J. Parallel Distrib. Comput. 72 (2012) 547–563

Contents lists available at SciVerse ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Replicated partitioning for undirected hypergraphs☆

R. Oguz Selvitopi, Ata Turk, Cevdet Aykanat *

Department of Computer Engineering, Bilkent University, 06800 Ankara, Turkey

ARTICLE INFO

Article history:
Received 23 February 2011
Received in revised form
27 September 2011
Accepted 12 January 2012
Available online 23 January 2012

Keywords:
Hypergraph partitioning
Recursive bipartitioning
Undirected hypergraphs
Replication
Iterative improvement heuristic

ABSTRACT

Hypergraph partitioning (HP) and replication are diverse but powerful tools that are traditionally applied separately to minimize the costs of parallel and sequential systems that access related data or process related tasks. When combined together, these two techniques have the potential of achieving significant improvements in performance of many applications. In this study, we provide an approach involving a tool that simultaneously performs replication and partitioning of the vertices of an undirected hypergraph whose vertices represent data and nets represent task dependencies among these data. In this approach, we propose an iterative-improvement-based replicated bipartitioning heuristic, which is capable of move, replication, and unreplication of vertices. In order to utilize our replicated bipartitioning heuristic in a recursive bipartitioning framework, we also propose appropriate cut-net removal, cut-net splitting, and pin selection algorithms to correctly encapsulate the two most commonly used cutsize metrics. We embed our replicated bipartitioning scheme into the state-of-the-art multilevel HP tool PaToH to provide an effective and efficient replicated HP tool, rpPaToH. The performance of the techniques proposed and the tools developed is tested over the undirected hypergraphs that model the communication costs of parallel query processing in information retrieval systems. Our experimental analysis indicates that the proposed technique provides significant improvements in the quality of the partitions, especially under low replication ratios.

1. Introduction

Models and methods based on hypergraph partitioning (HP) have been successfully used for different objectives in a wide range of areas such as parallel scientific computing [4,11,15,44], very large scale integration (VLSI) circuit layout design [1,32], parallel information retrieval (IR) [8], parallel volume rendering [9], and database systems [12,13,40].

A hypergraph is a generalization of a graph where hyperedges (nets) connect one or more vertices (cells). The HP problem is defined as the task of dividing the vertex set of a given hypergraph into disjoint subsets such that the cost (cutsize) is minimized while a certain balance constraint on the part weights is satisfied. The cutsize is generally a function of the nets that connect more than one part.

Hypergraphs can be used to represent different types of relation in a wide range of problems which can broadly be categorized into two as directed and undirected relations. Depending on the category of the relation, directed or undirected hypergraphs are used in the modeling. In undirected hypergraphs, a net is used to model an equally shared relation among the tasks/data represented by the vertices it connects. In directed hypergraphs, a net is used to model an input–output relation among the tasks/data represented by the vertices it connects.

We use the terms directional and undirectional HP models for indicating models based on partitioning of directed and undirected hypergraphs, respectively. We should note here that almost all of the state-of-the-art HP tools [2,14,26,43,45] are designed to partition undirected hypergraphs. Hence, some special techniques such as consistency condition [11] and the elementary hypergraph model [44] are utilized to model some types of directed relations correctly via undirectional HP models.

The schemes that combine vertex replication with HP models have only been studied for directional HP models in the context of VLSI circuit layout design. In these HP models, since the vertices generally model the gates or logic devices, replication corresponds to duplicating the same gate or logic device in multiple networks of a partitioned logic network. In this way, the number of connections between networks and the wiring density can be reduced at the expense of implementing the same logic in multiple networks.

In directional HP models, vertex replication may cause an increase in the cutsize, and it generally requires further replication of other vertices and nets. However, in undirectional HP models,

☆ This work is partially supported by the Scientific and Technological Research Council of Turkey (TÜBİTAK) under project EEEAG-109E019.

* Corresponding author.
E-mail addresses: reha@cs.bilkent.edu.tr (R. Oguz Selvitopi), atat@cs.bilkent.edu.tr (A. Turk), aykanat@cs.bilkent.edu.tr (C. Aykanat).

0743-7315/$ – see front matter © 2012 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2012.01.004

since an input–output relation does not exist between the vertices connected by a net, replication does not have such an effect. This forms the basic difference between vertex replication in directional and undirectional HP models. To the best of our knowledge, there are no studies in the literature addressing vertex replication schemes for undirectional HP models. In this study, we try to fill this gap. Note that, due to the above-mentioned fundamental difference in vertex replication, the techniques we present here are not directly applicable to directional HP models. Thus, replication in undirectional HP models requires specific techniques and tools tailored for this purpose.

1.1. Related work in directional HP models

Even though we do not address applications in the VLSI domain, we discuss replication schemes in this area, since, to our knowledge, VLSI circuit layout design is the only area where HP is applied together with replication, albeit in a directional partitioning framework.

Replication schemes in VLSI circuit layout design and partitioning arise in the form of gate replication to reduce pin counts and the interconnection cost of the partitioned circuits. These schemes can be categorized into two as one-phase schemes and two-phase schemes with respect to when the partitioning and replication are performed. In the one-phase approach, partitioning and replication are performed simultaneously, whereas in the two-phase approach replication is performed after obtaining a partition. In the one-phase approach, generally, extended versions of the Fiduccia–Mattheyses (FM) [17] heuristic are utilized [30,31]. In the two-phase approach, after obtaining a partition, linear programming or flow-network [22,33] formulations are used to achieve replication, and often, if needed, an extended FM heuristic is applied as the last step to find a feasible solution. Since this study focuses on performing replication and partitioning simultaneously, we briefly summarize the existing work on FM heuristics for directional graph/hypergraph partitioning with replication. In [30], an extended version of the FM algorithm for directional HP models is proposed to perform replication in two-way partitioned networks by introducing new definitions for cell/net states and cell gains. The authors of [31] introduce an extended version of the FM algorithm to achieve partitioning and replication, and they propose a new gain definition and objective function for this extended version. In [33], the authors use a modified FM algorithm applied over a replication graph which they obtain by a linear programming formulation. A detailed discussion and comparison of replication techniques in circuit partitioning can be found in [16].

1.2. Application

In order to show the validity of the algorithms proposed in our paper, we investigate undirectional HP models proposed for index partitioning of parallel IR systems [8,28], where replication is beneficial and commonly used [37]. Although we address the HP models used in parallel IR, our replication scheme can be used for any domain in which the underlying problem can be modeled as an undirected hypergraph.

In parallel IR systems, the index is partitioned across several machines [7,23,36,38,41], typically in a document-based or term-based fashion, in order to process very large text collections. In [42], it is remarked that replication is necessary for improving query throughput. The authors of [35] propose a bin-packing-based greedy algorithm that utilizes query logs to distribute terms to index servers. In their experiments, they replicate a small amount of most frequent terms and discover that replication is a powerful tool in reducing the average number of per-query servers, even under low replication ratios. In the distributed IR system of Google, the entire system is replicated [5]. A selective replication scheme that replicates inverted lists of high workload terms to improve load balancing in a pipelined and term-distributed IR system is investigated in [37].

In the HP models utilized for term-based distribution of inverted indices [28], the vertex v_i represents the term t_i and the task of retrieving its inverted list. The net n_j represents the query q_j and connects the subsets of vertices that represent the terms requested by that query. In this HP model, the nets have unit costs due to the infinite result cache capacity assumption.[1] The weight of a vertex is set equal to either the number of postings in the inverted list of the term represented by that vertex [8] or the multiplication of term popularity and the corresponding posting list size [37]. The balance constraint in the former vertex weighting scheme corresponds to maintaining storage balance, whereas the balance constraint in the latter vertex weighting scheme corresponds to maintaining computational workload balance. The partitioning objective of minimizing the cutsize corresponds to minimizing the communication volume during parallel query processing.

We introduce Fig. 1 to illustrate the relationship between the target application and undirectional HP models. Fig. 1(a) shows a sample term collection $\mathcal{T}$ that contains ten terms together with a query log $\mathcal{Q}$ that contains six queries. Fig. 1(b) shows the undirectional hypergraph model for this sample inverted index. As seen in Fig. 1(b), net n_1 connects vertices v_1, v_2, and v_3, since query q_1 requests the terms t_1, t_2, and t_3. Fig. 1(b) also shows a four-way partition of this hypergraph. Fig. 1(c) shows the distribution of the sample inverted index among four index servers ($IS_1, \ldots, IS_4$) that is induced by this four-way partition. For example, the index server IS_2 stores the terms t_3, t_4, and t_5 and their inverted lists since part $\mathcal{V}_2$ of the partition consists of the vertices v_3, v_4, and v_5.

The correspondence between vertex replication and the mentioned HP model is as follows. A net in this HP model represents the undirectional shared relation among the respective retrieval tasks that can be performed concurrently and independently on the inverted lists represented by the vertices connected by that net. Thus, vertex replication corresponds to replicating inverted lists of terms for further minimization of the communication volume. For a given query, the task associated with each data is only performed by one of the processors owning the replicas of that data. Thus, the proposed scheme incurs redundant storage (data replication) but does not incur redundant computation.

1.3. Contributions

There are five main contributions of this study. (1) The differences between vertex replication in directional and undirectional HP models are explained (Section 3). (2) A vertex replication scheme for undirectional HP models is proposed (Section 4). This replication approach is based on an iterative-improvement heuristic, and it achieves replication during partitioning. For this purpose, the FM heuristic is extended to support replication and unreplication of vertices in addition to vertex moves. This extended heuristic is called rFM, and it operates on a given two-way partition (bipartition) by introducing new gain definitions and vertex states. (3) In order to utilize rFM in a recursive bipartitioning (RB) framework, appropriate cut-net removal, cut-net splitting, and pin selection algorithms are proposed to correctly encapsulate the two most commonly used cutsize metrics (Sections 5 and 6). (4) The proposed vertex replication and bipartitioning scheme is integrated into the state-of-the-art multilevel HP tool PaToH [2] that uses

[1] This assumption simply states that each query is processed only once and its results are stored in the result cache. Further requests for the same query are responded from this result cache [10].

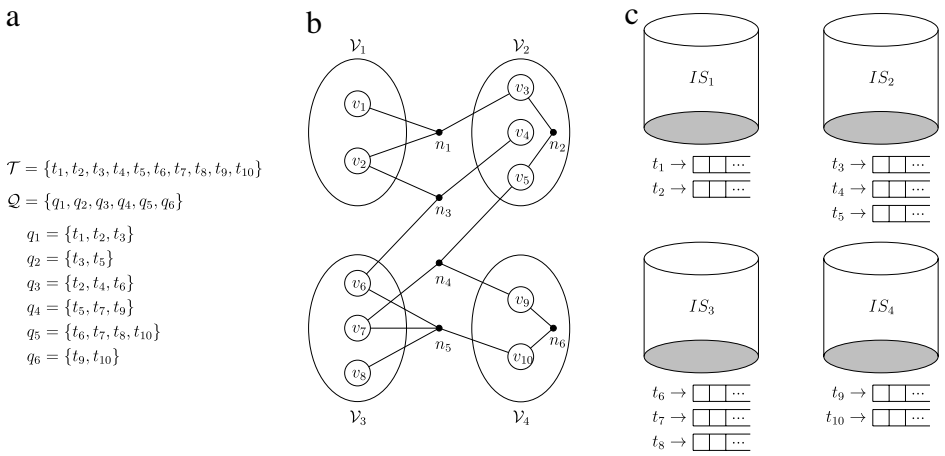

Fig. 1. The relation between an inverted index distribution and undirectional HP models. (a) A sample inverted index, (b) the corresponding hypergraph model, and (c) a four-way term-based inverted index distribution.

the RB paradigm to provide a replicated HP tool, `rpPaToH`. Specifically, the uncoarsening phase of the multilevel framework is modified by using rFM as a replicated partitioning and refinement tool. At each level of the uncoarsening phase, the rFM algorithm is run and the multilevel scheme is extended to support replicated vertices. (5) Detailed experimental analyses are performed over the hypergraph model of the sample application (Section 1.2) using synthetic and realistic datasets. The results obtained indicate that `rpPaToH` performs significantly better than a successful partitioning and replication scheme [28] for this application domain.

The rest of the paper is organized as follows. Section 2 gives the necessary background. Section 3 explains the differences between replication in directional and undirectional HP models. Section 4 describes the details of the rFM heuristic. Section 5 presents the proposed cut-net removal, cut-net splitting, and replication distribution schemes. Section 6 addresses the pin selection issue after obtaining a K-way partition. Section 7 discusses the results of the experiments that were carried out. Finally, Section 8 concludes.

2. Background and problem definition

2.1. Definitions and hypergraph partitioning problem

A hypergraph $\mathcal{H} = (V, N)$ is defined as a set of vertices $\mathcal{V}$ and a set of nets $\mathcal{N}$. Each net $n_j \in \mathcal{N}$ connects a subset of vertices. The set of vertices connected by net n_j is denoted as Vertices(n_j). The set of nets that connect vertex v_i is denoted as Nets (v_i). The vertices v_i and v_j are said to be neighbors if they are connected by at least one common net, i.e., Nets (v_i) $\cap$ Nets (v_j) $\neq \emptyset$. An (n_j, v_i) tuple denotes a pin of n_j where $v_i \in$ Vertices(n_j). The degree of a net n_j is equal to the number of vertices it connects, |Vertices(n_j)|. The total number of pins $P = \sum_{n_j \in \mathcal{N}} |\text{Vertices } (n_j)|$ denotes the size of a given hypergraph $\mathcal{H}$. A weight value $w(v_i)$ is associated with each vertex v_i, and a cost value $c(n_j)$ is associated with each net n_j. The cost function for a net easily extends to a subset of nets $\mathcal{M} \subseteq \mathcal{N}$, i.e., $c(\mathcal{M}) = \sum_{n_j \in \mathcal{M}} c(n_j)$.

$\Pi = \{\mathcal{V}_1, \ldots, \mathcal{V}_K\}$ is a K-way partition of $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ if each part $\mathcal{V}_k$ is a nonempty subset of $\mathcal{V}$, the parts are pairwise disjoint, and the union of K parts is equal to $\mathcal{V}$. The weight $W(\mathcal{V}_k)$ of a part $\mathcal{V}_k$ is the sum of the weights of the vertices in that part, i.e., $W(\mathcal{V}_k) = \sum_{v_i \in \mathcal{V}_k} w(v_i)$. A partition Π is said to be balanced if each part $\mathcal{V}_k \in \Pi$ satisfies the *balance constraint*:

$$W(\mathcal{V}_k) \leq (1 + \epsilon) W_{\text{avg}} \quad \text{for } k = 1, \ldots, K, \tag{1}$$

where $W_{\text{avg}} = W(\mathcal{V})/K$ and ϵ is the predetermined maximum imbalance ratio.

In a partition Π, a net is said to *connect* a part if it connects at least one vertex in that part. The *connectivity set* $\Lambda(n_j)$ of a net n_j is defined as the set of parts connected by n_j. The number of parts in the connectivity set of n_j is denoted by $\lambda(n_j) = |\Lambda(n_j)|$. A net is said to be *cut* or *external* if it connects more than one part ($\lambda(n_j) > 1$), and *uncut* or *internal* if it connects only one part ($\lambda(n_j) = 1$). The set of external nets in a partition Π is denoted as $\mathcal{N}_E$. The set of internal nets that connect a vertex v_i is denoted as InternalNets (v_i). Two *cutsize* metrics widely used in the literature to represent the cost of a partition Π are

$$\text{cutsize } (\Pi) = \sum_{n_j \in \mathcal{N}_E} c(n_j), \tag{2}$$

$$\text{cutsize } (\Pi) = \sum_{n_j \in \mathcal{N}_E} (\lambda(n_j) - 1) c(n_j). \tag{3}$$

The cost definitions in Eqs. (2) and (3) are called the *cut-net* metric and the *connectivity* metric, respectively. For example, the cut-net and connectivity metrics model the minimization of the communication volume in parallel sparse matrix vector multiplication utilizing collective and point-to-point communication schemes, respectively [11,44].

Given a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$, *hypergraph partitioning* can be defined as finding a K-way partition $\Pi = \{\mathcal{V}_1, \ldots, \mathcal{V}_K\}$ that minimizes the cutsize (Eqs. (2) or (3)) while maintaining the balance constraint (Eq. (1)). This problem is known to be NP-hard [32].

2.2. Iterative improvement heuristics for two-way HP

FM-based schemes [1,17] are widely used iterative-improvement heuristics to solve the HP problem. FM-based heuristics improve the cutsize of a bipartition by moving vertices from one part to the other. The gain of a vertex in these heuristics is generally defined as the reduction in the cutsize if that vertex were to be moved to its complementary part in a bipartition. FM heuristics can perform multiple *passes* over all vertices until the improvement in the cutsize drops below a certain threshold.

2.3. Recursive bipartitioning and multilevel frameworks

RB is the most commonly used method for obtaining a K-way partition of a hypergraph, although there are other methods based on direct K-way partitioning [3,27]. In the RB scheme, first a bipartition of the initial hypergraph is obtained, and then this

bipartition is decoded to construct two subhypergraphs using the cut-net removal and cut-net splitting techniques [2] to capture the cut-net and connectivity cutsize metrics, respectively. Then these two subhypergraphs are further bipartitioned in a recursive manner. This procedure continues until desired number of parts is reached (in $\log K$ recursion levels for K parts).

FM-based heuristics perform poorly on hypergraphs with high net degrees [3,27] and small vertex degrees [19]. To alleviate these problems, multilevel algorithms have been proposed [6,20] and applied to the HP problem, leading to successful HP tools such as PaToH [2], hMeTiS [26], Mondriaan [45], Zoltan [14], and ParKWay [43].

Multilevel methodology consists of coarsening, initial partitioning, and uncoarsening phases. In the coarsening phase, the original hypergraph is coarsened into a smaller hypergraph by a sequence of coarsening levels, where, in each level, various matching and clustering algorithms are used to form super-vertices from highly coherent vertices. Coherent vertices are the vertices that share high number of nets. In the initial partitioning phase, a bipartition of the coarsest hypergraph is obtained, and this coarsest hypergraph is projected back to the original hypergraph in the uncoarsening phase. At each level of the uncoarsening phase, FM-based or KL-based [29] refinement heuristics are used to improve the quality of the bipartitions.

3. Replication in directional versus undirectional HP models

There are two main differences between vertex replication in directional and undirectional HP models. (i) The replication of a vertex in directional HP models may bring internal nets to the cut and thus can increase the cutsize of a partition, and (ii) vertex replication generally requires further net and pin replication in directional HP models. However, these two cases are not valid for undirectional HP models.

In directed hypergraphs, the nets that connect a vertex v_i are categorized as input and output nets of v_i. In a dual manner, the vertices that are connected by a net n_j are categorized as input and output vertices of n_j. For example, in hypergraph representation of gate-level VLSI circuits for layout design [1] and column-net hypergraph representation of sparse matrices for parallel matrix–vector multiplication [11], nets have single input and multiple output vertices, which correspond to vertices having multiple input and single output nets.

In directional HP models, when an output vertex v_i of an internal net n_j is replicated, n_j becomes cut since any new instance of the replicated vertex v_i' must be fed by n_j on the part it is replicated to. Fig. 2 shows an example of vertex replication in a directed hypergraph. A sample bipartition on this directed hypergraph is illustrated in Fig. 2(a). Initially, the cutsize of the bipartition is one, assuming that the nets have unit costs. As shown in Fig. 2(b), when v_3 is replicated, n_1 and n_2 become cut since v_3 is an output vertex of these internal nets. Since v_3' has to be fed by both of these nets, pins (n_1, v_3') and (n_2, v_3') are generated in Fig. 2(b). Furthermore, when an external net n_j's input vertex v_i is replicated, n_j is generally replicated together with v_i to be able to save n_j from the cut. As shown in Fig. 2(b), when v_3 is replicated, n_3 is also replicated, leading to the addition of a new net n_3' and a new pin (n_3', v_3') in $\mathcal{V}_B$. In this way, we are able to save n_3 from the cut. However, since n_1 and n_2 become cut, the cutsize of the bipartition increases from one to two after the replication.

In contrast, in undirectional HP models, performing replication does not bring internal nets to the cut, and putting additional pins to the new instances of the replicated vertices may not be necessary, since a net represents a shared relation rather than a dependence among the vertices it connects. In other words, we can make a choice among the instances of a replicated vertex for a net in order to decide which one of these instances will represent that replicated vertex. This is done by putting a pin only to a single instance of the replicated vertex for that net. Fig. 3 shows an example of vertex replication in an undirected hypergraph. The initial bipartition is seen in Fig. 3(a), which is the undirected version of the directed hypergraph in Fig. 2(a) and has a cutsize of one. As opposed to replication of v_3 in Fig. 2, replication of v_3 in Fig. 3 does not bring any internal net to the cut, since, as seen in Fig. 3(b), the nets n_1 and n_2 are not required to feed v_3'. Instead, n_1 (or similarly n_2 and n_3) can "choose" to use either v_3 or v_3', since n_1 just needs to select an instance for this replicated vertex. In other words, n_1 has to have just one pin to an instance of the replicated vertex, which is selected to be the pin (n_1, v_3) in this example. We refer to this problem as the pin selection problem and address it in Section 6. After replication of v_3, the cutsize of the bipartition reduces from one to zero.

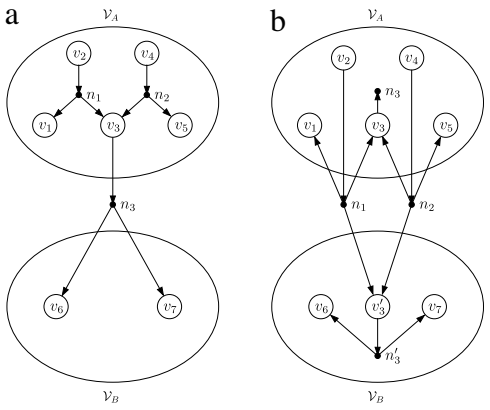

Fig. 2. Replication in a directed hypergraph. (a) Initial bipartition, (b) after replicating v_3.

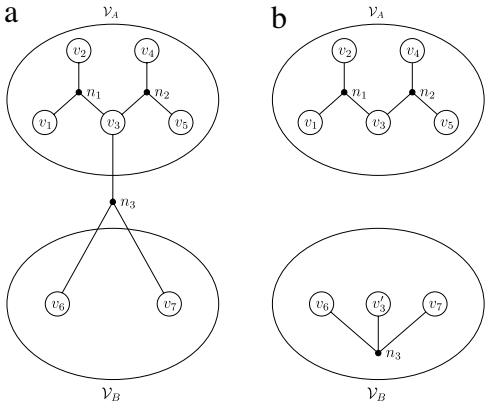

Fig. 3. Replication in an undirected hypergraph. (a) Initial bipartition, (b) after replicating v_3.

Having described the differences between vertex replication in directional and undirectional HP models, we set our focus on replication in undirectional HP models and define the *Replicated Undirected Hypergraph Partitioning* problem as follows: given an undirected hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$, an imbalance ratio ϵ, and a replication ratio ρ, find a K-way covering subset of $\mathcal{V}$, $\Pi^R = \{\mathcal{V}_1, \ldots, \mathcal{V}_K\}$ that minimizes the cutsize (Eqs. (2) or (3)) while satisfying the following constraints.

- *Balancing constraint:* $W_{\max} \leq (1+\epsilon)W_{\text{avg}}$, where
  $W_{\max} = \max_{1 \leq k \leq K} W(\mathcal{V}_k)$ and $W_{\text{avg}} = (1+\rho)W(\mathcal{V})/K$.
- *Replication constraint:* $\sum_{k=1}^{K} W(\mathcal{V}_k) \leq (1+\rho)W(\mathcal{V})$.

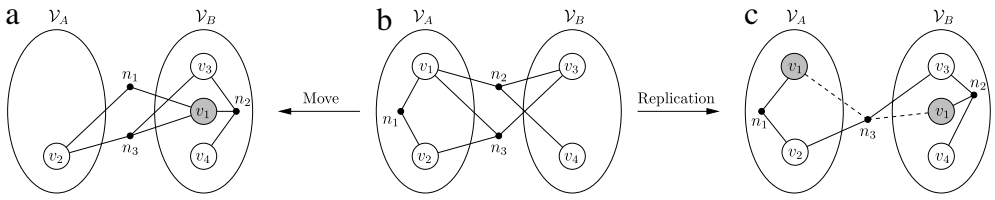

Fig. 4. Move and replication of a vertex. (b) Initial bipartition, (a) after moving v_1 from $\mathcal{V}_A$ to $\mathcal{V}_B$, and (c) after replicating v_1 from $\mathcal{V}_A$ to $\mathcal{V}_B$.

Note that $W_{\max}$ denotes the weight of the maximally weighted part, W_{avg} denotes the part weight under perfect balance, and $W(\mathcal{V})$ denotes the total vertex weight without replication.

4. Replicated FM (rFM)

We propose an extended FM heuristic which we call *replicated FM* (rFM) to address the *Replicated Undirected Hypergraph Partitioning* problem.

4.1. Definitions

In a two-way covering subset $\Pi^R = \{\mathcal{V}_A, \mathcal{V}_B\}$ of $\mathcal{V}$, a vertex can belong to $\mathcal{V}_A$, $\mathcal{V}_B$, or both of them if it is replicated, and hence it can be in one of three states, *A*, *B*, and *AB*:

$$\text{State}(v_i) = \begin{cases} A & \text{if } v_i \in \mathcal{V}_A \text{ and } v_i \notin \mathcal{V}_B, \\ B & \text{if } v_i \in \mathcal{V}_B \text{ and } v_i \notin \mathcal{V}_A, \\ AB & \text{if } v_i \in \mathcal{V}_A \text{ and } v_i \in \mathcal{V}_B. \end{cases}$$

Herein, a covering subset Π^R of $\mathcal{V}$ will be referred to as a replicated partition of $\mathcal{V}$, and subsets of Π^R will be referred to as parts of Π^R. Each instance of a replicated vertex is referred to as a replica. The number of non-replicated vertices in state *A* and connected by n_j is denoted as $\sigma_A(n_j)$. The number of non-replicated vertices in state *B* and connected by n_j is denoted as $\sigma_B(n_j)$. Similarly, the number of replicated vertices (not the number of replicas) that are connected by n_j is denoted as $\sigma_{AB}(n_j)$. Note that, according to the definitions,

$$|\text{Vertices}(n_j)| = \sigma_A(n_j) + \sigma_B(n_j) + \sigma_{AB}(n_j).$$

A net n_j in a two-way replicated partition is said to be *cut* if both $\sigma_A(n_j) > 0$ and $\sigma_B(n_j) > 0$. The *cut-state* of a net is used to describe whether that net is cut or not. A net n_j is said to be *internal* to $\mathcal{V}_A$ if $\sigma_B(n_j) = 0$ and it is said to be *internal* to $\mathcal{V}_B$ if $\sigma_A(n_j) = 0$. A net n_j can be considered *internal* to either $\mathcal{V}_A$ or $\mathcal{V}_B$ if $\sigma_A(n_j) = 0$, $\sigma_B(n_j) = 0$ and $\sigma_{AB}(n_j) > 0$.

rFM is an iterative-improvement heuristic that tries to improve the cutsize of a given two-way replicated partition by move, replication, and unreplication operations performed on vertices. The move and replication operations can only be performed on non-replicated vertices, whereas the unreplication operation can only be performed on replicated vertices. A non-replicated vertex has two gains, which are move and replication gains. Similarly, a replicated vertex also has two gains, which are unreplication from $\mathcal{V}_A$ and unreplication from $\mathcal{V}_B$ gains. The gain definitions are as follows.

- The *move gain*, $g_m(v_i)$, of a non-replicated vertex v_i is defined as the reduction in the cutsize if v_i were to be moved to the other part. The move gain of v_i is equal to the difference between the sum of the costs of the nets saved from the cut and the sum of the costs of the internal nets that are brought to the cut. Fig. 4(b) and (a) display the move of v_1 from $\mathcal{V}_A$ to $\mathcal{V}_B$. Moving v_1 from $\mathcal{V}_A$ to $\mathcal{V}_B$ brings net n_1 into the cut while saving net n_2 from the cut. Hence, $g_m(v_1) = c(n_2) - c(n_1)$. After the move operation, v_1 is locked. The locked vertices in the examples are illustrated by gray color.
- The *replication gain*, $g_r(v_i)$, is defined as the reduction in the cutsize if vertex v_i were to be replicated to the other part. The replication gain of v_i is equal to the sum of the costs of the nets saved from the cut. When a vertex is replicated, it cannot bring any internal net to the cut and thus cannot increase the cutsize. This forms the basic difference between the move and replication operations. Consequently, for any vertex v_i, we have $g_r(v_i) \geq 0$ and $g_r(v_i) \geq g_m(v_i)$. Fig. 4(b) and (c) show the replication of v_1 from $\mathcal{V}_A$ to $\mathcal{V}_B$. The replication of v_1 saves net n_2 from the cut as the move of v_1 does; however, net n_1 still remains as an internal net, as opposed to the move operation on the same vertex. Hence, $g_r(v_1) = c(n_2)$. In the examples, if a net is internal to a part and connects a replicated vertex, we illustrate this by putting a pin to the replica that is in the part of the internal net and omit the pin to the other replica. In contrast, if an external net connects a replicated vertex, the pins to the replicas of the replicated vertex connected by that net are displayed by dashed lines.
- The *unreplication gain*, $g_{u,A}(v_i)$ or $g_{u,B}(v_i)$, is defined as the reduction in the cutsize if a replica of the replicated vertex v_i were to be unreplicated from its part. Since unreplication of a replica cannot improve the cutsize, the maximum unreplication gain of a replica is zero. Thus, for any replicated vertex v_i, $g_{u,A}(v_i) \leq 0$ and $g_{u,B}(v_i) \leq 0$. A replica with an unreplication gain of zero implies that this replica is unnecessary and its removal will not change the cutsize. On the other hand, if the unreplication gain of a replica is negative, this implies that the replica is necessary and its unreplication will bring internal net(s) to the cut. Fig. 5 shows the unreplication of a necessary and an unnecessary replica. Initially, there are two replicas of v_1 in the bipartition in Fig. 5(b). The replica in $\mathcal{V}_A$ is necessary, and its unreplication causes the internal net n_1 to be cut, as seen in Fig. 5(a). On the other hand, the replica in $\mathcal{V}_B$ is unnecessary, and its unreplication does not change the cut set, as seen in Fig. 5(c). Hence, $g_{u,A}(v_1) = -c(n_1)$ and $g_{u,B}(v_1) = 0$.

4.2. Overall rFM algorithm

Replicated FM performs a predetermined number of passes considered on all vertices, where each pass comprises a sequence of operations (Algorithm 1). First, we compute the two possible gains for each vertex and initialize the pin distributions of the nets (line 1). At the beginning of each pass, we unlock all vertices to be able to perform operations on them (line 3). Then the algorithm enters the inner while loop (lines 4–7). In this loop, we first select a vertex and an operation (move, replication, or unreplication) to be performed on the selected vertex (line 5) according to the operation selection criteria described below. Then we perform the selected operation if it does not violate the size constraints on the weights of the parts (line 6). After the selected operation is performed on the vertex, the selected vertex is locked and the gain values of its unlocked neighbors and the pin distributions of the nets that connect this vertex are updated (line 7). A pass terminates when there are no more valid operations. At the end of a pass, a

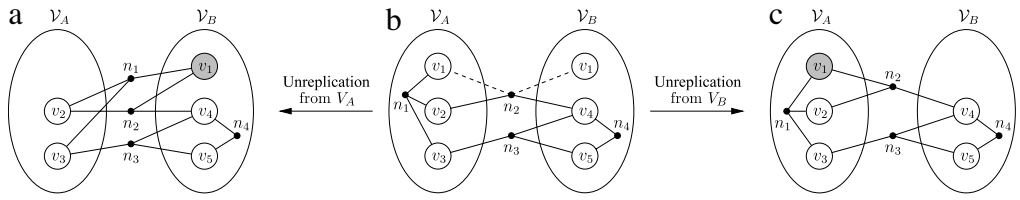

Fig. 5. Unreplication of instances of a replicated vertex. (b) Initial bipartition, (a) after unreplicating the replica of v_1 from $\mathcal{V}_A$, and (c) after unreplicating the replica of v_1 from $\mathcal{V}_B$.

rollback procedure is applied to the point where the bipartition with the minimum cutsize is seen (line 8).

The size constraint check performed during the operation selection is done as follows. (i) If the selected operation is a move or a replication, the new weight of the destination part if the selected operation were to be performed is computed, and, if it exceeds $(1+\epsilon)W_{\text{avg}}$, this operation is discarded, and (ii) if the selected operation is unreplication, it is checked if the weight of the part on which unreplication were to be performed drops below $(1-\epsilon)W_{\text{avg}}$, and, if it does, it is discarded. Furthermore, if the selected operation is replication, it is only performed if the total amount of replication performed up to that point plus the weight of the selected vertex does not exceed the allowed replication amount $\rho W(\mathcal{V})$.

Algorithm 1: Basic steps of rFM.

Input: $\mathcal{H} = (\mathcal{V}, \mathcal{N})$, $\Pi^R = \{\mathcal{V}_A, \mathcal{V}_B\}$
1 Initialize pin distributions, gains, and priority queues.
2 **while** *there are passes to perform* **do**
3 Unlock all vertices.
4 **while** *there is any valid operation* **do**
5 $(v, op) \leftarrow$ Select the vertex and the operation to perform on it.
6 Perform *op* on v, store the reduction in the cutsize, and lock v.
7 Update the gains of unlocked neighbors of v and the pin distributions of the nets in *Nets*(v).
8 Rollback to the point when minimum cutsize is seen.

Operation selection: We use a priority-based selection approach for determining the current operation and disallow some operations that do not satisfy certain conditions. The selection strategy is based on principles such as minimizing the number of unnecessary replicas, limiting the replication amount, and improving the balance. We give the highest priority to the elimination of unnecessary replicas. We do not perform unreplication operations with negative gains simply because such operations will degrade the cutsize. If there are no unnecessary replicas, we make a choice between move and replication by selecting the operation with the higher gain. Ties between the gains of the selected move and replication operations are broken in favor of the move operations. Any replication with a gain value of zero is disallowed since such operations will produce unnecessary replicas. However, the zero-gain moves that improve the balance are retained. Since, for any vertex v_i, $g_r(v_i) \geq g_m(v_i)$, in a single pass, the number of replication operations tends to outweigh the number of move operations. This issue can be addressed by the *gradient methodology*, which we discuss below.

Gradient methodology: The gradient methodology is used in FM heuristics that are capable of replication for directed graph models [34] to obtain partitions with better cutsize. The basic idea of the gradient methodology is to introduce the replication in the later iterations of a pass, especially when the improvement achieved in the cutsize by performing only move operations drops below a certain threshold. As mentioned in [16], early replication can have a negative effect on the final partition by limiting the algorithm's ability to change the current partition. Furthermore, by using the replication in the later iterations, the algorithm can climb out of the local minima reached by the move operations. In rFM, we adopt and modify the gradient methodology by allowing only move and unreplication operations until the improvement in the cutsize drops below a certain threshold, and then we allow replication operations.

Early exit: We use the early-exit scheme [18] to improve the run-time performance of rFM. In this scheme, if there are no improvements in the cutsize for a predetermined number of successive iterations, the current pass of the FM algorithm is terminated since it is unlikely to further improve the cutsize.

Locking: In conventional move-based FM algorithms, after moving a vertex, it is locked to avoid thrashing [17]. Similarly, in rFM, we also lock the operated vertex after performing a move, replication, or unreplication operation on that vertex.

Data structures: We maintain six priority queues keyed according to the gain values of the vertices with respect to type of operation. The heaps are implemented as binary heaps. For each part, we have three heaps for storing the move, replication, and unreplication gains. The two gains associated with a non-replicated vertex are stored in the move and replication heaps of the part that the vertex belongs to. Similarly, the two gains associated with the replicas of a replicated vertex have their unreplication gains stored in the unreplication heap of their respective parts.

4.3. Net criticality

The main power of rFM, like all FM-based algorithms, lies in its efficient linear-time gain update operations [17]. In this section, we present net criticality definitions that trigger updates on move, replication, and unreplication gains.

A net n_j is said to be *critical to part* $\mathcal{V}_k$, if an operation performed on a vertex $v_i \in \mathcal{V}_k$ can change the cut-state of n_j. Whenever an operation is performed on a vertex v_i, we check the criticality conditions of the nets that connect v_i. If the criticality condition of a net n_j that connects v_i changes, the other vertices that are connected by n_j are checked for gain updates. Each type of operation imposes different pin distributions for the criticality of nets; thus the criticality definition of a net is classified as move criticality, replication criticality, and unreplication criticality, according to the type of operation that causes a change in the cut-state of the respective net.

For a net to be move critical, it must connect at least two non-replicated vertices ($\sigma_A(n_j) + \sigma_B(n_j) > 1$), and it must either be an internal net or an external net with a single pin in one of the two parts. As seen in Table 1, a net n_j is move critical to $\mathcal{V}_A$ if ($\sigma_A(n_j) = 1$ and $\sigma_B(n_j) > 0$) or ($\sigma_B(n_j) = 0$ and $\sigma_A(n_j) > 1$), and to $\mathcal{V}_B$ if ($\sigma_A(n_j) = 0$ and $\sigma_B(n_j) > 1$) or ($\sigma_B(n_j) = 1$ and $\sigma_A(n_j) > 0$).

For a net to be replication critical, it must connect at least two non-replicated vertices ($\sigma_A(n_j) + \sigma_B(n_j) > 1$), and it must be an

Table 1
Criticality definitions for a net n_j to $\mathcal{V}_A$ and $\mathcal{V}_B$. For example, n_j is replication critical to $\mathcal{V}_A$ if $\sigma_A(n_j) = 1$ and $\sigma_B(n_j) > 0$.

n_j is	Move critical	Replication critical	Unreplication critical
To $\mathcal{V}_A$ if	$(\sigma_A(n_j) = 1$ and $\sigma_B(n_j) > 0)$ or $(\sigma_B(n_j) = 0$ and $\sigma_A(n_j) > 1)$	$\sigma_A(n_j) = 1$ and $\sigma_B(n_j) > 0$	$\sigma_B(n_j) = 0$ and $\sigma_A(n_j) > 0$ and $\sigma_{AB}(n_j) > 0$
To $\mathcal{V}_B$ if	$(\sigma_A(n_j) = 0$ and $\sigma_B(n_j) > 1)$ or $(\sigma_B(n_j) = 1$ and $\sigma_A(n_j) > 0)$	$\sigma_B(n_j) = 1$ and $\sigma_A(n_j) > 0$	$\sigma_A(n_j) = 0$ and $\sigma_B(n_j) > 0$ and $\sigma_{AB}(n_j) > 0$

external net with a single pin in one of the two parts. As seen in Table 1, a net n_j is replication critical to $\mathcal{V}_A$ if $(\sigma_A(n_j) = 1$ and $\sigma_B(n_j) > 0)$, and to $\mathcal{V}_B$ if $(\sigma_B(n_j) = 1$ and $\sigma_A(n_j) > 0)$. Note that the internal nets which are always move critical are never replication critical, since the replication of a vertex connected by an internal net cannot change the cut-state of that net. This difference is indicated in Table 1, where the conditions $(\sigma_B(n_j) = 0$ and $\sigma_A(n_j) > 1)$ and $(\sigma_A(n_j) = 0$ and $\sigma_B(n_j) > 1)$, which exist in the move-critical column, do not appear in the replication-critical column.

For a net to be unreplication critical, it must be an internal net that connects at least one non-replicated and one replicated vertex $(\sigma_A(n_j) + \sigma_B(n_j) > 0$ and $\sigma_{AB}(n_j) > 0)$. As seen in Table 1, a net n_j is unreplication critical to $\mathcal{V}_A$ if $(\sigma_B(n_j) = 0$ and $\sigma_A(n_j) > 0$ and $\sigma_{AB}(n_j) > 0)$, and to $\mathcal{V}_B$ if $(\sigma_A(n_j) = 0$ and $\sigma_B(n_j) > 0$ and $\sigma_{AB}(n_j) > 0)$. Note that external nets that connect a single non-replicated vertex in only one of the two parts, which are move critical, are never unreplication critical, since unreplication of a vertex connected by an external net cannot change the cut-state of that net. This difference is indicated in Table 1, where the conditions $(\sigma_A(n_j) = 1$ and $\sigma_B(n_j) > 0)$ and $(\sigma_B(n_j) = 1$ and $\sigma_A(n_j) > 0)$, which are shown in the move-critical column do not appear in unreplication-critical column.

4.4. rFM algorithm details

In this section, we present detailed explanations of some of the non-trivial concepts and algorithms used in rFM. The examples respect the basics of the operation selection criteria mentioned in Section 4.2. For the sake of simplicity, we assume that each net has unit cost, and we also overlook the balance constraints on part weights in the examples.

Initial gain computation. The initial gain computation, which is performed at the beginning of each pass of rFM, is given in Algorithm 2 and consists of two main loops. The first loop resets the initial gain values by traversing vertices (lines 1–7) and the second loop completes the initialization of gains by traversing all pins (lines 8–18). The move and replication gains are computed according to the external and critical nets that connect these vertices, whereas the unreplication gains are modified according to the internal and critical nets that connect these vertices.

The move and replication gains of the non-replicated vertices are initially set to their minimum possible values (lines 3–4). If a net n_j is external and move critical or replication critical, the move and replication gains of the vertices connected by n_j must be incremented by $c(n_j)$ (lines 12–13), since it can be saved from the cut with either one of these operations. In contrast to move and replication gains, unreplication gains are initially set to their maximum possible values (lines 6–7). If a net n_j is internal and thus unreplication critical, the unreplication gains of the replicas of the replicated vertices connected by n_j may need to be updated. The unreplication gains of the replicas that are in the same part with this internal net need to be decremented by $c(n_j)$ if n_j connects at least one non-replicated vertex that is in the same part with this net (lines 14–18).

Algorithm 2: Initial move, replication, and unreplication gain computation.

```
   Input: H = (V, N), Π^R = {V_A, V_B}
1  foreach v_i ∈ V do
2      if State(v_i) ≠ AB then
3          g_m(v_i) ← −c(InternalNets(v_i))
4          g_r(v_i) ← 0
5      else
6          g_{u,A}(v_i) ← 0
7          g_{u,B}(v_i) ← 0
8  foreach n_j ∈ N do
9      foreach v_i ∈ Vertices(n_j) do
10         if State(v_i) ≠ AB and n_j is external then
11             if (σ_A(n_j) = 1 and State(v_i) = A) or (σ_B(n_j) = 1 and
               State(v_i) = B) then ▷ n_j is critical to V_A or V_B
12                 g_m(v_i) ← g_m(v_i) + c(n_j)
13                 g_r(v_i) ← g_r(v_i) + c(n_j)
14         else if State(v_i) = AB and n_j is internal then
15             if σ_A(n_j) > 0 and σ_B(n_j) = 0 then ▷ n_j is critical to V_A
16                 g_{u,A}(v_i) ← g_{u,A}(v_i) − c(n_j)
17             else if σ_B(n_j) > 0 and σ_A(n_j) = 0 then ▷ n_j is critical to
               V_B
18                 g_{u,B}(v_i) ← g_{u,B}(v_i) − c(n_j)
```

Fig. 6(a) shows the pin distributions of the nets and the gain values of the vertices for a sample bipartition after Algorithm 2 is run on this sample. Nets n_4, n_5, and n_6 are cut; thus the cutsize of the bipartition in Fig. 6(a) is three. We use the notation $\sigma(n_j) = (\sigma_A(n_j) : \sigma_B(n_j) : \sigma_{AB}(n_j))$ to denote the pin distribution of n_j.

Gain updates after a move operation. Algorithm 3 shows the procedure for performing gain updates after moving a given vertex v^* from $\mathcal{V}_A$ to $\mathcal{V}_B$. The algorithm includes updating fields of v^* (lines 1–2), the pin distributions of Nets (v^*) (lines 4 and 16), and the gain values of neighbors of v^* (lines 5–15 and 17–27). The necessary field updates on v^* are performed by updating the state and locked fields of v^* to reflect the move operation. The pin distribution of each net $n_j \in$ Nets (v^*) needs to be updated by decrementing $\sigma_A(n_j)$ by 1 and incrementing $\sigma_B(n_j)$ by 1. When the pin distribution of n_j changes, its criticality may change with respect to the operation type. The change in the criticality of n_j may require various gain updates on the unlocked vertices connected by n_j.

After decrementing the number of vertices of n_j in $\mathcal{V}_A$ (line 4), we check the value of $\sigma_A(n_j)$ to see if the criticality of n_j has changed (lines 5 and 11). If $\sigma_A(n_j) = 0$, n_j becomes internal to $\mathcal{V}_B$ by becoming move critical and unreplication critical to this part, and if $\sigma_A(n_j) = 1$, n_j becomes move critical and replication critical to $\mathcal{V}_A$. Similarly, after incrementing the number of vertices connected by n_j in $\mathcal{V}_B$ (line 16), we check the value of $\sigma_B(n_j)$ to see if the criticality of n_j has changed (lines 17 and 23). If $\sigma_B(n_j) = 1$, it means that n_j was internal and hence was move critical and unreplication critical to $\mathcal{V}_A$, and if $\sigma_B(n_j) = 2$, it means that n_j was move critical and replication critical to $\mathcal{V}_B$. Under these conditions for n_j, the gains of the vertices connected by n_j should be checked for any update with respect to the corresponding part.

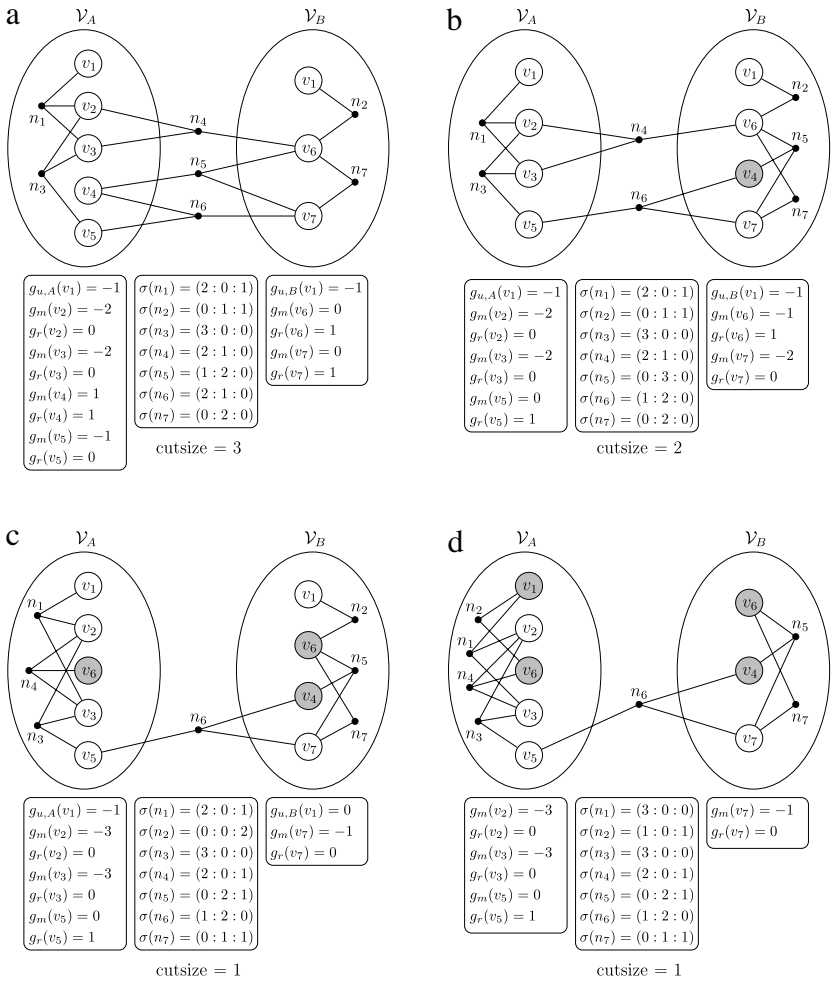

Fig. 6. Pin distributions of nets, gain values of vertices, and cutsize for a given bipartition. (a) Initial bipartition, (b) after moving v_4, (c) after replicating v_6, and (d) after unreplicating v_1 from $\mathcal{V}_B$. Gray vertices indicate locked vertices.

In Fig. 6(a), when we consider the selection criteria, the selected operation is going to be the move of v_4 whose gain is one. Fig. 6(b) shows the bipartition after running Algorithm 3 with the selected vertex v_4. After the move of v_4, n_5 is saved from the cut, and the cutsize of the bipartition becomes two.

Gain updates after a replication operation. Algorithm 4 shows the procedure for performing gain updates after replicating a given vertex v^* from $\mathcal{V}_A$ to $\mathcal{V}_B$. The procedure starts with changing the state of v^* to *AB* and locking both replicas of v^* (lines 1–2). Then, for each net n_j that connects v^*, the pin distributions of n_j are updated and checked for criticality condition changes (lines 6 and 17). Since v^* was in $\mathcal{V}_A$ before replication, $\sigma_A(n_j)$ is decremented by 1 and $\sigma_{AB}(n_j)$ is incremented by 1 to reflect that v^* is now a replicated vertex (lines 4–5). The replication of v^* from $\mathcal{V}_A$ does not change the $\sigma_B(n_j)$ value of any $n_j \in$ Nets (v^*); thus the criticality conditions that include $\sigma_B(n_j)$ need not be checked.

After the value of $\sigma_A(n_j)$ is decremented (line 4), n_j must be checked for criticality condition changes to see if there are any necessary gain updates for the neighbors of v^* (lines 6 and 17). If $\sigma_A(n_j) = 0$, n_j becomes move critical and unreplication critical to $\mathcal{V}_B$. In this condition, the move gains of the unlocked vertices and the unreplication gains of the unlocked replicas that are connected by n_j need to be decremented by $c(n_j)$ since n_j is internal now, and the move of any vertex or the unreplication of any replica connected by n_j would bring it to cut. If $\sigma_A(n_j) = 1$, n_j becomes move critical and replication critical to $\mathcal{V}_A$. The move or the replication of the only non-replicated vertex v_i connected by n_j in $\mathcal{V}_A$ can now save n_j from the cut, and thus the move and replication gains of this vertex must be incremented by $c(n_j)$.

After moving v_4, now we are to select another vertex to operate on in Fig. 6(b). There are two operations with the highest gain, which are the replication of v_5 and the replication of v_6, and the gain values of these operations are one. We select to replicate v_6. Fig. 6(c) shows the bipartition after running Algorithm 4 with v_6. After replication of v_6, we observe that n_4 is now uncut, and the cutsize becomes one.

Gain updates after an unreplication operation. Algorithm 5 shows the procedure for performing updates after unreplication of a given replica v^* from $\mathcal{V}_A$. The procedure starts with changing the state of v^* to *B* and locking it (lines 1–2). Then, for each net n_j that connects v^*, the pin distributions of n_j are updated and checked for criticality condition changes (lines 6 and 17). Since v^* was a replicated vertex before unreplication from $\mathcal{V}_A$, $\sigma_B(n_j)$ is incremented by 1 and $\sigma_{AB}(n_j)$ is decremented by 1 to reflect that v^* is now a non-replicated vertex in $\mathcal{V}_B$ (lines 4–5). The unreplication of v^* from $\mathcal{V}_A$ does not change the $\sigma_A(n_j)$ value of any $n_j \in$ Nets (v^*); thus the criticality conditions that include $\sigma_A(n_j)$ need not be checked.

After the value of $\sigma_B(n_j)$ is incremented (line 4), n_j must be checked for criticality condition changes to see if there are any necessary gain updates for the neighbors of v^* (lines 6 and 17). If $\sigma_B(n_j) = 1$, it means that n_j was move critical and unreplication critical to $\mathcal{V}_A$. In this case, the move and replication gains of the unlocked vertices and replicas that are in $\mathcal{V}_A$ and connected by n_j are incremented by $c(n_j)$, since n_j is not an internal net anymore.

Algorithm 3: Gain updates after moving v^* from $\mathcal{V}_A$ to $\mathcal{V}_B$.

Input: $\mathcal{H} = (\mathcal{V}, \mathcal{N})$, $\Pi^R = \{\mathcal{V}_A, \mathcal{V}_B\}$, $v^* \in \mathcal{V}_A$
$State(v^*) \leftarrow B$
Lock v^*
foreach $n_j \in Nets(v^*)$ **do**
  $\sigma_A(n_j) \leftarrow \sigma_A(n_j) - 1$
  if $\sigma_A(n_j) = 0$ **then** ▷ n_j becomes critical to $\mathcal{V}_B$
    foreach *unlocked* $v_i \in Vertices(n_j)$ **do**
      if $State(v_i) = B$ **then**
        $g_m(v_i) \leftarrow g_m(v_i) - c(n_j)$
      else if $State(v_i) = AB$ **then**
        $g_{u,B}(v_i) \leftarrow g_{u,B}(v_i) - c(n_j)$
  else if $\sigma_A(n_j) = 1$ **then** ▷ n_j becomes critical to $\mathcal{V}_A$
    foreach *unlocked* $v_i \in Vertices(n_j)$ **do**
      if $State(v_i) = A$ **then**
        $g_m(v_i) \leftarrow g_m(v_i) + c(n_j)$
        $g_r(v_i) \leftarrow g_r(v_i) + c(n_j)$
  $\sigma_B(n_j) \leftarrow \sigma_B(n_j) + 1$
  if $\sigma_B(n_j) = 1$ **then** ▷ n_j was critical to $\mathcal{V}_A$
    foreach *unlocked* $v_i \in Vertices(n_j)$ **do**
      if $State(v_i) = A$ **then**
        $g_m(v_i) \leftarrow g_m(v_i) + c(n_j)$
      else if $State(v_i) = AB$ **then**
        $g_{u,A}(v_i) \leftarrow g_{u,A}(v_i) + c(n_j)$
  else if $\sigma_B(n_j) = 2$ **then** ▷ n_j was critical to $\mathcal{V}_B$
    foreach *unlocked* $v_i \in Vertices(n_j)$ **do**
      if $State(v_i) = B$ **then**
        $g_m(v_i) \leftarrow g_m(v_i) - c(n_j)$
        $g_r(v_i) \leftarrow g_r(v_i) - c(n_j)$

Algorithm 4: Gain updates after replicating v^* from $\mathcal{V}_A$ to $\mathcal{V}_B$.

Input: $\mathcal{H} = (\mathcal{V}, \mathcal{N})$, $\Pi^R = \{\mathcal{V}_A, V_B\}$, $v^* \in \mathcal{V}_A$
$State(v^*) \leftarrow AB$
Lock v^*
foreach $n_j \in Nets(v^*)$ **do**
  $\sigma_A(n_j) \leftarrow \sigma_A(n_j) - 1$
  $\sigma_{AB}(n_j) \leftarrow \sigma_{AB}(n_j) + 1$
  if $\sigma_A(n_j) = 0$ **then** ▷ n_j becomes critical to $\mathcal{V}_B$
    foreach *unlocked* $v_i \in Vertices(n_j)$ **do**
      if $State(v_i) = B$ **then**
        $g_m(v_i) \leftarrow g_m(v_i) - c(n_j)$
        if $\sigma_B(n_j) = 1$ **then**
          $g_r(v_i) \leftarrow g_r(v_i) - c(n_j)$
      else if $State(v_i) = AB$ **then**
        if $\sigma_B(n_j) = 0$ **then**
          $g_{u,A}(v_i) \leftarrow g_{u,A}(v_i) + c(n_j)$
        else if $\sigma_B(n_j) > 0$ **then**
          $g_{u,B}(v_i) \leftarrow g_{u,B}(v_i) - c(n_j)$
  else if $\sigma_A(n_j) = 1$ **then** ▷ n_j becomes critical to $\mathcal{V}_A$
    foreach *unlocked* $v_i \in Vertices(n_j)$ **do**
      if $State(v_i) = A$ **then**
        $g_m(v_i) \leftarrow g_m(v_i) + c(n_j)$
        if $\sigma_B(n_j) > 0$ **then**
          $g_r(v_i) \leftarrow g_r(v_i) + c(n_j)$

If $\sigma_B(n_j) = 2$, it means that n_j was move critical and replication critical to $\mathcal{V}_B$. The net n_j connects two vertices in $\mathcal{V}_B$ and one of them, v^*, is already locked, and thus the move and replication gains of the other vertex, v_i, need to be decremented by $c(n_j)$, since this vertex can no longer save n_j from the cut.

In Fig. 6(c), after the replication of v_6, there is an unnecessary replica in $\mathcal{V}_B$ with an unreplication gain of zero. According to the selection criteria, the selected operation is the unreplication of the replica of v_1 in $\mathcal{V}_B$. Fig. 6(d) shows the bipartition after running Algorithm 5. The unreplication of an unnecessary replica cannot change the cutsize; thus, after the unreplication of the replica $v_1 \in \mathcal{V}_B$, the cutsize is still one.

Algorithm 5: Gain updates after unreplicating v^* from $\mathcal{V}_A$.

Input: $\mathcal{H} = (\mathcal{V}, \mathcal{N})$, $\Pi^R = \{\mathcal{V}_A, \mathcal{V}_B\}$, $v^* \in \mathcal{V}_A$
$State(v^*) \leftarrow B$
Lock v^*
foreach $n_j \in Nets(v^*)$ **do**
  $\sigma_B(n_j) \leftarrow \sigma_B(n_j) + 1$
  $\sigma_{AB}(n_j) \leftarrow \sigma_{AB}(n_j) - 1$
  if $\sigma_B(n_j) = 1$ **then** ▷ n_j was critical to $\mathcal{V}_A$
    foreach *unlocked* $v_i \in Vertices(n_j)$ **do**
      if $State(v_i) = A$ **then**
        $g_m(v_i) \leftarrow g_m(v_i) + c(n_j)$
        if $\sigma_A(n_j) = 1$ **then**
          $g_r(v_i) \leftarrow g_r(v_i) + c(n_j)$
      else if $State(v_i) = AB$ **then**
        if $\sigma_A(n_j) = 0$ **then**
          $g_{u,B}(v_i) \leftarrow g_{u,B}(v_i) - c(n_j)$
        else if $\sigma_A(n_j) > 0$ **then**
          $g_{u,A}(v_i) \leftarrow g_{u,A}(v_i) + c(n_j)$
  else if $\sigma_B(n_j) = 2$ **then** ▷ n_j was critical to $\mathcal{V}_B$
    foreach *unlocked* $v_i \in Vertices(n_j)$ **do**
      if $State(v_i) = B$ **then**
        $g_m(v_i) \leftarrow g_m(v_i) - c(n_j)$
        if $\sigma_A(n_j) > 0$ **then**
          $g_r(v_i) \leftarrow g_r(v_i) - c(n_j)$

4.5. Complexity analysis of rFM

Consider a single pass of rFM to be performed on an initial bipartition $\Pi^R = \{\mathcal{V}_A, \mathcal{V}_B\}$ of a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ with $V = |\mathcal{V}|$ vertices and P pins. Let V_r be the number of replicated vertices and V_s be the number of non-replicated vertices. Clearly, $V = V_r + V_s$. The initial gain computation takes $O(P)$ time since the vertices connected by each net are traversed as seen in Algorithm 2. After the initial gain computation is completed, these gain values are stored in six heaps. For each heap, it is required to perform a build-heap operation. The build-heap operations on two heaps storing move gains take a total of $O(V_s)$ time. Similarly, the build-heap operations on two heaps storing replication gains take a total of $O(V_s)$ time. This is because the total number of vertices in two heaps storing move gains and in two heaps storing replication gains are both equal to V_s. The build-heap operation on the heap storing unreplication gains of the replicas in $\mathcal{V}_A$ takes $O(V_r)$ time, and similarly the build-heap operation on the heap storing unreplication gains of the replicas in $\mathcal{V}_B$ takes $O(V_r)$ time, since each heap possesses V_r elements. Thus, the total time required for building heaps is equal to $O(V_r + V_r + V_s + V_s) = O(2V) = O(V)$.

The selection procedure consists of checking maximum gain values in six heaps, which takes $O(1)$ time. After selecting the gain value from one of the heaps with respect to the selection criteria, we perform an extract-max operation on the selected heap and a delete operation on another heap for the other gain value of the selected vertex (Section 4.2). Regardless of the selected heap, the extract-max and delete operations on the heaps are bounded by the number of total vertices, since the maximum number of elements in a single heap can be at most V. Thus, a single selection operation takes $O(1) + O(2 \log V) = O(\log V)$ time. In a single pass of rFM where all vertices are exhausted, we can make at most V selections. Consequently, the cost of selection in a single pass of rFM is equal to $O(V \log V)$.

As proved in the original FM heuristic [17], during an FM pass, the criticality state of a net changes at most three times due to the vertex locking mechanism adopted, which limits the number of gain updates by a constant factor. For our algorithm, Table 1 reveals that the criticality of a net n_j depends on its pin distributions, $\sigma_A(n_j)$ and $\sigma_B(n_j)$. More specifically, after an operation is performed, the

criticality of n_j changes if $\sigma_A(n_j)$ or $\sigma_B(n_j)$ observes the following changes:

$\sigma_A(n_j)$ from $2 \rightarrow 1$ or $1 \rightarrow 0$ or $0 \rightarrow 1$ or $1 \rightarrow 2$,

$\sigma_B(n_j)$ from $2 \rightarrow 1$ or $1 \rightarrow 0$ or $0 \rightarrow 1$ or $1 \rightarrow 2$.

Consider a vertex v_i connected by a net n_j. Recall that move of v_i from $\mathcal{V}_A$ to $\mathcal{V}_B$ will decrease $\sigma_A(n_j)$ by one and increase $\sigma_B(n_j)$ by one. Similarly, replication of v_i from $\mathcal{V}_A$ to $\mathcal{V}_B$ will decrease $\sigma_A(n_j)$ by one and unreplication of v_i from $\mathcal{V}_A$ will increase $\sigma_B(n_j)$ by one. Since replication of v_i will decrease $\sigma_A(n_j)$ or $\sigma_B(n_j)$ by one, replication of vertices connected by n_j can change the cut state of that net at most four times. Note that, when there are two vertices connected by n_j that are locked in two parts of a bipartition, that net will always be cut during that pass, and hence further criticality analysis will not be necessary for it. Also observe that a move to a part locks the moved vertex to that part and unreplication of a vertex from a part locks the other replica of that vertex in the other part. It is easy to see that the number of criticality changes that can be achieved by any combination of move and unreplication operations is at most five (the criticality of n_j can change at most four times for one part and one time for the other part). If we add criticality changes that can be achieved by replication as well, we can surmise that the criticality state of a net can change at most nine times during a pass. In fact, a more careful analysis reveals that the criticality state of a net can change at most seven times, since some changes either overlap with each other or cancel each other in the sense that, when one of them occurs for a net, the other cannot. Thus, in rFM, the number of criticality state changes for a net, and hence the number of gain updates for vertices is also bounded by a constant factor.

In rFM, since each vertex possesses two gains, in the worst case, both of these gain values may need to be updated, which doubles the number of gain updates for any vertex compared to FM. Each gain update requires an increase-key or decrease-key operation on the corresponding heap. Consequently, the complexity of gain updates in rFM is $O(2P \log V) = O(P \log V)$.

Given these complexity values, the complexity of a single pass of rFM is $O(P + V + V \log V + P \log V) = O(P \log V)$, since $P \geq V$.

4.6. rFM and multilevel framework

We utilize the rFM heuristic in a multilevel framework. In our multilevel approach we use the coarsening and initial partitioning phases of the conventional multilevel approach as is, and redesign the uncoarsening phase from scratch so that we can perform replication in this phase.

At each level of the uncoarsening phase, we perform multiple passes to refine the current bipartition. At the end of each level l, the bipartition Π_l^R on the coarser hypergraph $\mathcal{H}_l$ is projected back to the bipartition Π_{l-1}^R on the finer hypergraph $\mathcal{H}_{l-1}$. The projection includes the decomposition of each super-vertex in $\mathcal{H}_l$ to its constituent vertices in $\mathcal{H}_{l-1}$. The decomposition of a non-replicated super-vertex in $\mathcal{H}_l$ results in multiple non-replicated vertices in $\mathcal{H}_{l-1}$. Similarly, the decomposition of a replicated super-vertex in $\mathcal{H}_l$ results in multiple replicated vertices in $\mathcal{H}_{l-1}$. The existence of replicated vertices does not disturb the projection process. Clearly, the decomposition of a replicated super-vertex to its constituent replicated vertices will not change the cut-state of the nets this replicated super-vertex is connected by.

Unnecessary replicas tend to occur excessively at the beginning of each uncoarsening level due to the increase in the degrees of freedom after the projection of a coarser hypergraph to a finer hypergraph. Such replicas hamper the quality of the refinement and partitioning process if they are not removed, since (i) they consume the given replication amount needlessly, which may prevent the positive gain replications from being performed, and (ii) in the construction of the new hypergraphs for further levels of the RB, they can cause the new hypergraphs to become unnecessarily bigger. In operation selection, we give the unreplication of unnecessary replicas the highest priority (Section 4.2). This way, the majority of the unnecessary replicas are eliminated at the beginning of each uncoarsening level.

5. Recursive bipartitioning

To obtain K-way replicated partitions, we utilize rFM in a recursive bipartitioning framework, which requires cut-net removal and cut-net splitting techniques to be altered to support replication. In the RB framework, after each bipartitioning of $\mathcal{H} = (\mathcal{V}, \mathcal{N})$, two subhypergraphs $\mathcal{H}' = (\mathcal{V}', \mathcal{N}')$ and $\mathcal{H}'' = (\mathcal{V}'', \mathcal{N}'')$ are constructed from $\Pi^R = \{\mathcal{V}_A, \mathcal{V}_B\}$. In replicated or non-replicated HP, regardless of the underlying cost scheme (e.g., cut-net, connectivity), the vertex sets of $\mathcal{H}'$ and $\mathcal{H}''$ are equivalent to $\mathcal{V}_A$ and $\mathcal{V}_B$, respectively. That is,

$$\mathcal{V}' = \mathcal{V}_A \quad \text{and} \quad \mathcal{V}'' = \mathcal{V}_B.$$

Recall that, in replicated HP, $\mathcal{V}_A$ and $\mathcal{V}_B$ include the replicated vertices by definition (Section 4.1). In the construction of $\mathcal{H}'$ and $\mathcal{H}''$, the replicas of the replicated vertices of Π^R become non-replicated vertices of both $\mathcal{H}'$ and $\mathcal{H}''$, and the necessary pins are placed for these vertices. Consequently, the numbers of vertices and pins of the resulting hypergraphs are greater when compared to non-replicated HP.

In the following two subsections, for RB-based replicated HP, we show how to extend the cut-net removal and cut-net splitting schemes to capture the cut-net (Eq. (2)) and connectivity (Eq. (3)) cutsize metrics, respectively.

5.1. Cut-net removal

In the cut-net removal scheme for RB-based non-replicated HP, the internal net sets of $\mathcal{V}_A$ and $\mathcal{V}_B$ constitute the net sets of $\mathcal{H}'$ and $\mathcal{H}''$, respectively. Vertices connected by those internal nets will again be connected by the same nets in the new subhypergraphs. All cut-nets are discarded since they contribute to the cutsize only once.

In the cut-net removal scheme for RB-based replicated HP, the cut-nets are also discarded, since the definition of a cut-net in replicated HP does not take replicated vertices into account (a net n_j is cut if $\sigma_A(n_j) > 0$ and $\sigma_B(n_j) > 0$). Internal nets are kept for further bipartitionings as in non-replicated HP. The net sets of $\mathcal{H}'$ and $\mathcal{H}''$ are defined as follows:

$$\mathcal{N}' = \Big\{ n_j' \text{ with } \text{Vertices}(n_j') = \text{Vertices}(n_j) \text{ s.t. } n_j \in \mathcal{N} \text{ and } \sigma_B(n_j) = 0 \Big\},$$

$$\mathcal{N}'' = \Big\{ n_j'' \text{ with } \text{Vertices}(n_j'') = \text{Vertices}(n_j) \text{ s.t. } n_j \in \mathcal{N}, \sigma_A(n_j) = 0 \text{ and } \sigma_B(n_j) > 0 \Big\}.$$

In the construction of the new net sets, there is a subtle difference (indicated with $\sigma_B(n_j) > 0$) due to the nets that connect only replicated vertices. In this case, since such nets may be considered to be internal either to $\mathcal{V}_A$ or $\mathcal{V}_B$, we assumed them to be internal to $\mathcal{V}_A$, and added the necessary pins accordingly and discarded the pins to the replicas in $\mathcal{V}_B$. However, a more intelligent scheme can be devised to decide on which part these nets are considered to be internal. Fig. 7(a) shows an example of a cut-net removal scheme, where the cut-net n_k is removed while the internal nets n_i and n_j are preserved as n_i' and n_j'' for further bipartitionings.

5.2. Cut-net splitting

In the cut-net splitting scheme for RB-based non-replicated HP, the internal net sets of $\mathcal{V}_A$ and $\mathcal{V}_B$ will be in the net sets of $\mathcal{H}'$

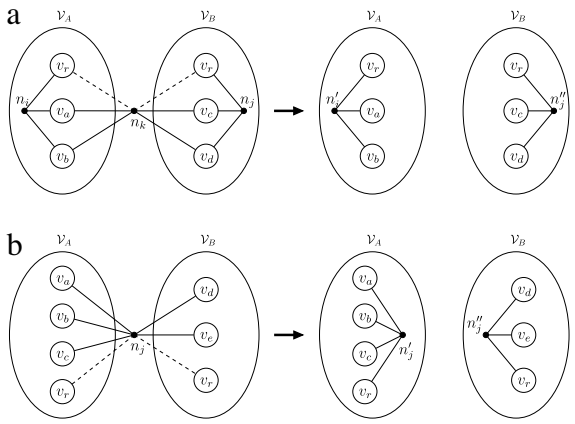

Fig. 7. (a) Cut-net removal. Cut-net n_k is removed while internal nets n_i and n_j are preserved for further bipartitionings. (b) Cut-net splitting. Cut-net n_j is split into two new nets n'_j and n''_j.

and $\mathcal{H}''$, respectively. Furthermore, each cut-net $n_j \in \Pi^R$ is split into two nets, n'_j and n''_j, where $\text{Vertices}(n'_j) = \text{Vertices}(n_j) \cap \mathcal{V}_A$ and $\text{Vertices}(n''_j) = \text{Vertices}(n_j) \cap \mathcal{V}_B$. Then, n'_j is added to $\mathcal{N}'$ if $\sigma_A(n'_j) > 1$ and n''_j is added to the $\mathcal{N}''$ if $\sigma_B(n''_j) > 1$. Clearly, $\sigma_B(n'_j) = 0$ and $\sigma_A(n''_j) = 0$.

In the cut-net splitting scheme for RB-based replicated HP, we need to add pins to the replicas of the replicated vertices in order to preserve the flexibility of performing move or replication operations on them in the newly constructed hypergraphs. The internal nets are kept for further bipartitionings as in non-replicated HP. The net sets of $\mathcal{H}'$ and $\mathcal{H}''$ are defined as follows:

$$\mathcal{N}' = \Big\{ n'_j \text{ with } \text{Vertices}(n'_j) = \text{Vertices}(n_j) \cap \mathcal{V}_A \text{ s. t. } n_j \in \mathcal{N}, \sigma_A(n_j) + \sigma_{AB}(n_j) > 1 \Big\},$$

$$\mathcal{N}'' = \Big\{ n''_j \text{ with } \text{Vertices}(n''_j) = \text{Vertices}(n_j) \cap \mathcal{V}_B \text{ s. t. } n_j \in \mathcal{N}, \sigma_B(n_j) + \sigma_{AB}(n_j) > 1 \text{ and } \sigma_B(n_j) > 0 \Big\}.$$

Notice that the new net definitions encompass both the internal nets and the external nets that are split. Similar to the cut-net removal scheme, there is a subtle difference (indicated with $\sigma_B(n_j) > 0$) due to the nets that connect only replicated vertices. Such nets are handled in the same way as they are handled in cut-net removal scheme. Fig. 7(b) shows the splitting of n_j into two distinct nets n'_j and n''_j and the addition of necessary pins for these nets for further bipartitionings. Note that a pin is added for each of the replicas of the replicated vertex v_r.

5.3. Replication amount distribution

The RB scheme consists of multiple bipartitions. The replication amount used in each bipartitioning can have an effect on the final cutsize. We consider two different replication amount distribution schemes in this work. (i) In the *level-wise replication* scheme, first, the total replication amount is divided by $\log K$, the number of levels of the recursion tree of RB, and then, for each specific level, the assigned replication amount is evenly distributed among the bipartitions in that level. (ii) In the *bisection-wise replication* scheme, the total replication amount is divided by $K - 1$, the number of bipartitionings in a K-way partitioning, and then distributed evenly among these bipartitionings.

6. Pin selection

After achieving a K-way replicated partition, in order to compute the cutsize we have to select the pins to the replicas of the replicated vertices connected by a net. The replication of a vertex v_i brings the problem of selecting replicas of v_i for each net it is connected by. If a net n_j connects replicated vertices, we need to decide which replicas of these replicated vertices will be "used" by n_j. This is required for a couple of reasons: (i) the cutsize of the final partition can only be computed after deciding from which part the replicas will be used, and (ii) the investigated real-world problem may enforce the nets to make a choice from which parts their replicas will be used. We propose a simple pin selection technique whose basic motivation is not to increase the cutsize with careless pin selection.

Fig. 8 shows two pin selection alternatives for a net n_j which connects three vertices v_r, v_s, and v_n in a three-way partition given in Fig. 8(a). The vertices v_r and v_s are replicated, each having three replicas, and v_n is non-replicated. In the examples for pin selection, after a selection is performed for a pin, this pin is indicated by a thick line. A selection alternative for n_j is seen in Fig. 8(b) where v_r and v_s are selected from $\mathcal{V}_1$ and $\mathcal{V}_2$, respectively and $\lambda(n_j) = 3$. A more careful selection alternative is shown in Fig. 8(c), where both v_r and v_s are selected from $\mathcal{V}_2$ and $\lambda(n_j) = 1$. This example illustrates how pin selection can be crucial in computing the cutsize of a given partition.

Let $nr(n_j, k)$ and $r(n_j, k)$ respectively denote the number non-replicated and replicated vertices that are connected by n_j in $\mathcal{V}_k$. Consider a net n_j that connects a replicated vertex v_i which has t replicas. We are to make a decision for n_j to select one of these replicas within the considerations mentioned above. Our replica selection algorithm is based on a greedy heuristic that consists of two stages.

The first stage of the algorithm is based on the following observation. Consider a cut-net n_j that connects at least one non-replicated vertex in $\mathcal{V}_k$ (i.e., $nr(n_j, k) > 0$). If $r(n_j, k) > 0$ too, then the pins of n_j to the replicas in $\mathcal{V}_k$ can be safely selected

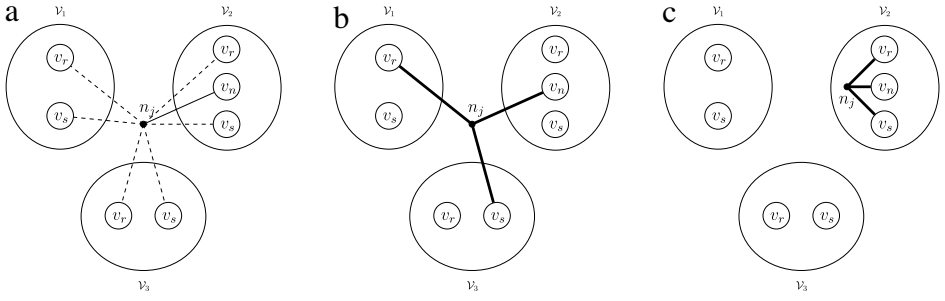

Fig. 8. Pin selection alternatives for net n_j. (a) Initial partition before selection, (b) after the first selection alternative, $\lambda(n_j) = 3$, and (c) after the second selection alternative, $\lambda(n_j) = 1$.

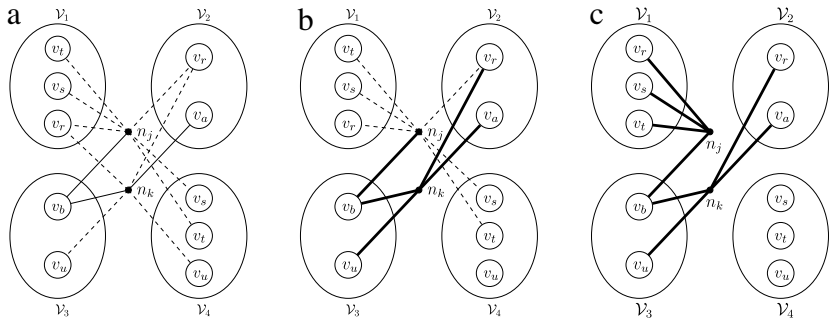

Fig. 9. (a) Initial partition before selection. (b) After the first stage of the pin selection algorithm. (c) After the second stage of the pin selection algorithm.

without degrading the cutsize. Using this observation, for each net n_j, the algorithm traverses n_j's pins to the vertices in $\Lambda(n_j)$ and selects the currently unselected pins to the replicas in $\mathcal{V}_k$ if $nr(n_j, k) > 0$. If a pin (n_j, v_i^k) is selected for the replicated vertex v_i during this process, all other pins (n_j, v_i^ℓ), where $\ell \neq k$, are deselected. Here, v_i^k denotes the replica of the replicated vertex v_i in $\mathcal{V}_k$. After the selection of pin (n_j, v_i^k) for n_j, the $r(n_j, k)$ value is decremented by one for each $\mathcal{V}_k$ that a replica of v_i resides in. At the end of the first stage, for each net n_j, n_j's pins to the non-replicated vertices are selected by default, since these are the only candidates for those pins. After the selection of pin (n_j, v_i) for n_j, where $v_i \in \mathcal{V}_k$ is non-replicated, the $nr(n_j, k)$ value is decremented by one. Fig. 9 illustrates our selection heuristic. In Fig. 9(a), there are two non-replicated (v_a and v_b) and four replicated (v_r, v_s, v_t, and v_u) vertices. Fig. 9(b) shows the resulting pins after the first stage of the pin selection algorithm is run. The selected pins $((n_j, v_b), (n_k, v_b), (n_k, v_u^3), (n_k, v_r^2), (n_k, v_a))$ are shown with thick lines. Note that deselected pins $((n_k, v_u^4), (n_k, v_r^1))$ are removed.

At the beginning of the second stage of the algorithm, all $nr(n_j, k)$ values are equal to zero. For a net n_j, if $r(n_j, k) > 0$ for at least one $\mathcal{V}_k$, the pin selection problem for n_j can be reduced to the set cover problem. This case can be seen in Fig. 9(b) for the pin selection process for n_j. In the example, our sets are $\mathcal{S}_1 = \{v_r, v_s, v_t\}$, $\mathcal{S}_2 = \{v_r\}$ and $\mathcal{S}_3 = \{v_s, v_t\}$, and we try to find a set cover of the ground set $\mathcal{S} = \{v_r, v_s, v_t\}$. Since the set cover problem is NP-hard [25], we adopt a simple greedy heuristic that has an approximation ratio of $\ln(n) + 1$ [24], where n is the number of total elements. Basically, in each iteration, this greedy heuristic selects the set that covers the largest number of uncovered elements so far and then removes the currently covered elements from remaining sets. When this algorithm is run for n_j in the example in Fig. 9(b), $\mathcal{S}_1$ will be selected as the covering subset which is illustrated in Fig. 9(c). According to this selection process, the pins (n_j, v_r^1), (n_j, v_s^1), (n_j, v_t^1) are selected whereas the pins (n_j, v_r^2), (n_j, v_s^4), (n_j, v_t^4) are deselected.

7. Experiments

7.1. Experimental setup

The proposed replication scheme is integrated into the multilevel HP tool PaToH [2]. We call this modified version of PaToH rpPaToH. In the experiments, we used the same parameters for PaToH and rpPaToH in the coarsening and the initial partitioning phases. We used agglomerative clustering (absorption clustering using pins) and greedy hypergraph growing algorithms in the coarsening and the initial partitioning phases, respectively. For both PaToH and rpPaToH, the imbalance ratio is set to $\epsilon = 0.10$ in all experiments. The boundary FM (BFM) refinement heuristic option is selected for PaToH, whereas the proposed rFM heuristic is used for rpPaToH. The number of passes for the refinement algorithms used is set to three for both tools. For the early-exit feature, the number of allowed operations which do not improve the cutsize is set to 100 for both PaToH and rpPaToH. We report results for five different K (16, 32, 64, 128, 256) and five different ρ (0.05, 0.10, 0.15, 0.20, 0.25) values. The results for $K = 16$ are omitted in Tables 3 and 5 due to lack of space.

Table 2
Dataset properties.

Dataset	Number of			Average
	Vertices	Nets	Pins	Net degree
CG	43,244	200,000	800,034	4.0
VG	146,742	500,000	1,999,540	4.0
WP	442,063	1,000,000	3,997,741	4.0
FB	2,880,004	4,000,000	16,001,754	4.0
AOL	1,339,596	11,867,848	38,907,550	3.3

All algorithms are implemented in C and compiled in gcc with the -O3 flag. Due to the randomized nature of PaToH, all of the partitioning results reported are the averages of ten runs. In the experiments, a six-core AMD Opteron with 2.1 GHz of clock frequency and 32 GB of main memory is used.

The performance of the proposed replication scheme and the replicated HP tool rpPaToH developed is evaluated on term-based partitioning of inverted indices for the parallel query processing application discussed in Section 1.2. The datasets are separated into two as realistic and semi-synthetic datasets. The realistic dataset used in the experiments is the AOL dataset [39], which consists of about 12 million queries and 1.3 million terms. We used synthetic datasets due to the difficulties in obtaining real-world query sets for IR. The synthetic datasets are in fact semi-synthetic in the sense that they are generated from real-world crawls downloaded from the Stanford WebBase Project [21]. The CG dataset is composed of pages collected from sites related to the California governor election on 09/30/2003. The FB dataset is composed of pages collected from Facebook on 09/08/2008. The WP dataset is composed of pages collected from Wikipedia in May 2006. The VG dataset is composed of pages collected from sites related with the Virginia Tech shooting on 04/23/2007. We generated query sets from each dataset consisting of queries between two to six terms, since 90% of Web search queries have between one and six terms [39]. Queries with single terms are not included in the datasets since they induce nets with single pins, and such nets do not incur any cost to the cutsize. In order to imitate real-world query sets, documents are randomly selected from these datasets, and then, from these selected documents, query terms are formed so that the query term frequencies follow a Zipfian distribution. Utilizing these query sets, we generated hypergraphs as described in [28] except the vertex weighting schema. Since vertex replication does not incur redundant computation due to the pin selection scheme described

Table 3
Percentage load imbalance (%LI) and percentage replication utilization (%RU) for PaToH+ MF and rpPaToH.

Dataset	ρ	$K = 32$			$K = 64$			$K = 128$			$K = 256$		
		PaToH+MF	rpPaToH		PaToH+MF	rpPaToH		PaToH+MF	rpPaToH		PaToH+MF	rpPaToH	
		%LI	%LI	%RU	%LI	%LI	%RU	%LI	%LI	%RU	%LI	%LI	%RU
CG	0.05	5.97	9.73	79	5.73	9.66	72	6.78	9.38	66	7.23	8.81	65
	0.10	5.17	9.52	88	5.63	9.38	85	5.89	9.01	79	6.99	8.76	74
	0.15	4.38	9.39	92	5.26	9.26	87	5.53	9.13	84	6.66	8.62	80
	0.20	4.06	9.21	93	4.31	9.17	91	5.08	8.78	86	6.16	8.53	83
	0.25	3.56	9.10	94	4.56	9.04	91	4.69	8.69	90	5.80	8.45	85
VG	0.05	6.82	8.79	79	6.57	9.10	76	6.61	9.07	72	6.57	8.98	68
	0.10	6.03	8.82	89	6.03	8.67	84	6.40	8.61	78	6.50	8.67	74
	0.15	5.23	7.85	93	5.68	8.31	89	6.20	8.19	85	6.01	8.37	80
	0.20	4.78	8.02	95	5.41	8.08	91	5.78	8.21	87	5.78	8.25	83
	0.25	4.05	7.44	96	4.81	7.97	92	5.05	8.18	89	5.72	8.06	85
WP	0.05	6.81	8.90	67	6.61	9.28	64	7.37	9.52	61	7.79	9.64	58
	0.10	6.15	9.44	73	5.87	9.44	70	6.68	8.78	65	7.66	8.50	62
	0.15	5.44	9.40	74	5.37	9.41	72	5.60	9.37	69	6.79	9.32	65
	0.20	4.47	9.19	75	5.18	9.21	73	5.35	9.18	70	5.83	9.16	67
	0.25	4.30	7.23	76	4.69	9.00	73	5.26	8.99	71	5.30	8.98	68
FB	0.05	9.00	5.74	70	9.11	5.87	69	8.91	6.90	68	8.49	6.96	66
	0.10	8.48	3.56	58	8.62	4.86	62	8.46	6.10	65	8.11	6.82	66
	0.15	7.89	1.66	51	8.18	3.52	55	8.14	4.84	59	7.73	5.74	61
	0.20	7.49	0.00	49	7.78	1.92	53	7.83	3.50	56	7.47	4.21	57
	0.25	6.84	0.00	44	7.36	0.77	50	7.48	2.76	53	6.90	3.63	55
AOL	0.05	5.74	9.19	99	6.47	8.99	97	6.56	9.54	95	7.22	9.46	92
	0.10	4.75	7.75	98	5.34	8.27	98	5.76	8.65	96	6.09	8.83	94
	0.15	4.15	6.65	98	4.79	7.78	97	5.11	8.06	97	5.57	8.43	95
	0.20	3.80	6.10	96	4.51	7.13	97	4.85	7.77	96	5.22	8.13	96
	0.25	3.69	4.44	91	4.03	6.39	95	4.60	6.73	96	4.88	7.52	95

Table 4
Cutsize averages of all datasets for four different schemes of rpPaToH normalized with respect to those of the bis + nor scheme.

ρ	Scheme	K				
		16	32	64	128	256
0.05	bis + nor	1.00	1.00	1.00	1.00	1.00
	bis + gra	0.88	0.91	0.89	0.93	0.93
	lev + nor	0.90	0.90	0.91	0.89	0.87
	lev + gra	0.81	0.81	0.81	0.83	0.84
0.10	bis + nor	1.00	1.00	1.00	1.00	1.00
	bis + gra	0.90	0.92	0.92	0.95	0.97
	lev + nor	0.91	0.90	0.88	0.87	0.86
	lev + gra	0.84	0.83	0.81	0.82	0.84
0.15	bis + nor	1.00	1.00	1.00	1.00	1.00
	bis + gra	0.93	0.93	0.96	0.96	0.96
	lev + nor	0.94	0.89	0.89	0.85	0.83
	lev + gra	0.87	0.83	0.83	0.82	0.82
0.20	bis + nor	1.00	1.00	1.00	1.00	1.00
	bis + gra	0.95	0.94	0.95	0.98	0.99
	lev + nor	0.89	0.87	0.86	0.84	0.82
	lev + gra	0.84	0.82	0.82	0.82	0.80
0.25	bis + nor	1.00	1.00	1.00	1.00	1.00
	bis + gra	0.95	0.95	0.95	1.00	1.00
	lev + nor	0.89	0.84	0.84	0.83	0.80
	lev + gra	0.84	0.81	0.80	0.81	0.79

in Section 6, the balance constraint is interpreted as balancing the number of terms assigned to each processor by using unit vertex weights in these hypergraphs. The characteristics of the hypergraphs generated from these datasets are given in Table 2.

7.2. Performance evaluations

In this section, we provide a thorough performance analysis of rpPaToH in terms of imbalance, replication utilization, and cutsize improvement.

Table 4 compares the performances of four different schemes utilized in rpPaToH in terms of cutsize averages for all datasets normalized with respect to those of bis + nor scheme. In the table, "lev" and "bis" respectively stand for the level-wise replication amount distribution scheme and the bisection-wise replication amount distribution scheme. The gradient methodology is indicated with "gra", whereas "nor" indicates that the gradient methodology is not used. For example, bis + gra means rpPaToH utilizes the bisection-wise scheme and gradient methodology. When the level-wise scheme is compared with the bisection-wise scheme (see the rows bis + nor and lev + nor, or bis + gra and lev + gra for a specific ρ value), the level-wise scheme achieves better cutsize values than the bisection-wise scheme. This may be due to the fact that the level-wise scheme distributes the given replication amount among bipartitionings proportional to the total vertex weights of the corresponding hypergraphs, whereas the bisection-wise scheme distributes the given replication amount evenly among bipartitionings, thus favoring the bipartitionings of (relatively) smaller hypergraphs at the deeper levels of the recursion tree of the RB scheme. As seen from the table, the gradient methodology performs better than its counterpart (see the rows bis + nor and bis + gra, or lev + nor and lev + gra for a specific ρ value). The reasons for this were explained in detail in Section 4.2. Henceforth, the results of rpPaToH with the level-wise scheme and gradient methodology are reported in subsequent tables and figures.

In parallel IR, one of the most commonly used replicated partitioning schemes is based on first hash-based partitioning of the inverted index and then replication of the posting lists of the most frequently occurring terms across all parts [37]. We call this scheme Hash+ MF. The recently proposed HP-based index partitioning scheme, which is summarized in Section 1.2, is reported to reduce the parallel query processing overhead significantly compared to hash-based partitioning. So, we replaced the hash-based partitioning in Hash+ MF with PaToH to obtain a more effective replicated inverted index partitioning scheme, which is referred here as PaToH+ MF. In our implementation

Table 5
Cutsize ($\times 10^3$) values for PaToH+ MF and rpPaToH.

Dataset	ρ	$K = 32$		$K = 64$		$K = 128$		$K = 256$	
		PaToH+MF	rpPaToH	PaToH+MF	rpPaToH	PaToH+MF	rpPaToH	PaToH+MF	rpPaToH
CG	0.05	359	220	412	300	445	362	468	410
	0.10	330	206	392	286	433	345	461	395
	0.15	305	194	373	267	422	329	453	380
	0.20	286	180	358	255	411	315	448	367
	0.25	266	168	344	241	401	303	441	354
VG	0.05	601	287	733	420	840	562	924	693
	0.10	539	267	672	395	790	539	887	660
	0.15	496	247	634	375	753	510	854	629
	0.20	458	226	605	349	724	485	832	603
	0.25	425	209	577	328	702	466	810	578
WP	0.05	657	355	867	494	1134	660	1406	855
	0.10	495	308	705	443	915	609	1176	780
	0.15	414	270	605	403	809	562	1034	724
	0.20	356	240	530	372	736	524	946	685
	0.25	312	212	477	336	675	492	882	649
FB	0.05	685	490	916	728	1183	1004	1518	1275
	0.10	512	306	740	511	963	761	1228	1032
	0.15	421	208	625	388	855	617	1090	879
	0.20	371	137	548	294	772	499	998	743
	0.25	337	97	492	221	710	412	934	651
AOL	0.05	3774	1254	6109	2005	8765	3363	11313	5235
	0.10	2161	780	4134	1495	6651	2620	9319	4305
	0.15	1451	551	3027	1180	5239	2177	7928	3518
	0.20	1073	427	2329	892	4333	1892	6905	3080
	0.25	820	299	1879	756	3692	1545	6122	2714
Averages of normalized cutsize values of PaToH+MF and rpPaToH with respect to PaToH+MF									
	0.05	1.00	0.54	1.00	0.60	1.00	0.66	1.00	0.71
	0.10	1.00	0.54	1.00	0.60	1.00	0.67	1.00	0.71
	0.15	1.00	0.53	1.00	0.60	1.00	0.66	1.00	0.71
	0.20	1.00	0.51	1.00	0.58	1.00	0.65	1.00	0.69
	0.25	1.00	0.49	1.00	0.56	1.00	0.63	1.00	0.68

of PaToH+ MF, first we obtain a K-way partition using PaToH. Then, we sort the vertices in non-increasing order with respect to their degrees and replicate them to each part in this order while respecting the given replication amount. This corresponds to replicating the most frequent terms and their inverted lists on the partition given by PaToH.

Table 3 displays the percentage load imbalance (%LI) and the percentage replication utilization (%RU) values obtained by PaToH+ MF and rpPaToH. These values are computed as follows:

$$\%\mathrm{LI} = \frac{W_{\max} - W_{\mathrm{avg}}}{W_{\mathrm{avg}}} \times 100,$$

$$\%\mathrm{RU} = \frac{\sum_{k=1}^{K} W(\mathcal{V}_k) - W(\mathcal{V})}{\rho W(\mathcal{V})} \times 100.$$

As seen in the table, both algorithms provide replicated partitions within the allowed imbalance values. The balancing performance of both algorithms is comparable, and there is no clear winner in this performance metric. Apart from that, it can be said that, as the given replication amount increases, the balance of the partitions obtained gets better. This is because replication can also be used to improve the balance of the partitions obtained.

The replication utilization values for the PaToH+ MF replication scheme are not presented in Table 3 since PaToH+ MF always utilizes 100% of the given replication amount. As seen in Table 3, rpPaToH does not fully utilize the given replication amount. This is because (i) replication operations with zero gain value are not allowed, which may prevent the replication operations from being performed even though the cutsize may be greater than zero and there is still a remaining replication amount to be used in the corresponding bipartitioning, and (ii) the gradient methodology uses more move operations, which limits the amount of replication performed in a pass of rFM. The remaining replication amount can be utilized by a post-processing step to rpPaToH such as MF or any other replication scheme to further improve the cutsize. However, the main purpose of the experiments in this section is to test the validity of RB-based replicated HP. Hence, the results with such post-processing enhancements are not reported here.

Table 5 displays the cutsize values obtained using PaToH+ MF and rpPaToH. Without any exceptions, rpPaToH performs significantly better than PaToH+MF in all experiments. In both schemes, as expected, the cutsize value decreases with increasing replication amount. The bottom of Table 5 displays the averages of normalized cutsize values of PaToH+ MF and rpPaToH with respect to those of PaToH+MF. As K increases for a fixed ρ, the average performance gap between rpPaToH and PaToH+ MF decreases. However, even for the highest K value of 256, rpPaToH reduces the cutsize by 29%, 29%, 29%, 31%, and 32% for the ρ values 0.05, 0.10, 0.15, 0.20, and 0.25, respectively. As ρ increases for a fixed K, the average performance gap between rpPaToH and PaToH+ MF increases gradually.

Figs. 10 and 11 are introduced for better illustration of relative cutsize performance comparisons of PaToH+ MF and rpPaToH with increasing K and ρ values, respectively. As seen in Fig. 10, with increasing K, the performance gap between PaToH+ MF and rpPaToH decreases for the CG, VG, and FB datasets and increases for the WP and AOL datasets. As seen in Fig. 11, with increasing ρ, the performance gap between PaToH+ MF and rpPaToH increases for FB, decreases for WP and AOL, and remains almost the same for the CG and VG datasets.

Table 6 displays the averages of the percentage reduction in cutsize values of rpPaToH over PaToH for all datasets. As expected, the percentage improvement of rpPaToH over PaToH increases with increasing ρ for a fixed K value. For a fixed replication amount, the percentage improvement of rpPaToH over

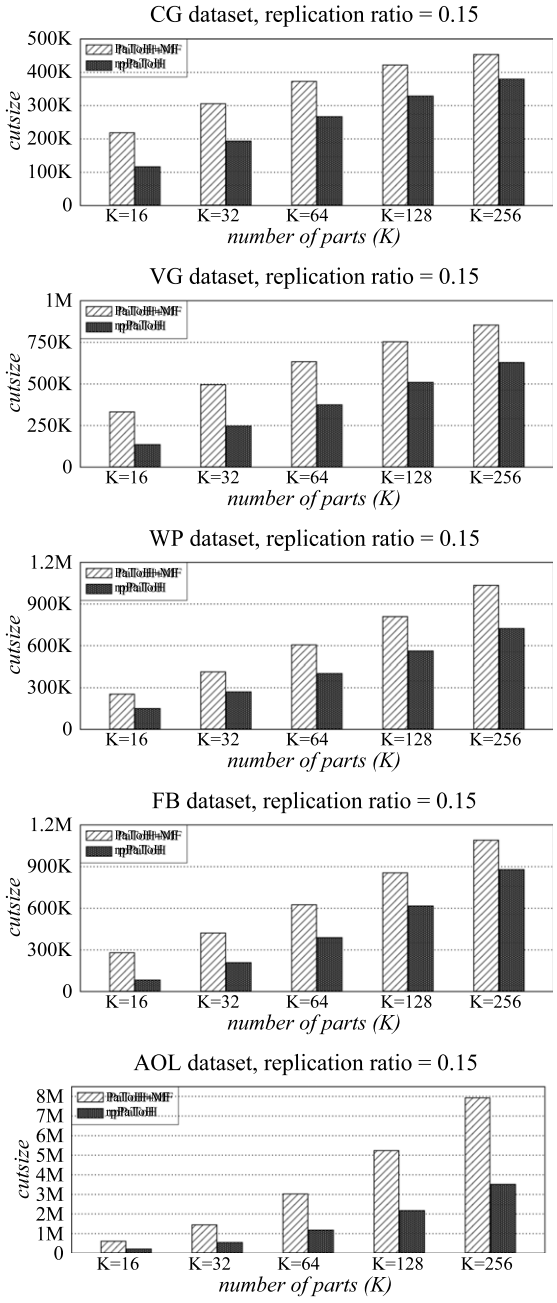

Fig. 10. Cutsize values of PaToH+ MF and rpPaToH with increasing K for $\rho = 0.15$.

Table 6
The averages of the percentage reduction in cutsize values of all datasets for rpPaToH over PaToH.

ρ	$K = 16$	$K = 32$	$K = 64$	$K = 128$	$K = 256$
0.05	76.76	68.28	60.69	52.47	44.64
0.10	81.01	72.75	65.02	56.87	49.36
0.15	83.98	75.80	68.30	60.32	53.11
0.20	86.19	78.43	71.02	63.06	55.95
0.25	87.69	80.55	73.42	65.42	58.37

PaToH decreases with increasing K. This experimental finding can be attributed to the fact that, with increasing K, the cutsize increases, and thus the ratio of the reduction in the cutsize due to fixed replication decreases.

Table 7 shows the run-time averages of all datasets for rpPaToH normalized with respect to those of PaToH. For a fixed

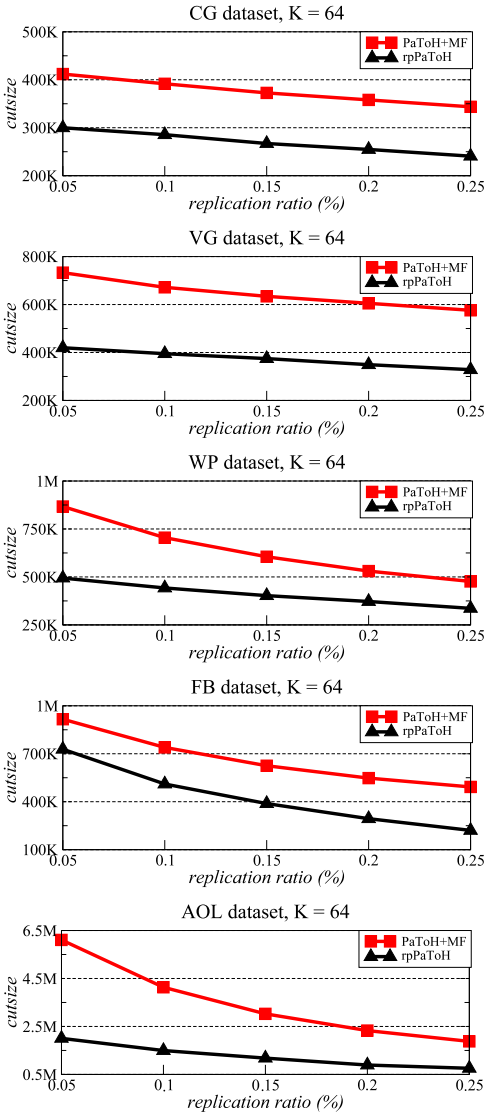

Fig. 11. Cutsize values of PaToH+ MF and rpPaToH with increasing ρ for $K = 64$.

K value, the run-time performance of rpPaToH degrades with respect to PaToH as ρ increases, since replication introduces new vertices and pins during the partitioning process. For a fixed ρ value, the run-time difference between PaToH and rpPaToH increases in favor of PaToH for increasing K. Our analysis reveals that this is mainly due to the run-time differences in the coarsening and the initial partitioning phases of PaToH and rpPaToH. Note that, after each bipartitioning in rpPaToH, the further coarsening and initial partitioning phases generally have to work on larger hypergraphs than those in PaToH. The larger the K, the greater the number of times these larger hypergraphs have to be bipartitioned, and hence the difference between PaToH and rpPaToH grows with increasing K. However, even for the largest $\rho = 0.25$ and $K = 256$ values, rpPaToH is only 2.78 times slower than PaToH on the average.

8. Conclusions and future work

A vertex replication scheme is proposed for undirectional HP models. The proposed scheme achieves replication during the partitioning process. Replication is performed using an extended version of the FM iterative-improvement heuristic (rFM) that operates on two-way partitions and is capable of replication

Table 7
Run-time averages of all datasets for rpPaToH normalized with respect to those for PaToH.

ρ	$K = 16$	$K = 32$	$K = 64$	$K = 128$	$K = 256$
0.05	1.76	1.90	2.03	2.20	2.40
0.10	1.78	1.94	2.17	2.37	2.55
0.15	1.83	1.97	2.28	2.41	2.58
0.20	1.82	1.99	2.36	2.55	2.70
0.25	1.91	2.03	2.42	2.55	2.78

and unreplication in addition to move of vertices. This two-way replicated partitioning scheme is used in a recursive bipartitioning framework to obtain K-way replicated partitions. Regarding the replicated vertices in a K-way replicated partition, a simple pin selection algorithm is proposed for the nets that connect replicated vertices. We developed a multilevel replicated HP tool, referred to as rpPaToH, by embedding our replication scheme into the uncoarsening phase of the multilevel HP tool PaToH. The validity of the proposed replication scheme is tested on one realistic and four semi-synthetic information retrieval datasets. rpPaToH is compared with a state-of-the-art replication scheme that replicates the most frequent terms to all parts to show that rpPaToH achieves better improvements in the cutsize within the allowed imbalance values with relatively low replication utilization. This work shows that vertex replication can be very effective in reducing the cutsize of the partitions obtained using HP by using little amount of replication.

As future research, we consider various ideas that can further improve the quality of the partitions. (i) Different operation selection strategies can be tested for rFM such as allowing zero gain replication or negative gain unreplication operations. (ii) The remaining replication amount can be used in a clever K-way replication heuristic to further improve the cutsize. Alternative ways of distributing the given replication amount between rpPaToH and this K-way replication heuristic can further be investigated.

References

[1] C.J. Alpert, A.B. Kahng, Recent directions in netlist partitioning: a survey, Integr. VLSI J. 19 (1995) 1–81.

[2] U.V. Çatalyürek, C. Aykanat, PaToH: partitioning tool for hypergraphs, Technical Report, Department of Computer Engineering, Bilkent University, 1999.

[3] C. Aykanat, B.B. Cambazoglu, B. Uçar, Multi-level direct k-way hypergraph partitioning with multiple constraints and fixed vertices, J. Parallel Distrib. Comput. 68 (2008) 609–625.

[4] C. Aykanat, A. Pinar, U.V. Çatalyürek, Permuting sparse rectangular matrices into block-diagonal form, SIAM J. Sci. Comput. 25 (2004) 1860–1879.

[5] L.A. Barroso, J. Dean, U. Hölzle, Web search for a planet: the Google cluster architecture, IEEE Micro 23 (2003) 22–28.

[6] T.N. Bui, C. Jones, A heuristic for reducing fill-in sparse matrix factorization, in: PPSC, pp. 445–452.

[7] B. Cahoon, K.S. McKinley, Performance evaluation of a distributed architecture for information retrieval, in: SIGIR'96: Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, New York, NY, USA, 1996, pp. 110–118.

[8] B.B. Cambazoglu, C. Aykanat, A term-based inverted index organization for communication-efficient parallel query processing, in: IFIP International Conference on Network and Parallel Computing, 2006.

[9] B.B. Cambazoglu, C. Aykanat, Hypergraph-partitioning-based remapping models for image-space-parallel direct volume rendering of unstructured grids, IEEE Trans. Parallel Distrib. Syst. 18 (2007) 3–16.

[10] B.B. Cambazoglu, F.P. Junqueira, V. Plachouras, S. Banachowski, B. Cui, S. Lim, B. Bridge, A refreshing perspective of search engine caching, in: Proceedings of the 19th International Conference on World Wide Web, WWW'10, ACM, New York, NY, USA, 2010, pp. 181–190.

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

[12] E. Demir, C. Aykanat, Efficient successor retrieval operations for aggregate query processing on clustered road networks, Inform. Sci. 180 (2010) 2743–2762.

[13] E. Demir, C. Aykanat, B.B. Cambazoglu, A link-based storage scheme for efficient aggregate query processing on clustered road networks, Inf. Syst. 35 (2010) 75–93.

[14] K. Devine, E. Boman, R. Heapby, B. Hendrickson, C. Vaughan, Zoltan data management service for parallel dynamic applications, Comput. Sci. Eng. 4 (2002) 90–97.

[15] N.J. Dingle, P.G. Harrison, W.J. Knottenbelt, Uniformization and hypergraph partitioning for the distributed computation of response time densities in very large Markov models, J. Parallel Distrib. Comput. 64 (2004) 908–920.

[16] M. Enos, S. Hauck, M. Sarrafzadeh, Evaluation and optimization of replication algorithms for logic bipartitioning, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 18 (1999) 1237–1248.

[17] C.M. Fiduccia, R.M. Mattheyses, A linear-time heuristic for improving network partitions, in: Proceedings of the 19th Design Automation Conference, DAC'82, IEEE Press, Piscataway, NJ, USA, 1982, pp. 175–181.

[18] J.R. Gilbert, E. Zmijewski, A parallel graph partitioning algorithm for a message-passing multiprocessor, Int. J. Parallel Program. 16 (1987) 427–449.

[19] M.K. Goldberg, M. Burnstein, Heuristic improvement technique for bisection of VLSI networks, in: Proceedings of the IEEE International Conference of Computer Design, pp. 122–125.

[20] B. Hendrickson, R. Leland, A multilevel algorithm for partitioning graphs, in: Proceedings of the 1995 ACM/IEEE Conference on Supercomputing (CDROM), Supercomputing'95, ACM, New York, NY, USA, 1995.

[21] J. Hirai, S. Raghavan, H. Garcia-Molina, A. Paepcke, Webbase: a repository of web pages, in: In Proceedings of the Ninth International World Wide Web Conference, pp. 277–293.

[22] J. Hwang, A. El Gamal, Optimal replication for min-cut partitioning, in: Proceedings of the 1992 IEEE/ACM International Conference on Computer-Aided Design, ICCAD'92, IEEE Computer Society Press, Los Alamitos, CA, USA, 1992, pp. 432–435.

[23] B.-S. Jeong, E. Omiecinski, Inverted file partitioning schemes in multiple disk systems, IEEE Trans. Parallel Distrib. Syst. 6 (1995) 142–153.

[24] D.S. Johnson, Approximation algorithms for combinatorial problems, in: Proceedings of the Fifth Annual ACM Symposium on Theory of Computing, STOC'73, ACM, New York, NY, USA, 1973, pp. 38–49.

[25] R. Karp, Reducibility among combinatorial problems, in: R. Miller, J. Thatcher (Eds.), Complexity of Computer Computations, Plenum Press, 1972, pp. 85–103.

[26] G. Karypis, R. Aggarwal, V. Kumar, S. Shekhar, Multilevel hypergraph partitioning: applications in VLSI domain, IEEE Trans. Very Large Scale Integr. 7 (1999) 69–79.

[27] G. Karypis, V. Kumar, Multilevel k-way hypergraph partitioning, in: Proceedings of the 36th Annual ACM/IEEE Design Automation Conference, DAC'99, ACM, New York, NY, USA, 1999, pp. 343–348.

[28] E. Kayaaslan, C. Aykanat, Efficient query processing on term-based-partitioned inverted indexes, Technical Report BU-CE-1102, Bilkent University, Computer Engineering Department, 2011. Also available at: http://www.cs.bilkent.edu.tr/tech-reports/2011.

[29] B. Kernighan, S. Lin, An efficient heuristic procedure for partitioning graphs, Bell Syst. Tech. J. 49 (1970) 291–307.

[30] C. Kring, A. Newton, A cell-replicating approach to minicut-based circuit partitioning, in: IEEE International Conference on, Computer-Aided Design, 1991, ICCAD-91, Digest of Technical Papers, 1991, pp. 2–5.

[31] R. Kužnar, F. Brglez, B. Zajc, Multi-way netlist partitioning into heterogeneous FPGAs and minimization of total device cost and interconnect, in: Proceedings of the 31st Annual Design Automation Conference, DAC'94, ACM, New York, NY, USA, 1994, pp. 238–243.

[32] T. Lengauer, Combinatorial Algorithms for Integrated Circuit Layout, John Wiley & Sons, Inc., New York, NY, USA, 1990.

[33] L.-T. Liu, M.-T. Kuo, C.-K. Cheng, T. Hu, A replication cut for two-way partitioning, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 14 (1995) 623–630.

[34] L.-T. Liu, M.-T. Kuo, S.-C. Huang, C.-K. Cheng, A gradient method on the initial partition of Fiduccia–Mattheyses algorithm, in: Proceedings of the 1995 IEEE/ACM International Conference on Computer-Aided Design, ICCAD'95, IEEE Computer Society, Washington, DC, USA, 1995, pp. 229–234.

[35] C. Lucchese, S. Orlando, R. Perego, F. Silvestri, Mining query logs to optimize index partitioning in parallel web search engines, in: Proceedings of the 2nd International Conference on Scalable Information Systems, InfoScale'07, ICST, Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, ICST, Brussels, Belgium, Belgium, 2007, pp. 43:1–43:9.

[36] A. MacFarlane, J.A. McCann, S.E. Robertson, Parallel search using partitioned inverted files, in: Proceedings of the Seventh International Symposium on String Processing Information Retrieval (SPIRE'00), IEEE Computer Society, Washington, DC, USA, 2000, pp. 209–220.

[37] A. Moffat, W. Webber, J. Zobel, Load balancing for term-distributed parallel retrieval, in: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR'06, ACM, New York, NY, USA, 2006, pp. 348–355.

[38] A. Moffat, W. Webber, J. Zobel, R. Baeza-Yates, A pipelined architecture for distributed text query evaluation, Inf. Retr. 10 (2007) 205–231.

[39] G. Pass, A. Chowdhury, C. Torgeson, A picture of search, in: Proceedings of the 1st International Conference on Scalable Information Systems, InfoScale'06, ACM, New York, NY, USA, 2006.
[40] S. Shekhar, C.-T. Lu, S. Chawla, S. Ravada, Efficient join-index-based spatial-join processing: a clustering approach, IEEE Trans. Knowl. Data Eng. 14 (2002) 1400–1421.
[41] A. Tomasic, H. Garcia-Molina, Performance of inverted indices in shared-nothing distributed text document information retrieval systems, in: Proceedings of the Second International Conference on Parallel and Distributed Information Systems, PDIS'93, IEEE Computer Society Press, Los Alamitos, CA, USA, 1993, pp. 8–17.
[42] A. Tomasic, H. Garcia-Molina, Performance issues in distributed shared-nothing information-retrieval systems, Inf. Process. Manage. 32 (1996) 647–665.
[43] A. Trifunović, W.J. Knottenbelt, Parallel multilevel algorithms for hypergraph partitioning, J. Parallel Distrib. Comput. 68 (2008) 563–581.
[44] B. Uçar, C. Aykanat, Revisiting hypergraph models for sparse matrix partitioning, SIAM Rev. 49 (2007) 595–603.
[45] B. Vastenhouw, R.H. Bisseling, A two-dimensional data distribution method for parallel sparse matrix–vector multiplication, SIAM Rev. 47 (2005) 67–95.

R. Oguz Selvitopi received his M.Sc. degree in Computer Engineering from Bilkent University, Turkey, in 2010. He is currently a Ph.D. candidate at Bilkent University. His research interests are parallel and distributed systems, scientific computing, and algorithms.

Ata Turk received his B.Sc. and M.Sc. degrees from the Computer Engineering Department of Bilkent University, Turkey, in 2002 and 2004, respectively. He is currently working towards a Ph.D. degree at Bilkent University. His research interests include parallel information retrieval and algorithms.

Cevdet Aykanat received his B.S. and M.S. degrees from Middle East Technical University, Ankara, Turkey, both in Electrical Engineering, and his Ph.D. degree from Ohio State University, Columbus, US, in Electrical and Computer Engineering. He was a Fulbright scholar during his Ph.D. studies. He worked at the Intel Supercomputer Systems Division, Beaverton, Oregon, US, 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, parallel computer graphics applications, parallel data mining, graph and hypergraph theoretic models for load balancing, high performance information retrieval systems, parallel and distributed databases, and grid computing. He has (co)authored about 60 technical papers published in academic journals indexed in the ISI, and his publications have received about 500 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 was appointed a member of IFIP Working Group 10.3 (Concurrent System Technology) in April 2004, a member of the EU-INTAS Council of Scientists in June 2005, and an Associate Editor of IEEE Transactions of Parallel and Distributed Systems in December 2008.