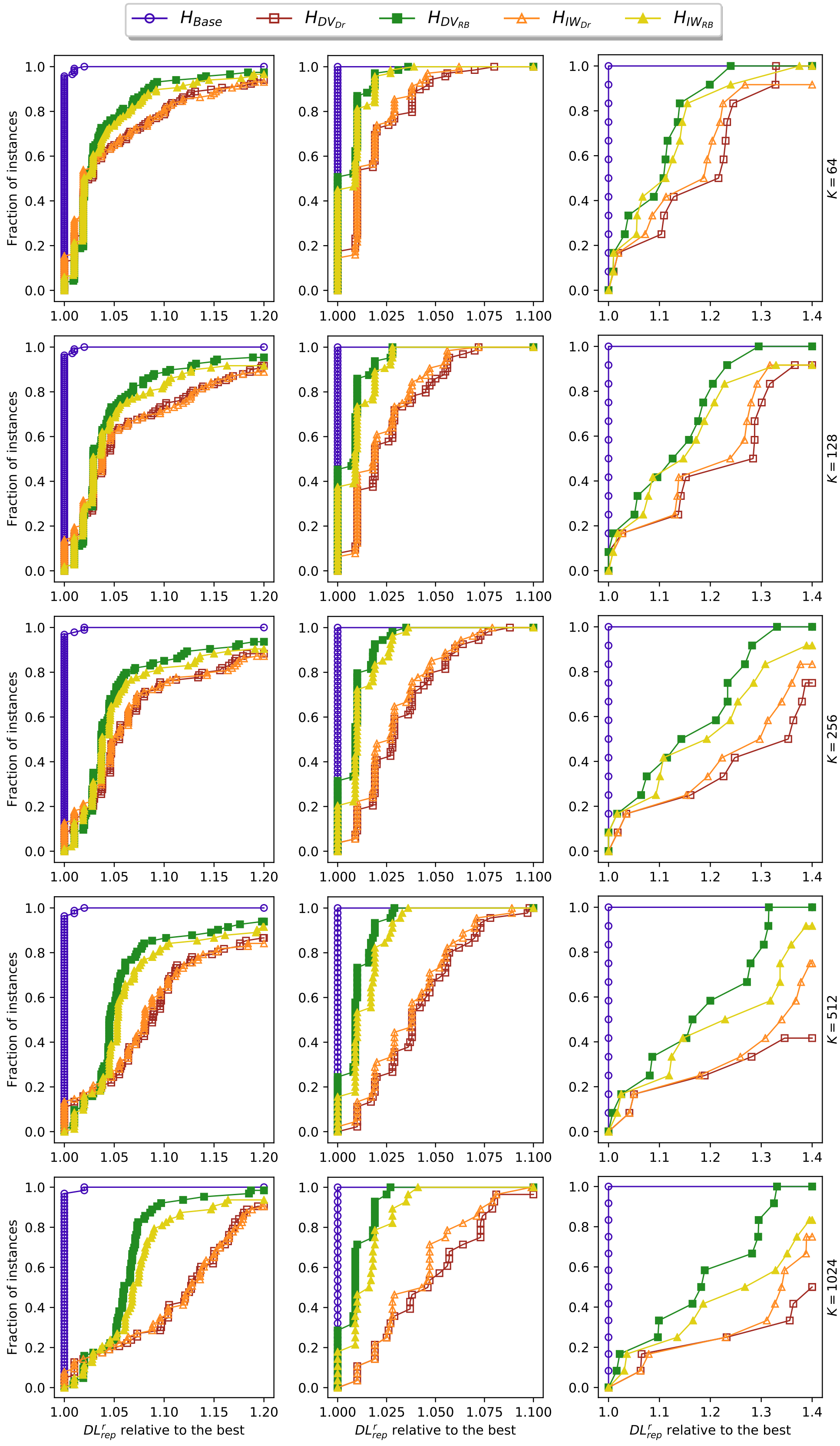


Fig. 8. *Performance profiles for the DL^r_{rep} metric on $K \in \{64, 128, 256, 512, 1024\}$*

posed models perform significantly better than a baseline model. We achieve up to 49 percent better performance on 1024 processors.

REFERENCES

[1] S. Acer, O. Selvitopi, and C. Aykanat, *Improving performance of sparse matrix dense matrix multiplication on large-scale parallel systems*, Parallel Computing, 59 (2016), pp. 71 – 96. Theory and Practice of Irregular Applications.

[2] S. Acer, O. Selvitopi, and C. Aykanat, *Optimizing nonzero-based sparse matrix partitioning models via reducing latency*, Journal of Parallel and Distributed Computing, 122 (2018), pp. 145 – 158.

[3] K. Akbudak, O. Selvitopi, and C. Aykanat, *Partitioning models for scaling parallel sparse matrix-matrix multiplication*, ACM Trans. Parallel Comput., 4 (2018).

[4] E. Angel, C. Chevalier, F. Ledoux, S. Morais, and D. Regnault, *FPT approximation algorithm for scheduling with memory constraints*, in Euro-Par 2016: Parallel Processing, Cham, 2016, Springer International Publishing, pp. 196–208.

[5] E. Angel, S. Morais, and D. Regnault, *A bi-criteria FPTAS for scheduling with memory constraints on graph with bounded tree-width*, arXiv preprint arXiv:2202.08704, (2022).

[6] C. Aykanat, B. B. Cambazoglu, F. Findik, and T. Kurc, *Adaptive decomposition and remapping algorithms for object-space-parallel direct volume rendering of unstructured grids*, Journal of Parallel and Distributed Computing, 67 (2007), pp. 77–99.

[7] R. Barat, *Load Balancing of Multi-physics Simulation by Multi-criteria Graph Partitioning*, PhD thesis, Thèse de doctorat dirigée par Pellegrini, François et Chevalier, Cédric Informatique Bordeaux 2017, 2017. 2017BORD0961.

[8] N. Bell, S. Dalton, and L. N. Olson, *Exposing fine-grained parallelism in algebraic multigrid methods*, SIAM Journal on Scientific Computing, 34 (2012), pp. C123–C152.

[9] R. H. Bisseling, *Parallel Scientific Computation: A Structured Approach using BSP (Second edition)*, Oxford University Press, USA, 2020.

[10] R. H. Bisseling and I. Flesch, *Mondriaan sparse matrix partitioning for attacking cryptosystems by a parallel block Lanczos algorithm – a case study*, Parallel Computing, 32 (2006), pp. 551 – 567. Algorithmic Skeletons.

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

[12] E. G. Boman and M. M. Wolf, *A nested dissection partitioning method for parallel sparse matrix-vector multiplication*, in 2013 IEEE High Performance Extreme Computing Conference (HPEC), 2013, pp. 1–6.

[13] B. B. Cambazoglu and C. Aykanat, *Hypergraph-partitioning-based remapping models for image-space-parallel direct volume rendering of unstructured grids*, IEEE Transactions on Parallel and Distributed Systems, 18 (2007), pp. 3–16.

[14] Ü. V. Çatalyürek and C. Aykanat, *Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication*, IEEE Transactions on Parallel and Distributed Systems, 10 (1999), pp. 673–693.

[15] Ü. V. Çatalyürek and C. Aykanat, *A hypergraph-partitioning approach for coarse-grain decomposition*, in Proceedings of the 2001 ACM/IEEE Conference on Supercomputing, SC '01, New York, NY, USA, 2001, Association for Computing Machinery, p. 28.

[16] Ü. V. Çatalyürek and C. Aykanat, *PaToH (partitioning tool for hypergraphs)*, in Encyclopedia of Parallel Computing, Springer, 2011, pp. 1479–1487.

[17] Ü. V. Çatalyürek, C. Aykanat, and B. Uçar, *On two-dimensional sparse matrix partitioning: Models, methods, and a recipe*, SIAM Journal on Scientific Computing, 32 (2010), pp. 656–683.

[18] Ü. V. Çatalyürek, E. G. Boman, K. D. Devine, D. Bozdag, R. Heaphy, and L. A. Riesen, *Hypergraph-based dynamic load balancing for adaptive scientific computations*, in 2007 IEEE International Parallel and Distributed Processing Symposium, 2007, pp. 1–11.

[19] Ü. V. Çatalyürek, E. G. Boman, K. D. Devine, D. Bozdağ, R. T. Heaphy, and L. A. Riesen, *A repartitioning hypergraph model for dynamic load balancing*, Journal of Parallel and Distributed Computing, 69 (2009), pp. 711–724.

[20] Ü. V. Çatalyürek, M. Deveci, K. Kaya, and B. Uçar, *UMPa: A multi-objective, multi-level partitioner for communication minimization*, Graph Partitioning and Graph Clustering, 588 (2013), p. 53.

[21] A. Cevahir, C. Aykanat, A. Turk, and B. B. Cambazoglu, *Site-based partitioning and repartitioning techniques for parallel pagerank computation*, IEEE Transactions on Parallel

and Distributed Systems, 22 (2011), pp. 786–802.

[22] C. Chevalier, F. Ledoux, and S. Morais, *A Multilevel Mesh Partitioning Algorithm Driven by Memory Constraints*, pp. 85–95.

[23] T. A. Davis and Y. Hu, *The University of Florida sparse matrix collection*, ACM Trans. Math. Softw., 38 (2011).

[24] M. Deveci, K. Kaya, B. Uçar, and Ü. V. Çatalyürek, *Hypergraph partitioning for multiple communication cost metrics: Model and methods*, Journal of Parallel and Distributed Computing, 77 (2015), pp. 69 – 83.

[25] K. Devine, B. Hendrickson, E. Boman, M. St. John, and C. Vaughan, *Design of dynamic load-balancing tools for parallel applications*, in Proceedings of the 14th International Conference on Supercomputing, ICS '00, New York, NY, USA, 2000, Association for Computing Machinery, p. 110–118.

[26] K. D. Devine, E. G. Boman, R. T. Heaphy, B. A. Hendrickson, J. D. Teresco, J. Faik, J. E. Flaherty, and L. G. Gervasio, *New challenges in dynamic load balancing*, Applied Numerical Mathematics, 52 (2005), pp. 133–152. ADAPT '03: Conference on Adaptive Methods for Partial Differential Equations and Large-Scale Computation.

[27] K. D. Devine, E. G. Boman, and G. Karypis, *Partitioning and load balancing for emerging parallel applications and architectures*, in Parallel Processing for Scientific Computing, Society for Industrial and Applied Mathematics, 2006, pp. 99–126.

[28] E. D. Dolan and J. J. Moré, *Benchmarking optimization software with performance profiles*, Mathematical programming, 91 (2002), pp. 201–213.

[29] J. Flaherty, R. Loy, P. Scully, M. Shephard, B. Szymanski, J. Teresco, and L. Ziantz, *Load balancing and communication optimization for parallel adaptive finite element methods*, in Proceedings 17th International Conference of the Chilean Computer Science Society, 1997, pp. 246–255.

[30] S. jun Liang, J. Cheng, and J. wei Zhang, *Research on data load balancing technology of massive storage system for wearable devices*, Digital Communications and Networks, (2020).

[31] G. Linden, B. Smith, and J. York, *Amazon.com recommendations: item-to-item collaborative filtering*, IEEE Internet Computing, 7 (2003), pp. 76–80.

[32] K. Liu, G. Xu, et al., *An improved hadoop data load balancing algorithm*, Journal of Networks, 8 (2013), p. 2816.

[33] H. Meyerhenke, *Dynamic load balancing for parallel numerical simulations based on repartitioning with disturbed diffusion*, in 2009 15th International Conference on Parallel and Distributed Systems, 2009, pp. 150–157.

[34] S. Morais, *Etude et obtention d'heuristiques et d'algorithmes exacts et approchés pour un problème de partitionnement de maillage sous contraintes mémoire*, theses, Université Paris Saclay, Nov. 2016.

[35] N. Patel and S. Chauhan, *A survey on load balancing and scheduling in cloud computing*, Int. Journal for Innovative Research in Science and Technology, 1 (2015), pp. 185–189.

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

[37] A. Pinar and B. Hendrickson, *Improving load balance with flexibly assignable tasks*, IEEE Transactions on Parallel and Distributed Systems, 16 (2005), pp. 956–965.

[38] O. Selvitopi, S. Acer, and C. Aykanat, *A recursive hypergraph bipartitioning framework for reducing bandwidth and latency costs simultaneously*, IEEE Transactions on Parallel and Distributed Systems, 28 (2017), pp. 345–358.

[39] O. Selvitopi and C. Aykanat, *Reducing latency cost in 2D sparse matrix partitioning models*, Parallel Computing, 57 (2016), pp. 1 – 24.

[40] C. Tzovas, M. Predari, and H. Meyerhenke, *Distributing sparse matrix/graph applications in heterogeneous clusters - an experimental study*, in 2020 IEEE 27th International Conference on High Performance Computing, Data, and Analytics (HiPC), 2020, pp. 72–81.

[41] B. Uçar and C. Aykanat, *Minimizing communication cost in fine-grain partitioning of sparse matrices*, in International Symposium on Computer and Information Sciences, Springer, 2003, pp. 926–933.

[42] B. Uçar and C. Aykanat, *Encapsulating multiple communication-cost metrics in partitioning sparse rectangular matrices for parallel matrix-vector multiplies*, SIAM Journal on Scientific Computing, 25 (2004), pp. 1837–1859.

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