A Graph Partitioning System For Natural Unbalanced Partitions

Sachin B. Patkar, Viraj Kumar, Bijit Hore, Hardeep Kaur
Department of Mathematics,
Indian Institute of Technology, Bombay,
Mumbai-400 076,
INDIA.
patkar@math.iitb.ac.in

Abstract: - We describe here a graph partitioning system that aims at producing naturally unbalanced multiway partitions of a large graph. This system makes use of ideas of 2-phase FM (coarsening,partitioning,refinement), multilevel refinement and network flows. The system has been built up as a loosely coupled system of independent modules, which may be replaced by equivalent modules, thus allowing us to try out several different strategies at different levels. In our strategy, the input netlist is first coarsened into a smaller netlist and the core network flow based partitioner then proceeds to partition this small coarsened netlist. This coarse partition is then lifted to a partition of the original netlist.
We also present the comparison of results of this approach applied to benchmark circuits with the well-established algorithm pmetis of Metis [5] package (Metis based algorithms [5],[2] have reported success for graph and netlist partitioning). The comparative study shows that this new approach indeed produces good quality scaled-cost multiway partitions.
Our partitioning system may be considered as an adaption of the classical Fiduccia-Mattheyses paradigm for producing unbalanced partitions. For efficiency and quality, we make use of the multilevel and 2-phase variations of the Fiduccia-Mattheyses paradigm rather than plain classical one itself.

Key-Words: -

1 Introduction

Graph Partitioning finds applications in various fields today. Scientific computing, circuit partitioning for various stages of VLSI design and task scheduling in multiprocessor systems are some of the important areas for which the problem of graph partitioning is a very central one.

The problem of partitioning a VLSI netlist into blocks, each of which meets the resource constraints of the FPGAs, is also of importance. The input to such a partitioner comprises of the netlist, cost estimates of the nodes in the netlist (ie. the resources of each type in the FPGA required by each type of the nodes of the netlist), and the resource constraints of an FPGA (such as the number of flipflops, the number of function generators, the number of CLBs and the number of I/O pins). Since a large variety of FPGA's are available with differing resource constraints, the idea of partitioning the graph underlying a netlist into natural (and possibly) unbalanced partitions has gained prominence. Several investigations have been carried out in this regard. The prominent approaches include MELO [3] based on spectral theory and the ratio-cut partitioner by Wei and Cheng [8] Recently the ideas from the theory of submodular functions have also been used successfully for computing partitions into 2 parts [7].

Most of the partitioning problems such as VLSI netlist partitioning problem reduce to graph partitioning. The data is represented by means of a graph $G = (V, E)$, where V is a set of vertices or nodes and E is a set of edges. A k-way partitioning of the graph is a division of the vertex set into k components. A good partition of the graph is one in which the number of edges that run across components is small.

The graph partitioning problem falls in the class of NP-hard problems. Solutions are typically based on heuristics or approximation algorithms.

Spectral methods are a popular choice for naturally unbalanced partitioning. The first key step in a typical spectral partitioner is that of the matrix formulation which is to be used for the spectral methods. One formulation is based on the Laplacian of the adjacency matrix of the graph. The next step of the process is to obtain the spectrum of the graph. The eigenvectors are used to find a geometric embedding

of the graph. Several intuitive ordering or clustering schemes use this embedding information to find multiway partitions of a given graph.

We have built a core for a multiway graph partitioning system that allows naturally unbalanced partitions to be found. It is a loosely coupled system consisting of several modules. Different strategies and approaches would be invoked by combining different modules in an appropriate order. One of the typical modules is *coarsener*. The *coarsener* primarily reduces the size of the problem, without destroying relevant information as much as possible, and it results in a smaller netlist. This smaller netlist is then processed by successive modules rather than the actual netlist which could be prohibitively large. This results in a partition of the coarsened netlist.

The central module is one that creates several multiway naturally unbalanced partitions of the coarsened graph.

Each of these partitions of the small coarsened graph is then projected back to the corresponding partition of the original graph and the last module of the system refines this partition to yield improved cut. This **refinement** module may be based on the multilevel refinement scheme that is a part of the powerful Metis package. Several other choices are possible and may be integrated in our loosely coupled system.

2 Partitioning Objectives

In general, a VLSI netlist or circuit is modeled well by a hypergraph. A hypergraph is a collection of vertices and hyperedges, where each hyperedge is simply a subset of vertices. Thus each hyperedge can model an electrical net joining a set of vertices.

A graph instance can be constructed from a given hypergraph by modeling each hyperedge e by a clique of weighted edges on the set of vertices of the given hyperedge. Each edge in the clique has a weight $\frac{4.w(e)}{p(p-1)}.\frac{2^p-2}{2^p}$ where p is the number of vertices in the hyperedge e and $w(e)$ is its weight. (This model is known as **partitioning-specific model**, cf. [6]).

Now the graph partitioning problem in general can be stated as follows:
Given a graph $G = (V, E)$, where V is the set of nodes $\{v_1, v_2, \ldots, v_n\}$ and E the set of edges, construct a k-way partitioning P_k which divides the nodes into a set of k disjoint subsets $P_k = \{C_1, C_2, \ldots, C_k\}$ so as to optimize some objective function $f(P_k)$.

A very common objective function is merely the size of the cut. Minimum such cut however, may divide the vertex set very unevenly and this is undesirable.

There are, in literature, objective functions which measure the size balancing effect of a partition. They are described below.

- **Ratio Cut**: Given $G = (V, E)$, partition V into disjoint U and W such that $cut(U, W)/(|U| \cdot |W|)$ is minimized.
- **Modified Scaled Cost** : For a k-way partition P_k of a graph G, the cost function is defined as

$$cost(P_k) = \sum_{h=1}^{k} \frac{E_h}{\min\{|\ C_h\ |, |V| - |C_h|\}}$$

  where, E_h is the number of cut edges that cross the cluster C_h. The scaled cost is then defined as $mscost(P_k) = \frac{1}{n(k-1)} \times cost(P_k)$.

  Note that this definition of the **modified scaled cost** is at least as harsh as the widely used usual notion of the **scaled cost**

$$scost(P_k) = \frac{1}{n(k-1)} \sum_{h=1}^{k} \frac{E_h}{|\ C_h\ |}$$

  Indeed, if a partition $P_k = \{C_1, C_2, \ldots, C_k\}$ of the vertex set of a graph $G(V, E)$ has at least one block, say C_h whose size is bigger than half the size of V, then the modified scaled cost is higher than the usual scaled cost. Otherwise the modified scaled cost is identical to the original notion of the scaled cost. We shall use both these measures to evaluate the partitions produced by our partitioner. Also note that these two notions of costs are identical in case of the perfectly balanced partitions such as the ones Metis tends to produce.

It is important to note that we shall also be using the new notion of the scaled cost which is harsher on the unbalanced partitions in which one of the parts is bigger than half the size of the vertex set V of the original graph. It will be shown with the help of experiments on the benchmark that our partitioning system shows improved results in the sense of both the variations of the notion of scaled cost.

3 Partitioning Methods

Since graph partitioning is a hard problem, most solution approaches are based on heuristics. There is a wide variety of heuristics that have been suggested and tried. Broadly, partitioning methods fall into two categories. The first variety includes those that work locally. The other set of methods are those that look at the global connectivity information while partitioning.

The classical algorithms due to Kernighan-Lin and Fiduccia-Mattheyses, which are now part of the standard texts (see [6]), were first effective methods for 2-way cuts, which could be classified as local search strategies. The major drawback however happens to be the arbitrary initial partitioning of the vertex set. These methods are popular as postprocessing or refining modules. After other methods arrive at a partition, these move-based methods are employed in order to take the partition to a local minimum. There are several other combinatorial schemes that work locally.

Among global methods, the ones based on spectra of associated matrices work quite well. As mentioned earlier, they form an important part of our system.

3.1 Spectral Formulation

We briefly discuss a matrix formulation obtained from the adjacency matrix of the graph.

3.1.1 The Laplacian formulation

Let A be the adjacency matrix of $G(V, E)$. Let a diagonal matrix D have entries d_{ii} of this matrix denoting the degree of the node v_i in the absence of edge weights and the sum of the weights of all edges that have v_i as an endpoint otherwise. The **Laplacian** of the graph is defined as $Q = D - A$.

A relationship between the optimal ratio cut cost and the second eigenvalue λ of $Q = D - A$ has been established by Hagen and Kahng (see for details [3]). Their theorem is stated below without proof.

Theorem 1 *Given a graph $G = (V, E)$ with adjacency matrix A, diagonal degree matrix D, and $\mid V \mid = n$, the second smallest eigenvalue λ of $Q = D - A$ yields a lower bound on the cost c of the optimal ratio cut partition, with $c \geq (\lambda/n)$.*

Following such ideas, researchers use the lower end of the spectrum of the Laplacian matrix. These eigenvectors of the Laplacian of the graph give an embedding of the graph in Euclidean space. Spectral partitioners then make use of a d-dimensional embedding of the graph from the eigenvectors corresponding to the d lowest eigenvalues of the Laplacian. The i^{th} components of each of these d eigenvectors give the d coordinates in R^d for the i^{th} node of the graph. After the embedding is obtained, various strategies are used to obtain clusters in the graph.

4 The Multilevel Scheme

In our Partitioning System, we make use of a multilevel algorithm for graph partitioning. The central idea is to first reduce the size of the graph repeatedly, till its size gets down to a few hundred vertices, i.e., to *coarsen* it. This small graph is then *partitioned* and the partitions are projected to the original graph to obtain a partitioning of the actual input. This last phase is the *uncoarsening* or *refinement* phase. Karypis et al [5] investigate some strategies for each of the three phases. We incorporate their multilevel scheme in our partitioner. As we shall see, the coarsening strategy gives partitions faster without sacrificing too much on the quality of the partition. The basic flow of the partitioner is described in some detail below.

Step 1. We first coarsen the original graph $G = (V, E)$ using a strategy for fusing pairs of nodes repeatedly to form an intermediate graph $G_1 = (V_1, E_1)$ with a smaller number of (super)nodes. One such strategy is to find a maximal matching of the graph and fuse vertices at the ends of the matching edges. Note that in this process some of the supernodes may become extremely large whereas others may consist of only a few nodes.

Step 2. Next, we coarsen the graph G_1 to G_2 using a similar strategy, but this time we may discard supernodes of small weight.

Step 3. Next, we use the relation between supernodes of G_2 and nodes of G_1 to obtain small cuts of G_1. We do this as follows : We construct a flow network $G_f = (V_f, E_f)$ where $V_f = V_1 \cup \{s, t\}$ where s and t are new (dummy) vertices, and $E_f = E_1$ and the capacity function $c : E_f \rightarrow R^+$ is given by $c_e = weight(e) \forall e \in E_f$.

Now, for each pair of supernodes U, V in G_2, we add infinite capacity edges between s and each vertex of G_1 in U and similarly we add infinite capacity edges between t and each vertex of G_1 in V. We now run the *Maxflow-Mincut algorithm* [4] on this flow network to obtain a small cut (S, T) of G_1 where S and T are respectively the source-side and sink-side vertices of the cut. Note that because of the infinite capacity edges S and T contain atleast the vertices in U and V respectively. This ensures that the cut is non-trivial. We now remove the infinite capacity edges and repeat the process for another pair of supernodes of G_2. This gives us a number of bipartitions of the node set of G_1.

It may be remarked that in case the number of supernodes of G_2 is large, the above approach may invoke the network flow algorithm quadratic number of times. To avoid this, in such situations, the network flow algorithm is used to compute minimum cuts separating each supernode from all the others simultaneously. Indeed this approach results in a theoretically 2-optimal solution to the multiway partitioning problem.

Step 4. We now use these sets of blocks of nodes to compute subpartitions of V_1. We compute these in the following manner : We create the *Overlap Graph* of the set of blocks. This is an undirected, unweighted graph each of whose node corresponds to a block and two nodes are adjacent **iff** the intersection of the blocks corresponding to them is non-empty. We enumerate all the *maximal independent sets* of this overlap graph. These enumerated sets (whose elements are blocks of elements), correspond to subpartitions of the set V_1.

The algorithm consists of growing branches of a tree by adding one vertex at a time such that the set always remains an independent set till it becomes a maximal independent set. We start with the root as a singleton set consisting of one vertex of the graph and a node at depth i is a set of size $i+1$. We stop growing a branch when we can't find any more vertices to add (i.e all remaining vertices have one or more edges incident on this set of nodes). Each set is an ordered set therefore avoiding generation of redundant sets amongst the different branches of the tree. We grow $\mid V \mid$ such trees starting with each distinct vertex of the graph as the root of the tree. Thus a tree with root v generates all the maximal-independent sets that contain v as the smallest labeled vertex.

Step 5. We now have a collection of subpartitions of G_1. We extend these to full partitions of G_1 by absorbing the extra vertices using a simple strategy, and then map these back to blocks of vertices in the original graph G.

Step 6. Finally, we use these blocks as the initial partition and run the refinement module on this. We make use of the multilevel refinement idea that has been popularized by packages such as Metis. Indeed we built one such system using the application programmers interface of MetisLib [5] to refine the above obtained partitions of the given graph. Metis library is extremely fast and thus our refinement works very fast. Standard Fiduccia-Mattheyses heuristic may also be used for the refinement purpose.

5 Details of the Coarsening Phase

During the coarsening phase, a sequence of smaller graphs $G_i = (V_i, E_i)$, is constructed from the original graph $G_0 = (V_0, E_0)$ such that $|V_i| < |V_{i-1}|$. A pair of vertices of G_i is combined to form a single vertex of the next level coarser graph G_{i+1}. Let V_i^v be the set of vertices containing the pair of vertices of G_i combined to form a vertex v of G_{i+1}. We call v as *supernode*. The weight of vertex v is equal to the sum of weights of the vertices in V_i^v. Also in order to preserve the connectivity information in the coarser graph, the edges incident on v are the union of the edges incident on the vertices in V_i^v. In case where more than one vertex of V_i^v contain edges to the same vertex u, the weight of the edge of v is equal to the sum of the weights of these edges. This coarsening method ensures the following properties.

- Every *edge-cut* of the coarser graph corresponds to a unique of the *edge-cut* of the original graph.
- A good partition of the coarser graph leads to a a good partition of the original graph.

The edge collapsing idea can be formally defined in terms of matchings. A **matching** of a graph is a set of edges, no two of which are incident on the same vertex. Thus the next level coarser graph G_{i+1} is constructed from G_i, by finding a matching of G_i and collapsing the vertices being matched into *supernodes*. The unmatched vertices are simply copied over to G_{i+1}. Since we are decreasing the size of the graph G_i, the matching should contain large number of edges. So, we use the **maximal matching** strategy. A matching is **maximal** if it is not possible to add any other edge to it without making two edges become incident on the same vertex. Several different heuristics for finding maximal matching are used as the algorithm to find maximum matching would not be practical in the sense of time complexity for rather large graphs although $O(|V|^3)$ algorithms are available for this purpose.

The following technique is used for computing a maximal matching in our approach.

Sorted Heavy Quotient Edge Matching (SHQEM) The edges are sorted according to the ratio of their edge-weights to the product of the weights of the two vertices joined by the edge. Those vertices are visited first which hava an adjacent edge of maximum ratio. This tends to preserve the ratioCut as observed empirically. We also ensure that no two vertices in the original graph from different block of the given bipartition are selected for merging.

At the end of the coarsening phase, we have a coarsened graph of appropriate size, and we also have a partition which corresponds to the given partition of the original graph which was one of those computed by running our network flow based cluster separator on the coarsened graph in conjunction with the maximal independent subsets algorithm.

6 Experimental Results

We have tested our approach on benchmark circuits. We used a subset of ACM/SIGDA benchmarks (available via World Wide Web at

http://ballade.cs.ucla.edu/cheese). These circuits were transformed into graph form by modelling a net using a random collection of edges connecting approx. $5 * |e|$ pairs of nodes of the net e (here $|e|$ denotes the number of nodes on which the net e is incident). This model has been suggested and employed in [2] as it preserves sparse-ness of the netlist which the partitioning specific model does not.

We conducted our experiments as follows: pmetis is a deterministic multiwat partitioner that takes the required number of blocks in a partition and the graph as input and computes good balanced partitions. We ran pmetis on each benchmark to obtain 2-way, 3-way and 4-way partitions. Our partitioner takes in as an input the graph and the *typical number of blocks*. It, however, produces in a single run, several (possibly unbalanced) multiway partitions, each with number of blocks less than the specified *typical number of blocks*. It is important to mention that we make only one run of our partitioning system on each benchmark. Just like pmetis, it is a deterministic partitioning system, too.

The following table lists a summary of the results of our experiments. The column **nparts** gives the number of blocks of the partition. The column **pmetis** gives the scaled cost computed by pmetis (of Metis package). The columns **mscost** and **scost** describe the modified scaled cost and the scaled cost of the best partitions produced by our system (Due to space limitation we are unable to give complete evidence about several different **better partitions** that were produced by our algorithm). Also recall that **modified scaled cost** is by definition harsher on the unbalanced partitions than the measure of **scaled cost**. More detailed results are available on request from the authors.

Next we present a series of tables, one for each benchmark circuits (primary01, test02, test03, test04, test05, test06, bm1), containing a few of the partitions obtained by our partitioner that are better (in the sense of scaled cost) than the corresponding multiway partitions obtained by pmetis.

7 Conclusion

Our partitioning system has reported success in finding natural unbalanced partitions, even when we employ the harsher meaure of modified scaled cost. A large number of applications in data clustering, graph drawing require identification of natural clusters. Our system can be of use in such applications.

Name	nparts	pmetis	mscost	scost
primary01	2	0.101	0.067	0.036
primary01	3	0.100	0.058	0.048
primary01	4	0.125		
test02	2	0.035	0.044	0.034
test02	3	0.043	0.032	0.029
test02	4	0.047	0.034	0.030
test03	2	0.036	0.022	0.014
test03	3	0.032	0.026	0.024
test03	4	0.044	0.030	0.030
test04	2	0.025	0.020	0.012
test04	3	0.028	0.017	0.017
test04	4	0.031	0.022	0.022
test05	2	0.013	0.014	0.012
test05	3	0.018	0.016	0.016
test05	4	0.024		
test06	2	0.076	0.053	0.028
test06	3	0.086	0.048	0.035
test06	4	0.090		
bm1	2	0.098	0.154	0.084
bm1	3	0.109	0.048	0.035
bm1	4	0.103		

Table 1: Comparison of pmetis and our partitioner in the sense of modified scaled cost and the scaled cost

Name	mscost	scost	block-sizes
primary01	0.067	0.036	69,763
primary01	0.065	0.037	126,706
primary01	0.067	0.048	259,573
primary01	0.058	0.049	182,575,75
primary01	0.077	0.077	374,389,69
primary01	0.082	0.082	346,418,68

Table 2: A few of the better partitions of primary01 obtained by our partitioners

Name	mscost	scost	block-sizes
test02	0.044	0.034	592,1071
test02	0.036	0.028	186,1309,168
test02	0.032	0.029	146,1012,505
test02	0.042	0.033	261,167,1235
test02	0.034	0.030	142,256,1102,161

Table 3: A few of the better partitions of test02 obtained by our partitioners

Name	mscost	scost	block-sizes
test03	0.024	0.014	313,1293
test03	0.022	0.014	328,1278
test03	0.032	0.029	897,709
test03	0.026	0.024	310,396,900
test03	0.029	0.026	282,370,954
test03	0.030	0.030	289,248,423,646
test03	0.032	0.032	300,257,422,627

Table 4: A few of the better partitions of test03 obtained by our partitioners

Name	mscost	scost	block-sizes
test04	0.020	0.012	293,1221
test04	0.021	0.013	286,1228
test04	0.022	0.018	592,922
test04	0.020	0.017	611,903
test04	0.017	0.017	290,686,538
test04	0.018	0.018	290,221,443,560
test04	0.022	0.022	294,322,333,555

Table 5: A few of the better partitions of test04 obtained by our partitioners

Name	mscost	scost	block-sizes
test05	0.018	0.010	397,2197
test05	0.014	0.012	1118,1476
test05	0.016	0.016	1033,859,702
test05	0.016	0.016	882,894,818
test05	0.016	0.016	1059,886,649

Table 6: A few of the better partitions of test05 obtained by our partitioners

Name	mscost	scost	block-sizes
test06	0.053	0.028	100,1651
test06	0.061	0.033	130,1621
test06	0.048	0.035	129,145,1477

Table 7: A few of the better partitions of test06 obtained by our partitioners

Name	mscost	scost	block-sizes
bm1	0.154	0.084	72,810
bm1	0.170	0.103	157,725

Table 8: A few of the better partitions of bm1 obtained by our partitioners

References

[1] C.J. Alpert and A.B.Kahng, Recent Directions in Netlist Partitioning: A Survey, *INTEGRATION, the VLSI Journal*, Vol. 19, 1995, pp.1-81.

[2] C. J. Alpert, L. Hagen and A. B. Kahng, A Hybrid Multilevel/Genetic Approach for Circuit Partitioning, *Proc. ACM SIGDA Physical Design Workshop*, 1996, pp. 100-105.

[3] C.J. Alpert, A.B.Kahng and D.S. Yao, Spectral Partitioning with Multiple Eigenvectors, *Discrete Applied Mathematics*, Vol. 90, 1999, pp. 3-26.

[4] A.V. Goldberg and R.E. Tarjan, A new approach to the maximum flow problem, *J. Assoc. Computing Mach.*, Vol. 35, 1988.

[5] G. Karypis and V. Kumar, METIS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices (version 4.0), *http://www.cs.umn.edu/ karypis*, 1998.

[6] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*, Bryan Press, New York, 1990.

[7] S.B. Patkar and H. Narayanan, Improving Graph Partitions using Submodular Functions, *to appear in Special Issue of* **Discrete Applied Mathematics** *on Submodular Functions and Applications*

[8] Y.-C Wei and C.-K Cheng, An improved two-way partitioning algorithm with stable performance, *IEEE Trans. Computer Aided Design*, Vol 10, No. 12, 1991, pp. 1502-1511.