rpPaToH: **R**e**p**licated **Pa**rtitioning **To**ol for **H**ypergraphs*

R. Oguz Selvitopi
Computer Engineering Department
Bilkent University
Ankara, 06800 Turkey
reha@cs.bilkent.edu.tr
roguzsel@gmail.com

Ata Turk
Computer Engineering Department
Bilkent University
Ankara, 06800 Turkey
atat@cs.bilkent.edu.tr

Cevdet Aykanat
Computer Engineering Department
Bilkent University
Ankara, 06800 Turkey
aykanat@cs.bilkent.edu.tr

July, 2012

*This work is partially supported by Turkish Science and Research Council under grant 109E019.

Abstract

Hypergraphs are widely used to model various problems from different domains. With respect to modeled problem domain, the utilized hypergraph partitioning models can either be directed or undirected. The quality of the partitions obtained using hypergraph partitioning can further be improved by using vertex or net replication. The replication in directional hypergraph partitioning models is a well investigated subject in VLSI domain supported by various algorithms and tools. However, replication in undirectional hypergraph partitioning models is an immature research area. To fill this gap, we propose novel algorithms for performing vertex replication in undirectional hypergraph partitioning models. Our approach is one-phase, i.e., replication is performed simultaneously with the partitioning. This is achieved by using an extension of the FM heuristic, called replicated FM (rFM), that support replication and unreplication operations in addition to move operations. This algorithm is used as the main refinement heuristic in the multilevel framework. Later, rFM is utilized in a recursive bipartitioning framework to obtain K-way partitions. Pin selection algorithms are proposed to compute the final cutsize values. These algorithms and methods are realized in a tool called rpPaToH that performs vertex replication in undirected hypergraphs. This technical report describes the replicated hypergraph partitioning in short and gives detailed information about how to use the rpPaToH tool.

Keywords: replication, hypergraph, undirected, hypergraph partitioning, replicated hypergraph partitioning, iterative improvement heuristic, recursive bipartitioning

Contents

1 Introduction

Replication in directional hypergraph partitioning (HP) models (note that this report assumes a background for hypergraph partitioning, see [2] for definitions and background on this subject) is a well-studied problem in VLSI domain [4, 6, 8, 9, 10]. The purpose of replication in this domain is to reduce the interconnection cost of the partitioned circuits and the pin counts by replicating gates, which are modeled as vertices. There are various approaches for replication in directional HP models such as one-phase iterative-improvement-based heuristics [8, 9] that generally extend the basic FM heuristic [5] or the two-phase approaches that utilize linear programming [6] or flow-network formulations [10].

The replication in undirectional HP models in rather an immature research area and greatly differs from the replication in directional HP models [11]. There are two basic differences between replication in directional and undirectional HP models. First, vertex replication in directional HP models may bring internal nets to the cut and can increase the cutsize of a partition. Second, whenever a vertex is replicated in directional HP models, this replication may require further vertex/net replication which is due to the input-output relation that is inherent in the nets of the directed hypergraphs. These two cases are specific to directed hypergraphs and are not valid for undirected hypergraphs, thus the methods proposed for directional HP models are not directly applicable for undirectional HP models. Fig. 1 and Fig. 2 illustrate replication in directed and undirected hypergraphs, respectively. The replication of v_3 in Fig. 1 increases cutsize and requires further net replication whereas this is not the case for the replication of v_3 in Fig. 2.

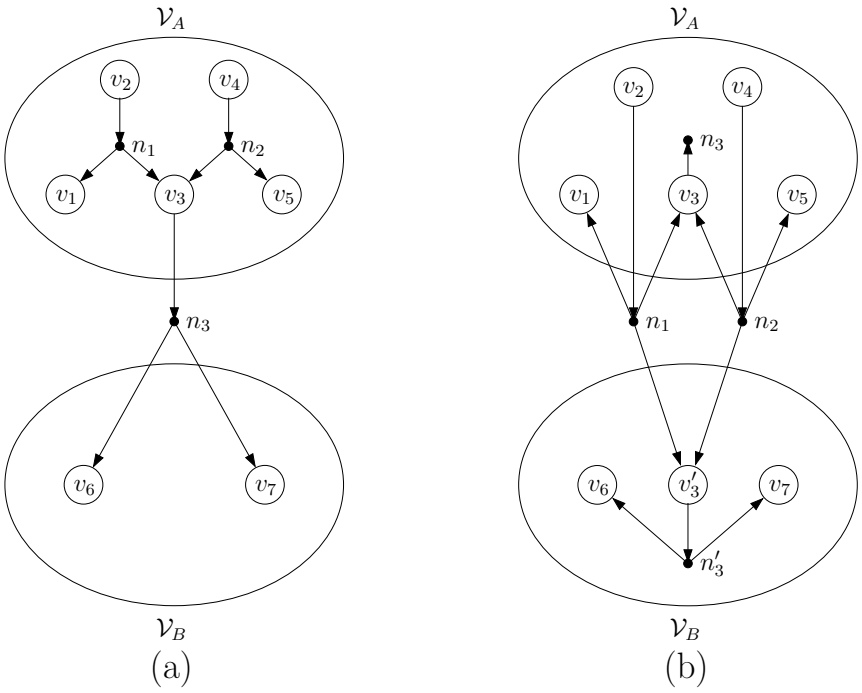

Figure 1: Replication in a directed hypergraph: (a) Initial bipartition, (b) after replicating v_3.

In hypergraph partitioning the purpose is to obtain a K-way partition (Π) of a given hypergraph $\mathcal{H}=(\mathcal{V},\mathcal{N})$ where each part $\mathcal{V}_k \in \Pi$ satisfies the balance constraint:

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

where $W_{avg} = W(\mathcal{V})/K$, ϵ is predetermined maximum imbalance ratio, and W is the weight function. The objective is to minimize the cutsize, which can be either the cut-net metric,

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

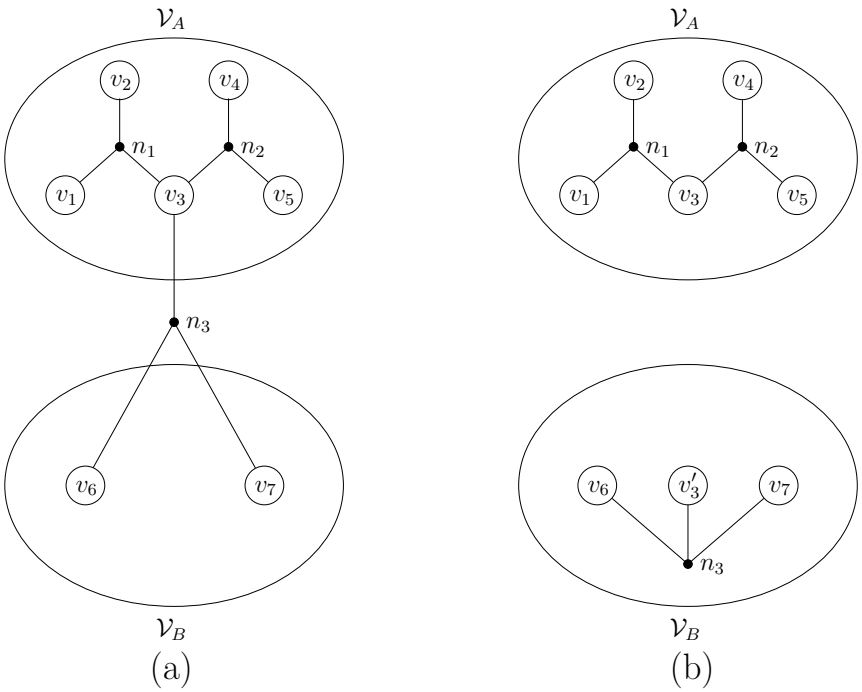

Figure 2: Replication in an undirected hypergraph: (a) Initial bipartition, (b) after replicating v_3.

or the connectivity metric,

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

Here, n_j denotes the net j and $c(n_j)$ denotes its cost. Only the external nets are considered ($\mathcal{N}_E$) in both cutsize metrics. In the connectivity metric in Equation 3, each net n_j's connectivity set must be used in cutsize computation, which is denoted as $\lambda(n_j)$.

In the replicated HP problem, compared to the HP problem, there is one more parameter, the replication ratio, ρ, which is a coefficient of the total vertex weight. Although both problems try to optimize the same objective, replicated HP problem has one more constraint regarding replication ratio. In addition, the balance constraint must be updated to include replication. Hence, given an undirected hypergraph $\mathcal{H}=(\mathcal{V},\mathcal{N})$, an imbalance ratio ϵ, and a replication ratio ρ, we are to find a K-way replicated partition (possibly overlapping) satisfying the balance constraint:

$$W_{max} \leq (1+\epsilon)W_{avg}, \text{ where } W_{max} = \max_{1 \leq k \leq K} W(\mathcal{V}_k) \text{ and } W_{avg} = (1+\rho)W(\mathcal{V})/K, \tag{4}$$

and the replication constraint:

$$\sum_{k=1}^{K} W(\mathcal{V}_k) \leq (1+\rho)W(\mathcal{V}). \tag{5}$$

Note that $W(\mathcal{V})$ denotes the total vertex weight without replication while $W(\mathcal{V}_k)$ for each part stands for the part weights with replication included.

We fill the necessity for replication in undirected hypergraphs by proposing a replicated hypergraph partitioning tool `rpPaToH` [11]. `rpPaToH` is an extension to the successful hypergraph partitioning tool `PaToH` [2] and performs vertex replication during the partitioning process by using a move, replication and unreplication capable iterative improvement heuristic based on the basic FM heuristic.

2 Replicated Hypergraph Partitioning for Undirected Hypergraphs

`rpPaToH` utilizes various algorithms and methodologies to achieve partitioning and replication simultaneously. In the core of these algorithms and methodologies is the replicated FM heuristic (rFM) that

basically performs vertex replication. rFM is mainly utilized as the refinement algorithm in the uncoarsening phase of the multilevel framework. In this way, we are able to perform partitioning and replication at the same time. To obtain K-way replicated partitions, rFM is utilized in a recursive bipartitioning framework supported by `rpPaToH`. To this end, `rpPaToH` utilizes new net splitting techniques to supoort vertex replication for both cutsize metrics given in Equations 2 and 3. Furthermore, to compute the final cutsize or because of the requirements of the modeled application, it might be necessary to determine which instances of the replicated vertices (replicas) will be used for the nets of the given hypergraph. For this purpose, `rpPaToH` employs a basic set-cover heuristic. To utilize the given replication amount efficiently, `rpPaToH` supports two replication distribution alternatives.

2.1 Replicated FM (rFM) Heuristic

Replicated FM heuristic is an extension to the basic FM heuristic [5] that runs on two-way partitions (bipartitions) and is capable of move, replication and unreplication of vertices. To support replication and unreplication of vertices, it defines a new state for vertices to indicate that a vertex can also be replicated, besides that they can be in one of two parts. rFM supports these three operations (move, replication and unreplication) by defining three types of gains: move, replication and unreplication gains. A replicated vertex can only have unreplication gains whereas a non-replicated vertex can have move and replication gains. Figs. 3 and 4 display examples of these operations.

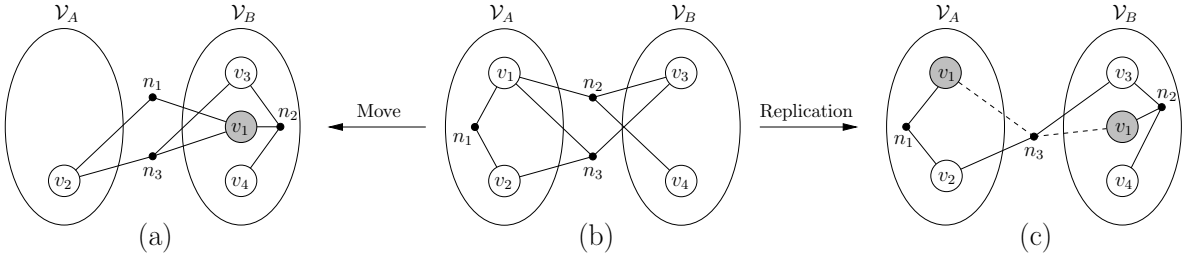

Figure 3: Move and replication of v_1.

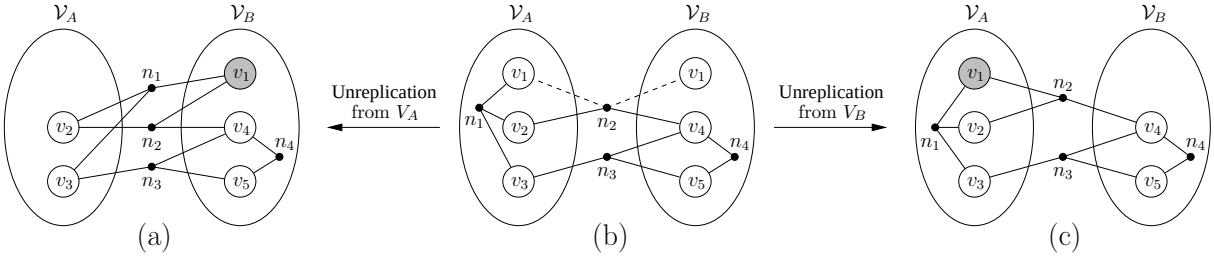

Figure 4: Unreplication of instances of v_1.

rFM is used in the uncoarsening phase of the multilevel framework to be able to perform partitioning and replication simultaneously. The coarsening and the initial partitioning phases are used as is (i.e., they are not altered to adopt the replicated vertices). rFM is linear in complexity (see [11] for a detailed analysis).

An extension to rFM heuristic is the *gradient* methodology in which the replication is used in a more intelligent manner. In this method, vertex replication is introduced in later stages of the rFM heuristic, where the improvement achieved by only move operations drops below a pre-determined threshold. After this threshold, rFM is allowed to replicate vertices.

2.2 Recursive Bipartitioning (RB)

There are two basic methods to obtain K-way partitions for both graphs and hypergraphs. One of them is direct K-way partitioning [1, 7], and the other one, which is the most widely used, is the recursive

bipartitioning [3]. We utilized RB in `rpPaToH`. In the RB scheme, firstly a bipartition of the initial hypergraph is obtained, and then this bipartition is decoded to construct two sub-hypergraphs using the cut-net removal and cut-net splitting techniques (Section 2.2.1). Then, these two sub-hypergraphs are further bipartitioned in a recursive manner till the desired number of parts is obtained.

2.2.1 Net Removal and Splitting

There are two basic cutsize computation metrics in hypergraph partitioning, the cut-net (Equation 2) and the connectivity (Equation 3) metric. These two equations require different net removal and splitting techniques to correctly capture the corresponding cost metric as mentioned in [2]. In `rpPaToH`, these techniques are enhanced to support the replicated vertices. Like in `PaToH`, the cut-net removal technique in `rpPaToH` requires removal of the nets in the cut. However, there is one exception in `rpPaToH` which occurs when a net only connects nothing but replicated vertices. In such cases, this net is not in the cut and thus can be considered internal to any one of the parts, which is chosen to be the first part in `rpPaToH`.

For the cut-net splitting technique, we need to add the pins to replicas of the replicated vertices of the split nets in order to perform move and/or replication operations on these vertices in further bipartitionings. Likewise in the cut-net removal technique, the same exception occurs for the cut-net splitting technique which is handled in the same way.

2.2.2 Replication Amount Distribution

RB scheme consists of multiple bipartitionings. The given replication amount can be distributed in various ways to these bipartitionings. In `rpPaToH`, two alternative replication amount distribution schemes are considered. The first of them is *level-wise replication* scheme where the given replication amount is distributed evenly among the *levels* of the recursion tree of the RB scheme. The other one is *bisection-wise replication* scheme where the given replication amount is distributed evenly among the *bipartitionings* performed by the RB scheme. Note that `rpPaToH` may not be able to utilize all of the replication amount it is provided. The used replication amount (a percentage of the given replication amount) is given in the output statistics of the program.

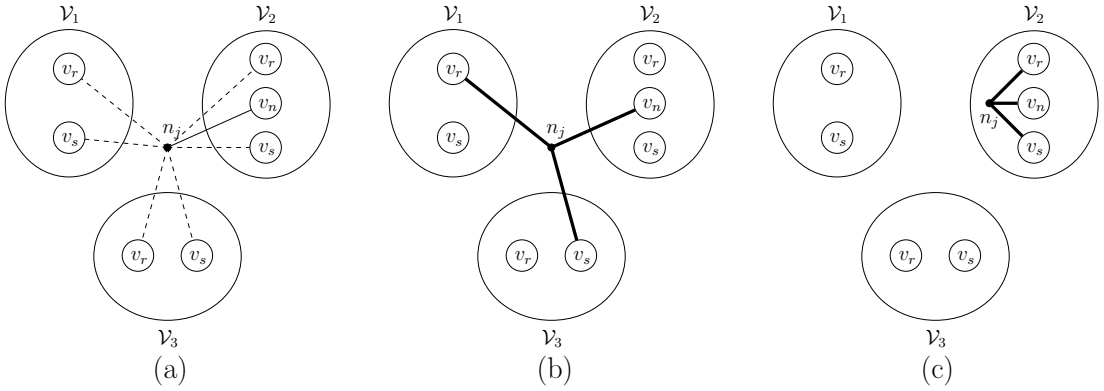

Figure 5: Pin selection alternatives for net n_j: (a) Initial partition before selection, (b) after the first selection alternative, $\lambda(n_j) = 3$, and (c) after the second selection alternative, $\lambda(n_j) = 1$.

2.3 Pin Selection

After obtaining a K-way replicated partition, for each net that connects at least one replicated vertex, it is necessary to decide which instance of the replicated vertices will be used by the nets that connect them. This is required for two basic reasons. First, the final cutsize computation requires to make a decision for selecting replicas for the nets that connect replicated vertices. Second, the modeled application may

require nets to make a choice about which parts the replicas will be used. This is done by using a set-cover-based heuristic (see [11] for details). Fig. 5 shows different pin selection alternatives for net n_j which connects two replicated vertices (v_r and v_s) and one non-replicated vertex (v_n).

3 Stand-Alone Program

`rpPaToH` is distributed as a stand-alone executable that accepts its parameters from command-line. It supports most of the parameters supported by `PaToH` besides the replication related parameters. Note that `rpPaToH` currently supports the replicated partitioning process for only the number of parts that are power of 2! `rpPaToH` can be run as follows:

```
> rppatoh <hypergraph-file> <number-of-parts> RR=<repl_ratio>
  [[parameter1] [parameter2] ....]
```

There are three mandatory arguments: the hypergraph file, the number of parts, and the replication ratio. The optional parameters must be provided beginning with two letter abbreviation, followed by an equal sign and the desired value of that parameter. See `PaToH` manual [3] for more information on these parameters.

The replication related parameters are as follows:

- *RR:* **R**eplication **R**atio. This value is multiplied by the total vertex weight to get the total replication amount. Note that this is not the percentage of the total vertex weight. Thus, for example if this parameter is given a value of 0.5 (RR=0.5), this means the replication amount will be 50% of the total vertex weight.

- *RD:* **R**eplication **D**istribution method. There are two alternatives for utilizing the given replication amount: *level-wise* and *bisection-wise*. Thus, this parameter can get two values: "level" and "bisection". In most cases, level-wise replication obtains better results and thus it is the default value.

- *RT:* **R**eplication **T**ype. This parameter chooses the replicated refinement heuristic to be used. It can be the basic rFM heuristic (indicated via "basic" value) or the gradient rFM heuristic (indicated via "gradient" value). In most cases, gradient rFM performs better and it is set to be the default replicated refinement heuristic.

- *WS:* **W**rite the **S**cheduling information to file or not. As mentioned in Section 2.3, replicated vertices bring the problem of replica selection for nets. This parameter is for writing the pin selection information using the proposed set-cover-based heuristic to the file. Note that different applications may require different pin selection strategies to best utilize the given replicated partitioning scheme. Thus, this information can be seen as a recommendation of `rpPaToH`. Also note that `rpPaToH`'s final cutsize computation relies on this heuristic. This is a boolean parameter (0/1) and its default value is 1.

```
> rppatoh ken-11.u 4 RR=0.25

 ++++++++++++++++++++++++++++++++++++++++++++++++++++++
 +++ PaToH v3.0 (c) Nov 1999, by  Umit V. Catalyurek +++
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++

 ######################################################
 ### rpPaToH v1.0 January 2011, by R. Oguz Selvitopi ###
 ######################################################
```

Table 1: `rpPaToH` parameter information

Parameter	Abbreviation	Type	Value
Replication-related Parameters			
`repl-ratio`	RR	float	must be provided (> 0.0)
`repl-dist`	RD	string	"level" (default), "bisection"
`repl-type`	RT	string	"basic", "gradient" (default)
`write-scheduling`	WS	int	0, 1 (default)
Various parameters heavily utilized in the replication process			
`write-info`	WI	int	0,1
`ref-passcnt`	RP	int	[1-]
`ref-maxnegmove`	RN	int	[5-]
`imbal`	IB	float	[0.00-]
`init-imbal`	II	float	[0.00-]
`final-imbal`	FI	float	[0.00-]
`fast-initbal-mult`	FB	float	[0.5-2.0]
Other parameters (supported by `rpPaToH` and also found in `PaToH`) UM, PQ, DP, VO, MT, FL, FM, CT, CK, ID, NM, FS, NT, IT, CP, CM, PA, IR, IA, TB, SD, A<p>, OD, UW			
Excluded parameters (found in `PaToH` but not in `rpPaToH`) BO, NI, BV, SV, NR, TR, DI, DF, RA, RL, RF, LC, HC, HM, HD, 2D, FX, SP, JN			

```
*********************************************************************************
Hypergraph : ken-11.u   #Cells : 14694   #Nets : 14694   #Pins : 82454
*********************************************************************************
 4-way partitioning results of rpPaToH:

 'Con - 1' Cost:  2588
 Replication usage:   20407 / 20613 (99.0 %)
 Part Weights   :
        Min =       24366 (5.2 %)
        Max =       26793 (4.2 %)
---------------------------------------------------------------------------------
I/O          :       0.016
I.Perm/Cons.H:       0.005  ( 1.4%)
Splitting    :       0.004  ( 1.1%)
Coarsening   :       0.241  (61.5%)
Partitioning :       0.038  ( 9.8%)
Uncoarsening :       0.074  (18.8%)
Finalizing   :       0.005  ( 1.3%)
Total        :       0.392 sec.
---------------------------------------------------------------------------------
```

In the example above, `rpPaToH` is run to partition and replicate the hypergraph file "ken-11.u" to obtain an 4-way replicated partition with 25% replication amount. This given replication ratio (RR) parameter is multiplied with the total vertex weight to obtain the replication amount (so if the total vertex weight is 1000, the replication amount will be 1000*0.25=400 if RR=0.25). The output gives very basic information about the replicated partitioning process and the obtained replicated partition. According to the output, the cutsize is 2588 (with the connectivity metric). As mentioned, `rpPaToH` may not utilize all of the given replication amount. For the given replication amount of 20613, `rpPaToH`

utilized 99.0% of this amount. Next line gives information about the load imbalance for minimum and maximum weighted parts, which are 5.2% and 4.2%. Then, somewhat detailed timing information is given. It took 0.392 seconds for `rpPaToH` to obtain a 4-way replicated partition of the given hypergraph file "ken-11.u".

A few more examples that exploit the replication related parameters are given below:

```
> rppatoh ken-11.u 32 RR=0.25 RT=gradient RD=bisection WI=1 WS=0
> rppatoh ken-11.u 1024 RR=1.75 RT=basic RD=level WS=1 RP=3 IB=0.20 MT=11 PA=3
```

Table 1 gives information about the replication related parameters, their default values, some `PaToH` parameters that are also heavily utilized in the replication process, the parameters supported by `rpPaToH`, and the excluded `PaToH` parameters. For `PaToH` parameters that are also supported by `rpPaToH`, the user is urged to see Section 5 of `PaToH` manual [3] as they can have a great impact on the quality of the replicated partitions obtained by `rpPaToH`.

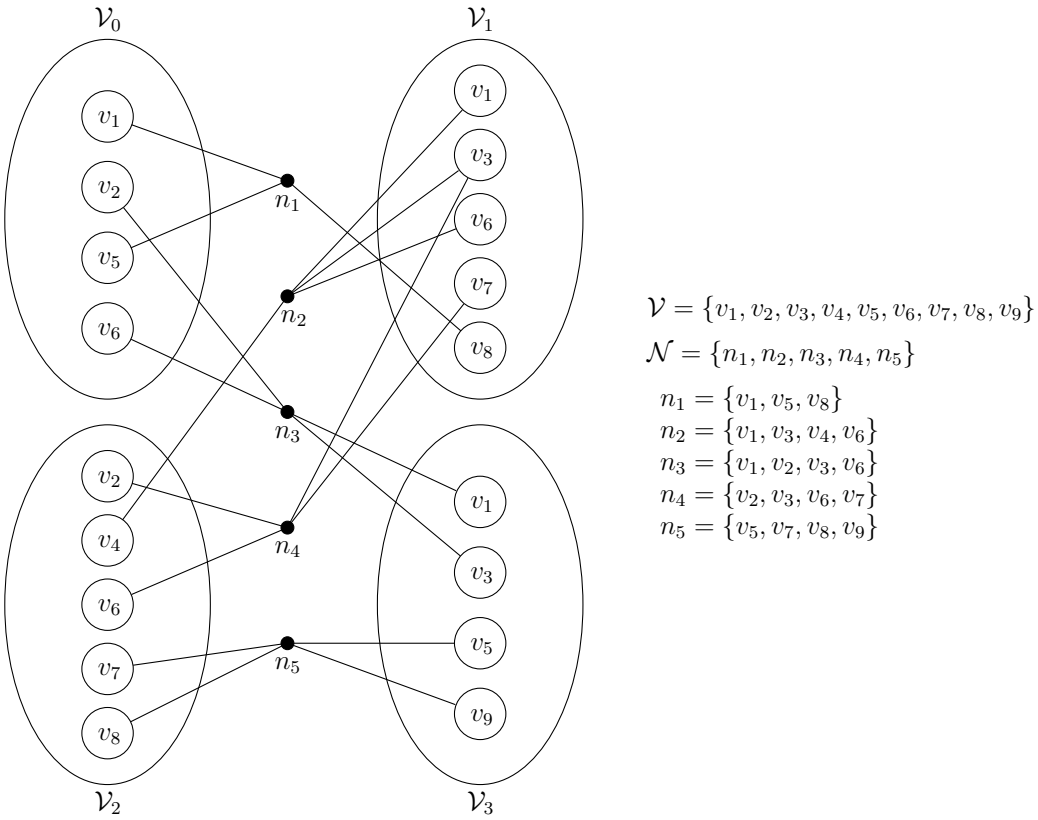

Figure 6: A 4-way replicated partitioned hypergraph with nine vertices and five nets.

3.1 Input and Output File Format

As input file format, `rpPaToH` uses the same format as `PaToH` (see [3] for `PaToH` manual). However, `rpPaToH`'s output file format and the number of output files it produces is different from those of `PaToH`'s. Fig. 6 shows an example of a hypergraph that replicated and partitioned into four parts. The Fig. 7(a) is the file for the input hypergraph.

`rpPaToH` produces two output files:

- *Partitioning Information:* This file indicates the replicated partitioning information about the vertices and which parts they belong to (this file is produced by default, it can be turned off via setting WI=0). If the original hypergraph file name is "hygr-file", then the file that includes the

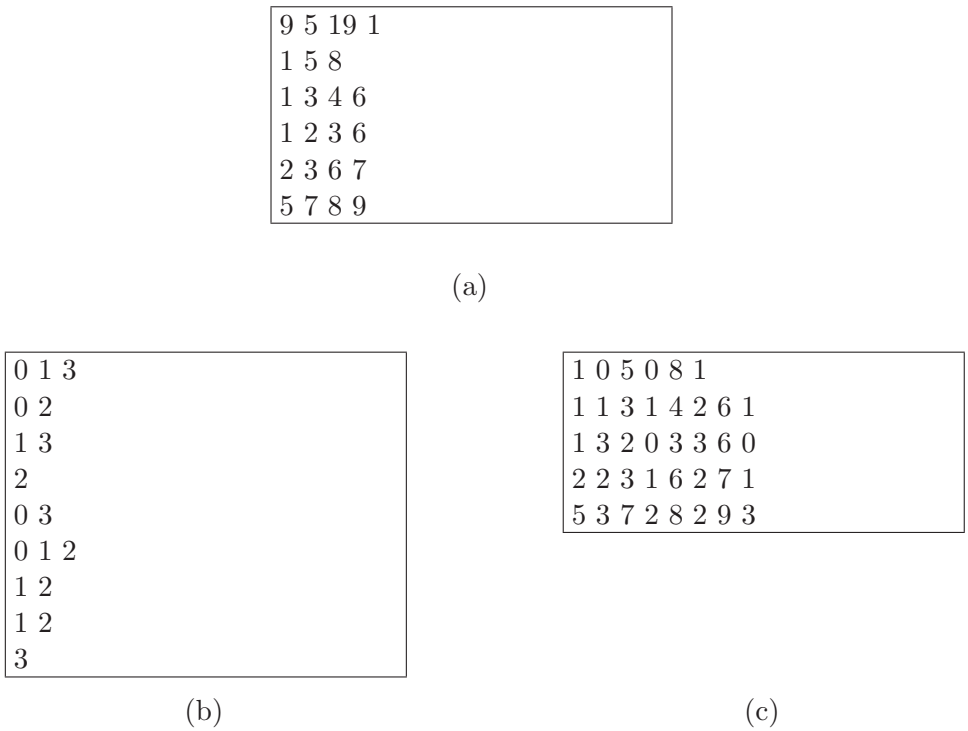

Figure 7: (a) The input hypergraph file corresponding to the hypergraph illustrated in Fig. 6, (b) Output part information of the replicated partitioned hypergraph in Fig. 6, (c) Output scheduling (pin selection) information of the replicated partitioned hypergraph in Fig. 6.

replicated partitioning information becomes "hygr-file.part.$<K>$" where K is the number of parts. Each line in this file indicates a vertex and lines are numbered with respect to vertices (if there are n vertices, there will be n lines). Each line consists of part numbers separated by a single space. Fig. 7(b) shows the replicated partitioning information of the hypergraph given in Fig. 6. For example, line 6 corresponds to v_6, whose entries are "0 1 2". It means v_6 is replicated to parts 0, 1, and 2, as seen in Fig. 6.

- *Scheduling Information:* This file indicates the pin selection information for the nets about which replicas of the replicated vertices they use (this file is produced by default, it can be turned off via setting WS=0). If the original hypergraph file name is "hygr-file", then the file that includes the replicated partitioning information becomes "hygr-file.schedule.$<K>$" where K is the number of parts. Each line of this file corresponds to a net and a line's contents are composed of the vertices connected by the corresponding net and their selected part information. This information is given in tuples, i.e., it consists of $<v_i, K>$ pairs where K is the part the v_i is assigned to. Fig. 7(c) shows the scheduling information of the replicated partitioned hypergraph given in Fig. 6. For example, line 5 corresponds to n_5, whose entries are "5 3 7 2 8 2 9 3". It means n_5 uses v_5 from part 3, v_7 from part 2, v_8 from part 2, and v_9 from part 3, as seen in Fig. 6.

References

[1] Cevdet Aykanat, B. Barla Cambazoglu, and Bora Uçar. Multi-level direct k-way hypergraph partitioning with multiple constraints and fixed vertices. *J. Parallel Distrib. Comput.*, 68:609–625, May 2008.

[2] Umit Catalyurek and Cevdet Aykanat. Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. *IEEE Trans. Parallel Distrib. Syst.*, 10:673–693, July 1999.

[3] Ümit V. Çatalyürek and Cevdet Aykanat. Patoh: partitioning tool for hypergraphs. Technical report, Department of Computer Engineering, Bilkent University, 1999.

[4] M. Enos, S. Hauck, and M. Sarrafzadeh. Evaluation and optimization of replication algorithms for logic bipartitioning. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 18(9):1237 –1248, September 1999.

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

[6] James Hwang and Abbas El Gamal. Optimal replication for min-cut partitioning. In *Proceedings of the 1992 IEEE/ACM international conference on Computer-aided design*, ICCAD '92, pages 432–435, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press.

[7] George Karypis and Vipin Kumar. Multilevel k-way hypergraph partitioning. In *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, DAC '99, pages 343–348, New York, NY, USA, 1999. ACM.

[8] C. Kring and A.R. Newton. A cell-replicating approach to minicut-based circuit partitioning. In *Computer-Aided Design, 1991. ICCAD-91. Digest of Technical Papers., 1991 IEEE International Conference on*, pages 2 –5, November 1991.

[9] Roman Kužnar, Franc Brglez, and Baldomir 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, pages 238–243, New York, NY, USA, 1994. ACM.

[10] Lung-Tien Liu, Ming-Ter Kuo, Chung-Kuan Cheng, and T.C. Hu. A replication cut for two-way partitioning. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 14(5):623 –630, May 1995.

[11] R. Oguz Selvitopi, Ata Turk, and Cevdet Aykanat. Replicated partitioning for undirected hypergraphs. *J. Parallel Distrib. Comput.*, 72(4):547–563, April 2012.