2015 IEEE International Parallel and Distributed Processing Symposium Workshop

K-Ways Partitioning of Polyhedral Process Networks: a Multi-Level Approach

Riccardo Cattaneo, Mahdi Moradmand, Donatella Sciuto, Marco D. Santambrogio
Politecnico di Milano, Milan, Italy
{riccardo.cattaneo, donatella.sciuto marco.santambrogio}@polimi.it
mahdi.moradmand@mail.polimi.it

***Abstract*—Process Networks(PNs)-based models of computation have proven as a successful framework for describing multiple kinds of applications in the Reconfigurable Hardware (RH) domain. Due to their intrinsically parallel and reactive behavior, and well-known techniques to automatically manipulate some of their instances, they are very amenable to Field Programmable Gate Arrays (FPGAs). One problem associated with PNs is that the number of nodes is usually proportional with the parallel portions of computation, and a tool to automatically map tasks to FPGAs is required when multiple FPGAs are employed to improve performance (via increased parallelism). While it is possible to solve this problem in an exact manner via dynamic programming approaches, this is not the case when practical graphs are under examination, i.e. graphs with potentially thousands nodes. In this work we extend a well-known graph partitioning technique, namely Multi-Level K-ways partitioning algorithm, in order to cope with such scenario.**

I. Introduction

General Partitioning Problem (GPP) plays a major role in data analysis, machine learning, computer science, engineering, and related fields. Most graph partitioning algorithms optimize a ratio between the cut and the size of the partitions, leading to an NP-Complete problem [1]. However, this makes it impractical to partition large networks, which is the reason why an entire field arose to cope with this problem, namely Approximated Graph Partitioning.

Given an un-weighted graph G with V nodes and E edges and given a number K, the Graph Partitioning Problem is to divide the V nodes into K parts such that the number of edges connecting nodes in different parts is minimized given the condition that each part contains roughly the same number of nodes. If the graph is weighted, i.e. the nodes and edges have weights associated with them, the problem considers the sum of the weights of the edges connecting nodes in different parts, while roughly keeping the weights in each partition the same. The problem can be reduced into one where the graph is split into N parts and then merging these nodes to build a smaller graph with fewer nodes, intrinsically easier to partition in the so called initial partitioning phase. In the approximated version of this problem, adequate (possibly random) heuristics are employed to do so [2], [3].

Among the many successful heuristics for partitioning large, highly interconnected graphs, the Multi-Level Graph Partitioning approach stands apart for both the average quality of the result (i.e.: difference in the resulting final partitions and cut sizes and those generated via the solution of an equivalent optimal problem) and the execution time, usually confined to few minutes on large instances (millions of nodes and arcs) on commodity-level machines. In this approach the graph is recursively contracted to create smaller and smaller graphs which should reflect the same basic structure as the input graph [4]. After that, an initial partitioning algorithm is applied to the smallest graph, in order to obtain a seeding partitioning. Then, each partition of this initial partitioning is further de-contracted (un-coarsening) and, at each level, a local search method is used to improve the subsequent partitioning (decontraction/uncoarsening step) induced by the coarser level. The Fiduccia-Mattheyses heuristic for refining the partition after initial partitioning step is employed in this (and other) work to improve the edge cut [3].

Although several successful Multi-Level partitioners have been developed in the last two decades, to the best of our knowledge, none cope with a specific scenarios. Suppose to have a graph (G,V) representing an application. Each node (which we will call *process*) represents a potentially recurrent, potentially periodic task, while edges (which we will call *channels*) represent FIFOs between processes. In this scenario, each process is further characterized by an amount of resources required in order to implement such process p on an FPGA (R_p), and channels are characterized by an amount of sustained data transferred. Additionally, we want to fully exploit this model to compute (i.e. execute processes and data transfers) in parallel, on a multi-FPGA system. In this case, between each FPGA involved in the system, only B_{max} data can be transferred each unit of time, and each FPGA has an amount of resource R_{max}. This is a basic yet accurate representation of the common scenario where a multi-FPGA is designed. In this case, partitioning of the network (for mapping purposes) must take into account how many processes can run onto a single FPGA, and which nodes to map onto which FPGA, in order to cope with given resource constraints. First constraint is related to cut size between each pair of final partitions. In order to meet this constraint we must consider the cut size not only in the original graph but also between each final partition, so that the cut size between each pair of partitions is less or equal to B_{max}. The second constraint is related to resources consumed by each node (and eventually, by each partition). These two constraints, along with the problem formulation, makes up for the novel contribution of this work.

In this work we present an algorithm that seeks and finds the

978-1-4673-7684-6/15 $31.00 © 2015 IEEE
DOI 10.1109/IPDPSW.2015.17

solution of the Approximated Graph Partitioning Problem in order to satisfy two major constraints that arise when mapping process networks (like Polyhedral Process Networks or Khan Process Networks, to name just a few) onto FPGAs. We use a classical approach to ratio problems where we repeatedly ask whether the solution is greater than or less than some constant which refers to our constraints, based on the Multi-Level Approach.

The rest of the dissertation proceeds as follows. As the state of art of GPP is vast, Section II presents a thorough review of it in order to understand where this work fits and what problem we addressed. Section III reviews, in particular, the basics of Multi-Level, K-ways partitioning. Section IV describes how we extended previous work in order to cope with the mapping problem at hand. Section V presents experimental results, and the dissertation is ended by Section VI with comments and future work.

II. Related Work

There has been a large amount of research on GPP so that we refer the readers to [5]–[7] for most of the material. Since finding an optimal partitioning is NP-Complete (and is a well-known, solved problem [8]), one is forced to set up for approximation algorithms in order to find a solution (even though non optimal, in the general case) in a practical amount of time. The part of the investigation in this area concentrates on approaches to solving the Two-Ways Partitioning Problem (TWPP) for bi-sectioning the graph (partitioning the graph into exactly two parts), which is also a NP-Complete Problem. One of the primary attempts and maybe the most well-known heuristic algorithm for partitioning graphs was described in [9], which takes two separate sets as an initial solution of the problem, and trades pairs of nodes between them in order to obtain a candidate solution.

Branch and Bound($B\&B$) strategy solves the partitioning in the case of $K = 2$, for general weighted graphs have also been presented in [10]. Yan and $Hsiao$ have presented a fuzzy clustering algorithm to solve the GBP and apply it to Circuit Partitioning [11]. Other authors have been presented methods based on $Genetic\ Algorithms$ [12], $Divide\ \&\ Conquer$ approximation algorithms [13] and even $Ant\ Colony$ optimization [14]. $Linear\ programming(LP)$ methods became more popular after being shown that they were able to find better cuts over KL.

Spectral methods additionally got vastly used, since they were faster and produced great results. These are focused on the computation of eigenvectors of the adjacency matrix. Several works have used such techniques like [15]–[17]. As an alternative, Multi-Level algorithms for partitioning graphs were initially presented by [18] and [19]. Regularly such Multi-Leveling systems match pairs of adjacent nodes to define new merged graphs and recursively iterate this procedure in order to make a graph with arbitrary nodes. The coarsest graph is then partitioned and the partitions is refined on all the graphs back to the original graph.

Besides to heuristics and approximate algorithms for solving the GPP, many authors have analyzed the lower bounds of known algorithms and in special case of graphs(e.g. [20] and [21]).

Since GP is a hard problem, practical solutions are focused on heuristics. There are two broad categories of methods, Local and Global which we consider here in greater detail.

A. Local Search Methods

The partitioning can be described as breaking a graph into sub-graphs and recursively do it in this way under some constraints in order to make a graph with arbitrary nodes or less than a specific marginal number. Here we will describe this problem by a method known as iterative improvement.

The idea behind iterative improvement is to begin with an initial solution, and make a new solution iteratively until we have a solution that is "good enough". Optimality is measured with respect to a given goodness criteria.

Most iterative improvement techniques are greedy. In a greedy algorithm, the new solution is accepted only if it is better than the old one. Non-Greedy methods (like: hill-climbing algorithms) will sometimes accept a solution that is worse than the existing solution, the reason being that hill-climbing algorithms are used is to avoid getting trapped in local minima. A hill-climbing algorithm can sometimes climb out of a local minimum and find a better solution by temporarily accepting a solution that is worse than the existing solution.

Two well known local methods in the context of iterative algorithms for GPP are are *Kernighan-Lin* and *Fiduccia-Mattheyses* algorithms, which were the first two-way cuts heuristics adopted by local search strategies. Their significant disadvantage is the arbitrary initial partitioning of the node set, which might have a negative affect on final solution quality. Broadly speaking, given a partition of a graph, a local search algorithm tends to enhance an objective function by moving nodes between partitions. These algorithms let a node move at most once during one iteration of the algorithm. More costly local search algorithms such as Tabu Search eliminate this restriction as far as possible, i.e. a node can be moved different times during one iteration. However, today majority of the methods for enhancing a given partition are variations of the FM algorithm.

1) Kernighan-Lin Algorithm: The $Kernighan-Lin(KL)$ Algorithm is one the most popular algorithm for the TWPP. *KL* algorithm works as follows:

1) The initial partition is generated Randomly. Create two sub-graphs G_1, and G_2. If the graph has N nodes, the first $\frac{n}{2}$ are assigned to G_1, and the rest are assigned to G_2.
2) A solution is acceptable only if both sub-graphs contain more or less the same number of nodes.
3) The goodness of a solution is equal to the number of graph edges that are cut between partitions.
4) The technique for generating new solutions from old solutions is to select a subset of nodes from G_1, and a subset of nodes from G_2 and swap them. To maintain acceptability, we always select two subsets of the same size.

KL drawbacks are:

1) handling of unit node weights only,
2) handling of exact bi-sections only,
3) time complexity of a pass is high, $O(n^3)$.

2) Fiduccia-Mattheyses Algorithm: There have been great improvements made to the KL algorithm. The most imperative change is a slight adjustment of the algorithm and the decrease in running time that was provided by $Fiduccia - Mattheyses(FM)$ [22]. $Fiduccia$ and $Mattheyses$ suggested following modifications:

1) Only one node is moved at a time,
2) The consecutive moves are made in the opposite directions,
3) The algorithm maintains a sorted list of candidate interior nodes for moving to the other sub-graph, and updates it after each move.

They succeed to decrease the complexity for a single pass to $O(n)$ by using modern data structures. Like the KL strategy, the FM strategy performs passes where each node is moved at most once, and the best bi-section observed during an iteration is used as input for the next iteration. In any case, instead of selecting pairs of nodes, the FM method chooses just single nodes for moving. *Fiduccia-Mattheyses* balanced the algorithm and adopted adequate date structures such that the asymptotic running time of their local search algorithm is reduced to linear time $O(n)$.

B. Global Search Methods

Global search relies on the properties of the entire graph and do not rely on an arbitrary initial partition.

One such technique (specifically aimed at solving the TWPP) is to formulate it as a quadratic optimization problem. However, due to the nature of the optimization problem, realistic graphs still result unmanageable. For this reason, a class of graph partitioning methods, called Spectral Methods – the most common example of which is Spectral Partitioning, where a partition is derived from the analysis of the spectrum of the adjacency matrix – relax this optimization. Spectral techniques have been enhanced by several works like [23], [24]. Unfortunately, to the best of our knowledge, none of the previous methods contemplate the partitioning of applications in the presence of simultaneous resource and bandwidth problem constraints (or the equivalente in the respective formulations).

Other methods contemplate Multi-Way Spectral Bisection Algorithm and Parallel Graph Partitioning [25], [26] and Multi Level, K-Ways Partitioning. As this last technique is the basis for this work, we detail the inner workings in the following Section.

Previous work – as presented in this brief recall of the state of art – focuses on heuristically minimizing the cut size associated to the partitionings found. *However, as the techniques focus on such minimization, to the best of our knowledge, none address the problem that we approach in this work: a cut size minimization algorithm with novel constraints tightly related to the reconfigurable hardware domain.*

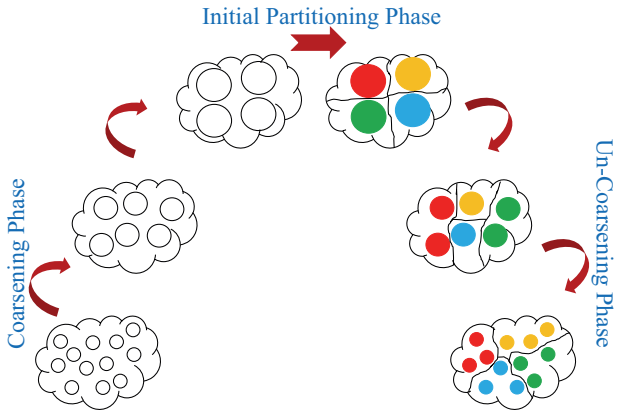

Fig. 1: Multi-Level Scheme [27]

III. Multi-Level, K-ways Partitioning

[19] formulated this strategy as it is known now. The Multi-Level approach to GP comprises of three main phases, which are reported in Figure 1.

In the **contraction (coarsening)** phase, a hierarchy of graphs is created. The most common way to build this hierarchy is to iteratively identify matching $M \subseteq E$ and contract the edges in M. Contraction should rapidly reduce the size of the input and each computed level should reflect the global structure of the input network.

Contraction is halted when the graph is small enough to be directly **partitioned** using some costly other algorithm like the ones described in the previous Section (such as KL, FM algorithms and spectral partitioning).

In the **un-coarsening** phase, matching nodes and arcs – which had been previously been merged together in the coarsening phase – are iteratively un-coarsened.

During un-contraction of matching graphs, a local improvement algorithm moves nodes between partitions to enhance the cut-size or balancing constraint. Generally variants of the *FM* algorithm are used. The vision behind this technique is that a good partition at one level will also be a good partition on the next finer level, so that local search will rapidly find a good solution. Moving a node on a coarse level hierarchy typically corresponds to the movement of a whole set of node movements of the finest level of the hierarchy. Intuitively, the Multi-Level scheme has a global view on the optimization problem on the coarse levels of the hierarchy and a very local view on the finest levels with respect to the primary one.

[15] is the first work to report a linear time $O(n)$ implementation of this scheme to obtain $K - Partitions$ (using Recursive Multi-Level Bi-section only on the coarsest level and a direct $K - Way$ local search algorithm). A variant of the Multi-Level algorithm has been proposed in [2]. Their $n-level$ approach is based on the extreme idea of contracting only one single edge between two consecutive levels of the Multi-Level hierarchy. During un-coarsening, local search is done highly localized around the un-constructed edge. Using complicated data structures their algorithm requires sub-linear time on real graphs.

Compared with Multi-Level Spectral Bisection, Multi-Level K-Way partitioning is usually two orders of magnitude faster, and produces partitioning with generally smaller edge-cuts. This is why we employed this basic scheme for the implementation of our partitioning algorithm, which is described in Section IV.

IV. Algorithm's Internals

The proposed method is based on a variant of the aforementioned Multi-Level, K-Ways Partitioning (MLKWP) scheme.

In the proposed algorithm, the input graph is coarsened to a parametrized size (default is 100). However, it is not un-coarsened and refined back to the original graph in just one step. Rather, it is un-coarsened up to a certain intermediate level and then coarsened back to the lowest level if needed. This process of un-coarsening and refining up to an intermediate level and coarsening again to the lowest level is repeated a number of parametrized times, depending on whether we are already meeting the constraints or not.

At each iteration, we generate different intermediate clusterings, that are compare *a posteriori* using a goodness function; the best (i.e. the one that is nearest to meeting the constrains) is chosen as the "correct" intermediate un-coarsening candidate. This step incentives rapid convergence while accounting for broad exploration of different clusterings.

After the coarsening phase, we try to meet the K different partitions with the help of initial partitioning phase.

A. Coarsening Phase

In the coarsening phase we use three type of different matchings in order to better explore different results given multiple search strategies:

- *Random Maximal Matching*,
- *Heavy Edge Matching*,
- *K-Means Matching*.

Random Maximal Matching Nodes of a graph are randomly visited. If there is a node u which is not matched, then one of its un-matched neighboring nodes is randomly selected. Two nodes are said to be adjacent if there exists an edge that is incident to those two nodes. If there exists such a node v, the edge (u, v) is included in the matching and the nodes u and v are marked as matched. Node u remains un-matched in the random matching if there is no un-matched adjacent node v. The goal in the GP is to minimize the sum of the weights of the edges between the nodes on the boundary of the parts of the graph. Using a randomized algorithm, a maximal matching can be found so a randomized matching method may not always produce satisfactory results for every graph. In order to decrease the edge cut value, heavy edge matching [3] can be used.

Heavy Edge Matching As the name suggests, the edges are sorted according to their weights and matching begins by selecting the heaviest edge. All the edges are visited in descending order and edges with un-matched end points are selected. This heuristic is used when the graph size has been reduced substantially so that not much work is done in sorting the edges.

K-Means Matching Clusters are formed on the basis of their weight; a subset of near nodes is chosen accordingly.

The main objective and theme of this method is to divide the graph into smaller partitions and based on the concept that it first divides the problem into multiple sub-partitions by dividing the total number of the nodes by the number of sub problem you want and assign the nodes to the partitions which is near to the specific cluster [28].

We use in this work all three heuristics algorithms (*Random*, *HEM*, *K-Means*) to get the matching. These heuristics are employed at different times, multiple times, in order to find the best matching for the given graph. Each time we compare the results of the three heuristics with each other and choose the best one.

Once we obtain the matching of nodes to coarsened graphs, we create a map from the nodes in the un-coarsend graph to those in the coarsened graph. Then, using the matches and the map, the coarser graph is built, ready for the next iteration of the coarsening step. Thea adjacency matrix of the coarsened graph is adjusted according to the new incidence between coarser nodes in the graph. The edge weights, in particular, are all copied over but when the matched nodes have a common neighbor: in this case weights are merged into one and the new edge has a weight equal to the sum of the weights of the merged edges. Similarly, the new node gets the sum of the weights of the merged nodes. Any duplicate edge resulting from the process is merged together with their weights added.

The coarsening phase of our algorithm continues until few nodes remain (for example 100 nodes – this is a parameter in our implementation). The resultant most coarsened graph is considered an initial partitioning for the initial partitioning phase.

B. Initial Partitioning Phase

After reducing the original graph into multiple sub-partitions we produce an initial partitioning of it, with a number of partitions much lesser than the required one.

We adopt a greedy approach, as it is a heuristic that usually yields good results. Specifically, we partition the graph in such a way that we have a balanced number of resources in each part (note how balancing resources is not a priority in our case, while meeting the resource and bandwidth constrain is). After that, we check the bandwidth between each pair of partitions and use the FM algorithm to meet the bandwidth constraint.

We start off with the heaviest nodes. After finding the heaviest one, we'll take it in the first partition among K partitions available and add its neighbors (which are connected via edges to this node) as long as the total number of resources assignable to each partition (R_{max}) is not violated. After this we apply the same for the other partitions as far as all nodes assigned to exactly one partition. Since this method is sensitive to the initial node selection, the whole process is repeated with a parametrized number of randomly chosen initial nodes (10 is default). Since the coarsest graph is no more than a few hundred nodes (100 is our default), running this algorithm K times does not add much to the total partitioning cost. The final partitioning that gives the best cut-size is returned.

After this allocation we pay attention to the remaining nodes (if any) which are not assigned to any partitions. First we try to put each remaining node in accordance to its resources to the first partition which has biggest free space for that node and do it for all remaining nodes. If after this step there are still nodes to assign, we assign each node to the partition which has the biggest free space even though this implies violating the R_{max} constraint. After this step we check the Bandwidth between each pair. If it doesn't meet the constraints we use an *FM*-based algorithm to minimize it as far as possible. Partitions will be changed and nodes will move between partitions as far as constraints met.

We then un-coarsen as necessary, as described in the next Section, in order to obtain the right number of partitions, each meeting the constraints.

C. Un-Coarsening Phase

During the un-coarsening (refining) phase, the initial partition of the coarsest graph is projected onto the lower level, finer graph. This procedure is repeated until a partition is projected onto the top level graph and is refined to obtain the final partition and cut-size and resource allocation for the graph. The mapping vector is used to project the coarse graph partition onto the finer graph. But if we do not met constraints, we go back to coarsening phase and then partitioning phase (randomly), cyclically. If after a predetermined number of iterations a feasible partitioning is still missing, a message will signal that partitioning with these constraints is either impossible or we have to give the tool more time (i.e.: iterations) to compute such solution.

V. Experimental Setting

We compare *METIS* and *GP* using random generated graphs. We employ particularly small instances in the following part of this Section in order to visualize the different behavior of the two tools when partitioning the given networks. In all cases, these graphs represent Process Networks generated via suitable tools. Each *process* (i.e.: node) is characterized by an amount of resources required to implement such process on an FPGA (only one resource is considered at this time, for example LUTs) and each *channel* (i.e.: edge) is characterized by an amount of bandwidth consumed. Only bandwidth outgoing from and incoming to different partitions consume bandwidth – we assume that there is enough bandwidth on the FPGA to sustain enough computation between nodes belonging to the same partition (i.e.: FPGA).

We synthetically generated few graphs with the following goal in mind: *to demonstrate that GP can always partition the given network while respecting resource and bandwidth constraints (or fail while doing so) while METIS always partitions, regardless of said constraints*. Graphs are represented as incidence matrices, and are given as inputs to MATLAB.

$Table(5.1) \sim Table(5.3)$ compare the results obtained running both *METIS* and *GP*. Various *GP* parameters are used across all experiments . For *METIS*, we used the default parameter values and decode the results in $Matlab$ in order to compare the results with *GP*.

We compare:

1) Local Edge Cut (i.e. bandwidth insisting between each pair of partitions),
2) Maximum Resources Allocation (i.e. the maximum amount of resources consumed by all partitions),
3) Algorithm's Execution Runtime,
4) Global Edge Cut Sum.

The machine we employed is a 2.53 GHz Intel(R) Core(TM) i5-M 460 CPU with 8GB RAM running $Ubuntu14.0464bit$. The code runs under $MATLABR2013a$. $METIS5.1.0$ is used for comparisons. We refer to the Graph Partitioner of this work as *GP*.

A. Experiment 1

We consider a graph with 12 nodes and 33 edges for the first experiment. Maximum bandwidth constraint is 16 units. Maximum resources constraint is 165 units.

As it is possible to see in experimental table I – red font – *METIS* violates both constraints while *GP* meets both of them. However, the size of the cut is slightly bigger for *GP*, which is a consistent result as *METIS* tries to minimize the overall cut, but generally violating bandwidth and/or resource constraint. *GP* does not violate any partition-to-partition bandwidth constraint, but it fails at *globally* minimizing the edge cut. Actually, it does, under the bandwidth constraint.

Figure 2 reports the unpartitioned graph (radius of nodes proportional to weight), Figure 3 the same graph with weight and edges allocation, Figure 4 partitioning with *GP*, and Figure 5 partitioning with *METIS*.

B. Experiment 2

In $Table\ II$ we consider a graph with 12 nodes and 30 edges for the second experiment. We apply the following constraints: 25 for bandwidth and 130 for resources. *METIS* violates bandwidth while meeting (incidentally) resources, while, again, *GP* meets both of them.

Incidentally, the local refinement strategy employed translates, in this graph, in a better overall global cut, as reported in Table II.

Figure 6 reports the unpartitioned graph (radius of nodes proportional to weight), Figure 7 the same graph with weight and edges allocation, Figure 8 partitioning with *GP*, and Figure 9 partitioning with *METIS*.

C. Experiment 3

In the last experiment, whose data are shown in $Table\ III$, we consider a graph with 12 nodes and 32 edges. I apply the following constraints: 20 for bandwidth and 78 for resources. *METIS* violates bandwidth while meeting (incidentally) resources, while, again, *GP* meets both of them.

Figure 10 reports the unpartitioned graph (radius of nodes proportional to weight), Figure 11 the same graph with weight and edges allocation, Figure 12 partitioning with *GP*, and Figure 13 partitioning with *METIS*.

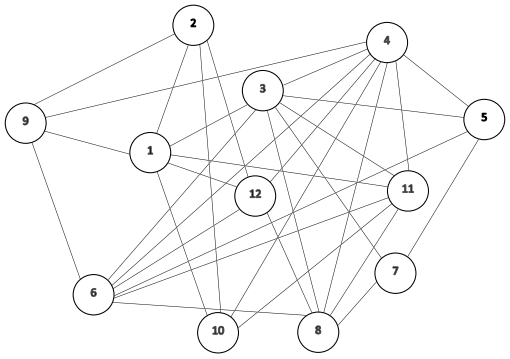

Fig. 2: Un-partitioned sample graph 1 before weighting and resource allocation.

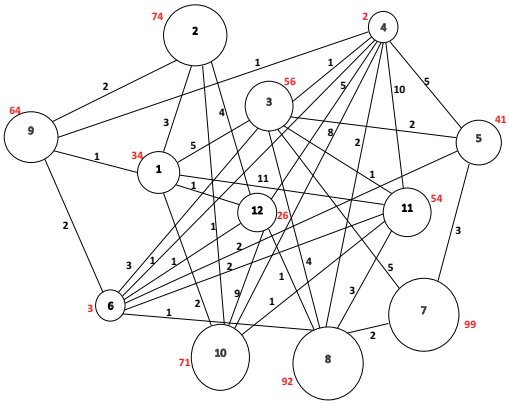

Fig. 3: Un-partitioned sample graph 1 after weighting and resource allocation

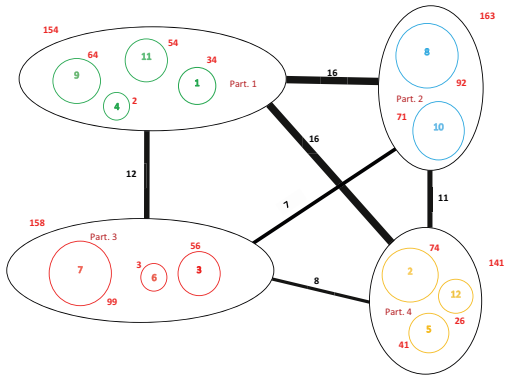

Fig. 4: Partitioning of the sample graph 1 with *GP* algorithm, both constraints are met, constraints are : bandwidth = 16 and resources = 163.

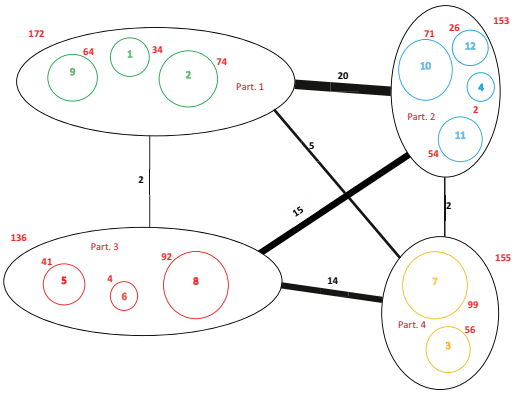

Fig. 5: Partitioning of the sample graph 1 with *METIS* algorithm, both constraints are violated, constraints are : bandwidth = 16 and resources = 163.

D. Summary

As it is possible to see from experimental Figures and summary Tables, *GP* can always (on the test cases) partition without violating given constraints, which is not guaranteed to be true with *METIS*. Additionally, in our test cases the increase in cut size is near to negligible; however, this might not be the case if we employed stricter constraints.

VI. Conclusions

We presented a novel approach to partitioning a process network in the presence of simultaneous bandwidth and resource constraints, based on the Multi-Level, K-Ways approach already known in literature. We developed a tool that extends $METIS$ in that it copes with situations where partitioning must happen within precise bandwidth and resource constraints. Future work contemplates the test of this system on actual multi-FPGA based systems where the mapping of potentially large application graphs (process networks) is a difficult task to do by hand.

References

[1] E. Mezuman and Y. Weiss, "Globally optimizing graph partitioning problems using message passing," in *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*, 2012.

[2] V. Osipov and P. Sanders, "n-level graph partitioning," *In Proceedings of the* 18^{th} *European Conference on Algorithms: Part I*, vol. 6346, pp. 278–289, 2010.

[3] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *Society for Industrial and Applied Mathematics Journal of Scientific Computing*, vol. 20, pp. 359–392, 1998.

[4] P. Sanders and C. Schulz, "Distributed evolutionary graph partitioning." SIAM.

[5] P. O. Fjällström, "Algorithms for graph partitioning : A survey," *Linköping Electronic Articles in Computer and Information Science*, vol. 3, 1998.

[6] K. Schloegel, G. Karypis, , and V. Kumar, "Graph partitioning for high performance scientific simulations," *In The Sourcebook of Parallel Computing*, pp. 491–541, 2003.

[7] C. Bichot and P. Siarry, *Graph Partitioning*, 2011.

[8] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1979.

[9] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: application in vlsi domain," *Proceedings of the* 34^{th} *annual Design Automation Conference*, p. 526ñ529, 1997.

[10] C. Roucairol and P. Hansen, "Cut cost minimization in graph partitioning," *Numerical and Applied Mathematics*, pp. 585–587, June 1989.

[11] J. Yan and P. Hsiao, "A fuzzy clustering algorithm for graph bisection," *Information Processing Letter*, vol. 52, December 1994.

[12] T. N. Bui and B. R. Moon, "Genetic algorithm and graph partitioning," *IEEE Trans-actions on Computers*, vol. 45, no. 7, July 1996.

[13] S. R. G. Even, J. Naor and B. Schieber, "Fast approximate graph partitioning algorithms," *In SIAM Journal on Computing*, vol. 28, pp. 639–648, July 1997.

[14] T. N. Bui and L. C. Strite, "An ant system algorithm for graph bisection," *In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '02), W. B. Langdon et al. (Eds.)*, pp. 43–51, 2002.

[15] B. Hendrickson and R. Leland, "An improved spectral graph partitioning algorithm for mapping parallel computations," *SIAM Journal on Scientific Computing*, vol. 95, pp. 452–469, 1995.

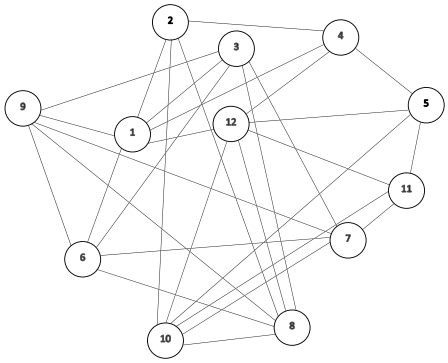

Fig. 6: Un-partitioned sample graph 2 before weighting and resource allocation.

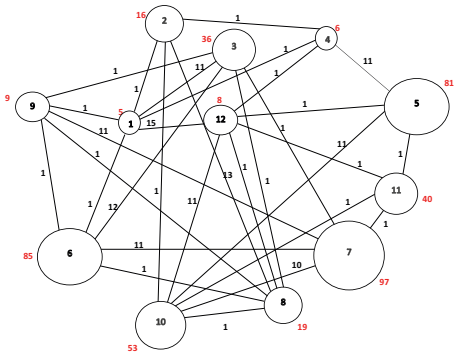

Fig. 7: Un-partitioned sample graph 2 after weighting and resource allocation.

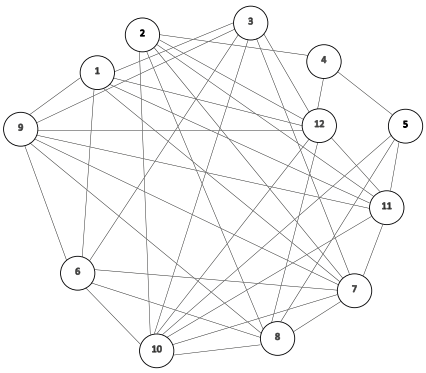

Fig. 10: Un-partitioned sample graph 3 before weighting and resource allocation.

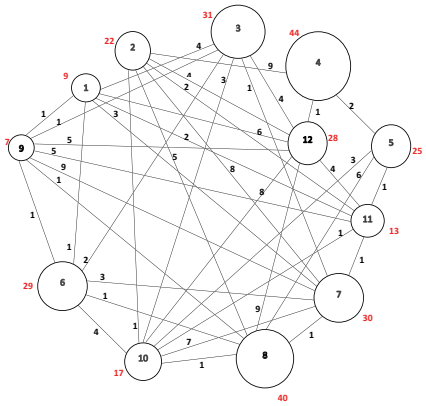

Fig. 11: Un-partitioned sample graph 3 after weighting and resource allocation.

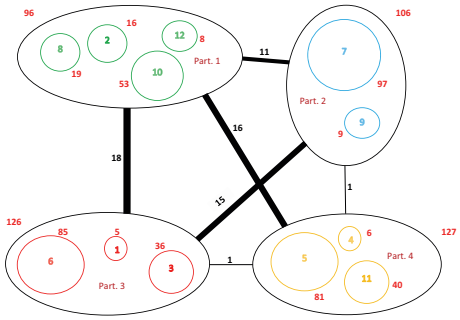

Fig. 8: Partitioning of the sample graph 2 with *GP* algorithm, both constraints are met, constraints are : bandwidth = 25 and resources = 130.

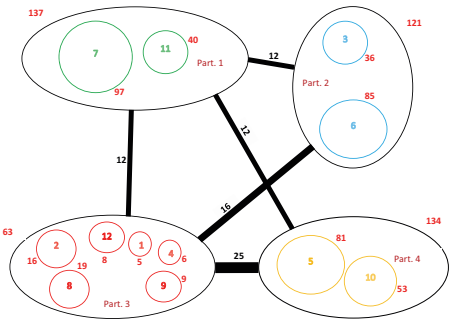

Fig. 9: Partitioning of the sample graph 2 with algorithm, resources is violated while bandwidth is met, constraints are : bandwidth = 25 and resources = 130.

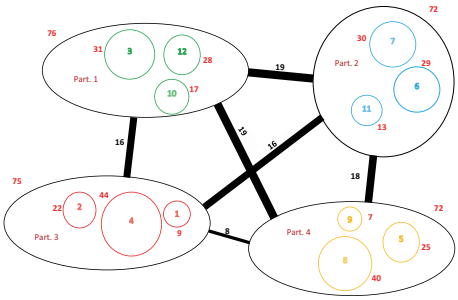

Fig. 12: Partitioning of the sample graph 3 with *GP* algorithm, both constraints are met, constraints are : bandwidth = 25 and resources = 130.

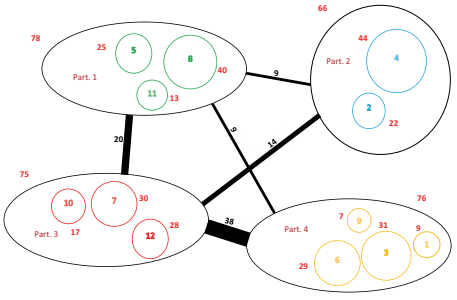

Fig. 13: Partitioning of the sample graph 3 with *METIS* algorithm, resources is violated while bandwidth is met, constraints are : bandwidth = 25 and resources = 130.

Algorithms	K = 4			
	Total Edge-Cuts	Total Time(S)	Maximum Resource Allocation	Maximum Local bandwidth
METIS	58	0.02	**172**	**20**
GP	70	0.33	163	16

EXPERIMENT I: Number of Nodes = 12, Number of Edges = 33, both constraints are violated in *METIS* and in *GP* both constraints are met.

Algorithms	K = 4			
	Total Edge-Cuts	Total Time(S)	Maximum Resource Allocation	Maximum Local bandwidth
METIS	77	0.02	**137**	25
GP	62	0.25	127	18

EXPERIMENT II: Number of Nodes = 12, Number of Edges = 30, resource is violated in *METIS* but bandwidth is met and in *GP* both constraints are met.

Algorithms	K = 4			
	Total Edge-Cuts	Total Time(S)	Maximum Resource Allocation	Maximum Local bandwidth
METIS	90	0.02	78	**38**
GP	96	7.76	76	19

EXPERIMENT III: Number of Nodes = 12, Number of Edges = 32, bandwidth is violated in *METIS* but resource is met and in *GP* both constraints are met.

[16] F. Rendl and H. Wolkowicz, "A projection technique for partitioning the nodes of a graph," *Annals of Operations Research*, vol. 58, pp. 155–179, 1995.

[17] S. T. D. A. Spielman, "Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems," *In Proceedings of the* 36^{th} *Annual ACM Symposium on Theory of Computing*, pp. 81–90, 2004.

[18] S. T. Barnard and H. D. Simon, "A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems," *In Proceedings of the* 6^{th} *SIAM Conference on Parallel Processing for Scientific Computing*, pp. 711–718, 1993.

[19] B. Hendrickson and R. Leland, "A multilevel algorithm for partitioning graphs," *In Proceedings of the ACM/IEEE Conference on Supercomputing*, p. 626ñ657, December 1995.

[20] R. K. U. Feige and K. Nissim, "Approximating the minimum bisection size (extended abstract)," *In Proceedings of the 32nd Annual ACM symposium on Theory of computing*, May 21-23 2000.

[21] R. H. E. M. Arkin, "Graph partitions with minimum degree constraints," *Journal of Discrete Mathematics*, pp. 55–65, August 1998.

[22] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partition," *in Proc.* 19^{th} *Design Automation Conf.*, pp. 175–181, 1982.

[23] R. B. Boppana, "Eigenvalues and graph bisection , an average-case analysis (extended abstract)," *In Proceedings of the* 28^{th} *Symposium on Foundations of Computer Science*, pp. 280–285, 1987.

[24] A. Pothen, H. D. Simon, and K. P. Liou, "Partitioning sparse matrices with eigenvectors of graphs," *SIAM Journal on Matrix Analysis and Applications*, vol. 11, pp. 430–452, 1990.

[25] S. Barnard, "Pmrsb: Parallel multilevel recursive spectral bisection," *In Proc. Supercomputing*, 1995.

[26] J. Gilbert and E. Zmijewski, "A parallel graph partitioning algorithm for a message-passing multiprocessor," *International Journal of Parallel Programming*, pp. 498–513, 1987.

[27] P. K. AURORA, "Multi-level graph partitioning," 2007.

[28] M. U. Khan, "Use multilevel graph partitioning scheme to solve traveling salesman problem," Master's thesis, Department Of Computer Engineering, Dalarna University, Sweden, 2010.