Multi-Way VLSI Circuit Partitioning Based on Dual Net Representation

Jason Cong
Department of Computer Science
University of California, Los Angeles, CA 90024

Wilburt Labio and Narayanan Shivakumar
Department of Computer Science
Stanford University, Stanford, CA 94305

Abstract

In this paper, we study the area-balanced multi-way partitioning problem of VLSI circuits based on a new dual netlist representation named the hybrid dual netlist (HDN), and propose a general paradigm for multi-way circuit partitioning based on dual net transformation. Given a netlist we first compute a K-way partitioning of nets based on the HDN representation, and then transform the K-way net partition into a K-way module partitioning solution. The main contribution of our work is in the formulation and solution of the K-way module contention (K-MC) problem, which determines the best assignment of the modules in contention to partitions while maintaining user-specified area requirements, when we transform the net partition into a module partition. Under a natural definition of binding function between nets and modules, and preference function between partitions and modules, we show that the K-MC problem can be reduced to a min-cost max-flow problem. We present an efficient solution to the K-MC problem based on network flow computation. We apply our dual transformation paradigm to the well-known K-way FM partitioning algorithm (K-FM) and show that the new algorithm, named K-DualFM, reduces the net cutsize by 20% to 31% compared with the K-FM algorithm. We also apply the same paradigm to the K-MFFC-FM algorithm, a K-FM algorithm based on maximum fanout-free cone (MFFC) clustering reported in [10], and show that the resulting algorithm, K-DualMFFC-FM reduces the net cutsize by 15% to 26% compared with K-MFFC-FM. Furthermore, we compare the K-DualFM algorithm with EIG1[18] and Paraboli [26], two recently proposed spectral-based bipartitioning algorithms. We showed that K-DualFM reduces the net cutsize by 56% on average when compared with EIG1 and produces comparable results with Paraboli.

1. Introduction

The K-way partitioning problem is one of partitioning the modules in a network into K subsets (partitions) of "approximately" the same size while minimizing the number of interconnections between the K partitions. This problem has many applications in VLSI circuit design ranging from circuit layout to logic simulation and emulation.

The existing partitioning algorithms in the literature can be classified into two-way partitioning (bipartitioning) algorithms and multi-way partitioning algorithms. The bipartitioning algorithms include the iterative improvement methods, the analytical methods, the min-cut based method, and the net-based partitioning method. Some of the best known iterative improvement based partitioning methods include the Kernighan-Lin (KL) algorithm [22], the Fiduccia-Mattheyses (FM) algorithm [15], the FM-algorithm with look-ahead scheme [24], and the simulated annealing approach [23, 17]. The analytical methods include both the use of a linear placement formulation with the quadratic objective function, which is solved by computing the second smallest eigenvector of the Laplacian matrix of the given network [14, 2, 4, 18], and the use of the linear placement formulation with a linear objective function, which is solved by an iterative method in [26]. The min-cut based method uses the maximum flow algorithm to compute a series of minimum cuts in the given circuit in

order to obtain an area-balanced cut with small cut size [29]. The net-based partitioning approach first computes a bipartitioning of the nets, and then transforms the net partitioning solution into a module partitioning solution [20, 9].

The multi-way partitioning algorithms include the recursive bipartitioning by Kernighan and Lin [22], a generalization of the FM-algorithm with lookahead by Sanchis [27], the primal-dual algorithm [30], and a generalization of the graph spectral-based partitioning method to multi-way ratio-cut by Chan, Schlag, and Zien [5].

To reduce the computational complexity for partitioning very large circuits, cluster-based partitioning methods have been introduced. In this approach clusters are identified and collapsed, and the resulting clustered network is partitioned using existing partitioning methods. Clustering methods include random-walk clustering [8, 19], multicommodity-flow based clustering [31], clique based clustering [13], geometric embedding with min-diameter clustering [1], and clustering based on maximum fanout-free cones (MFFCs) [10]. Partitioning with module replication [25, 21] and the communication-complexity based partitioning method [3] have also been proposed to further reduce the amount of interconnections.

Since the objective of the partitioning problem is to minimize the number of nets to be cut, we believe that assigning nets, instead of modules, to partitions will lead to better partitioning solutions in general. The net-based bipartitioning algorithm by Cong, Hagen, and Kahng [9] is therefore of particular interest to us. This algorithm first computes a bipartitioning of the nets using the graph spectral method, and then transforms a net bipartitioning solution into a module bipartitioning solution by solving the *module contention problem* (to be discussed in Section 3 in detail). It was shown that the module contention problem for bipartitioning can be solved optimally by computing a minimum vertex covering in a bipartite graph, and very encouraging experimental results were reported. However, the minimum vertex covering formulation for the module contention problem is inherent to bipartitioning and cannot be easily generalized to multi-way partitioning.

In this paper, we introduce a new dual netlist representation named *hybrid dual netlist (HDN)* and propose a general paradigm for multi-way circuit partitioning based on dual transformation. Given a netlist, we first compute a K-way partition of the nets based on the *HDN* representation, and then transform the K-way net partition into a K-way module partition. The main contribution of our work is the formulation and solution of the *K-way module contention (K-MC)* problem. We introduce a *binding function* between modules and nets, and a *preference function* between modules and partitions in determining the best assignment of modules in contention to partitions. We show that the K-MC problem can be formulated as a min-cost max-flow problem, and we present two efficient network flow based algorithms for solving the K-MC problem under static preference functions and dynamic preference functions.

The rest of the paper is organized as follows. We present the problem formulation and terminologies in Section 2. Section 3 presents the hybrid dual netlist representation and our K-way partitioning algorithms based on the dual transformation paradigm. Section 4 presents experimental results. We conclude the paper in Section 5 with some observations and directions for future work. An extended abstract of this paper was presented in the 1994 International Conference for Computer Aided Design (ICCAD'94) [12].

2. Problem formulation

Given a netlist NL to be partitioned into K partitions, we use $M = \{ m_1, m_2, ..., m_p \}$ to denote the set of modules in NL, $N = \{ n_1, n_2, ..., n_q \}$ to denote the set of nets in NL, and $P_1, P_2, ..., P_K$ to denote the K partitions, where p is the number of modules, and q is the number of nets in NL. The modules may have different areas.

An *optimal area-balanced K-way partitioning solution* of a given netlist NL satisfies the following conditions:

(i) Each module is assigned to exactly one partition.

(ii) The total area of the modules in each partition are within the user-specified area bounds, i.e.

$$(1-\alpha)\cdot \frac{A}{K} \le A_i \le (1+\alpha)\cdot \frac{A}{K}$$

for each partition P_i, where A is the total area of all the modules in NL, A_i is the total area of all the modules in a partition P_i, and α is a user-specified parameter controlling the allowable slack in the area constraint.

(iii) The number of nets cut is minimized.

Given a netlist NL (for example, shown in Fig. 1(a)), we introduce the following definitions:

(i) **Netlist Hypergraph:** $NHG = (V(NHG), H(NHG))$, where each vertex in $V(NHG)$ represents a module m_i $(1 \le i \le p)$ and each hyperedge in $H(NHG)$ represents a net n_j $(1 \le j \le q)$ (see Fig. 1 (b)).

(ii) **Net Intersection Graph (NIG):** $NIG=(V(NIG), E(NIG))$, where each node in $V(NIG)$ represents a net n_i $(1 \le i \le q)$, and there is an edge in $E(NIG)$ between n_i and n_j iff $n_i \cap n_j \neq \phi$ (i.e. the two nets share common modules). Note that NIG is a graph instead of a hypergraph (see Fig. 1 (c)).

(iii) **Dual Netlist Hypergraph (DNHG):** $DNHG=(V(DNHG), H(DNHG))$ where each node in $V(DNHG)$ represents a net and each hyperedge in $H(DNHG)$ represents $N(m_i)$, the set of nets incident to module m_i $(1 \le i \le p)$ (see Fig. 1 (d)).

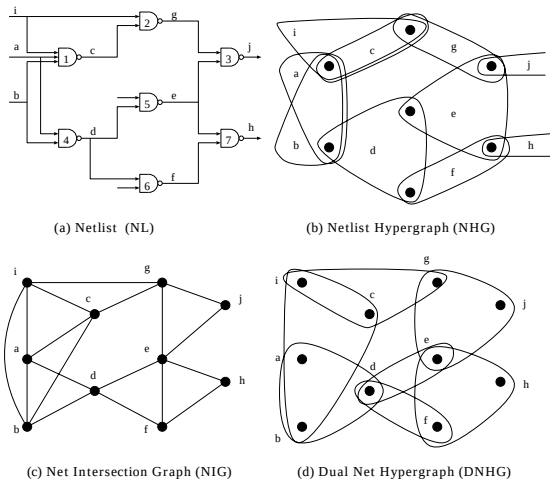


Figure 1 Different Circuit Representations

3. Multi-way circuit partitioning based on dual transformation

In this section, we first review IG-MATCH, the bipartitioning algorithm based on dual transformation by Cong, Hagen, and Kahng [9] and discuss its limitations for multi-way circuit partitioning. Then, we describe the K-DualFM algorithm in detail in Subsections 3.2 to 3.6 and use it to illustrate how to apply our new dual transformation technique to a multi-way partitioning algorithm. Finally, in Subsection 3.7, we present the K-DualMFFC-FM algorithm, which applies our dual transformation technique to the K-MFFC-FM algorithm, a K-FM based algorithm with MFFC clustering [10].

3.1. Review of the IG-MATCH algorithm

The IG-MATCH algorithm partitions the *NIG* using a spectral based method, and then transforms the net partitioning to a module partitioning solution. Given a netlist shown in Fig. 2(a), and a net bipartitioning solution P_1 and P_2 of the *NIG* shown in Fig. 2(b), IG-MATCH assigns all the modules in nets a and b (i.e. modules 1 and 4) to P_1, and all the modules in nets h and j (i.e. modules 3 and 7) to P_2. This assignment assures that nets a, b, h and j will not be cut in the resulting module partitioning solution. Note that these three nets are not incident to any edge cut in the partitioning of *NIG*. Each of the remaining unassigned modules are part of nets that intersect (i.e. share modules with) another net in the other partition. When two (or more) nets reside in different partitions and share some module m_k, the question arises as to which partition the contended module m_k should be assigned to. This is the *module contention (MC)* problem: in Fig 2(b), modules 2, 5 and 6 are in contention since they are shared by nets on both sides. A bipartite graph B was constructed in [9] as follows. The nodes in B represent the nets on the boundary of the partitioning solution of *NIG*. For each net n_i in P_1 and net n_j in P_2 sharing some module m_k, we introduce an edge between n_i and n_j. For example, Fig. 2(c) shows the bipartite graph B, a sub-graph of *NIG*, for the net bipartitioning of *NIG* in Fig. 2(b). It was shown that for any assignment of modules in contention, the set of nets being cut in the resulting module partitioning solution form a vertex cover in B. Therefore, the module contention problem is reduced to the one of computing a minimum vertex cover in a bipartite graph, which can be solved optimally in polynomial time. It is easy to see that nets d and g form the minimum vertex

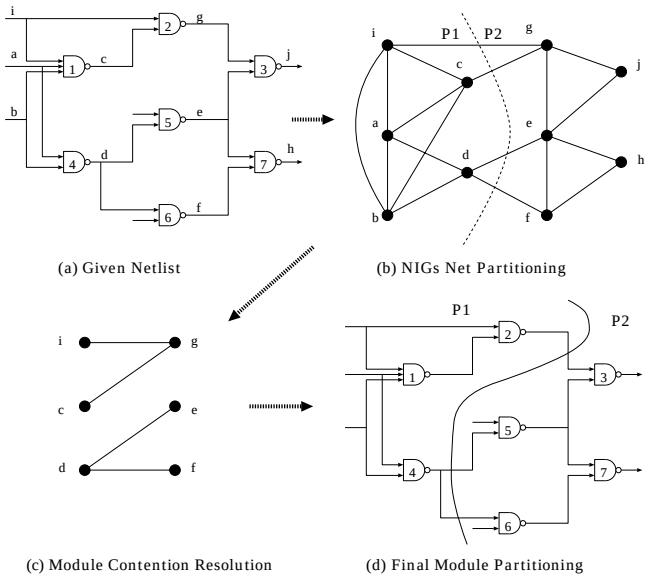


Figure 2 Illustration of the IG-MATCH Algorithm

cover of the bipartite graph shown in Fig. 2(c). Therefore, we decide to allow nets d and g to be cut while keeping nets i, c, e, and f intact in solving the module contention problem, which leads to the module bipartitioning solution shown in Fig. 2(d).

It is clear that the minimum vertex covering formulation for the module contention problem is inherent to bipartitioning and cannot be easily generalized to multi-way partitioning. Also, the minimum vertex cover formulation cannot handle the area balance constraint effectively. The objective of [9] was to minimize ratio-cut size, and therefore, the area balance constraint was not an issue in their formulation. Our work to be presented in the remainder of this section gives a novel and more general formulation of the module contention problem for area-balanced multi-way partitioning that leads to efficient solutions. In the next a few subsections, we shall present the K-DualFM algorithm in detail and use it to illustrate our dual transformation technique for multi-way circuit partitioning.

3.2. Overview of K-DualFM algorithm

Our dual netlist based K-FM partitioning algorithm, named K-DualFM, consists of the following phases:

(1) We first *convert the net hypergraph to a dual net representation*. The dual net representation used in K-DualFM, named *hybrid dual netlist (HDN)*, is a combination of *NIG* and *DNHG*.

(2) *Assign nets to partitions.* We use the K-FM partitioning algorithm [27, 15] to partition the nets into K partitions.

(3) We transform the net partitioning solution into a module partitioning solution by solving the *K-way module contention problem* (K-MC). We show that the K-MC problem can be formulated as a min-cost max-flow problem and we present two efficient algorithms to solve the K-MC problem.

(4) We further *improve the module partitioning solution* using again the K-FM iterative improvement algorithm.

The subsequent subsections describe these phases in detail.

3.3. Generating dual netlist representations

The net intersection graph (*NIG*) was used in [9] since the graph spectral based algorithm used in their net partitioning algorithm applies only to graphs and cannot be used for hypergraphs. However, we notice that for many test circuits, there are a number of nets incident to the same module (see Fig. 3(a)), and these nets will form a large clique (complete graph) in the *NIG* (see Fig. 3(b)). In this case, the memory requirement for storing *NIG* is high and partitioning *NIG* also tends to be more difficult and time intensive. Moreover, since *NIG* can be very dense when large nets exist in the circuit, iterative improvement based partitioning algorithms may easily be trapped in local optima. On the other hand, the dual net hypergraph (*DNHG*) (Fig. 3(c)) defined in Section 2 is more economical in terms of memory requirement when compared to the *NIG*. However, our study shows that use of *DNHG* directly as the dual representation does not give the best partitioning results since *DNHG* representation does not distinguish the number of nets contending for a module. To avoid these problems, we introduce a threshold parameter *CF* when constructing the net intersection graph. When the number of nets incident to the same module is more than *CF*, we connect these nets by a hyperedge instead of a large clique. The resulting dual netlist representation is called the *hybrid dual netlist* (*HDN*) representation. Note that if we set *CF* to be 2, then the *HDN* is the same as the *DNHG*. In general, *HDN* (shown in Fig. 3(d)) is a combination of *NIG* and *DNHG*. Our experimental results confirm that net partitioning based on *HDN* produces better results than those based on *NIG* or *DNHG* representations. The *HDN* was constructed in two steps:

(i) Construct the *Dual net hypergraph*.

(ii) For each hyperedge with no more than *CF* nodes, replace it with a clique.

In our implementation, *CF* was chosen to be 5. Note that the *HDN* is a hypergraph in general. Since we use the K-FM algorithm for net partitioning (see next sub-section), a hypergraph representation presents no problem to us.

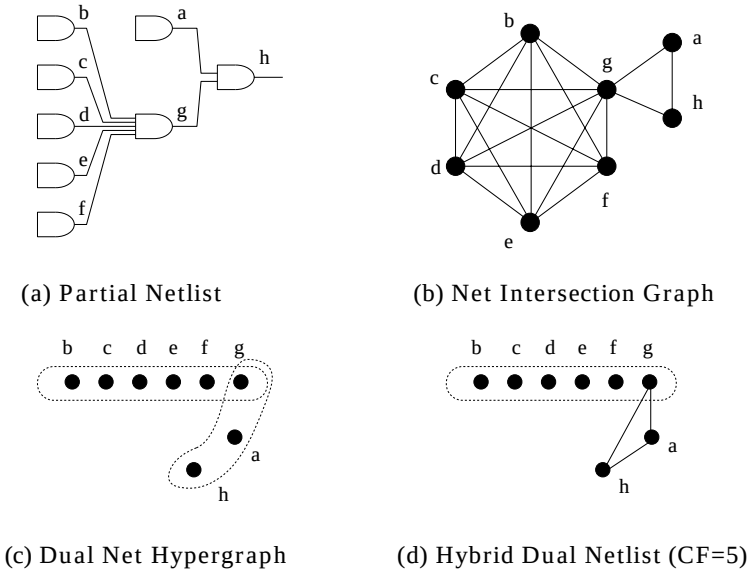
3.4. Partition of dual netlist representation

After constructing the *HDN* hypergraph, we use the K-FM algorithm to compute a K-way partitioning of *HDN* to obtain a K-way partitioning of the nets in the original netlist. We want to minimize the number of edges cut in *HDN* so that the subsequent module contention is easier to solve. For example, a "bad" net partitioning shown in Fig. 4(a) may result in all the modules in all nets being in contention. However, for a good net partitioning as shown in Fig. 4(b), modules in nets a, f, and g can be assigned according to the net partitioning, and only the shared modules in nets b, c, d and e will be in contention. Therefore, it is very important to obtain a good net partitioning based on the K-way partitioning of *HDN*. In our algorithm, we compute an initial net partitioning, using the following simple deterministic, greedy algorithm.

(i) Sort the nets in descending order of their sizes (in terms of the number of modules).

(ii) Assign the largest unassigned net to the partition where most of its assigned modules reside (without violating area constraints).

(iii) Move the unassigned modules that are a part of the selected net to the same partition determined in step 2.



(a) Partial Netlist (b) Net Intersection Graph

(c) Dual Net Hypergraph (d) Hybrid Dual Netlist (CF=5)

Figure 3 Advantage of Hybrid Dual Netlist

(iv) Repeat steps 2 and 3 until all nets are assigned.

In addition, we also generate a number of random initial net partitions if sufficient CPU time remains. We then apply the K-FM algorithm to the net partitions (greedy and random) to improve the K-way partitions of *HDN*.

3.5. Solution to the K-Way Module Contention problem (K-MC)

The main contribution of this paper is the general formulation of the K-way module contention (K-MC) problem and the efficient methods of solution, which allow us to transform a K-way net partition (obtained by any partitioning algorithm) to a K-way module partition such that the number of nets cut is minimized. We shall discuss our formulation and the methods of solution in detail in this subsection.

3.5.1. Problem statement

We say that a module m is in contention if there exist two nets n_i and n_j containing m such that n_i and n_j are in two partitions in the net partitioning solution. We use M_{cont} to denote the set of modules in contention.

The K-MC problem is *to assign modules in* M_{cont} *to proper partitions* so that the total number of nets being cut is minimized. If we start with a good net partitioning (which is usually the case after applying K-FM algorithm on *HDN*), the size of M_{cont} is much smaller than the number of modules in the original netlist. From our experiments, we see that for the MCNC benchmarks, the percentage of modules in contention ranges from 50 - 62% for K = 2, 45 - 60% for K = 3, 40 - 50% for K = 4, and 30 - 42% for K = 5. Therefore, the K-MC problem is much simpler than the original K-way partitioning problem, and judging by the trend of our results, it gets simpler with increasing K.

3.5.2. Binding function and preference function

Good solutions to the K-MC problem should minimize the number of nets being cut under the area constraint. Since this problem is NP-Hard in general, we resort to efficient heuristic algorithms. We introduce a function to estimate the number of nets cut when a module is assigned to partition P_i. Intuitively, a net n_j has a high affinity for a module m_k in contention if it has a high probability of being satisfied (uncut) after attracting m_k into its partition, and a low affinity if it is most likely to be cut even after obtaining m_k. We introduce a *binding function* (bf) to measure this affinity between a

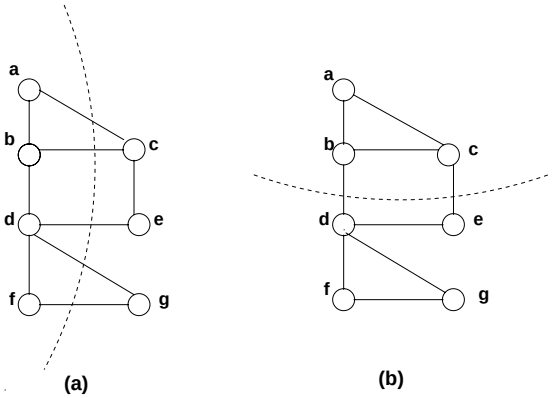


Figure 4 Comparison of two net-partitioning solutions

net and a module. Let n_j be a net and $m_k \in n_j$ be a module in contention. The binding function $bf\ (n_j, m_k)$ should depend on the following factors:

(i) *$S(n_j)$, the number of modules in n_j*: As the number of modules in a net increases, the probability of the net being satisfied (uncut) is reduced. So the $bf\ (n_j, m_k)$ should be inversely proportional to the net size $S(n_j)$.

(ii) *$C(n_j)$, the number of modules in contention in net n_j*: If $C(n_j)$ is high, the probability of the net being satisfied is low. Hence, the $bf\ (n_j, m_k)$ should be inversely proportional to C(n_j).

(iii) $S\ (n_j) - C(n_j)$, *the number of modules of net n_j in its partition already*, i.e. the number of modules in n_j not in contention. The $bf\ (n_j, m_k)$ should be directly proportional to this factor since the probability of the net being satisfied increases as this number increases.

Therefore, we consider the ratios $\frac{S(n_j) - C(n_j)}{S(n_j)}$ and $\frac{S(n_j) - C(n_j)}{C(n_j)}$ to be of primary importance in determining the binding function. From the two ratios, we define the *binding function* of net n_j for module m_k to be

$$bf\ (n_j, m_k) = \frac{(S(n_j) - C(n_j))^2}{S(n_j) \times C(n_j)}$$

We define that $bf\ (n_j, m_k) = 0$ if m_k is not in n_j. Also, if two modules m_k and m_l in net n_j are already assigned to two different partitions (i.e. n_j is already cut), then $bf\ (n_j, m_i) = 0$ for any $m \in n_j$. Based on the definition of the binding function, we define the *preference function* $pf\ (P_i, m_k)$ between a module m_k in contention and a partition P_i as follows:

$$pf\ (m_k, P_i) = \sum_{n \in P_i} bf\ (n, m_k)$$

That is, the preference function between module m_k and partition P_i is the sum of binding function values between m_k and all nets in partition P_i. Our objective is to find an optimal assignment of the modules in M_{cont} to the partitions such that the *cumulative preference* over all assignment edges is maximized.

3.5.3. Flow-based formulation of the K-MC problem

We use the min-cost max-flow algorithm[16, 11] to compute the optimal module assignment. First, we construct an *assignment network (AN)* as follows. We construct a bipartite graph in which the nodes represent the modules in M_{cont} and the partitions in P and each directed edge (m_k, P_i) connects module m_k to partition P_i. Then, we add a source node s to AN and connect it to every module node m_k in AN. Similarly, we add a sink node t to AN and connect every partition node P_i to the sink t. Fig. 5 shows an example of the assignment network. For each edge e in the assignment network, we define its capacity $cap(e)$ and cost $cost(e)$ as follows:

(i) if $e = (s, m_k)$, $cap(e) = 1$, $cost(e) = 0$;

(ii) if $e = (P_i, t)$, $cap(e) = cap(P_i)$, $cost(e) = 0$, where $cap(P_i)$ is the number of modules that partition P_i can accept without violating its area constraints.

(iii) if $e = (m_k, P_i)$, $cap(e) = 1$, $cost(e) = MAX - pf\ (P_i, m_k)$, where MAX is a positive constant larger than any preference function value.

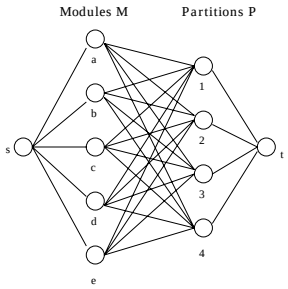


Figure 5 Assignment Network (AN)

Let $m = |M_{cont}|$. In general, we have

$$\sum_{i=1}^{q} cap(P_i) \geq m_{cont} \ (= |M_{cont}|). \qquad \text{(C1)}$$

That is, the total excess capacity in all partitions is larger than the number of modules in contention. (It is easy to show that when all modules are uniform in size, condition C1 is always true.) When module sizes vary significantly, we can only use $cap(P_i)$ to estimate the maximum number of modules allowed in P_i without violating the area balance constraint, and such estimation usually tends to be conservative (based on the largest or average module size). In principle, if we relax the area slack parameter α as defined in Section 2, we can always satisfy condition C1. In the cases where C1 is not satisfied, we shall present a dynamic area updating scheme in Section 3.5.4 that satisfies the area balance constraint.

Lemma 1 The value of the maximum flow in the assignment network is m_{cont} when condition C1 is true.

Proof According to the min-cut max-flow theorem, we need only to show that the size of a min-cut in the assignment network is m_{cont}. First, we show that the size of a min-cut is at least m_{cont}. Let $(X, \bar{X})$ be a min-cut where $s \in X$ and $t \in \bar{X}$. Let M_X be the set of module nodes in X and $M_{\bar{X}}$ be the set of module nodes in $\bar{X}$. Let $m_X = |M_X|$.

Case 1: Suppose that there exists a partition node P_i in $\bar{X}$. Then, the total capacity of the edges between P_i and nodes in M_X is m_X and the total capacity of the edges between the source and the edges in $M_{\bar{X}}$ is $m_{cont} - m_X$. Therefore, the cut size of $(X, \bar{X})$ is at least $m_X + (m_{cont} - m_X) = m_{cont}$.

Case 2: Suppose that there is no partition node in $\bar{X}$. Then, the min-cut must be a cut between the sink node and the remaining nodes in AN. Since condition C1 is satisfied, the cut size is at least m.

Hence, in both cases, the cut size of $(X, \bar{X})$ is at least m. Moreover, it is easy to see that the size of a min-cut is no more than m since the cut between the source and the remaining nodes is of size m. Therefore, we conclude that the size of any min-cut in AN is m when condition C1 is satisfied. □

Theorem 1 The min-cost max-flow in the assignment network induces a module assignment whose total preference function is maximum, when condition C1 is true.

Proof According to Lemma 1, when condition C1 is true, the value of a max-flow in N is m. Given a max-flow f, we can induce a module assignment as follows: Because of the capacity constraints in N and the integer flow property, it is easy to show that a max-flow f of value m satisfies the following property: for each m_i, there is exist a unique P_j such

that $f(m_i, P_j) = 1$ and $f(m_i, P_k) = 0$ for $k \neq j$. We assign module m_i to partition P_j if $f(m_i, P_j) = 1$, and obtain a module assignment solution. Moreover, the module assignment solution satisfies the partition area constraints due the capacity constraints on edges (P_j, t)'s.

On the other hand, given a module assignment solution satisfying the partition area constraints, we can derive a max-flow f as follows: If module m_i is assigned to partition P_j, we define $f(s, m_i) = f(m_i, P_j) = 1$ and $f(m_i, P_k) = 0$ for $k \neq j$. Furthermore, we define $f(P_j, t)$ to be the sum of all incoming flows at node P_j. It is easy to verify that f is a max-flow of value m.

Therefore, each max-flow f in N induces a K-MC module assignment solution satisfying the partition area constraints, and vice versa. Moreover, the cost of the max-flow f is

$$cost(f) = m \cdot MAX - \sum_{m \in M_{cont},\, p \in P,\, f(m,p)=1} pf(m, p).$$

Since MAX is a constant, maximizing the total preference in a module assignment solution is equivalent to minimizing $cost(f)$. Hence, a min-cost max-flow in N induces a module assignment with the maximum total preference function. □

We use the augmenting path algorithm [16, 11] for computing a minimum-cost maximum-flow in the assignment network. We start with a flow of value zero. At each step, we compute the minimum cost augmenting path in the residual graph of the assignment network. Then, we augment the flow value by one, and update the residual graph of the assignment network. The augmentation process stops after m steps. It can be shown that the time complexity of the minimum-cost maximum-flow computation is $O(K \cdot m^2 + m^2 \cdot \log m)$ using Fibonacci heaps. Our implementation has time complexity $O(K \cdot m^3)$ using a simple shortest path algorithm. After we obtain a min-cost max-flow, we can determine the assignment of modules in contention in linear time.

When condition C1 is not satisfied (it occurs in rare cases when the area slack parameter α is very small and/or the module sizes vary significantly), the max-flow in the assignment network has a value less than m, which means that some modules in M_{cont} are left unassigned. One option would be to recompute the actual residual capacities at the termination of the flow algorithm, and repeat the min-cost max-flow algorithm until all modules are assigned. In theory, we may have to go through several such iterations of flow computation until all modules are assigned. In the next sub-section, we present a dynamic area updating scheme that requires exactly one flow computation.

3.5.4. Dynamic updating of area constraint

The flow-based algorithm for the K-MC problem presented in the previous sub-section computes the capacity of the edge from a partition to the sink using the *number* of modules that a partition could accommodate based on the *average* module size. This scheme works well when all modules are more or less uniform in size. However, in some circuits (such as Test02 in the MCNC benchmark suite), there exist a few modules (such as macro blocks) whose areas are much larger than the rest of modules. In order to effectively handle the case when module sizes vary considerably, we introduce an area balancing scheme which first removes and assign the modules of excessively large areas from contention, and then assigns the remaining modules in contention based on an accurate area capacity constraint which is *dynamically updated* during flow computation.

Let A be the cumulative area of all modules, K be the number of partitions and α the user-specified slack. We first compute $Ex = \{ m \mid area(m) \geq (1+\alpha) \cdot \frac{A}{K} \}$, which is the set of modules that cannot be accommodated even into the largest allowed partition size. We then assign each of these modules to a single partition since combining any of them with other modules in the same partition will further violate the area balance constraint[1]. The residual area (total area of remaining modules), denoted by A_{res}, is $A - area(Ex)$ where $area(Ex)$ is the cumulative area of the modules in Ex. Our new area balance constraint will then require that the $K - |Ex|$ partitions satisfy

$$(1-\alpha) \cdot \frac{A_{res}}{K - |Ex|} \leq A_i \leq (1+\alpha) \cdot \frac{A_{res}}{K - |Ex|}$$

It is easy to see that $|Ex| < K$ when $\alpha > 0$.

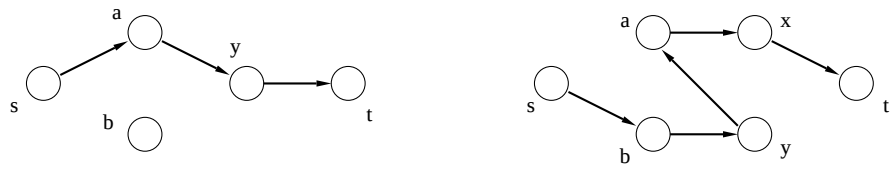
Observe that each augmenting path in our min-cost max-flow computation assigns one more module to a partition with possible re-assignment of some other modules. For example, Fig. 6 (a) shows a simple augmenting path in the residual graph of the assignment network, which assigns module a to partition y. Fig. 6 (b) shows a more general augmenting path, which assigns module b to partition y and reassigns module a from partition y to partition x. Hence, after each augmenting path computation, we know the assignment of the new module and (possible) reassignments of other modules. Therefore, we can update the area capacity constraints of all affected partitions.

We modify our flow algorithm as follows. We use the same definitions for capacity and flow as earlier, but we are *selective* about augmenting flow along min-cost augmenting paths in the residual graph. We define two extra arrays $current[P_i]$ and $capacity[P_i]$ to denote the *actual* total area of the modules assigned to P_i so far, and the *actual* maximum allowable area of P_i, respectively. The new min-cost max-flow algorithm from a *conceptual point of view* can be described as follows:

While there exists an augmenting path in the residual graph
  Find the min-cost augmenting path $p = (s, m_{i_1}, P_{j_1}, m_{i_2}, ..., m_{i_t}, P_{j_t}, t)$ such that
    for all assignments $m_i \rightarrow P_j$ specified by p, $(current[P_j] + area(m_i)) > capacity[P_j]$ holds
  Augment flow along path p and update the residual graph
  Update $current[P_j]$ for all affected partitions as follows
    if $(m_i, P_j) \in p$ then $current[P_i] = current[P_i] + area(m_i)$
    if $(P_j, m_i) \in p$ then $current[P_i] = current[P_i] - area(m_i)$
  Update $capacity[P_j]$ for all affected partitions accordingly
End-while

That is the modified min-cost max-flow algorithm accepts augmenting paths specifying assignments that satisfy the *actual* partition area constraints by maintaining two arrays reflecting the areas due to the current module assignment. One concern is the number of augmenting paths that we have to reject before we find one which satisfies the area constraint. In fact our algorithm incorporates checking of area constraint into the best-first search process for computing the min-cost augmenting path, so that we generate only the min-cost augmenting path which satisfies the area constraint. This is based on a simple observation that there are three types of edges in the residual graph of the assignment network:

[1] Given an area balance constraint specified by a slack parameter α as shown in Section 2, it is NP-complete to decide if there is a partitioning solution satisfying such a constraint when the modules are not uniform in size. As a result, it is not always possible for the user to specify a slack parameter which guarantees an area-balanced partitioning solution. The best any practical partitioning algorithm can do is to try to minimize the violation of area balance constraint.



(a) Simple Assignment of a Module (b) Assignment of a Module by Reassignment of other modules

Figure 6 Augmenting paths in the residual graph of the assignment network

(i) the edges between the source and module nodes

(ii) the edges between module nodes and partition nodes

(iii) the edges between partition nodes and the sink

The min-cost augmenting path algorithm is based on the Dijsktra's shortest path algorithm [11], which performs a best-first search starting from the source and keeps expanding the node reachable from the source with the minimum path cost in the residual graph. A min-cost source-to-sink path is found when the sink is reached in the best-first search process. In our modified augmenting path algorithm, we follow the edges of type (i) and type (ii) in the residual graph during best-first search, but not the edges of type (iii). Instead, we use the values maintained in *capacity* and *current* arrays when deciding if we can extend the current min-cost path from a partition node to the sink. If extending the path to the sink (which will form an augmenting path) does not violate the area balance constraint of the partitions involved in the path, we obtain a min-cost augmenting path which satisfies the area balance constraint. Otherwise, we continue to process the next node in the best-first search ordering. With this modification, we can guarantee to generate a min-cost augmenting path which satisfies the *actual* area capacity constraint for each partition and the complexity of the algorithm is the same as that for regular min-cost max-flow using the shortest path algorithm.

3.5.5. Dynamic updating of binding functions during flow computation

One problem with the flow-based solution presented in an earlier sub-section is that the static *preference functions* are not reflective of the changes of the binding functions of the nets as more and more modules are assigned during flow computation. In Fig. 7, when m_1 is assigned to P_2 based on n_3's strong affinity, it is clear that $bf\ (n_4, m_2)$ should increase since the probability of its net being satisfied is increased, and hence the $pf\ (P_2, m_2)$ increases. Also, $bf\ (n_1, m_2)$ should be assigned zero since n_1 is being cut, and $pf\ (P_1, m_2)$ should be updated accordingly.

Binding functions can be updated efficiently as follows after module m_k is assigned to partition P_i. Let $N(m_k) = N_1 \bigcup N_2$, where N_1 is the subset of nets that are placed in P_i and N_2 are the subset of nets that are not in P_i. Let M_{cont} be the set of modules that are still in contention.

(i) Remove module m_k from the set of modules in contention.

$$M_{cont} \leftarrow M_{cont} - \{m_k\}$$

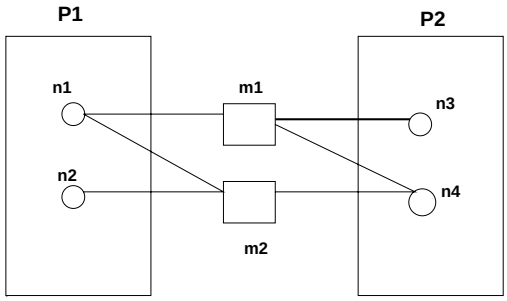


Figure 7 Example of dynamic update of binding factors

(ii) For each net n $\in N_1$, recompute the bf 's and update the pf 's associated with P_i incrementally.

for each net n $\in N_1$
  for all modules $m_j \in$ n, and $m_j \in M_{cont}$
    $pf\ (P_i, m_j) = pf\ (P_i, m_j) - (bf_{old}\ (n, m_j) - bf_{new}\ (n, m_j))$
    (Note that $bf_{old}\ (n, m_j) < bf_{new}\ (n, m_j)$ since $C(n)$ is reduced by one)

(iii) For each net n $\in N_2$, assign its bf 's to zero, and update the pf 's of all the partitions but P_i incrementally as follows:

for each net n $\in N_2$
  for each $m_j \in N_2$, and $m_j \in M_{cont}$
    $pf\ (P_i, m_j) = pf\ (P_i, m_j) - bf_{old}\ (n, m_j)$
    (Note that $bf_{new}\ (n, m_j) = 0$)

Such dynamic updating of binding functions and preference functions can be easily incorporated in our min-cost max-flow algorithm after each flow augmentation. As noted earlier, each flow augmenting path assigns one more module to a partition and also possibly specifies re-assignment of some other modules (see Figures 6 (a) and (b)). Therefore, after finding each augmenting path Q_i, we can update the binding functions related to the modules in Q_i, and then update the related preference functions accordingly. Therefore, the edge costs in the assignment network may change after each flow augmentation. Even with dynamic updating of edge costs in the assignment network, we can still show that flow augmentation stops after m steps. So, we have the following results:

Theorem 2 Assume that each module has a constant average degree in the netlist. With dynamic updating of area constraints, binding functions and preference functions, the K-MC problem can be solved in $O(K \cdot m^2 + m^2 \cdot \log m)$ time based on min-cost max-flow computation in the assignment network, where K is the number of partitions and m is the number modules in contention.

Proof Since we do not update edge capacity in flow computation, according to Lemma 1, the value of a maximum flow is still m and the algorithm stops after m flow augmentations. After each flow augmentation, we need to update the preference functions of the modules along the augmenting path, which takes at most $O(K \cdot m)$ time The overall complex-

ity[2] is then $O(m \cdot (K \cdot m + m^2 \cdot \log m + K \cdot m))$, which is still $O(K \cdot m^2 + m^2 \cdot \log m)$. □

In real-life CMOS VLSI designs, modules are connected on average to a few nets and hence our assumption of a constant average degree for modules is valid. The K-DualFM algorithm with dynamic updating of area constraints, binding functions in the flow computation is denoted as K-DualFM/DF, and the flow computation with static binding functions is denoted as K-DualFM/SF. From the results shown in the next section, we shall see that in general K-DualFM/DF produces better partitioning solutions compared to K-DualFM/SF. The increase in computation time due to dynamic updating of edge costs is negligible due to the incremental updating.

3.6. Refinement of module partitioning solution

After solving the K-MC problem we apply another pass of the K-FM partitioning algorithm to further refine the module partitioning solution. However, we observe that in all test cases the K-FM based refinement step converges very quickly with very few module moves, which is a strong indication that the K-FM module partitioning solution obtained from K-way net partitioning and module contention resolution is of very high quality.

3.7. K-DualMFFC-FM algorithm

Subsections 3.2 to 3.6 described the K-DualFM algorithm in detail and illustrated how to combine our dual netlist transformation based multi-way partitioning paradigm with the K-FM algorithm. To further demonstrate the power and effectiveness of our multi-way circuit partitioning paradigm based dual netlist representation, we present in this subsection how to apply our paradigm to the K-MFFC-FM algorithm by Cong, Li and Bagrodia [10].

To reduce the computational complexity of partitioning very large circuits, and to take the advantage of the signal direction information, Cong, Li and Bagrodia proposed a clustering method based on maximum fanout-free cones (MFFCs) [6, 7] as a preprocessing step to their partitioning algorithm. The MFFC decomposition technique was first proposed for combinational circuits [6] for duplication-free technology mapping of lookup-table based FPGAs. Let $input(v)$ denote the set of nodes which are the fanins of node v, and let $output(v)$ denote the set of nodes which are the fanouts of node v. For a node v in the network, a cone of v denoted C_v, is a subgraph of logic gates (excluding primary inputs (PIs)) consisting of v and its predecessors such that any path connecting a node in C_v and v lies entirely in C_v. We call v the root of C_v. A fanout-free cone (FFC) at v, denoted by FFC_v, is a cone of v such that for any node $u \neq v$ in FFC_v, $output(u) \subseteq FFC_v$. The maximum fanout free cone (MFFC) of v, denoted by $MFFC_v$, is a FFC of v such that for any non-PI node w, if $output(w) \subseteq MFFC_v$, then $w \in MFFC_v$. It is not difficult to show that MFFC is unique for every node, and any FFC of v is contained in $MFFC_v$. Clearly, if a gate u is in $MFFC_v$, its value is used solely for generating the output at gate v (and its descendants). Therefore, it is very natural to cluster u and v together. In general, all the gates in a single $MFFC_v$ can be considered to be closely related, since they are used solely for computation of v. The MFFC-based clustering algorithm considers both signal direction and logic dependency, and produces a natural clustering solution in linear time. The K-MFFC-FM algorithm reported in [10] applies the K-FM algorithm directly on the clustered circuits to get very promising results.

We applied our dual net transformation paradigm to K-MFFC-FM and obtained the K-DualMFFC-FM algorithm. In the K-DualMFFC-FM algorithm, modules were clustered using the MFFC algorithm, and then a net partitioning of the *HDN* of the clustered network was obtained using the K-FM algorithm. Then, the K-MC problem was solved using the

[2] This complexity analysis is based on using a Fibonacci heap. It is easy to see by similar analysis that dynamic updating does not increase the com-

min-cost max-flow technique, and the solution was further improved using the K-FM algorithm. By our convention, the algorithm with dynamic updating of binding functions is named K-DualMFFC-FM/DF and the algorithm with static binding functions is named K-DualMFFC-FM/SF. The experimental results for both K-DualFM and K-DualMFFC-FM are reported in the next section.

4. Experimental results

We have implemented both the K-DualFM/SF and K-DualFM/DF algorithms on SUN SPARC workstations. We compared the two algorithms with the conventional K-FM algorithms on a set of MCNC benchmark circuits (Test02-06, PrimGA1, PrimGA2) and 5 large circuits provided by the Hewlett-Packard Research Lab (CPU, GA, FPU, GA2, FPU2). Circuits CPU, GA and FPU consist of lookup-tables (for multi-FPGA implementation). Circuits FPU2 and GA2 are the original netlists of FPU and GA before technology mapping.

Table 1 shows the characteristics of the benchmark circuits, including the number of modules (gates and I/O pads) , number of nets, maximum module area versus total module area, the maximum number of modules in a net (max net), and the maximum number of nets incident to the same module (deg).

Tables 2(a)-2(d) show the comparison of K-DualFM/SF and K-DualFM/DF with the K-FM algorithm for K ranging from 2 to 5. For each example, the K-FM algorithm was run 20 times, each on a random initial module partitioning. In order to obtain a fair comparison, we make sure that the runtimes of K-DualFM/SF and K-DualFM/DF are comparable with that of the K-FM algorithm. As a result, the K-DualFM algorithms were run once with the greedy net partition, and then approximately 10 times[3], each on a random initial net partitioning of the dual netlist representation (HDN). The area slack parameter α was set to be 10% in both K-FM and K-DualFM algorithms.

Circuit	# Modules *	# Nets	Max module/ Total area	Max net	Deg
Test02	1724	1721	18776/57051	306	57
Test03	1664	1618	5295/22229	225	54
Test04	1541	1658	19928/42040	422	60
Test05	2650	2751	33509/71895	608	59
Test06	1813	1674	22/16968	388	6
PrimGA1	914	902	9/3432	18	9
PrimGA2	3121	3029	9/8373	37	9
cpu	947	1729	1/947	115	8
GA	6198	11049	1/6198	330	8
FPU	8046	15901	1/8046	245	8
GA2	25452	25628	**1/25452	337	57
FPU2	30669	33826	**1/30669	330	129

Table 1. Characteristics of the Test Circuits.

plexity of min-cost max-flow even if regular heaps or linear searches are used for choosing the min-cost node to expand.

*The number of modules include the I/O pads.

**For GA2 and FPU2 module areas were not given. The areas were assumed to be 1.0.

[3] The number of runs varies from example to example in order to match the runtime of 20-run K-FM algorithm on the same example in order to obtain fair comparison. The MCNC circuits and circuits CPU, GA, and FPU from HP usually ranged from 10 - 12 runs, while the unmapped circuits FPU2 and GA2 from HP ranged between 5 and 8 runs.

Circuit	K-FM	K-DualFM/ SF	K-DualFM/ DF
Test02	245	130	130
Test03	175	140	137
Test04	46	60	60
Test05	42	59	43
Test06	219	106	106
PrimGA1	104	77	42
PrimGA2	585	246	214
cpu	263	341	175
GA	496	157	214
FPU	488	409	344
GA2	448	440	492
FPU2	450	632	570
overall	1.527	1.229	1.0

Table 2 (a) Comparison of K-DualFM against K-FM for K = 2

Circuit	K-FM	K-DualFM/ SF	K-DualFM/ DF
Test02	114	97	97
Test03	271	219	199
Test04	1017	817	823
Test05	1069	898	767
Test06	476	424	408
PrimGA1	209	188	175
PrimGA2	767	270	231
cpu	951	882	796
GA	2270	2127	2038
FPU	1238	1468	1383
GA2	5772	4181	2829
FPU2	5673	2860	3776
overall	1.466	1.081	1.0

Table 2 (b) Comparison of K-DualFM against K-FM for K = 3

Circuit	K-FM	K-DualFM/ SF	K-DualFM/ DF
Test02	1013	796	796
Test03	589	518	496
Test04	1143	913	876
Test05	1809	1665	1103
Test06	723	519	487
PrimGA1	292	270	231
PrimGA2	1303	907	821
cpu	716	682	626
GA	3917	3263	3021
FPU	3178	1654	1654
GA2	10112	4915	2968
FPU2	7300	5684	5552
overall	1.568	1.148	1.0

Table 2 (c) Comparison of K-DualFM against K-FM for K = 4

Circuit	K-FM	K-DualFM/ SF	K-DualFM/ DF
Test02	963	700	637
Test03	645	618	598
Test04	1213	1052	933
Test05	1743	1563	1013
Test06	868	629	592
PrimGA1	290	251	206
PrimGA2	1104	907	821
cpu	734	631	601
GA	4643	3296	3340
FPU	4100	3131	3047
GA2	11401	7051	5246
FPU2	11455	7586	7055
overall	1.465	1.139	1.0

Table 2 (d) Comparison of K-DualFM against K-FM for K = 5

One can see from Tables 2(a) - 2(d) that the K-DualFM/DF algorithm consistently outperforms the K-FM algorithm by a significant margin, from about 20% to 31% reduction in net cutsize for K = 2 through 5. The K-DualFM/SF algorithm produces results with about 12% to 21% cutsize reduction as compared to K-FM for K = 2 through 5. The K-DualFM/DF algorithm in general outperforms the K-DualFM/SF algorithm for all the circuits tested, and indicates that dynamic updating of binding functions is indeed useful. In terms of efficiency, the K-DualFM/SF algorithm is generally faster than the K-DualFM/DF algorithm, and the difference increases as the number of partitions increases. Each run of K-DualFM/SF for FPU2 took around 6500 seconds and 8200 seconds for K = 3 and K = 4, respectively on a SUN SPARC10 workstation, while each run for K-DualFM/DF took around 6900 seconds and 9000 seconds, respectively.

There are also improvements, though less dramatic, when the reduction in average net cutsize is considered as opposed to the minimum net cutsize. For instance, the K-DualFM/DF algorithm reduces the average net cutsize (when compared to the K-FM algorithm) by 12% for the MCNC circuits for K = 4. Similarly, the K-DualFM/SF reduces the average net cutsize by 10%. The standard deviations of the net cutsizes from the three algorithms are comparable (of 9% - 13%).

The *CF* parameter was set to 5 for both K-DualFM algorithms for all the results shown here. The results may vary considerably if we change the value of *CF*. As we change the *CF* parameter from 2 to 10, the cutsize obtained by the K-DualFM algorithms varies by 18% on average, with a standard deviation 12%. However, the range of the *CF* parameters is very small since it is related to the degree of the modules in a netlist. For instance, 99% of the modules in all of the MCNC benchmark and HP Research Lab circuits have a degree of less than 9. Thus one can easily find a good range for the *CF* value.

We compared the K-DualFM algorithm with both EIG1 [18] and Paraboli [26], two recently reported spectral-based methods specialized to solve the bipartitioning problem. EIG1 uses a quadratic objective function, while Paraboli considers the linear objective function and obtains impressive improvement over EIG1 with considerable longer computation time. We present comparisons of K-DualFM with these two algorithms in Table 3 for circuits in the ACM/SIGDA benchmark suite based on the results reported in [26]. The area slack was set to 10%, and the number of runs for K-DualFM was set to be 10 since it leads to comparable runtime consumed by EIG1. We see that in general the K-DualFM algorithms produce bipartitioning solutions with comparable net cutsizes as Paraboli while consuming much less computation time, and consistently outperform EIG1 by a significant margin (56% net cutsize reduction on average) while consuming the same amount of time. Moreover, K-DualFM has the following advantages over EIG1 and Paraboli: (i) The

area balance constraint cannot be modeled naturally in the EIG1 and Paraboli formulation, but is easily accommodated in K-DualFM. (ii) EIG1 and Paraboli cannot handle some practical constraints such as the pre-assignment of certain modules to partitions, but K-DualFM can accommodate such constraints easily by pre-assigning these modules in the initial partitioning solution and marking them 'locked' throughout all phases of the partitioning. (iii) Furthermore, EIG1 and Paraboli are designed specifically for bipartitioning while K-DualFM is applicable to general multi-way partitioning.

Finally, we report the results obtained by the K-DualMFFC-FM algorithm, which was described in Section 3.7. The K-MFFC-FM algorithm was developed to partition a circuit to multiple processors for parallel logic simulation. Since the goal is to minimize inter-processor communication, Cong, Li and Bagrodia [10] modified their objective function to minimize the number of nets cut while ignoring the connections between gates and input/output pads. (This is because in their application a primary input variable can be made available for every processor, and generating an output to an output pad is equivalent to a write operation to local disk with no communication overhead.) Our K-DualMFFC-FM algorithm was modified accordingly to use the same objective function for fair and accurate comparison. Tables 4(a)-4(b) show the comparison of the K-DualMFFC-FM algorithms with the original the K-MFFC-FM[4] [10] under the same objective function for the ICSAC85 benchmark suite (which consist of only combinational circuits and are suitable for MFFC clustering). The area slack was set to 5%, and K-MFFC-FM was run 20 times, and the K-DualMFFC-FM[5] algorithms were run from 10 - 12 times for comparable runtimes. The β value that bounds the maximum size of the cluster was set to 1/3 for both algorithms since higher values (like 1/2) sometimes did not produce area-balanced partitions for the K-MFFC-FM algorithm. From Tables 4 (a) and (b), we see that the K-DualMFFC-FM algorithms have on the average 15 to 26% smaller cutsize than K-MFFC-FM for the ICSAC85 benchmark suite.

5. Conclusion and possible future extensions

The results in this paper show convincingly that net partitioning based methods produce better solutions to the multi-way circuit partitioning problem than direct module partitioning. Our formulation and solution to the K-way module contention problem provide a general and effective frame-work to convert a K-way net partitioning solution to a K-way module partitioning solution. Both the K-DualFM/SF and K-DualFM/DF algorithms can be extended easily to handle many practical constraints, such as I/O bound constraint on each partition, pre-specified assignment of modules to

Cicuit	EIG1	Paraboli	K-DualFM/ SF	K-DualFM/ DF
balu	83	41	27	27
struct	102	40	44	38
primary1	81	53	72	52
biomed	729	135	123	107
s13207	241	91	110	110
s15850	215	91	140	93
s9234	227	74	77	77
industry2	620	193	804	493
	2.85	1.002	1.228	1.0

Table 3 Comparison of K-DualFM against EIG1 and Paraboli for K = 2

[4] K-MFFC-FM is abbreviated to K-MFM in Table 4.

[5] K-DualMFFC-FM is abbreviated to K-DualMFM in Table 4.

Cicuit	K-MFM	K-DualMFM/ SF	K-DualMFM/ DF
c880	42	21	21
c1355	17	17	17
c1908	84	68	68
c2670	161	162	144
c3540	178	153	142
c5315	186	111	99
overall	1.441	1.046	1.0

Table 4 (a) Comparison of K-DualMFFC-FM against K-MFFC-FM for K = 4

Cicuit	K-MFM	K-DualMFM/ SF	K-DualMFM/ DF
c880	60	52	52
c1355	43	41	39
c1908	119	92	92
c2670	208	193	193
c3540	229	212	205
c5315	260	184	172
overall	1.206	1.026	1.0

Table 4 (b) Comparison of K-DualMFFC-FM against K-MFFC-FM for K = 8

partitions, etc.

Our partitioning results for the test circuits, ranging from 1000 to 31,000 modules, prove that our K-DualFM algorithm is scalable in terms of the size of the circuit. Other partitioning algorithms (such as the graph spectral based method or the random-walk based method) may fail to produce solutions for large circuits due to high memory usage (e.g. to store the Laplacian matrix or a random-walk of quadratic length in terms of the number of modules) or speed inefficiency. The memory and speed efficiency of the K-DualFM algorithm enables us to handle problems of much larger sizes.

When the problem size is not too large, more elaborate K-way partitioning algorithms other than the simple K-FM algorithm can also be used to produce a better net partitioning of the dual netlist representation. Improvement on the net partitioning solutions usually leads to improvement on the resulting module partitioning solution.

In this paper, we concentrated primarily on minimizing the number of nets cut subject to area constraints. Our general framework is applicable to other objective functions also. For example, our methods can be extended to handle the following two objectives:

(F1) The ‘‘wirability’’ objective function: This is a popular objective function used to minimize the number of inter-chip wire crossings. Our dual netlist approach can be used to optimize this function by setting binding functions between a net and any of its contended modules to one. In this case, one can show that a min-cost max-flow in the assignment network corresponds to a module assignment solution with the minimum total wire crossing.

(F2) Performance-based objective function: Some recently proposed objectives optimize for system performance under timing constraints. For example, Shih, Kuh and Tsay[28] proposed an algorithm to minimize the wirability objective function subject to thermal, I/O pin, and timing constraints in addition to area constraints. In (F1) we showed how to minimize the wirability function. The thermal and I/O pin constraints can be handled in a similar way as the

area constraint. Hence, the algorithm proposed in Section 3.5.4 can be modified to accommodate these two constraints (with two additional arrays). The timing constraints can be handled by assigning locations to partitions such that each pair of partitions will be at a certain distance from each other. We can then ensure that timing constraints are satisfied when the preference functions are being updated in the algorithm (as explained in Subsection 3.5.5, Steps (ii) and (iii)). In particular, once a module, belonging to a net n, is assigned to a partition P, the unassigned modules of n are constrained to be placed in partitions that are sufficiently "close" to P. This constraint can be satisfied by making the cost of the edges from these unassigned modules to "distant" partitions prohibitively high.

We also observed that the choice of *CF* parameter defined in Subsection 3.3 may affect the partitioning results considerably. We would like to develop effective algorithms to automatically compute the value of CF parameter based on the characteristics of the circuits so that the K-DualFM algorithm can consistently produce better partitioning results.

6. Acknowledgments

The authors would like to thank Phil Kuekes and Greg Snider at Hewlett-Packard Laboratory for providing benchmark circuits. This work is partially supported by ARPA/CSTO under contract J-FBI-93-112, the National Science Foundation Young Investigator Award award number MIP9357582, and grants from AT&T Bell Laboratories, Hewlett-Packard, Xerox Foundation, Xilinx under the California MICRO program and NY1 Award matching program.

References

[1] C. J. Alpert and A. B. Kahng, ‘‘Geometric Embeddings for Faster (and Better) Multi-Way Netlist Partitioning,’’ *Proc. ACM/IEEE Design Automation Conf.*, pp. 743-748, June 1993.

[2] E. R. Barnes, ‘‘An Algorithm for Partitioning the Nodes of a Graph,’’ *SIAM J. Alg. Disc. Math.*, Vol. **3**, pp. 541-550, 1982.

[3] M. Beardslee and A. Sangiovanni-Vincentelli, ‘‘Heuristic Methods for Communication-Based Logic Partitioning,’’ *4th ACM/SIGDA Physical Design Workshop*, pp. 199-210, April 1993.

[4] R. Boppana, ‘‘Eigenvalues and Graph Bisection: An Average-Case Analysis,’’ *IEEE Symp. on Foundations of Computer Science*, pp. 280-285, 1987.

[5] P. Chan, M. Schlag, and J. Zien, ‘‘Spectral K-Way Ratio-Cut Partitioning and Clustering,’’ *Proc. 30th ACM/IEEE Design Automation Conf.*, June 1993.

[6] J. Cong and Y. Ding, ‘‘On Area/Depth Trade-off in LUT-Based FPGA Technology Mapping,’’ *Proc. 30th ACM/IEEE Design Automation Conf.*, pp. 213-218, June 1993.

[7] J. Cong and Y. Ding, ‘‘On Area/Depth Trade-off in LUT-Based FPGA Technology Mapping,’’ *IEEE Trans. on VLSI Systems*, Vol. **2**, pp. 137-148, June 1994.

[8] J. Cong, L. Hagen, and A. B. Kahng, ‘‘Random Walks for Circuit Clustering,’’ *IEEE 4th Int’l ASIC Conf.*, pp. P14-2.1, Sept. 1991.

[9] J. Cong, L. Hagen, and A. B. Kahng, ‘‘Net Partitions Yield Better Module Partitions,’’ *IEEE 29th Design Automation Conference*, pp. 47-52, June 1992.

[10] J. Cong, Z. Li, and R. Bagrodia, ‘‘Acyclic Multi-Way Partitioning of Boolean Networks,’’ *Proc. ACM/IEEE 31st Design Automation Conf.*, pp. 670-675, June 1994.

[11] T. Cormen, C. Leiserson, and R. Rivest, *Algorithms*, MIT Press, Cambridge, MA (1990).

[12] J. Cong, W. Labio, and N. Shivakumar, ''Multi-Way VLSI Circuit Partitioning Based on Dual Net Representation,'' *Proc. IEEE Int'l Conf. on Computer-Aided Design*, pp. 56-62, Nov. 1994. Also available as UCLA Computer Science Department Tech. Report CSD-940029

[13] J. Cong and M. Smith, ''A Bottom-up Clustering Algorithm with Applications to Circuit Partitioning in VLSI Designs,'' *ACM/IEEE Design Automation Conf.*, pp. 755-760, June 1993.

[14] W. Donath and A. Hoffman, ''Lower Bounds for the Partitioning of Graphs,'' *IBM J. Res. Dev.*, pp. 420-425, 1973.

[15] C. Fiduccia and R. Mattheyses, ''A Linear Time Heuristic for Improving Network Partitions,'' *ACM/IEEE Design Automation Conf.*, pp. 175-181, 1982.

[16] L. R. Ford and D. R. Fulkerson, *Flows in Networks*, Princeton Univ. Press, Princeton, N.J. (1962).

[17] J. Greene and K. Supowit, ''Simulated Annealing without Rejected Moves,'' *Proc. Int'l Conf. on Computer Design*, pp. 658-663, 1984.

[18] L. Hagen and A. B. Kahng, ''Fast Spectral Methods for Ratio Cut Partitioning and Clustering,'' *Proc. IEEE Int'l Conf. on Computer-Aided Design*, pp. 10--13, 1991.

[19] L. Hagen and A. B. Kahng, ''A New Approach to Effective Circuit Clustering,'' *Int'l Conf. on Computer-Aided Design*, pp. 422-427, Nov. 1992.

[20] L. Hagen and A. B. Kahng, ''New Spectral Methods for Ratio Cut Partitioning and Clustering,'' *IEEE Trans. on CAD*, pp. 1074-1085, Sept. 1992.

[21] J. Hwang and A. El Gamal, ''Optimal Replication for Min-Cut Partitioning,'' *Int'l Conf. on Computer-Aided Design*, pp. 432-435, Nov. 1992.

[22] B. Kernighan and S. Lin, ''An Efficient Heuristic Procedure for Partitioning of Electrical Circuits,'' *Bell System Technical J.*, Feb. 1970.

[23] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, Jr., ''Optimization by Simulated Annealing,'' *Science*, Vol. **220**, pp. 671-680, May 1983.

[24] B. Krishnamurthy, ''An Improved Min-Cut Algorithm for Partitioning VLSI Networks,'' *IEEE Trans. on Computers*, Vol. **33**, pp. 438-446, 1984.

[25] C. Kring and A. R. Newton, ''A Cell-Replicating Approach to Mincu-Based Circuit Partitioning,'' *IEEE Int'l Conf. on Computer-Aided Design*, pp. 2-5, Nov. 1991.

[26] B. M. Riess, K. Doll, and F. M. Johannes, ''Partitioning Very Large Circuits Using Analytical Placement Techniques,'' *Proc. ACM/IEEE 31st Design Automation Conf.*, June 1994.

[27] L. Sanchis, ''Multiple-Way Network Partitioning,'' *IEEE Trans. on Computers*, Vol. **38**, pp. 62-81, 1989.

[28] M. Shih, E. Kuh, and R. Tsay, ''Performance-Driven System Partitioning in Multi-Chip Modules,'' *Proc. ACM/IEEE Design Automation Conf.*, pp. 53-56, June 1992.

[29] H. Yang and D. F. Wong, ''Efficient Network Flow Based Min-Cut Balanced Partitioning,'' *Proc. IEEE Int'l Conf. on Computer-Aided Design*, pp. 50-55, Nov. 1994.

[30] C. W. Yeh, C. K. Cheng, and T. T. Lin, ‘‘A General Purpose Multiple-Way Partitioning Algorithm,’’ *Proc. 28th ACM/IEEE Design Automation Conf.*, June 1991.

[31] C. W. Yeh, C. K. Cheng, and T. T. Lin, ‘‘A Probabilistic Multicommodity-Flow Solution to Circuit Clustering Problems,’’ *Int’l Conf. on Computer-Aided Design*, pp. 428-431, Nov. 1992.