Spectral-Based Multi-Way FPGA Partitioning

Pak K. Chan*, Martine D.F. Schlag,†and Jason Y. Zien

Computer Engineering
University of California, Santa Cruz
Santa Cruz, California 95064 USA

Abstract— Recent research on FPGA partitioning has focussed on finding minimum cuts between partitions without regard to the routability of the partitioned subcircuits. In this paper we develop a spectral approach to multi-way partitioning in which the primary goal is to produce routable subcircuits while maximizing FPGA device utilization. To assist the partitioner in assessing the routability of the partitioned subcircuits, we have developed a theory to predict the routability of the partitioned subcircuits prior to partitioning. Advancement over the current work is evidenced by results of experiments on the standard MCNC benchmarks.

I FPGA partitioning

The design flow using commercial FPGA tools involves technology mapping, placement and routing. Since FPGA devices have relatively low density, the use of multiple FPGAs is often required to implement a large circuit. A large circuit has to be decomposed or partitioned into subcircuits for a multiple-FPGA realization, as shown in Fig. 1.

Modifications of standard iterative mincut-based partitioning algorithms have been applied to FPGA partitioning. In [10], Kuznar *et al* considered the problem of partitioning a circuit for heterogeneous FPGA systems. Their cost function was the total-dollars to implement the circuit. This approach is perhaps suitable from the standpoint of developing a new multiple-FPGA board to realize the given circuit. However, it doesn't consider the (labor) cost to layout the FPGA systems, where regularity is one of the major design issues. In addition, most existing multiple FPGA systems are homogeneous systems; for example, the Quickturn emulators and reconfigurable FPGA-based computing engines. So there is a need for partitioning algorithms for homogeneous FPGA systems.

Given a large circuit, a partitioner generates many subcircuits. Only when *all* the subcircuits have been successfully placed and routed, will the multiple FPGAs realize the large circuit. An important issue that has not been considered in the literature is the routability of the partitioned subcircuits. In this paper, we shall see that the manner in which a circuit is partitioned can determined whether *all* the subcircuits can be automatically routed. We present a theory to predict the routability of partitioned subcircuits prior to partitioning: pre-partitioning routability prediction. The routability predictor can assist a user in determining the number of partitions, and the selection of FPGA devices (when possible) to realize the large circuit.

Knowing the parameters that would affect the routability of the subcircuits, we devise a spectral-based partitioning algorithm to decompose a large circuit into subcircuits. Routability of the partitioned subcircuits is the primary concern. The inputs to our partitioner include partition size and cut constraints. Although spectral-based partitioning algorithms have not been recognized for their ability to realize hard constraints, we shall demonstrate that spectral-based partitioning algorithms are excellent candidates for routability reasons.

In this paper, we shall present two main results on spectral-based partitioning targeted for homogeneous FPGA systems:

1. A theory to predict the routability of the partitioned subcircuits to assist the partitioner in assessing the routability of the partitioned circuits prior to partitioning.
2. A k-way, spectral partitioning method which handles both partition-size and cut-size constraints. The partitioner's primary goal is to generate routable subcircuits while maximizing logic utilization.

We compare our partitioning results to those in the

*Supported in part by NSF Grant MIP-9196276 & MIP-9223740 & MICRO Program of University of California with matching support from Xilinx Inc.

†Supported in part by NSF Grant MIP-9223740 & MICRO Program of University of California with matching support from Xilinx Inc..

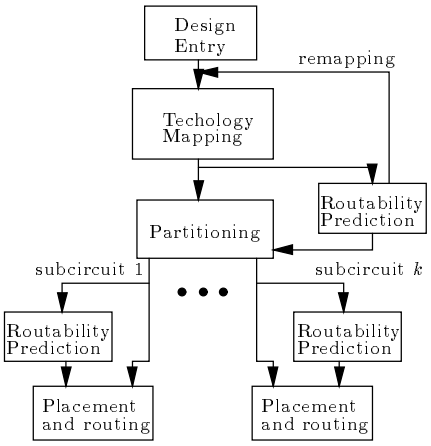

Figure 1: Multiple FPGAs design flow.

literature using the standard MCNC Partitioning 1993 benchmarks. Our partitioner yields partitions which are predictable in the routability sense, while still producing fewer partitions in comparison to results reported in the literature.

II Routability of subcircuits

A routability barometer has been suggested in [4] to predict the routability of a single FPGA circuit. We attempt to extend the technique to predict the routability of the partitioned subcircuits. There are two ways to approach this. The straightforward approach is to partition the circuit, and apply the single-FPGA routability predictor to the partitioned subcircuits. We call this post-partitioning routability prediction. A more ambitious approach is to predict the routability of the subcircuits prior to partitioning, and we call this pre-partitioning routability prediction. This approach has the benefit that it can assist a user in determining the number of partitions, and the suitable FPGA devices required to generate routable subcircuits. Given a single FPGA circuit, a single-FPGA routability predictor benefits from the fact that the structure of the circuit can be extracted, hence the average wire length can be estimated with accuracy. Pre-partitioning routability prediction for multiple FPGAs is harder because the structures of the partitioned circuits can only be estimated before partitioning, and the predictions might not be as accurate as the single-FPGA case.

Figure 1 depicts the roles of both routability predictions in the multiple-FPGA design flow. Given a large circuit to be partitioned, the design flow also suggests two possible methods to control the routability of the partitioned subcircuits:

1. remap the given circuit to achieve 100 percent routable subcircuits [13, 3], and
2. partition the given circuit to facilitate the routability of the subcircuits.

Here, we shall focus on the second method.

Simply put, the routability prediction for a single FPGA circuit involves estimating the channel width requirement of the circuit for routing completion [8]. The parameters of the circuit involved are:

1. γ: the average number of pins emanating from each Configurable Logic Block (CLB), we shall refer to γ as the pins-per-CLB ratio,
2. β: the average number of pins on a net (we refer to β as the pins-per-net ratio), and
3. L: the average wire length of the circuit after placement.

The *average* channel width requirement for routing completion is [8, 4]:

$$W = \frac{1}{2}\left[\gamma \cdot \left(1 + \frac{\beta - 2}{\beta}\right) L\right] \ . \qquad (1)$$

Routability is determined by comparing W with the FPGA device's channel width. We present a theory to predict the *average* channel width requirement of subcircuits before partitioning by predicting the average pins-per-CLB and pins-per-net ratios of the partitioned subcircuits (before the circuit is partitioned). The inputs to the routability predictor are:

1. the total number of CLB pins of the (original) circuit,
2. the total number of nets in the circuit,
3. the average wire lengths of the subcircuits,
4. the number of partitions k, and
5. the average number of I/O pins per partition.

The first two parameters can be calculated from the (unpartitioned) circuit. The average wire lengths of the subcircuits can only be estimated (see Equation (8) later). The last two parameters depend on the devices in the FPGA system. Assuming that the pins-per-CLB ratios do not vary widely among the partitioned subcircuits, on the average the pins-per-CLB ratio of a partitioned subcircuit is:

$$\overline{\gamma}_p \approx \gamma = \text{total CLB pins/total CLBs} \ . \qquad (2)$$

Similarly, the average pins-per-net ratio, $\overline{\beta}_p$, of the partitioned subcircuits is

$$\text{(CLB + IO pins in a partition)/nets in a partition} \qquad (3)$$

The average pins-per-net ratio $\overline{\beta}_p$ can be estimated with good accuracy, as we shall see. Let

1. $pins$ denote total CLB pins before partitioning,
2. $pins_p$ denote CLB pins in partition p,
3. $nets$ denote total nets before partitioning,
4. $nets_p$ denote nets (with one or more pins) contained in partition p,
5. IO_b denote total IOs before partitioning (sum of all IO pins in the original circuit),
6. IO_a denote total IOs after partitioning, (total number of IO pins in all the partitioned subcircuits),
7. io_p denote the number of IO pins in partition p, and
8. C denote the number of nets cut by partitioner.

The number of nets before partitioning and after partitioning can be related as:

$$nets = \sum_{p=1}^{k} nets_p - (IO_a - IO_b) + C \qquad (4)$$

which essentially states that "nets" are conserved. Equation (4) follows from the observation that when a net is partitioned into t pieces exactly t new I/Os will be introduced if the net had no I/O pin to begin with, while only $t-1$ new I/Os will be introduced if it had an I/O pin (if it was already cut). The nets cut by partitioner (C) term only counts the nets cut by the partitioner which were not connected to the original IOs. Equation (4) is exact, and on the average:

$$\overline{nets}_p = (nets + IO_a - IO_b - C)/k \qquad (5)$$

so the average pins-per-net ratio is

$$\begin{aligned} \overline{\beta}_p &\approx \frac{\text{average no. of pins in a partition}}{\text{average no. of nets in a partition}} \qquad (6) \\ &= (\overline{pins}_p + \overline{io}_p)/\overline{nets}_p \\ &= k \times (\overline{pins}_p + \overline{io}_p)/(nets + IO_a - IO_b - C) \\ &= (pins + k \times \overline{io}_p)/(nets + IO_a - IO_b - C). \end{aligned}$$

We'll need to make some approximation of Equation (4) before Equation (6) can be useful. We approximate: $C \approx (IO_a - IO_b)/\min(k, \beta)$ since these are the nets that are not connected to the IOs. On the average these nets fanout to $\frac{k}{}$ or β chips, which ever is smaller. Also, $IO_a = k \times \overline{io}_p$. So, we deduce

$$\overline{\beta}_p \approx \frac{(pins + k \times \overline{io}_p)}{nets + (k \times \overline{io}_p - IO_b)(1 - 1/\min(k, \beta))} \qquad (7)$$

Given a circuit, we can extract the number of pins, the number of nets, and the number IO pins, so we know β. If we know the number of partitions and the average number of IO pins used after partitioning, we should be able to predict the average pins-per-net ratio $\overline{\beta}_p$ prior to the actual partitioning. A good guess of $\overline{io}_p$ is the number of I/O pins available on the FPGA device. From Equations (1), (2), and (7), we can predict the routability of the partitioned circuits even before partitioning if the average wire length can be estimated. Our approach is to calculate L based on the logic utilization of the FPGA devices and the FPGA device used:

$$L = \sqrt{\text{logic\_utilization}} \times L_{device\_type} \qquad (8)$$

where L_{device_type} depends solely on the FPGA device type used. We shall validate our routability theory and Equation (7) through experiments presented in Section IV.

Equation (7) identifies an important aspect of partitioning for routability; the number of IOs ($\overline{io}_p$) has some bearing on the pins-per-net ratio (hence the routability) of the partitioned subcircuits. Equation (7) indicates that $\overline{\beta}_p$ *decreases* with increasing $\overline{io}_p$, this suggests that a routability-driven partitioner should maximize the IO utilization, whenever it is possible. Intuitively, a partitioner decomposes the nets of a circuit into inter-partition net and intra-partition nets. Equation (4) expresses the law of "conservation of nets." Large cut sizes imply smaller fanout inside the FPGA devices, making the subcircuits more routable.

There are tradeoffs between routability and the cost of devices to implement the partitioned subcircuits. Routability generally increases with reduced utilization, whereas cost decreases with utilization. The art of routability-driven partitioning is to seek the fine balance between routability and cost. It is not obvious that mincut-based algorithms can handle this seemingly paradoxic set of constraints. Typically, mincut-based partitioning algorithms minimize inter-partition cuts [14]. We have devised a spectral-based partitioner that maximizes the IO utilization while minimizing the number of FPGA devices. This will be discussed in the next section.

III Spectral partitioning

We provide some background to understand spectral-based partitioning. We shall also discuss the subtlety involved in choosing the proper graph matrix (Laplacian or the adjacency matrix) in order to handle both partition-size and cut-size constraints. An instance of the graph partitioning problem consists of a graph, $G = (\mathcal{V}, \mathcal{E})$ with vertices, $\mathcal{V} = \{\nu_1, \nu_2, \ldots, \nu_n\}$, and weighted edges where the weight of edge $e = (\nu_i, \nu_j)$, represents the cost of putting ν_i and ν_j in separate partitions. The problem is to find a partition of the set of vertices $\mathcal{P} = \{P_1, P_2, P_3, ...P_k\}$ for a given k, which optimizes some cost criterion based on the weights of the edges cut and/or the sizes of the partitions.

The *adjacency matrix* of G is the $n \times n$ matrix $A(G) = [a_{ij}]$ where a_{ij} is the weight of the edge between nodes ν_i and ν_j. The *degree matrix* of G is the $n \times n$ matrix $D(G) = [d_{ij}]$ defined by, $d_{ij} = \begin{cases} \sum_{k=1}^{n} a_{ik} & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$. The *Laplacian* of G is the $n \times n$ symmetric matrix $Q(G) = D(G) - A(G)$.

III-A Laplacian or adjacency matrix?

Spectral partitioning forms clusters of vertices based on the embedding implied by the eigenvectors V of a graph matrix, which can be the Laplacian Q, or the adjacency matrix A of the graph. To minimize the ratio-cut cost metric, researchers used the Laplacian Q to obtain an embedding [11, 2] of the eigenvectors. This is the correct approach since spectral embedding of the Laplacian appears to be closely related to the ratio-cut cost metric.

Let Π be the set of all k-way partitions of a graph G, and E_h be the total weight of the edges in G having exactly one endpoint in partition P_h. In [5]

the authors show that,

$$\min_{X^T X = I} trace(X^T Q X) \leq \min_{\mathcal{P} \in \Pi} \sum_{h=1}^{k} \frac{E_h}{|P_h|} \qquad (9)$$

which provides a lower bound on k-way ratio-cut cost metric. The first k eigenvectors V of the Laplacian Q satisfy this inequality and the eigenvectors can be used to form the projector, VV^T, as an approximation of the ratioed partition matrix [5]. Note that minimization of the ratio-cut cost metric implies that partition size and partition cut constraints *are not* imposed simultaneously on the partitions. This is a consequence of the definition of the ratio-cut cost metric.

The adjacency matrix $A(G)$ of a graph G is a less popular choice of the matrix to obtain an embedding. Based on Donath/Hoffman's result [7], Rendl/Wolkowicz [12] derived an upper bound on the weight of the edges uncut (E_{uncut}) by a partition satisfying pre-determined partition sizes. If $m_1 \geq m_2 \geq \ldots \geq m_k$ are the target partition sizes, $M = diag(m_1, ..., m_k)$, and $\lambda_{n-k+1} \leq \lambda_{n-k+2} \leq \ldots \leq \lambda_n$ are the largest k eigenvalues of the adjacency matrix, then

$$\sum_{i=n-k+1}^{n} m_i \lambda_i(A) = \max_{X^T X = I} \left\{ trace(M X^T A X) \right\} \geq 2 E_{uncut}.$$

If all the partitions have the same size, $m_1 = m_2 = \ldots = m_k = m$, then the above equation can be simplified to:

$$\sum_{i=n-k+1}^{n} \lambda_i(A) = \max_{X^T X = I} \left\{ trace(X^T A X) \right\} \geq \frac{2 \times E_{uncut}}{m}$$

which provides an upper bound on the number of edges uncut. The last k eigenvectors V of the adjacency matrix A satisfy this inequality and the eigenvectors can be used to form the projector VV^T. It is an approximation of the ratioed partition matrix [5]. Consequently the partitions will be equal-sized, roughly speaking. Hence, the adjacency matrix A is a better choice than the Laplacian Q to partition a circuit for a homogeneous FPGA system.

Equation (7) states that the pins-per-net ratio decreases with increasing number of I/O pins used. So, it is meaningful for a partitioner to find suboptimal cuts for routability purposes. First, a spectral-based partitioner has the inherent property that it can generate a wide selection of cuts. It is easy to see why. For example, from Equation (9), if we exclude the first eigenvector v_1 from the minimization, then the lower bound will be increased:

$$\min_{\substack{X^T X = I \\ X^T v_1 = 0}} trace(X^T Q X) = \sum_{h=2}^{k+1} \lambda_i(Q) \geq \sum_{h=1}^{k} \lambda_i(Q).$$

This has the implication that more cuts (consequently more I/O pins) would be obtained by using intermediate eigenvectors to induce the partitions (A similar argument holds for the adjacency matrix). Second, some net models used in transforming hypergraphs into graphs are known to produce more cuts than the others, we may use those net models to increase the cuts. As the last resort, we can use a different graph matrix. For example, use the Laplacian in place of the adjacency matrix.

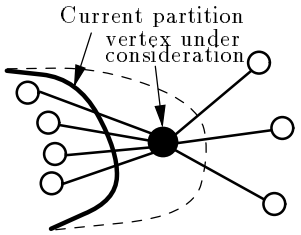

Figure 3: Ranking heuristic based on Cuts.

III-B From Eigenvectors to Partitions

Given the logic capacity and I/O capacity constraints, we present a procedure to use the eigenvector embedding of a graph matrix to construct partitions that satisfy the constraints. As in [5], our approach to k-way partitioning is to "reverse engineer" the partitions from the embeddings implied by the eigenvectors of A, $V = [v_{n-k+1}, ..., v_n]$ (or the intermediate eigenvectors $V = [v_{n-k+1-q}, ..., v_{n-q+1}]$, for some $q \leq n$ if so desired). [1] Given the eigenvectors, we measure the cosine of the angle between the two row vectors i and j of V (or the column vectors of V^T). These *directional cosines* provide a measure of the proximity of the vertices relative to each other. This strategy, in effect, identifies all the '1's in the partitioning matrix P implied by the projector VV^T. The first step is to determine a vertex of largest magnitude to serve as the *prototype* (seed) for the first partition. The first prototype is selected by magnitude, and the rest are determined by their relative (anti)orthogonality with respect to the existing prototypes.

A vertex is said to be *allocated* if it has been assigned and committed to a partition. After the determination of the prototype, the rest of the unallocated vertices are ranked based on their anti-orthogonality with respect to the prototype. The ranked vertices are sorted in a heap data structure, which is updated as vertices are extracted from the heap.

An additional factor to be considered in the ranking of the vertices is the number of *cuts* incurred by including a vertex in a partition. The ranking function for the vertices is:

$$rank = anti\_orthogonality - f \times cuts\_incurred \qquad (10)$$

where f is a weight parameter associated with the heuristics. We use $0.5 < f < 2.5$; a typical value is 1. We have two different strategies. We call them the *pcut* and *pcluster* heuristics, respectively. Figure

[1] We use the first k eigenvectors of the Laplacian graph matrix, and the last k eigenvectors of the adjacency matrix.

```
KPF_Partition(I/O constraint, logic constraint) {
        Remove high-fanout nets in the hypergraph and transform it to a graph
        Find k eigenvectors of this graph, V

        Associate each row of V with its vertex in the original hypergraph
        Mark all vertices unallocated
        h=0;
        Select vertex with largest-magnitude vector as 1st prototype

        while (there are unallocated vertices) {
                h++;
                if h ≠ 1 select vertex most orthogonal to previous
                        prototype and use it as the prototype for partition h
                Build heap based on the ranking function, Eq. (10)
                while partition h has not reached logic capacity {
                        remove largest cost vertex, ν, from heap
                        assign ν to partition h
                        update cut tally for each net to which ν belongs
                        update heap cost for all vertices adjacent to ν
                        put ν on Rollback list if I/O capacity is exceeded
                        reset Rollback list if I/O capacity is not exceeded
                }
                Remove vertices on Rollback list from h and mark them unallocated
        }
        Print partition results
}
```

Figure 2: Pseudo-code for spectral **KPF** partitioning method.

3 illustrates a vertex (in solid) which, with the *pcut* heuristics, is considered to have saved $4-3=1$ cut if incorporated into the current partition; whereas with the *pcluster* heuristics, it is considered to have *saved* 4 cuts (or incur -4 cuts).

The logic capacity and I/O constraints are satisfied by using the procedure **KPF** as outlined in Fig. 2. In essence, the partitions are built one at a time. A prototype is the first entry of a partition. One at a time, vertices are extracted from the heap and tentatively assigned to the current partition. If the I/O capacity is exceeded, then the vertex's ID is entered into a rollback list. The rollback list is reset if the number of I/Os drops within the bound. Upon each extraction of a vertex from the heap, the costs of vertices in the heap are incrementally updated according to the cost function, Eqn. (10). The extraction is repeated until the logic capacity constraint is exceeded. Rollback commences if there are entries in the rollback list; it rolls back to the last point where the I/O constraint is met. Then all vertices in the feasible partition are marked "allocated."

IV Experimental results

IV-A KPF K-way FPGA partitioner

We implement our k-way partitioning algorithm with both clustering heuristics *pcuts* and *pcluster*, and the best results of the two heuristics are presented. We refer to our k-way spectral-based partitioning algorithm as **KPF**.

We ran the graphs derived from the MCNC FPGA partitioning 1993 benchmarks for the Xilinx XC3000 series. The hypergraphs of the benchmarks were transformed into graphs by using Frankle's clique expansion net model [9], or the star graph net model. Unlike the clique net model, the star graph net model produces a very sparse graph matrix, which accelerates the eigensolver. On the other hand, this hypergraph model generates auxiliary vertices that have to be filtered out from the eigenvector embedding. We have observed that the star graph net model tends to produce more partitions than the clique net model. Unless otherwise stated, the partitioning results reported are produced by Frankle's clique net model.

Also, the hypergraphs of the benchmarks were pre-processed to remove high-fanout nets of degree greater than 99. Nets whose degree were greater than 99 were removed in order to reduce storage space and processing time. This step is essential in facilitating the eigensolver (Scott/Parlett implementation of the Lanczos algorithm) to run on a low-end SPARC station with only 32 Mbyte of memory. The high-fanout nets are only excluded during the calculation of the eigenvectors. Once the eigenvectors are computed, all nets are considered in determining the partitions.

A user supplies the desired FPGA package type, number of eigenvectors (h) desired, a balancing constraint, the circuit, and the net model desired to the partitioner. The balancing constraint determines the ratio of the partition sizes of the last two partitions. The partitioner transforms the circuit to a graph (adjacency) matrix according to the net model. The Lanczos algorithm is used to find the last h eigenvectors; the partitioner then generates the partitions. The number of partitions might be different from the number of eigenvectors specified.

The partitioned subcircuits are translated to the Xilinx `map` format. The routability predictor `kop` was applied to all the subcircuits. All the subcircuits are subsequently placed and routed using the vendor's tool `apr` (version XACT 5.0) with the default options.

Overall, the results in Table 1 demonstrate that

Circuit	Kunzar *et al*'s results		**KPF** results	CLB	IOB	elapsed
	# of chips	CLB util %	# of chips	util %	util %	time in sec.
c3540	{0,0,3,0,0}	66	{0,0,3,0,0}	66	99	38
c5315	{2,1,2,0,0}	73	{0,0,4,0,0}	65	94	64
c6288	{0,0,4,2,0}	81	{0,0,0,4,0}	93	77	84
c7552	{0,0,4,0,0}	85	{0,0,4,0,0}	85	99	83
s5378	{0,0,1,0,1}	82	{0,0,0,0,2}	60	72	38
s9234	{0,0,0,1,1}	77	{0,0,0,0,2}	71	69	69
s13207	{3,5,4,0,0}	72	{0,0,11,0,0}	58	98	202
s15850	{0,0,2,2,1}	83	{0,0,0,0,4}	66	91	171
s38584	{0,5,15,4,1}	75	{0,0,0,16,0}	81	95	1710

Circuit	Chou (SC) *et al*'s results	**KPF** results
s13207	{0,0,0,0,6}	{0,0,0,0,6}(routable)
s15850	{0,0,0,0,3}	{0,0,0,0,3} (unroutable); {0,0,0,0,4} (marginal)
s38417	{0,0,0,0,10}	{0,0,0,0,8} (unroutable); {0,0,0,0,9} (marginal)
s38584	{0,0,0,0,14}	{0,0,0,0,11}(marginal); {0,0,0,0,12}(marginal)

Table 1: Summary of **KPF** (k-way partitioning for homogeneous FPGA systems). Partitioning results using a SUN IPC workstation with 32 Mbyte of memory. The number of chips generated by the partitioners is presented as {# of XC3020, # of XC3030, # of XC3042, # of XC3064, # of XC3090} Xilinx XC3000 FPGAs. Kuznar *et al*'s partitioner [10] is for heterogeneous FPGA systems. Chou's partitioner (SC) [6] is for homogeneous FPGA hardware emulators.

our spectral-based graph partitioner **KPF** is capable of handling hard constraints and producing good partitions in reasonable time. We used the results from [10, 6] as references. The partitioner of Kuznar *et al* [10] is targeted for heterogeneous systems, while our partitioner **KPF** is targeted for homogeneous systems. We only list our homogeneous partitioning results in Table 1. However, our partitioner is also capable of generating very competitive heterogeneous results. For example, in [10] the authors use a monetary cost function, their partitioner generates a device distribution of $\{0, 0, 1, 0, 1\}$ meaning one XC3042 and one XC3090 FPGA to implement the benchmark circuit `s5378` (routability results are not reported in [10]). But without balancing constraints, our partitioner generates a better device distribution $\{1, 0, 0, 0, 1\}$ meaning one XC3020 and one XC3090 FPGAs to implement the circuit, and the average CLB utilization is 99.6%. Also, our result is less expensive than [10] with a monetary cost function. But the vendor's placement and routing tool `apr` [1] cannot complete the routing of this XC3090 subcircuit, the tool reports 50 unrouted nets! This demonstrates that partitioning without consideration for routability could be an exercise of futility. Even though several parallel processing researchers reported that in terms of cuts, spectral-based partitioning results can be further improved by applying mincut-based partitioning algorithm such as Kernighan and Lin; we do not resort to this post-processing step for routability considerations. The elapsed times (not CPU time) reported in Table 1 do not include the time taken by an `awk` script to translate the partitioning results generated by **KPF** into the Xilinx `map` format.

IV-B Routability prediction results

The routability of all the subcircuits are predicted before they are placed and routed, and the accuracy of the pre-partitioning routability predictor is reported in the last column of Table 2. As seen from the second column in Table 2, the pins-to-cell ratios of the MCNC benchmark circuits are relatively low. All subcircuits predicted to be "routable" were automatically routed in the first time by `apr`. On the other hand, subcircuits predicted to be "marginal" didn't complete the routing the first time by `apr`. Ten placement and routing runs were applied to those subcircuits and the average number of unrouted nets was recorded.

The majority of the subcircuits are predicted to be routable and this was later verified, as given in Table 2. Of particular interest is the circuit `s15850` with 3 partitions which was predicted to be *unroutable*, so we increased the number of partitions to 4, and the predictor predicts marginal routability. The prediction was verified to be correct, as depicted in the first five entries, last column of Table 3. This table also illustrates that spectral-based partitioners can generate a wide variety of cuts, and hence partitioning results. For instance, one of the 4 subcircuits generated by using the last 4 eigenvectors (of the adjacency matrix) has 7.7 unrouted (average) nets after 10 `apr` runs, and this subcircuit has higher cell utilization than the others. Instead, by using the 2nd to 4th eigenvectors our partitioner generates slightly more balanced and routable subcircuits, as indicated in the 4th entry of Table 3. Last, the star net model generates 5 very routable subcircuits, but the utilizations of the subcircuits are low.

We also include two hard-to-route circuits, `alu4` and `misex3` from the MCNC combinational circuit benchmark suite to test our partitioner and routability predictor. Initially, the circuit `alu4` uses two XC3042PG132's and circuit `misex3` uses two XC3020PC84's to implement the subcircuits, respectively. The routability predictor predicts the subcir-

Circuit	pins/clb predicted by eqn (2)	pins/net predicted by eqn (7)	pins/clb actual average	pins/net actual average	CLB % util	IOB % util	# of partitions	# of unrouted nets in subcircuits		pre-partition prediction
								Average	Minimum	
c3540	5.56	2.94	5.58	3.05	66	99	3	0,0,0	0,0,0	√
c5315	5.59	3.29	5.68	3.15	65	94	4	0,0,0,0	0,0,0,0	√
c6288	4.53	2.46	4.53	2.50	93	77	4	0,0,0,0	0,0,0,0	√
c7552	5.36	3.10	5.35	3.05	85	99	4	0,0,0,0	0,0,0,0	√
s5378	5.24	3.13	5.23	3.19	60	72	2	0,4.55	0,0	M
s9234	5.11	3.11	5.05	3.15	71	69	2	0,0.5	0,0	M
s13207	4.70	2.68	4.83	2.79	58	98	11	0,0,...,0	0,0,...,0	√
s15850	4.87	2.93	4.90	3.07	66	91	4	2.8,0,0,0	0,0,0,0	M
s38584	5.08	3.27	5.15	3.38	81	95	16	0,0,...,0	0,0,...,0	√
alu4	6.45	3.20	6.45	3.18	58	80	2	3,15	1,11	M
misex3	6.49	3.36	6.49	3.30	90	89	2	0,10.55	0,6	M

Table 2: Routability results of the subcircuits produced by the partitioner **KFP** using the clique hypergraph model. The predicted and measured (average) pins/net ratios of the benchmark circuits are reported; as well as the prediction of the pre-partitioning routability predictor (√=routable, M=Marginal, U=Unroutable). Subcircuits with zero average unrouted nets indicate successful *first-time* placement and routing completion. Nonzero average unrouted nets are the averages of 10 placement and routing runs.

net model	number of partitions	eigenvectors	# of CLBs in subcircuits	avg. unrouted nets in subcircuits
clique	–	last 2	–	–
clique	3	last 3	300,299,243	10.3 ,2.5,0
clique	4	last 4	303,214,150,175	7.7 ,0,0,0
clique	4	last 2nd to 4th	294,191,230,127	1.4 ,0,0,0
clique	4	last 3rd to 5th	249,165,197,231	0 ,0,0,0
star	5	last 3	203,193,127,116,203	0,0,0,0,0

Table 3: Partitioning and routability results of subcircuits of benchmark circuit `s15850` generated by using different net models and eigenvectors of the graph adjacency matrix.

Circuit	package	net model	subcircuit1		subcircuit2		avg. unrouted nets subcircuit1,subcircuit2
			IOBs	CLBs	IOBs	CLBs	
alu4	3042-PG132	clique	73	72	81	95	3.0, 15.0
alu4		star	90	77	96	90	0.6, 11.9
misex3	3020-PC84	clique	55	57	53	58	1.9, 6.5
misex3		star	64	56	64	59	0.5, 4.8

Table 4: Routability results of subcircuits of `alu4` and `misex3` after 10 placement and routing runs, showing the effect of increased cuts on routability.

cuits to be marginally routable, and this was verified, as shown in Table 2.

We repartition both circuits using the star net model, the results are shown in Table 4. The number of cuts (IOBs) produced by using the star net model is higher than the clique net model. The subcircuits are more "routable" with increased IOB usage, as indicated by the drop in the average number of unroutable nets in Table 4. We also applied the pre-partitioning routability predictor to check to see if the circuit `misex3` would be routed using a different package type, say XC3030PQ100, which has 100 CLBs and 80 IOBs. The predictor predicts positive results and this was verified to be the case.

V Conclusion

In this paper, we have presented partitioning results of a spectral-based partitioner targeted for homogeneous FPGA systems. The partitioner handles both partition-size and cut-size constraints. Routability of the partitioned subcircuits is the primary concern. To assist the partitioner in assessing the routability of the partitioned subcircuits prior to partitioning. We feel that routability prediction is valuable and should play a pivotal role in the partitioning for multiple-FPGA systems.

References

[1] XILINX*: The Programmable Gate Array Data Book.* 2100 Logic Drive, San Jose, CA 95124, 1993.

[2] C. J. Alpert and A. B. Kahng. Multi-way partitioning via spacefilling curves and dynamic programming. 31^{st} *DAC*, pg. 652–657, CA, June 1994.

[3] N. Bhat and D. Hill. Routable Technology Mapping for LUT FPGAs. *ICCD*, pg. 95–98, Oct. 1992.

[4] P. K. Chan, M. Schlag, and J. Zien. On routability prediction for field-programmable gate arrays. 30^{th} *DAC*, pg. 326–330, Texas, June 1993.

[5] P. K. Chan, M. Schlag, and J. Zien. Spectral k-way ratio-cut partitioning and clustering. *IEEE Trans. on CAD*, pg. 1088–1096, Sept. 1994.

[6] N.-C. Chou, L.-T. Liu, C.-K. Cheung, W.-J. Dai, and R. Lindelof. Circuit partitioning for huge logic emulation systems. 31^{st} *DAC*, pg. 244–249, CA, June 1994.

[7] W. Donath and A. Hoffman. Lower bounds for the partitioning of graphs. *IBM J. R & D*, pg. 420–425, 1973.

[8] A. El Gamal. Two-Dimensional Stochastic Model for Interconnections in Master Slice Integrated Circuits. *IEEE Trans. on CAS*, pg. 127–138, Feb. 1981.

[9] J. Frankle and R. M. Karp. Circuit placements and cost bounds by eigenvector decomposition. *IEEE ICCAD-86*, pg. 414–417, Santa Clara, CA, Nov 1986.

[10] R. Kuznar, F. Brglez, and K. Kozminski. Cost minimization of partition into multiple devices. 30^{th} *DAC*, pg. 315–320, Texas, June 1993.

[11] A. Pothen, H. D. Simon, and K.-P. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM Journal of Matrix Analysis and Applications*, 11(3):430–452, 1990.

[12] F. Rendl and H. Wolkowicz. A projection technique for partitioning the nodes of a graph. Tech. report, University of Waterloo, Apr. 1991.

[13] S. Trimberger and M.-R. Chene. Placement-Based Partitioning for Lookup-Table-Based FPGAs. *ICCD*, pg. 91–94, Oct 1992.

[14] N.-S. Woo and J. Kim. An efficient method of partitioning circuits for multiple-FPGA implementation. 30^{th} *DAC*, pg. 202–207, Texas, June 1993.