

Replication for Logic Partitioning

A Project Report

Submitted to the Graduate School

In Partial Fulfillment of the Requirements for the Degree

Master of Science

Field of Computer Science and Engineering

By

Morgan Enos

September 1996

Abstract

Replication offers an outstanding potential for cut size reduction. However, current work on replication does not provide any insight into the effectiveness of replication when used in conjunction with today's optimized partitioners. This work demonstrates that through a careful integration of replication into an optimized partitioner we can significantly improve the effectiveness of most replication algorithms. By developing novel techniques which capitalize on the structure of the partitioner, and by experimenting with the parameters associated with most replication algorithms, we can improve the effectiveness of most replication algorithms by approximately ten percent. This allows us to produce a cut set reduction of nearly forty percent under quite conservative assumptions and small circuit expansion.

1 Introduction

Bipartitioning is pervasive throughout VLSI CAD, being an essential step in any divide-and-conquer heuristic designed to simplify the task of working with large problem sizes by creating multiple smaller problem instances. In many cases we are willing to tolerate a small increase in problem size to reduce the communication between problem instances; a multi-FPGA system is one example of such a case. In multi-FPGA systems, large logic networks are partitioned into multiple sub-networks each of which can be mapped onto a single FPGA device. Because FPGAs are pin-limited devices, particular emphasis is placed upon minimizing the number of nets which span multiple partitions, for such nets consume pins in the corresponding FPGA devices, and may therefore expand the number of devices required to implement a design. Another aspect of pin-limitation is the presence of residual gate capacity; therefore, logic replication becomes an attractive addition to the partitioners repertoire, for the gate capacity exists to make it feasible, and the promise of a reduction in cut nets makes it valuable.

Much of the current work regarding replication has achieved improvements in cut size of approximately 30 percent; however, since these algorithms tend to use a non-optimized Fiduccia-Mattheyses algorithm (hereafter referred to as FM) for comparison or as a base for replication, and since many techniques exist which can significantly improve cut sizes for the basic FM, it remains unclear if such gains in the cut set are merely an artifact of relatively poor original cuts, or if replication is a genuinely valid tool for improving cut size beyond that possible through ordinary

move based tools. This work intends to address this issue, and provide a comparison of the various replication algorithms to determine their relative strengths, and to find ways in which they can be improved.

To promote the feasibility of replication in partitioning, and to facilitate the use of replication in recursive bipartitioning, this work limits itself to small circuit expansions (approximately seven percent). Several novel techniques are developed which enhance the partitioners ability to generate small cut sizes under such austere circumstances. This work illustrates that with careful integration of current replication techniques, and through various enhancements designed to capitalize on a partitioner's structure, we can achieve excellent cut sizes with relatively little circuit expansion.

1.1 Optimized FM

Each of the replication algorithms we will examine are either adaptations of FM or justify their utility by comparison against FM. The basic FM algorithm is as follows [4]:

```

Generate an initial partition;
while the number of cut nets is decreased
begin-1
  while there exists an unlocked node whose movement does not result in
    a partition size violation
  begin-2
    Select an unlocked node whose movement results in the most
      favorable change in the number of cut nets without violating
      partition size limitations;
    Move node to opposite partition;
    Lock node;
  end-2
  Re-establish the partition generated by above loop which had the
    smallest number of cut nets;
  Unlock all nodes;
end-1

```

Figure 1. The Fiduccia - Mattheyses Algorithm.

FM is a widely used algorithm, and as such, a number of ideas have emerged in the literature which improve upon the basic design. By a judicious combination of these ideas, significant improvements in performance can be realized. This is the premise behind Strawman, for this partitioner actually consists of many techniques which achieve a synergistic reduction in cut size [6]. As the following table illustrates, Strawman produces results which are better than the current state-of-the-art.

Mapping	Basic FM	Strawman	EIG1	Paraboli	FBB
s9234	70	42	166	74	70
s13207	108	57	110	91	74
s15850	170	44	125	91	67
s38584	168	49	76	55	47
s35932	162	47	105	62	49
Geom. Mean	128.5	47.5	112.7	73.1	60.3
Normalized	2.705	1.000	2.373	1.539	1.269

Table 1. Comparison of partitioners. Partitions are between 45% and 55% of total circuit size. FM and Strawman results are from [6], EIG1 results from [1, 7], and Paraboli results from [12].

The various techniques used by Strawman to achieve its superior results are outlined below. A more detailed explanation of these techniques can be found in [6]:

- **Random initial partitioning**

The initial partition is generated by randomly placing nodes into partitions.

- **Size Model**

The Strawman partitioner uses a size model specialized for FPGA partitioning. In this model each node has a size of *number of inputs - 1*. This tends to be a realistic size model for FPGA partitioning since the technology mapper will often assign gates to a CLB (configurable logic block) in such a way that the output of one gate is the input of the other, and hence the function implemented by the two gates requires one less input than the total number of inputs in both gates. So if one gate required M inputs, and the other P inputs, the combination could be implemented by a lookup-table (the implementation of a CLB in Xilinx 3000 series FPGA devices [16]) of $M + P - 1$ inputs.

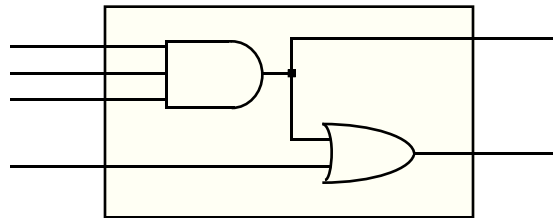


Figure 2. A three-input and two-input gate are mapped to a four-input CLB

Therefore *number of inputs - 1* is a reasonable approximation to the input capacity of a CLB which a node will consume. This size model has the further advantage of assigning inverters and I/O nodes a size of zero. The former is appropriate since a one input function does not require an extra input on a LUT, and the latter is appropriate since there is usually a separate space on the device for handling I/O nodes, and therefore there is no particular interaction between logic capacity and I/O node distribution beyond the min-cut objective to which we aspire. Generally a uniform size model (each node has size one) will provide better cut values; however, our size model will provide a partitioning of logic which results in a better balanced CLB distribution after technology mapping. For purposes of comparison, all implemented algorithms which operate on non-technology mapped circuits will use our size model.

- **Presweeping**

Presweeping is a clustering step designed to be used previous to all other techniques. In our size model certain nodes may have size zero (such as inverters, and I/O nodes), and may therefore under certain circumstances be clustered with a neighboring node without ever changing the optimal cut size. Two such circumstances are as follows: a size zero node with a single net may be clustered with any other connected node; a size zero node with exactly two nets, one of which is a two terminal net, may be clustered with the other node on the two terminal net. Since such clusterings can never increase the optimal cut size, they may be performed previous to all other techniques, and need never be undone.

- **Krishnamurthy Gains**

During the course of the FM algorithm there may be many nodes with the same movement gain (where gain is defined as the difference in the number of cut nets before and after movement). Krishnamurthy tried to encode some foresight into the selection among these nodes by adding “higher-level” gains as a tie-breaking mechanism [6][10]. A net contributes an n th level gain of $+1$ to a node S on the net if there are $n-1$ other nodes on the net in the same partition as S which are unlocked, and zero which are locked. A net contributes a -1 n th level gain to a node S on the net if there are $n-1$ unlocked nodes on the net in the partition opposite to S , and zero locked. The first level gain is identical to the standard gains of the FM algorithm. If a net contributes positively toward the n th level gain of a node, it implies that the net can be uncut by the movement of the node, and $n-1$ further node movements. If a net contributes negatively toward the n th level gain of a node it implies that by moving this node, we will lose an

opportunity to have the net uncut in $n-1$ further node movements. We use three gain levels ($n = 3$) since additional levels of gain appeared to slow down the partitioning without improving the resulting quality.

- **Recursive Connectivity Clustering**

Nodes are examined randomly and clustered with the neighbor to which they have the greatest connectivity as defined by the following equation [6][15]:

$$connectivity_{ij} = \frac{bandwidth_{ij}}{size_i * size_j * (fanout_i - bandwidth_{ij}) * (fanout_j - bandwidth_{ij})}$$

Equation 1. Connectivity metric for connectivity clustering.

Here $bandwidth_{ij}$ is the total bandwidth between nodes i and j , where each n -terminal net connecting nodes i and j adds $1/(n-1)$ to the total. This equation is aimed at encouraging the clustering of nodes which are highly interconnected and of small size. Without the size constraints on the nodes, larger nodes would tend to hoard the bandwidth, and hence be disproportionately chosen by neighboring nodes as a partner in clustering. Eventually the network would come to be dominated by a few large nodes, and any structural information designed to be uncovered by the clustering would be lost. This method is applied recursively until, due to partition size limitations, no further clusters can be formed.

- **Iterative Unclustering**

If we retain the clustering information at each level of the hierarchy, where the n th level of the hierarchy consists of all clusters created as a result of the n th application of connectivity clustering, we may perform iterative unclustering. We begin by performing FM (with higher level gains) upon the clusters at the top of the hierarchy (those clusters which exist after the final round of connectivity clustering). We then uncluster the top level of the hierarchy, revealing those clusters which existed previous to the final application of connectivity clustering. We repeat this process of FM and unclustering until we have unclustered the entire hierarchy. A final round of FM is then performed on the nodes and sweep clusters of the original network. The technique of iterative unclustering is based upon the work in [2].

Throughout this paper we will be using the cut sizes produced by Strawman to judge the effectiveness of the various replication algorithms we examine, and in many cases, we will be trying to extend the techniques of Strawman to those replication algorithms which are based upon the unimproved FM.

1.2 Problem Definition

Let $H = (V, E)$ be a hypergraph where V is the set of nodes of the hypergraph, and E is the set of hyperedges. For each hyperedge $e \in E$, $e = (s, U)$ where $s \in V$, and $U \subseteq V$, and s is designated the source of the hyperedge, and each $u \in U$ a sink of the hyperedge. Then given H and positive integers pX and pY , the replication problem is then to find X, Y , such that $X, Y \subseteq V$, $X \cap Y = R$, $X \cup Y = V$, which minimizes the size of the cut set C and does not violate the size constraints: $|X| \leq pX$ and $|Y| \leq pY$. A hyperedge $e \in E$, $e = (s, U)$ is an element of the cut set C if $s \in X - R$ and $U \cap Y \neq \emptyset$ or $s \in Y - R$ and $U \cap X \neq \emptyset$. In this formulation R is denoted as the Replication set.

1.3 Previous Work

Newton and Kring introduced a minimum cut replication algorithm in 1991 [9]. This was a fairly straight-forward adaptation of the basic FM algorithm to support replication in hypergraphs. Shortly thereafter Hwang and El Gamal solved the minimum cut replication problem optimally when provided with an initial partition, and unlimited partition size [8]. However, this formulation only applied to graphs (not hypergraphs), and a relatively ineffective heuristic was substituted when working with hyper-graphs or partition size limitations. Kuznar, Brglez, and Zajc developed a replication technique specifically for technology mapped circuits in 1994 [11]. This technique utilizes functional information inherent in the design to determine the replication set. In 1995, Yang/Wong improved upon

the Hwang/El Gamal algorithm by reducing replication set size, and adapting their flow-based partitioning method for use in place of the Hwang/El Gamal heuristic [18]. Also in 1995, Liu et. al. introduced their own flow-based formulation for use with graphs (not hypergraphs) without partition size limitations, and their own move based heuristic for use with hyper-graphs or problem instances with partition size limitations [13]. Also by Liu et. al. was an algorithm that uses replication to derive a good initial partition for use by an FM based partitioner [14].

1.4 Replication Algorithms

This paper focuses on min-cut replication. Furthermore, we are primarily interested in those algorithms which produce good cuts quickly (approximately the same average running time as FM). We chose not to implement the Gradient decent based FM of Liu, et. al [14]. This algorithm is somewhat slow, and it would be unlikely to perform well on the small circuit expansions with which we are working. We believe the algorithms we have selected are exhaustive of current work on replication given our two criteria: min-cut objective, and speed.

Replication for cut minimization attempts to seek out cells that when replicated across the partition reduce the number of nets in the cut set. Replication reduces the number of cut nets in a partition by creating redundancy in signal pathways.

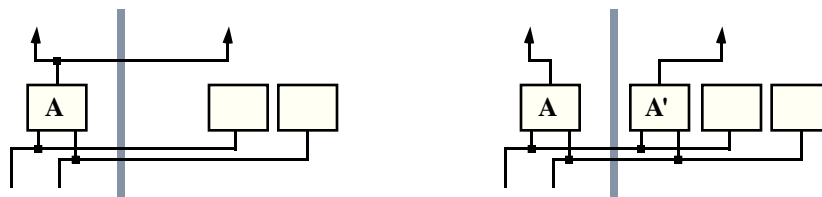


Figure 3. An example of node replication. Node **A** in the circuit at left is replicated into **A** and **A'**, yielding a gain of 1.

As the above figure illustrates, the output net of a replicated cell need no longer span partitions since each partition has its own copy of the cell to generate the signal. However, to ensure that the correct signal is generated, every input net of the original cell must also input the replicated cell. Hence, the gain of cell replication is:

$$| \text{cut output nets of cell} | - | \text{uncut, and } \textit{unreplicated} \text{ input nets of cell} |$$

Equation 2. Gain of cell replication.

By *unreplicated* input net we mean to refer to input nets of the cell that have a replicated source. Since such nets can never be cut they are excluded from the gain calculation. Note specifically that unlike standard partitioning, the direction of a net now matters, for in cell replication the circuit is modified, and the gain is determined based upon whether the net uses the cell as a sink, or as a source. So in partitioning with replication, the partitioning is always performed upon a directed graph.

One possible point of contention between our work and the work of others is in the replication of I/O nodes, or Input Buffers (IBUFs) to be more specific. We regard every IBUF to be the sink of some external source-less net, and when the IBUF is replicated across the partition, this net becomes cut. We prefer this convention since the replication of such IBUFs is not free, for the IBUF consumes a pin in the partition to which it was replicated; however, we recognize the fact that an external net consumes fewer additional pins (one) than an internal cut net (two), and therefore our convention is somewhat conservative.

To determine the effectiveness of the various replication algorithms, we run each algorithm thirty times on a SPARC5 workstation with each of the following benchmarks, and limit ourselves to a maximum of seven percent circuit expansion.

benchmark	# cells	# nets	# iopads	# pins
s5378	3225	3176	88	8241
s9234	6098	6076	45	15026
s13207	9445	9324	156	23442
s15850	11071	10984	105	27209
C6388	1956	1924	64	7046
biomed	6494	6442	77	24537
primary2	3035	3028	128	11294
struct	1952	1920	64	5535
s35932	19880	19560	359	55420
s38584	22451	22173	294	61309
s38417	25589	25483	138	64299

Table 2. Characteristics of benchmark circuits.

Seven percent expansion was chosen because the Strawman partitioner requires that each partition have a size ceiling of at least 51% of the circuit size to allow for the movement of large clusters, and we wanted to allow each partition to grow an addition 5% after partitioning to provide some room for the replication algorithms to operate. This implies that each partition can expand up to 53.6% ($0.51 * 1.05 \cong 0.536$) of the original circuit size, yielding a circuit expansion of approximately seven percent. Such small expansion is preferred since these techniques may be applied recursively to divide a large circuit among many devices, and therefore the overall circuit expansion can become quite large rather quickly. When using Strawman to produce partitions for comparison against the various replication algorithms, we allow each partition to grow to 53.6% of the original circuit size.

2 Newton/Kring Replication

The algorithm proposed by Newton and Kring (hereafter referred to as N/K) is a fairly straight-forward adaptation of FM to support cell replication [9]. In the FM algorithm, a node can exist in one of two states dependent upon which partition it is currently located. Newton and Kring introduce a third state – replication, in which a node exists in both partitions simultaneously. Since there are now three states, three gain-arrays are required: one for each partition, and one for nodes in both partitions (replicated). Also, since each state must have a transition to each of the others, there must be two moves available to each cell from its current state. Therefore gain arrays must support both transitions, and hence have two entries for each cell.

The Newton/Kring algorithm also has two parameters: a replication threshold, and an unreplication threshold. The replication threshold is the gain value under which no replication moves are allowed. The unreplication threshold is the value above which a unreplication move is forced to occur regardless of the presence of higher gain moves. Newton and Kring found that an unreplication threshold of zero always produced the best results. This implies that we always insist upon unreplication when it is associated with a positive gain. It was also found that a replication threshold of one was superior, and therefore it was the best strategy to allow replication only if it reduced the cut value. Our own experiments confirm the value of using an unreplication threshold of zero; however, various values of the replication threshold were found useful in different situations as detailed below.

2.1 Basic Newton/Kring

After the circuit has been fully unclustered (iterative unclustering), a final iteration of FM (with higher-level gains) is performed upon the gate-level nodes and sweep clusters. Instead of performing FM, we perform the N/K variant (without higher-level gains) to reduce the cut size via replication. This is essentially the N/K algorithm performed on an initial partition which has been optimized – a modification proposed by Newton and Kring which was found to be useful in partitioning larger benchmarks.

Benchmark	Best Strawman	Cumulative Execution Time (min.)	Best N/K	Cumulative Execution Time (min.)	Replication (percent) of Best
s5378	60	5.0	41	5.2	6.6
s9234	45	8.8	30	8.9	7.1
s13207	67	17.3	53	17.9	7.0
s15850	52	19.6	43	20.2	7.1
C6288	50	3.4	33	3.3	2.6
biomed	161	38.3	136	36.7	5.9
primary2	133	8.2	114	7.7	7.1
struct	33	2.5	33	2.5	4.7
s35932	46	124.2	39	122.2	2.6
s38584	50	94.0	35	92.5	2.6
s38417	55	93.5	51	94.8	4.2
Mean	60.8	17.4	48.2	17.3	4.2

Table 3. Comparison of Strawman against Basic N/L for thirty iterations of each algorithm. Geometric mean used on cut values and execution time, arithmetic mean used with replication percent.

Unlike the Newton/Kring experiments, we found that a replication threshold of zero produced superior results. This, in large part, is due to the fact that we are using gate level circuits, and therefore each node is single output. If we were to insist on a replication threshold of one, every net on the node would have to be cut for a node to be considered for replication. Since we are already using a fairly optimized initial partition, this is a rather unlikely event, and overly restrictive.

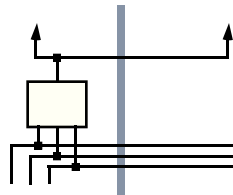


Figure 4. A node which has a replication gain of one.

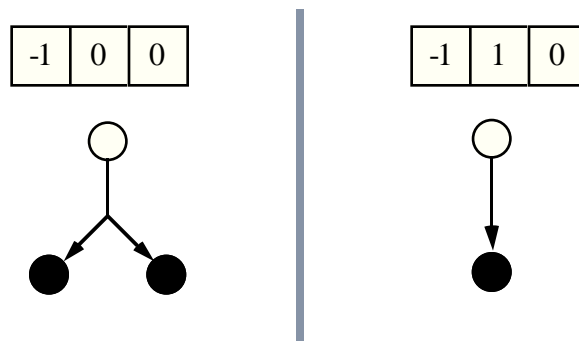


Figure 5. Higher-level gains of replicated nodes. White nodes are replicated versions of same original node.

2.2 Higher-Level Gains

There are perhaps several ways to extend higher-level gains to include replication moves; however, one of the simplest, and most efficient is simply to set the higher-level gains of replication moves to $-max_fanin$, where

max_fanin is the largest degree of any node in the network. This tends to force ordinary moves to occur before replication moves of equal first level gain. This was deemed desirable since we always wish to remove nets from the cut set without circuit expansion when possible. For ordinary moves we can retain the original definition of higher-level gains. For unreplication moves, we can retain the original definition for all levels greater than one. This is true because once the unreplication move has been performed (the first level gain), both the node and the net revert to the original state for which the higher-level gains were formulated.

In the example in Figure 5, we would prefer to unreplicate the node by moving the right copy to the left partition, for then we can more easily uncut the net by subsequent moves. This is exactly the information that the higher-level gains communicate to us. The following table compares the basic N/K implementation with a maximum gain level of one (the usual FM gain definition) against an N/K implementation with a maximum gain level of three.

Benchmark	Best Strawman	Best N/K	Best N/K w/ higher-level gains	Cumulative Execution Time (min.)	Replication (percent) of Best
s5378	60	41	39	5.3	5.4
s9234	45	30	32	9.2	7.0
s13207	67	53	48	18.3	7.1
s15850	52	43	43	20.7	4.4
C6288	50	33	33	3.3	2.6
biomed	161	136	137	37.2	7.1
primary2	133	114	114	8.1	7.0
struct	33	33	33	2.6	4.7
s35932	46	39	39	121.3	1.1
s38584	50	35	35	91.5	2.4
s38417	55	48	49	94.8	4.8
Mean	60.8	48.0	47.7	17.6	4.5

Table 4. Comparison of Basic N/K vs. N/K with higher-level gains.

Because of the superior performance of N/K with Higher-Level Gains (albeit marginally), all future N/K variants in this paper will be performed with higher-level gains.

Benchmark	Best Strawman	Best N/K	Best N/K with Clustering	Cumulative Execution Time (min.)	Replication (percent) of Best
s5378	60	39	48	5.3	7.0
s9234	45	32	33	9.4	6.9
s13207	67	48	49	18.8	7.1
s15850	52	43	39	22.2	7.0
C6288	50	33	34	3.1	2.0
biomed	161	137	126	34.8	7.1
primary2	133	114	110	6.9	7.2
struct	33	33	33	2.2	0.0
s35932	46	39	28	124.7	7.1
s38584	50	35	46	90.5	6.6
s38417	55	49	47	95.7	2.7
Mean	60.8	47.7	47.6	17.1	6.2

Table 5. Comparison of N/K with higher-level gains against N/K with recursive clustering.

2.3 Extension to Clustering

It was hoped that by earlier application of N/K, better results might be achieved via increased opportunity for replication. So rather than perform N/K style replication only after complete unclustering, N/K replication was performed at each clustering level in place of the Strawman FM algorithm. To facilitate replication at each level of the clustering hierarchy, the total allowed circuit expansion was evenly distributed among all levels, and so if there were seven clustering levels, and we allowed up to seven percent circuit expansion via replication, then N/K would be allowed to expand the circuit by one percent at each clustering level. Such expansion is cumulative, so at each clustering level, the total amount of circuit expansion allowed is greater than the previous (one percent greater in this example). Without this incremental circuit expansion, replication in the first couple of clustering levels would inevitably expand the circuit maximally, and therefore severely limit replication in succeeding levels. This promotes early, and relatively ignorant replication over later replication, and tends to produce poorer results.

2.4 Variable Thresholds

The essential problem of using replication earlier in the partitioning is that much of the replication is ignorant, and must be undone in later iterations. This is particularly problematic when performed in conjunction with clustering since the replication of a single cluster may give rise to the replication of a great many gate-level nodes, and it may never be possible to undo all of the damage. Variable replication thresholds were introduced as a means of discouraging early (and therefore ignorant) replication. The replication threshold (as defined by Newton/Kring) is a gain threshold below which no replication should be allowed. Therefore if a replication potential were set at two, a replication move could only be performed if a gain of two were realized – the total number of cut nets were reduced by two as a result of the replication. As mentioned previously, when using a gate-level representation of a circuit, any replication threshold above zero is overly restrictive since all nodes are single output. However, clusters may be multi-output, so a threshold above zero now becomes reasonable, and since clusters may also be quite large, we want to be assured that their replication is significantly beneficial. We use a replication threshold which is: $\text{floor}(\text{cluster_level} / 2)$. This discourages replication at the higher clustering levels when nodes are quite large, and the partition relatively unoptimized – conditions in which replication can be most ignorant and most damaging. We also use incremental circuit expansion as described in the previous section.

Benchmark	Best Strawman	Best N/K with Clustering	Best N/K with Var. Thres.	Cumulative Execution Time (min.)	Replication (percent) of Best
s5378	60	48	44	5.3	7.1
s9234	45	33	32	9.6	7.1
s13207	67	49	50	19.3	5.8
s15850	52	39	40	22.5	4.2
C6288	50	34	33	3.7	0.1
biomed	161	126	138	35.7	7.1
primary2	133	110	114	7.5	7.1
struct	33	33	33	2.8	0.0
s35932	46	28	23	130.1	7.1
s38584	50	46	46	92.1	4.1
s38417	55	47	48	96.8	1.6
Mean	60.8	47.6	47.0	18.1	5.1

Table 6. Comparison of N/K with clustering against N/K with variable thresholds.

As Results indicate, variable thresholds improve best cut values, and reduce the amount of replication performed.

2.5 Gradient Method

As the FM algorithm progresses, the partitions become steadily better. Early in the algorithm there may be many high gain moves since the partition is relatively poor, but as the algorithm proceeds high gain moves become rare. Therefore high gain moves can be as much an indication of a relatively poor partition as an intelligent move, and this renders threshold inhibition of replication a somewhat vulnerable means of control. By exerting a more temporal control over replication we can be more certain that we are pushing the partition in the right direction. If we activate replication only in the later stages of FM, we can afford to use a lower replication threshold value since the partition is already fairly good, and we've nearly exhausted any improvement we are going to achieve via standard FM movements. Replication becomes a valuable tool for tunneling out of any local minimum the FM algorithm is settling into.

We choose to "turn on" replication when the cut sizes between successive inner-loop iterations of the Strawman FM algorithm change by less than ten percent (hence the term gradient). When replication is "turned on", we use a replication threshold value of zero. This procedure is applied at each clustering level in place of the Strawman FM algorithm. The circuit is expanded incrementally as in the previous two clustering algorithms.

Benchmark	Best Strawman	Best N/K with Var. Thres.	Best N/K Gradient	Cumulative Execution Time (min.)	Replication (percent) of Best
s5378	60	44	43	5.9	4.0
s9234	45	32	32	10.7	4.3
s13207	67	50	43	20.6	7.1
s15850	52	40	41	24.5	5.8
C6288	50	33	33	3.9	0.0
biomed	161	138	130	38.6	4.7
primary2	133	114	114	9.2	7.0
struct	33	33	33	2.8	0.0
s35932	46	23	23	132.0	6.4
s38584	50	46	32	100.3	3.0
s38417	55	48	49	104.0	0.8
Mean	60.8	47.0	44.7	19.7	3.5

Table 7. Comparison of N/K with variable threshold against N/K with gradient.

The N/K algorithm, by incorporating replication moves into FM, significantly increases the size of the solution space. This tends to overwhelm the N/K algorithm and often results in cuts which are significantly poorer than cuts produced by the Strawman FM algorithm without replication. By introducing replication in the late stages of Strawman FM we allow the N/K algorithm to proceed from a local minimum produced by Strawman FM, and therefore greatly reduce the size of the space which the N/K algorithm must search. This, in general, leads to better solutions than the N/K algorithm could produce without such assistance.

3 DFRG

Another move based approach modeled upon FM is the Directed Fiduccia-Mattheyses on a Replication Graph or DFRG [13]. The replication graph is a modification of the original circuit as inspired by a dual network flow formulation from a linear program designed to solve the replication problem optimally (without size constraint). However, since we do have size constraints, and since our circuits are typically modeled by hyper-edges rather than edges we perform a "directed FM" upon the network instead of pushing flow through it.

To create the replication graph we duplicate the network, and reverse the direction of each of the duplicated nets (connecting the duplicated nodes). So each node that is a source of an original net has a duplicate node which is a

sink in the duplicate net, and of course, each sink of an original net has a duplicate which is a source of the duplicate net. Also, original nodes are connected to duplicate nodes via a two terminal net of infinite weight (can never be cut) in which the original node is the source of the net.

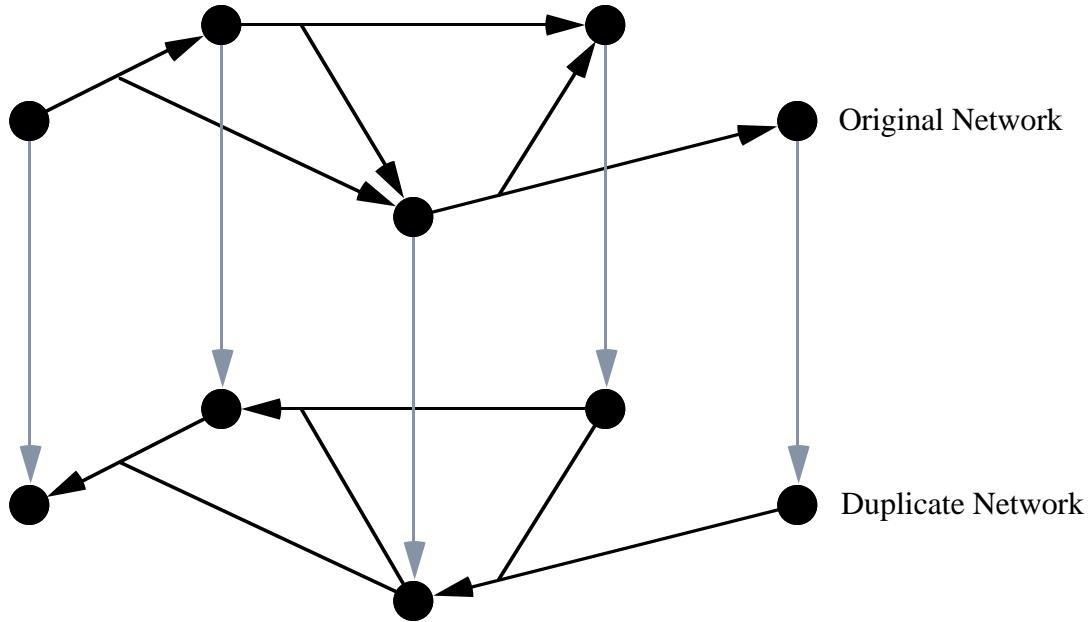


Figure 6. Replication graph. Gray arrows are infinite weight edges.

Directed FM is then performed upon this graph. Directed FM is essentially the same as standard FM, but with a different gain function. In directed FM an initial partition is generated, and then one side of the partition is chosen arbitrarily to be the “source” partition. Nets are only considered cut if a source of the net exists in the “source” partition, and a sink of the net exists in the “non-source” partition. Since each net now has a duplicate, it is possible for a net to contribute twice to the cut set if both the original and the duplicate net are cut. However, if every net in the original circuit has at most one source, then both nets cannot be cut simultaneously, for it would require that the source of the original net be in the “source” partition, and the duplicate of this source node (the sole sink of the duplicate net) be in the “non-source” partition. This would imply that the infinite weight net connecting these two nodes is cut.

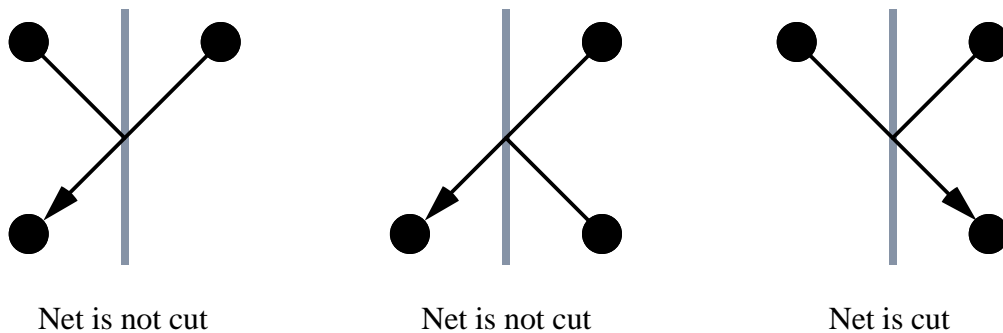


Figure 7. Example of cut definition in directed partitioning. The source partition is on the left in each figure.

After completion of the directed FM, all nodes of the original circuit which are in the “non-source” partition and have duplicates in the “source” partition are replicated. Assume for a moment that no replication occurs, this condition can be enforced by adding an additional net of infinite weight between duplicate nodes and original nodes as shown below, and thus forcing original nodes and their duplicates to coexist in the same partition.

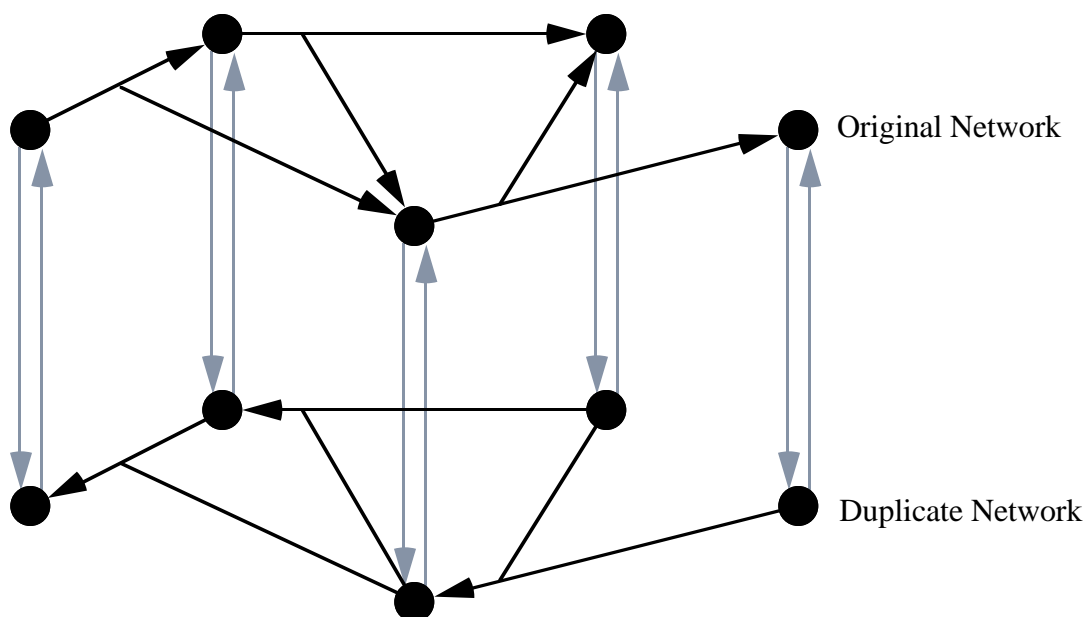


Figure 8. Modified replication graph. Gray edges have infinite weights.

For DFRG to produce a valid cut size (without replication), all original nets which have nodes spanning the two partitions must be considered cut. Therefore, if the original net N has nodes in both partitions, then DFRG must either consider this net to be cut, or the duplicate net N' to be cut. If the net N has its source in the “source” partition, it must have a sink in the “non-source” partition, and therefore directed FM would consider this net to be cut. If the net N has its source in the “non-source” partition, it must have a sink in the “source” partition. The duplicate node of this sink is a source of the duplicate net N' , and it also is in the “source” partition. The duplicate node of the source of N is a sink of the duplicate net N' , and this node is in the “non-source” partition, and therefore the duplicate net N' is cut. Hence, when there is no replication, all original nets which have nodes in both partitions are considered cut by DFRG. Since (as shown earlier) a net and its duplicate cannot both be cut, the cut size of the replication graph is identical to the original graph.

Let us now remove the infinite weight edge that we just added. This allows original nodes in the “non-source” partition to have duplicate nodes in the “source” partition (we still cannot have original nodes in the “source” partition, and duplicates in the “non-source” partition because this would imply that an infinite weight net is cut). Original nodes in this state are considered replicated. For this to be a valid replication, all of the nets which input the replicated node must be cut (or have a replicated source), and all of the nets which use the replicated node as a source must not be cut. Let R be an arbitrary replicated node, R' be its duplicate in the duplicate circuit, N an arbitrary input net to R in the original circuit, and N' the duplicate of N in the duplicate circuit. Exactly one of three conditions must exist:

- (1) N is cut.
- (2) N' is cut.
- (3) The source of N is replicated, and neither N nor N' is cut.

First assume (1) is false. This implies that the source of N (all nets of the original circuit have at most a single source) is in the “non-source” partition. Since R is replicated, R' is in the “source” partition, and since R is a sink of N , R' is a source of N' . Now, the sink of N' (all nets in the duplicate circuit have a single sink) is the duplicate node of the source of N , and the source of N is in the “non-source” partition. If the sink of N' is in the “non-source” partition, the source of N is not replicated and N is cut. Therefore condition (2) prevails. If the sink of N' is in the “source” partition, the source of N is replicated, and N is uncut, therefore condition (3) prevails.

Now assume (1) is true. Then (2) is false as discussed previously (only one of N, N' may be cut), and the source of N must be in the “source” partition, and hence cannot be replicated, and therefore (3) is also false. Therefore our conditions on the input nets of a replicated node are satisfied. Now we examine the output net of a replicated node. An output net of a replicated node can never be cut, so if N is an output net of the replicated node R , both N and its duplicate net N' must be uncut. This condition is satisfied since R , the source of N , is in the “non-source” partition, and therefore N cannot be cut. N' is also uncut because its single sink, R' , is in the “source” partition. Thus for any node designated for replication, all nets adjacent to the node are in the appropriate state, and therefore the cut size as determined by the DFRG algorithm reflects the cut size of the circuit after replication.

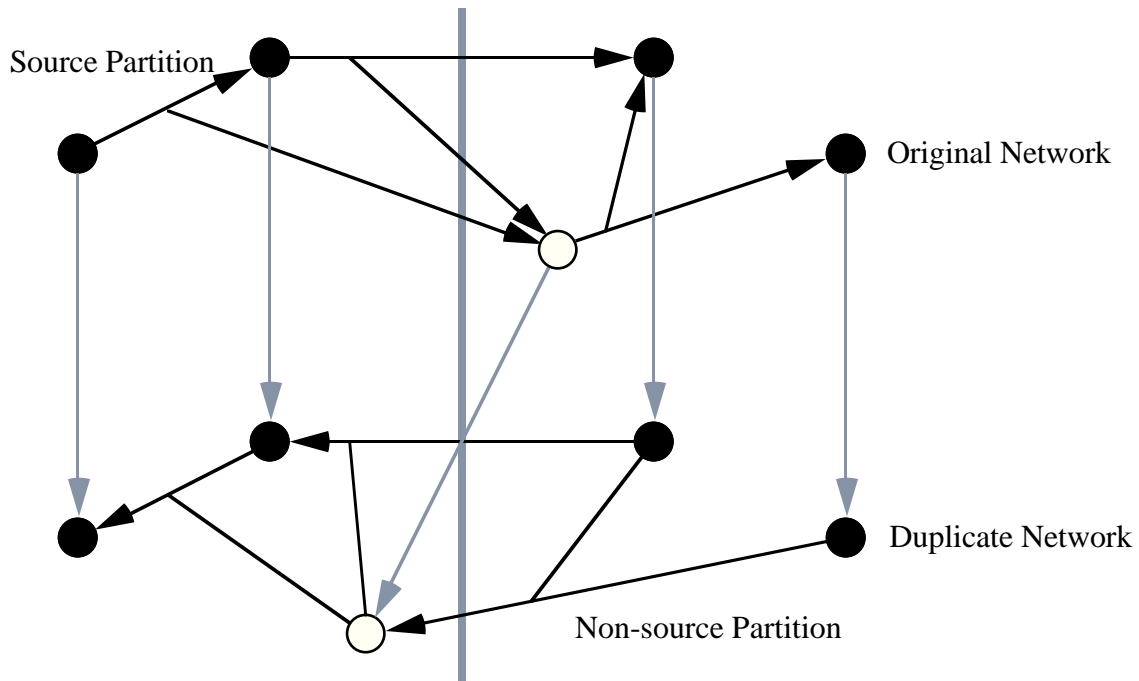


Figure 9. White node is designated for replication.

The size constraint on the “non-source” partition is enforced by limiting the total size of original circuit nodes in the “non-source” partition. This is valid since this size includes the size of each replicated node plus the size of all non-replicated nodes in the “non-source” partition. The size constraint on the “source” partition is enforced by limiting the total size of duplicated circuit nodes in the “source” partition. This set of duplicated nodes contains a copy of each replicated node plus the size of all original nodes in the “source” partition. The latter part of this statement is true because if an original node exists in the “source” partition, the duplicate node must also be in the “source” partition, otherwise the infinite weight net connecting the two nodes would be cut.

The replication of input buffers represents a further complication to this algorithm. We wish to penalize such replication since it necessarily introduces an extra pin into a partition, however simply introducing an external net whose sink is the input buffer will not suffice (as it did with N/K). Such sourceless nets can never be cut in directed

FM since they have no source in the original network, and no sink in the duplicated one. To work around this, the replication graph was modified by adding a net of unit weight between an IBUF and its duplicate, and designating the duplicate IBUF as the source of the net. Therefore if the IBUF were replicated, the duplicate IBUF would be in the “source” partition and the original IBUF would be in the “non-source” partition, and this net would contribute to the size of the cut set.

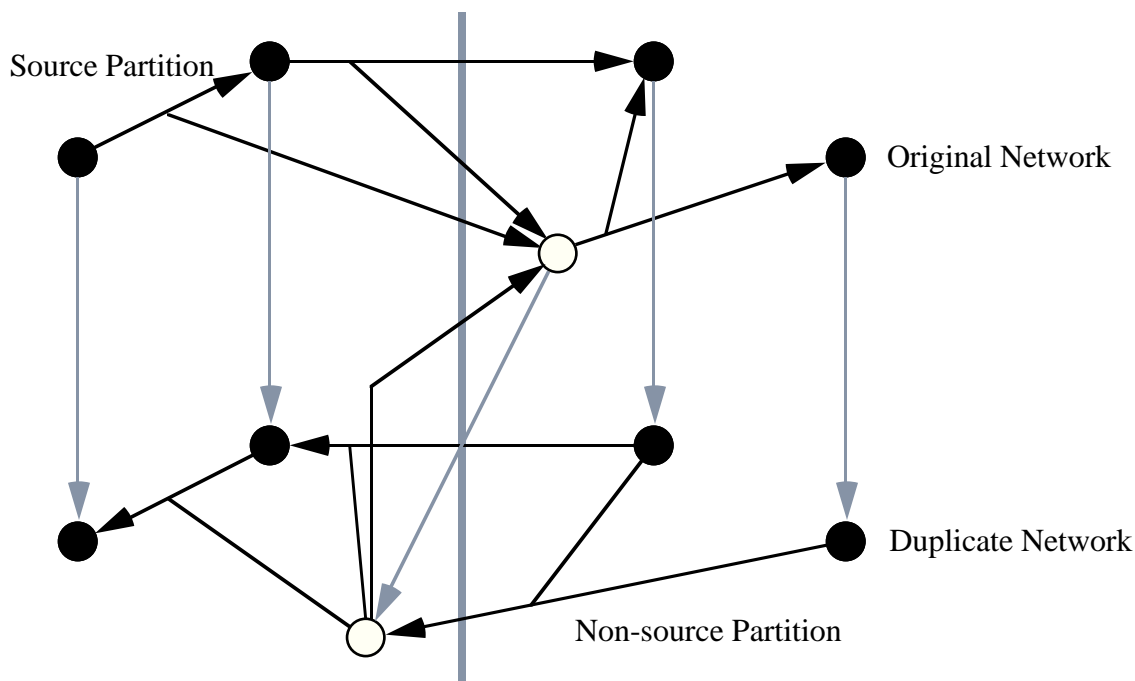


Figure 10. The white node is an IBUF which is currently designated for replication.

3.1 Basic DFRG

We obtain an initial partition via Strawman and use DFRG as a post-processing step upon the gate-level nodes and sweep clusters. This is essentially DFRG as performed upon a good initial partition.

3.2 Higher-Level Gains

A net can be uncut either by moving all sources to the “non-source” partition, or by moving all sinks to the “source” partition. By moving a source, one does not lose any opportunities for uncutting a net via sink movements, and by moving a sink, one does not lose any opportunities for removing the net from the cut set via source movements. Since source and sink motion operate somewhat independently in determining the state of a net, we use the following formulation: a cut net contributes +1 to the n th level gain of a source node S if S is in the “source” partition, and there are $n-1$ other unlocked sources of the net in the “source” partition, and zero locked sources in the “source” partition; a cut net contributes a -1 to the n th level gain of a source node S if S is in the “non-source” partition, and there are $n-1$ unlocked sources in the source partition, and zero locked sources in the “source” partition; a cut net contributes +1 to the n th level gain of a sink node T if T is in the “non-source” partition and there are $n-1$ other unlocked sinks of the net in the “non-source” partition, and zero locked sinks in the “non-source” partition; a cut net contributes a -1 to the n th level gain of a sink node T if T is in the “source” partition and there are $n-1$ unlocked sinks in the “non-source” partition.

For ease of implementation, only cut nets contribute to the higher level gains of a node. Essentially a cut net contributes positively to the n th level gain of a node if the node is a source and the net can be uncut in $n-1$ further

source movements, or the node is a sink and can be uncut in $n-1$ further sink movements. And a net contributes negatively to the n th level gain of a node if the node is a source and we lose an opportunity to have the net uncut in $n-1$ further source movements, or the node is a sink and we lose an opportunity to have the net uncut in $n-1$ further sink movements.

Benchmark	Best Strawman	Best DFRG	Cumulative Execution Time (min.)	Replication (percent) of Best
s5378	60	40	9.3	7.0
s9234	45	34	15.5	3.9
s13207	67	54	29.4	5.7
s15850	52	45	32.0	4.1
C6288	50	33	5.7	2.1
biomed	161	136	56.7	5.2
primary2	133	114	14.1	6.6
struct	33	33	4.0	4.8
s35932	46	39	172.9	1.0
s38584	50	37	132.4	0.5
s38417	55	48	136.5	0.4
Mean	60.8	48.9	27.8	3.5

Table 8. Comparison of Basic DFRG against Strawman.

Benchmark	Best Strawman	Best Basic DFRG	Best DFRG w/ Higher-Level Gains	Cumulative Execution Time (min.)	Replication (percent) of Best
s5378	60	40	41	10.2	6.5
s9234	45	34	34	17.3	6.0
s13207	67	54	47	31.7	5.3
s15850	52	45	44	36.1	3.6
C6288	50	33	33	6.3	0.3
biomed	161	136	136	64.0	5.4
primary2	133	114	116	16.2	4.8
struct	33	33	33	6.4	4.8
s35932	46	39	39	181.2	0.9
s38584	50	37	35	144.2	2.9
s38417	55	48	48	143.2	1.3
Mean	60.8	48.9	48.2	31.6	3.5

Table 9. Comparison of DFRG with higher-level gains against basic DFRG.

3.3 Extension to Clustering

Because DFRG more than doubles the size of the network before partition, it was thought that DFRG would benefit considerably if the problem size were reduced through clustering. We perform DFRG at each clustering level in place of the Strawman FM algorithm, and gradually expand the maximum circuit size as in the N/K extension to clustering. The clustering is performed upon the original graph, and the replication graph is created from the clustered graph at each clustering level. We retain the use of higher-level gains as explained in the previous section.

We could undoubtedly further improve cut sizes if at each clustering level we preprocess the original graph with Strawman FM before creating the replication graph and executing DFRG. However, we would have to modify

Strawman FM to support the motion of replicated nodes, and it therefore becomes identical to N/K with an infinite replication threshold. Hence we reserve this modification for the combinations section on page 30.

Benchmark	Best Strawman	DFRG w/ Higher Level Gains	DFRG with Clustering	Cumulative Execution Time (min.)	Replication (percent) of Best
s5378	60	41	41	16.7	6.8
s9234	45	34	32	28.1	5.3
s13207	67	47	63	47.2	7.1
s15850	52	44	45	59.8	5.9
C6288	50	33	33	10.4	2.4
biomed	161	136	107	85.7	4.8
primary2	133	116	121	25.3	6.0
struct	33	33	33	7.9	0.0
s35932	46	39	24	280.3	5.6
s38584	50	35	29	221.6	3.2
s38417	55	48	45	223.4	3.4
Mean	60.8	48.2	44.9	48.2	5.1

Table 10. Comparison of DFRG with higher-level gains against DFRG with clustering.

4 Functional Replication

Functional Replication is quite similar in form to N/K, but is specialized for partitioning onto Xilinx-based (3000 series) devices [11]. Rather than using gate-level nodes as the base element for partitioning, functional replication uses configurable logic blocks (CLBs), and therefore the circuit must be technology mapped prior to partitioning.

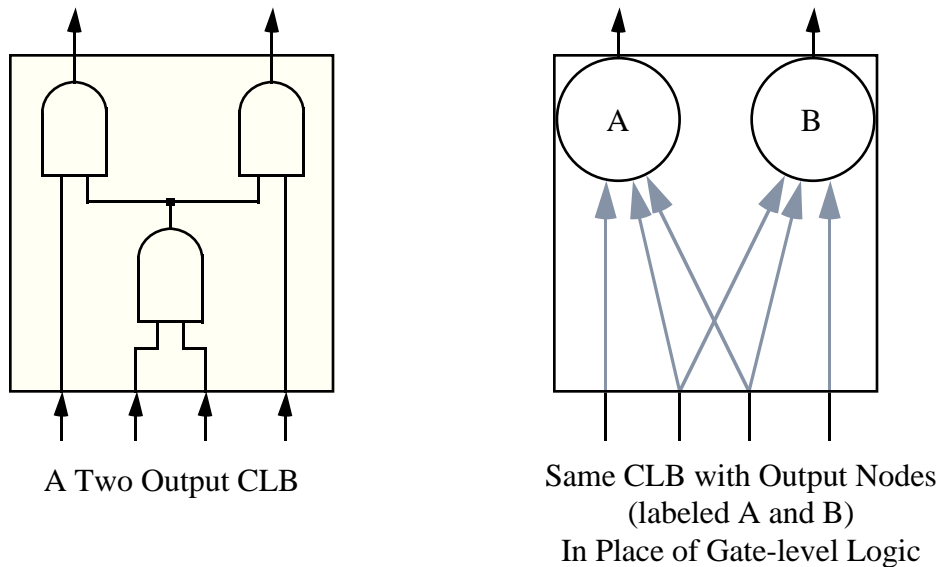


Figure 11. How a standard CLB (left) is viewed under functional replication (right).

The gain function of Functional replication is most easily described by considering each CLB to consists of a cluster of nodes in which there is exactly one node for each output net of the CLB. Each of these “output” nodes is connected to a unique output net, and to all input nets of the CLB which are required to generate the correct signal

on the output net. Hence, each output node of a CLB represents the gate-level logic which the technology mapper assigned to the CLB to generate the output signal.

For the following description, we consider each CLB to be a cluster of output nodes, and each net adjacent to a CLB to be connected to the appropriate “output” nodes rather than to the CLB itself. We have two types of movements in Functional Replication. The first type of movement is the movement of an entire CLB cluster of “output” nodes across the partition – this is called a standard movement, and it is identical to the usual sort of movement in FM. The second type of movement is the process of removing a single “output” node from a CLB cluster, moving it across the partition, and forming a new CLB cluster which consists of this single “output” node – this is called a replication movement. By describing a replication movement in this way, we see that replication no longer inherently forces each input net of the replicated object (CLB in this case) into the cut set. An input net of a replicated CLB is only moved into the cut set (presuming it is not already there) if, as a result of the “output” node movement, it is now connected to “output” nodes in each partition. Furthermore, an output net of a replicated CLB is no longer forced out of the cut set. Only a single output net of the original CLB is affected by the replication, and this is the output net corresponding to the moved “output” node. All other output nets remain in their previous state (cut or uncut). The output net corresponding to the moved “output” node may become cut or uncut depending on whether the output net is now connected to “output” nodes in both partitions (as in a standard FM style movement). As the above discussion indicates, the replication in Functional Replication is more akin to unclustering and standard movement than true replication and the concomitant creation of signal redundancy.

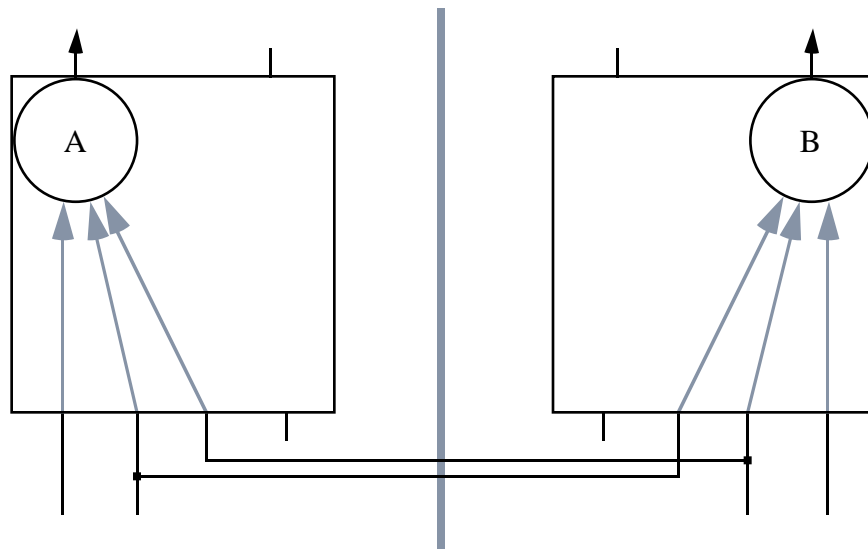


Figure 12. A replicated two-output CLB.

In our implementation of Functional Replication we set a hard limit of seven percent CLB expansion, and use a threshold replication potential of zero. To calculate the replication potential of a CLB we simply count the number of input nets of the CLB which are adjacent to a single “output” node – this number is the replication potential. Any CLB which has a replication potential below the threshold replication potential is excluded from replication moves. We further exclude single output CLBs from Functional Replication since a single output CLB cannot be split into two distinct output functions. CLBs with high replication potentials tend to force relatively few input nets into the cut set when replicated, and therefore contribute disproportionately to cut size reduction via replication. By modulating replication set size through the threshold replication potential the bulk of the gain of replication is retained even at small circuit expansions. However, since we are explicitly limiting circuit expansion to seven percent to allow comparison between replication algorithms, we do not require the replication inhibiting effect of the

replication potential threshold, and therefore we set it to its lowest value – zero – to realize the maximum gain from replication.

Benchmark	Strawman Unmapped Benchmark	Strawman Mapped Benchmark
s5378	60	61
s9234	45	59
s13207	63	113
s15850	45	81
s35932	46	129
s38584	50	119
s38417	51	100
Mean	53.1	90.7

Table 11. Comparison of Strawman on mapped and unmapped benchmarks.

Because we are moving clusters of gate-level nodes (CLBs) rather than gate-level nodes, one would naturally expect that the cut sizes resulting from the partitioning of mapped circuits to be somewhat inflated since we are working with a rather small subspace of the true solution space. The fact that the cut values are so much worse than that of their gate-level counterparts (see above table) is perhaps indicative of tech-map clusters being non-conducive to partitioning [6]. The replication moves of Functional Replication serve to ameliorate the negative effects of tech map clustering by introducing a limited form of dynamic unclustering. However, since we are not performing much in the way of true replication, we have not significantly altered the size of our solution space, and we would therefore expect Functional Replication, if it were a reasonable algorithm, to perform roughly on par with Strawman. The fact that Functional Replication does perform slightly better than Strawman is somewhat surprising, and perhaps due to the limited amount of gate-level replication implicit in the replication moves (output nodes of the same CLB need not contain mutually exclusive logic). This implicit replication and the gate-level clustering inherent in the formation of “output” nodes may perhaps foreshadow future algorithms which use replication in preprocessing techniques to make circuits easier to partition.

Benchmark	Best Strawman	Best DFRG w/Clustering	N/K w/Gradient	Best Functional Replication w/ Rep. Pot. Zero	Replication (percent) of Best
s5378	60	41	43	55	1.0
s9234	45	32	32	48	4.2
s13207	67	63	43	79	5.5
s15850	52	45	41	59	3.1
s35932	46	24	23	42	2.4
s38584	50	29	32	50	1.6
s38417	55	45	49	30	2.2
Mean	53.1	38.1	36.6	49.9	2.9

Table 12. Comparison of functional replication with other replication algorithms.

Nevertheless, the cut values produced by Functional Replication are significantly worse than other replication algorithms. Such an inflated cut size might be justified if recursive bipartitioning resulted in a large expansion of the number of CLBs required to map a circuit. This is a legitimate concern, for one would expect at least some small increase in the total number of CLBs required to technology map a partitioned circuit since the mapper is presented with a number of smaller problem sizes, and therefore may not have the flexibility in the mapping that it would have if presented with the circuit in its entirety. However, our results indicate that such expansion is rather moderate, and it is quite possible that the recursive bipartitioning of mapped circuits via Functional Replication

would produce significantly higher expansions, for in general several rounds of recursive bipartitioning are required to distribute a circuit onto FPGA devices.

Benchmark	Percent Expansion
s5378	6.2
s9234	10.5
s13207	-0.6
s15850	1.5
s35932	14.0
s38584	1.9
s38417	-0.1
Mean	4.8

Table 13. Benchmarks were recursive partitioned prior to technology mapping. Three to six rounds of recursive partitioning were required to produce partitions small enough to fit onto FPGA devices after technology mapping. When technology mapping was completed, the total number of CLBs required for the mapping of the partitioned circuit was calculated and compared to the total number of CLBs required for the mapping of the original circuit without partitioning. This table logs the resulting expansion.

5 Flow Based Techniques

The final pair of algorithms are significantly different from the preceding in that they are flow based rather than move based. The flow based algorithms strive to capitalize on a flow network formulation which derives an optimal minimum-cut replication set (without size bounds) when provided with an initial partition. To develop this technique, we require the following two definitions [3][5]:

A *flow network*, $G = (V,E)$, is a directed graph in which each edge, $(u,v) \in E$, has a non-negative capacity, and there exists two special nodes $S, T \in V$ designated as a source and sink respectively.

A *flow* in G is a function $f: V \times V \rightarrow \Re$ such that

- (1) $f(u,v) \leq c(u,v)$, $u,v \in V$, where $c(u,v)$ is the capacity of the edge $(u,v) \in E$
- (2) $f(u,v) = -f(v,u)$, $u,v \in V$
- (3) $\sum_{v \in V} f(u,v) = 0$, $u \in V - \{S,T\}$

Also needed is the Ford-Fulkerson max-flow/min-cut Theorem [5]:

For any network the maximal flow value from S to T is the size of some minimum cut in G .

5.1 The Optimal Algorithm [8]

Given a graph $G = (V,E)$, and an initial partition of the vertices (X,Y) such that $X \cap Y = \emptyset$, and $X,Y \subset V$, we select a partition arbitrarily (let us choose X), and form a flow network by creating a node S , which will be the source of the network, and clustering with S the various nodes in X which we do not want replicated, and which we would like to act as a source of flow in the network. We also create a node T , which will be the sink of the network, and cluster with T all nodes which are in the opposite partition, and are attached to edges which have their source in X . All edges which have their source in the partition opposite to X are removed from the network, and all remaining edges are given a capacity of one.

We push as much flow from S to T as we can, and thus by the Ford-Fulkerson Theorem we produce a minimum cut. We find the minimum cut closest to T by a breadth-first search from T , and every node on the sink side of the

minimum cut, and in partition X is designated for replication. This is actually a modification proposed by the authors of [18] to reduce the amount of replication – the original algorithm finds the minimum cut closest to S by a breadth-first search from S . By replicating these designated nodes we place all edges in the minimum cut into the cut set, and guarantee that any other edge which has its source in X is not in the cut set. The original graph is then reestablished; we select the opposite partition, and we repeat the above (with Y substituted for X). By finding the replication subset which minimizes the cut edges which have their source in X , and then finding the replication subset which minimizes the cut edges which have their source in Y , we have found the replication set which minimizes the number of cut edges for the partition (X,Y) .

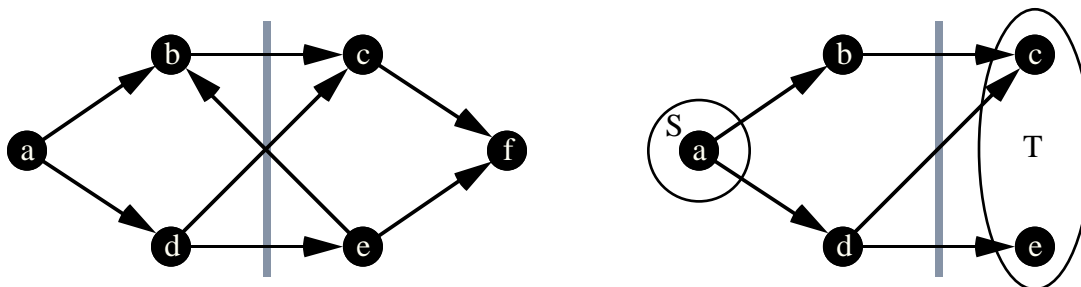


Figure 13. Creating a flow network from a directed graph.

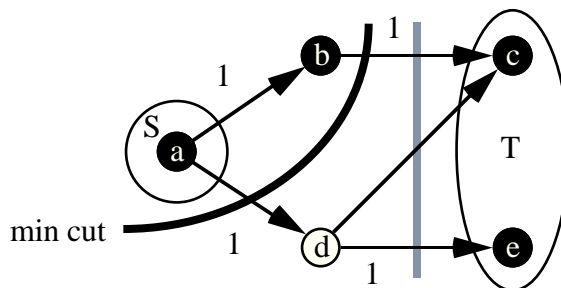


Figure 14. Deriving replicated nodes from flow network. White node is replicated.

The above algorithm has two limitations: it applies only to directed graphs (not hypergraphs), and it assumes unbounded partition size. By extending this algorithm to incorporate directed hyper-edges or size constraints, we lose optimality, and therefore must resort to heuristics.

5.2 Hyper-MAMC (Min-Area Min-Cut) [18]

To use the network flow formulation we must reduce hyperedges to edges. Each hyperedge is replaced with a hyper-node. We connect the source of the former hyperedge to the hypernode with an edge of unit capacity. For each sink of the former hyperedge we add an infinite capacity edge from the hypernode to the sink. This construction limits each hyperedge to unit flow. Furthermore, the construction which replaced the hyperedge can only be cut at one place – between source and hypernode, for in max-flow/min-cut formulations, the min-cut details a bottle-neck which prevents further flow from source to sink, and obviously infinite capacity edges can never contribute to such a bottleneck, and hence cannot be part of the minimum cut. Because this construction can only be cut at one place, and since it can only support unit flow, this guarantees that each hyperedge contributes at most once to the minimum cut derived from the maximum flow.

As in the optimal algorithm we select a partition arbitrarily, and form a flow network. We then push as much flow from S to T as possible, deriving a minimum cut. However, rather than replicated each node on the target side of the minimum cut, we first determine if this amount of replication would exceed the partition size limitation. If so, we

determine all hyperedges which have their source clustered with the network source, and which have positive flow. A sink of such a hyperedge is selected, provided it is in the same partition as the network source, and it is clustered with the network source. This has two implications: this selected node can no longer be replicated; and the selected node can now act as a source of flow in the network. We then attempt to push more flow through this new flow network; if successful a higher max-flow is obtained implying we have produced a larger “min-cut”, and usually a smaller replication set. This process of incrementing the flow is repeated until the size of the replication set falls within acceptable parameters. Once we have derived a sufficiently small set for replication, we dismantle the flow network, select the opposite partition, and repeat the above.



Figure 15. Reduction of hyper-edges to normal edges. White node is the hyper-node.

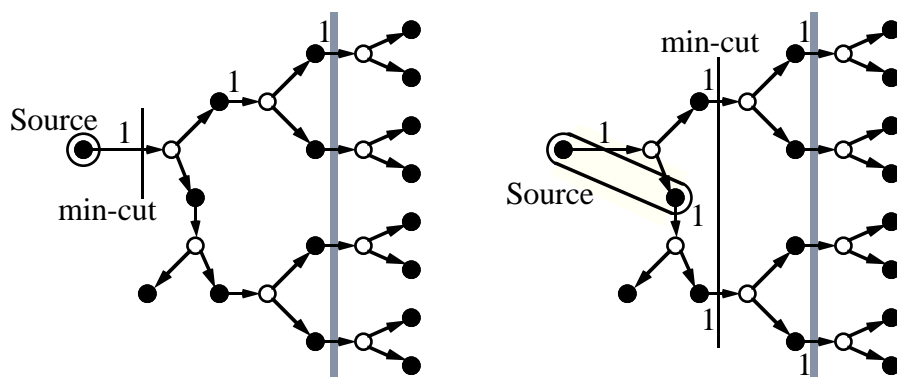


Figure 16. Incrementing the flow reduces the replication set and increases the cut size.

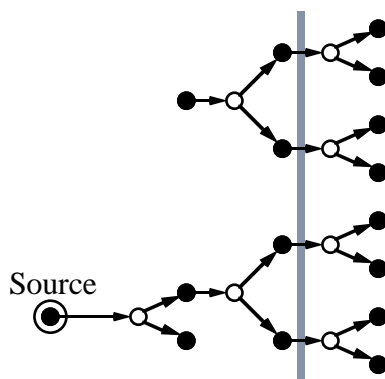


Figure 17. Cannot reach top nodes from the network source. Such nodes will always be in the replication set.

A peculiarity of this algorithm, which is particularly pronounced when working with small circuit expansions, is that we may exceed partition size limits, yet there may be no node available to cluster with the source for flow

incrementing. This is caused by an excessive number of nodes in the active partition which are unreachable from the network source. However, this is a fairly rare event, and when it occurs we default to a directed-FM (see MC-rep below) on the modified network graph.

There are several choices which influence the behavior of this algorithm. We will examine each one.

5.2.1 Node Choice for Source Clustering

When creating the flow network, we must choose a number of nodes to cluster together to create the network source. The most obvious candidates for source clustering are Input Buffers or IBUFs. These nodes tend to be liberally scattered about both partitions, and presumably every node in the graph is reachable through some IBUF. However, it appears that a random node selection tends to produce better results. This is likely due to the variability in node selection which it introduces, providing the algorithm with an opportunity to get “lucky” with its source choice when iterated multiple times.

Random node selection introduces a complication into the algorithm. We insist that replicated IBUFs cut an external net, and therefore add to the cut set. But these external nets have no sources, and hence can never have any flow. Such nets can never contribute to the minimum cut in a flow based algorithm, and therefore can never be in the cut set. To eliminate this problem, all external nets use the network source as their source.

When choosing nodes randomly to cluster with the source, we select the number of nodes as a function of initial directed cut size. The initial directed cut size is the number of hyperedges which have their source in the chosen partition, and one or more sinks in the opposite partition. This is simply the number of hyperedges which span both partitions in the newly created flow network. In the worst case, the minimum cut is the initial directed cut size (implying no replication occurred), and therefore the maximum flow can never be higher than this value. It seemed a reasonable convention to choose the number of sources based upon the maximum amount of flow we may have to push through the network. The following table details the various source selection strategies. Columns with the heading RAND indicate that a random node selection strategy was used. The number following the RAND specification is the multiplier used to determine the total number of source nodes selected. For example, RAND 0.25 indicates that a random node selection was employed where a total of $0.25 * \text{directed_cut_size}$ nodes were selected.

Benchmark	Best Hyper-MAMC IBUF	Best Hyper-MAMC RAND 2.0	Best Hyper-MAMC RAND 1.0	Best Hyper-MAMC RAND 0.5	Best Hyper-MAMC RAND 0.25	Best Hyper-MAMC RAND 0.125
s5378	46	44	42	41	39	40
s9234	31	32	29	29	28	29
s13207	45	46	48	47	46	45
s15850	40	39	39	38	37	37
C6288	34	34	33	33	33	33
biomed	70	92	79	75	73	83
primary2	115	120	112	109	110	110
struct	34	33	33	33	33	33
s35932	39	39	39	39	39	39
s38584	26	28	26	25	25	25
s38417	40	40	37	37	37	38
Mean Time	24.7	21.7	22.5	23.4	24.3	21.9
Mean Cut	43.2	44.5	42.5	41.8	41.2	42.0

Table 14. Effect of source choice on cut size. Initial partitions generated by Strawman with a maximum partition size of 51% of the logic

Selecting randomly a number of sources equal to one-quarter the directed cut size produces the best cuts. Therefore all future hyper-MAMC variants in this paper will incorporate this strategy of source selection.

5.2.2 Initial Partition Variation

The hyper-MAMC algorithm attempts to find the optimal replication set given the initial partition, but the initial partition itself may vary considerably in quality, and therefore skew the efforts of hyper-MAMC. One may attempt to improve the quality of the initial partition by varying the size bounds within the maximum size threshold we allow for replication (53.6%). We have been previously using partition bounds of 51%/49%, to reserve the maximum room for the operation of the hyper-MAMC algorithm, but 52%/48% and 53%/47% partition thresholds are also valid possibilities, and should give us better initial partitions for the hyper-MAMC algorithm at the expense of hindering replication in some situations. The results indicate that improving initial partition quality does indeed improve the cuts created by hyper-MAMC, but we must leave the algorithm some room to operate. All future hyper-MAMC variants in this paper will use an initial partition size limit of 52%/48% circuit size.

Benchmark	Best Strawman	Initial Partition size limit of 49%/51%	Initial Partition size limit of 48%/52%	Initial Partition size limit of 47%/53%
s5378	60	39	40	41
s9234	45	28	31	35
s13207	67	46	44	47
s15850	52	37	38	39
C6288	50	33	33	33
biomed	161	73	61	84
primary2	133	110	113	107
struct	33	33	33	33
s35932	46	39	23	39
s38584	50	25	25	25
s38417	55	37	36	38
Mean Time	17.4	24.3	24.3	21.8
Mean Cut	60.8	41.2	39.1	43.1

Table 15. Effect of initial partition size limit on Hyper-MAMC.

5.2.3 Node Choice for Incrementing Flow

When faced with an overly large replication set, the hyper-MAMC algorithm will attempt to increment the flow by choosing a node to cluster with the network source. We can attempt to find the best node choice by selecting each node, and determining which node produces the greatest decrease in the size of the replication set, or the smallest increase in maximum flow, or both. This greedy strategy of node choice does not necessarily produce the best cut sizes; however, it does tend to produce good results. Unfortunately this exhaustive node search is generally too time consuming, and therefore the hyper-MAMC algorithm includes a threshold parameter for use with node choice [17]. When incrementing the flow, and choosing from among a threshold number of nodes (or less), we determine the effects of incrementing the flow to each node, and finally increment the flow to that node which produced the best results (least increase in max-flow with ties broken by choosing the largest decrease in total size of nodes replicated). If we have greater than a threshold number of nodes to choose from, we choose a node for source clustering arbitrarily.

As Table 16 illustrates, small threshold values have little impact on partition quality. Apparently the number of occasions in which the node choice falls below the threshold value are very few on these benchmarks. Unfortunately, using a higher threshold value tends to slow the algorithm considerably. To avoid this performance penalty, we decided to select the initial partition (as produced by Strawman) which produced the best final cut size

after performing hyper-MAMC with a size threshold of zero, and then run a high threshold hyper-MAMC on only this initial partition. This amortizes the time cost of a single high threshold hyper-MAMC over the previous thirty iterations of hyper-MAMC.

Benchmark	Best Strawman	Best Threshold zero	Best Threshold five	Best Threshold ten	Best Threshold twenty-five
s5378	60	40	40	40	40
s9234	45	31	31	31	31
s13207	67	44	44	44	44
s15850	52	38	38	38	38
C6288	50	33	33	33	33
biomed	161	61	61	61	61
primary2	133	113	113	113	113
struct	33	33	33	33	33
s35932	46	23	23	23	23
s38584	50	25	25	25	25
s38417	55	36	36	36	36
Mean Time	17.4	21.9	24.3	23.4	37.6
Mean Cut	60.8	39.1	39.1	39.1	39.1

Table 16. Effect of small threshold values on partition quality.

Benchmark	Best Strawman	Amortized Threshold fifty	Amortized Threshold one-hundred	Amortized Threshold two-hundred
s5378	60	40	40	40
s9234	45	31	28	28
s13207	67	44	44	44
s15850	52	38	38	38
C6288	50	33	33	33
biomed	161	61	58	58
primary2	133	113	108	108
struct	33	33	33	33
s35932	46	23	23	23
s38584	50	25	25	25
s38417	55	36	36	36
Mean Time	40.6	26.3	28.1	32.0
Mean Cut	60.8	39.1	38.4	38.4

Table 17. Amortizing high threshold Hyper-MAMC over twenty Strawman partitions.

As expected, high threshold values improve cut size by facilitating a more intelligent approach to source node selection. It was therefore surmised that selecting the best of a number of node choices was too good an idea to relegate to a single promising partition, or to situations in which we had only a few nodes from which to choose. Instead, each time we attempt to increment the flow we randomly choose a threshold number of nodes, and select the best for incrementing the flow.

5.3 MC-Rep [8]

We reduce hyper-edges to edges as in the hyper-MAMC algorithm (the authors of [8] use a different hyperedge representation, but it is far more cumbersome), then as in the optimal algorithm we select a partition arbitrarily, and

form a flow network. We then push as much flow from S to T as possible, deriving a minimum cut. However, rather than replicated each node on the target side of the minimum cut, we first determine if this amount of replication would exceed the partition size limitation. If the size of the replication set is too large, we dismantle the flow network, and re-establish the original hypergraph. All nodes which were not in the selected partition are permanently locked, and a directed FM is performed upon the hypergraph. In this directed FM, hyperedges are cut only if the source of the hyperedge is in the selected partition (and is unreplicated), and at least one sink is in the opposite partition. We only allow replication and unreplication moves – no standard FM moves are allowed – and only by nodes which were originally in the selected partition (all other nodes are permanently locked). In this way, a replication set is derived which meets the size criterion. We then select the opposite partition, and repeat as above.

Benchmark	Best Strawman	Random Node Choice w/ Threshold of five	Cumulative Execution Time (min.)	Percent Replication of Best
s5378	60	39	16.4	5.5
s9234	45	28	35.6	7.0
s13207	67	42	121.5	7.0
s15850	52	38	107.8	4.9
C6288	50	33	6.2	0.9
biomed	161	58	233.6	6.7
primary2	133	106	49.4	6.0
struct	33	33	4.4	1.4
s35932	46	23	173.8	5.5
s38584	50	25	289.6	4.8
s38417	55	34	283.1	5.5
Mean	60.8	37.8	59.7	4.6

Table 18. Hyper-MAMC with selection of best of random five nodes for incrementing flow.

We apply the lessons learned from hyper-MAMC and randomly cluster a number of nodes equal to twenty-five percent of the directed cut size to create the network source. We also use Strawman to generate an initial partition with 48%/52% size limits.

Benchmark	Best Strawman	Best MC-REP	Cumulative Execution Time (min.)	Percent Replication of Best
s5378	60	44	6.6	5.2
s9234	45	37	11.1	5.7
s13207	67	61	24.3	5.5
s15850	52	50	26.6	2.4
C6288	50	33	4.4	0.9
biomed	161	127	53.8	4.9
primary2	133	121	10.8	5.2
struct	33	33	3.0	1.4
s35932	46	23	130.9	5.5
s38584	50	25	103.6	4.8
s38417	55	44	103.3	3.7
Mean	60.8	46.3	21.7	3.3

Table 19. Comparison of Strawman against MC-Rep.

The MC-Rep algorithm performs significantly worse than the hyper-MAMC algorithm. This is due to the fact that we are using rather small circuit expansions, for the MC-Rep algorithm almost always throws the flow network

solution away (because it fails to meet the size bounds), and performs the directed FM. Hyper-MAMC, by retaining the flow graph and modifying it gradually, remains closer to the optimal formulation, and therefore derives superior results.

6 Combinations

DFRG, N/K and Hyper-MAMC each approach the replication problem from a different angle, and therefore there is every reason to believe that a combination of these algorithms may produce better results than any single algorithm.

6.1 N/K and DFRG

This is a likely combination since they are both move based, but operate on significantly different graphs. This suggests that the topology of their solution spaces are substantially different, and therefore the local minima of one need not be a local minima of the other. By combining these two algorithms one could therefore expect solutions which are superior to either operating alone. Since the DFRG performs quite well with clustering, at each clustering level of the Strawman algorithm the Strawman FM was replaced with a variable threshold N/K followed by a DFRG. This is essentially the DFRG as extended to clustering, but at each level the partition is optimized prior to DFRG processing.

Benchmark	Strawman	Best N/K Gradient	DFRG with Clustering	N/K w/ Var. Thres. Followed by DFRG	Cumulative Execution Time (min.)	Percent Replication of Best
s5378	60	43	41	39	12.3	4.9
s9234	45	32	32	24	21.8	6.2
s13207	67	43	63	52	37.3	6.9
s15850	52	41	45	38	46.9	6.6
C6288	50	33	33	33	7.9	0.1
biomed	161	130	107	113	69.7	7.1
primary2	133	114	121	101	17.6	6.3
struct	33	33	33	33	6.6	0.0
s35932	46	23	24	23	262.4	5.5
s38584	50	32	29	31	202.2	2.1
s38417	55	49	45	44	202.0	3.9
Mean	60.8	44.7	44.9	42.0	39.1	5.3

Table 20. Combination of N/K with variable threshold, and DFRG extended to clustering.

6.2 All Post-processing

The results of the N/K and DFRG combination while better than either algorithm alone, still could not match the performance of hyper-MAMC. To capitalize on the performance of the hyper-MAMC algorithm, it was decided to let the standard hyper-MAMC algorithm (without random node choice for incremental flow since this significantly slows the algorithm) produce the original replication set, and then let each of the basic N/K and DFRG algorithms attempt to improve upon this base. The initial partition was generated by Strawman with 48%/52% partition size limits.

7 Circuit Expansion

Having found that hyper-MAMC produces the best results, we use our optimized hyper-MAMC algorithm with random node selection to determine the effects of increased circuit expansion on cut size. Each data point in the graph was derived by choosing the best of ten runs of our optimized hyper-MAMC algorithm at the selected circuit expansion. Recall that our optimized hyper-MAMC implementation used an initial circuit partition of 52%/48%

implying a size variation of four percent, or fifty-seven percent of our 7 percent total circuit expansion ($4/7 \cong 0.57$). For larger circuit expansion we have a wider range of choices for the initial partition size variation, but in all cases, the best results were achieved by allowing the initial partition to vary by fifty to seventy-five percent of the allowed circuit expansion.

Benchmark	Strawman	hyper-MAMC, N/K, DFRG	Cumulative Execution Time (min.)	Percent Replication of Best
s5378	60	38	10.8	4.0
s9234	45	26	19.4	7.1
s13207	67	39	41.6	6.8
s15850	52	37	43.0	7.1
C6288	50	33	6.6	0.4
biomed	161	60	81.3	6.0
primary2	133	100	19.2	7.0
struct	33	33	6.0	1.4
s35932	46	23	164.7	5.5
s38584	50	25	135.9	4.8
s38417	55	35	139.2	5.1
Mean	60.8	37.2	34.1	5.1

Table 21. Hyper-MAMC, followed by N/K, and finished with DFRG.

Cut Size Reduction as a Function of Circuit Expansion

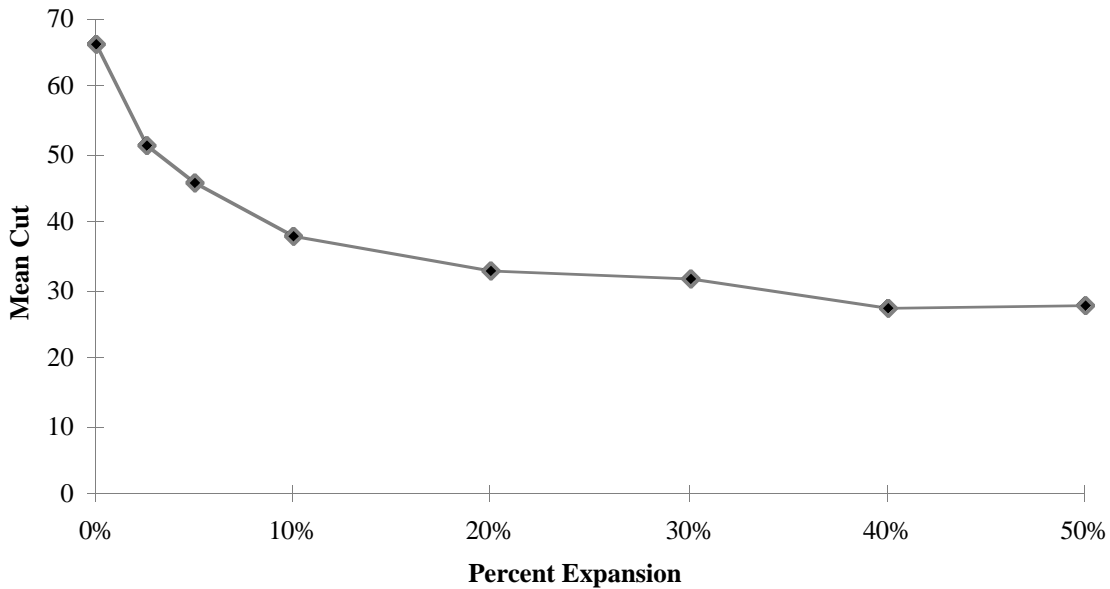


Figure 18. Graph of cutsize for various restrictions on allowed circuit expansion.

8 Future Work

A possible avenue for future research is node replication with the purpose of making graph partitioning easier for standard move based algorithms. Such replication could take place at the clustering or unclustering phase of the Strawman partitioner to create better cluster separation. This sort of replication could also take place dynamically

(as in the Functional Replication algorithm), to allow the partitioner to move fractions of a cluster which has been cleaved through replication. Another area of exploration is clustering to take full advantage of replication. Current clustering algorithms are created with an eye toward the aggregation of nodes which are unlikely to be separated via standard FM style movements. However, the notion of which nodes should be clustered may be somewhat modified if we were to agglomerate on the basis of replication style movements. Still another area of future work could be in local logic manipulations to reduce cut size. Equivalent logic transformations could be applied as a post-processing technique to reduce the cut size, and may or may not involve circuit expansion or logic replication.

9 Conclusion

By adding replication to the partitioning problem we have significantly expanded the solution space. This expansion tends to overwhelm the move-based replication algorithms (N/K, DFRG), for they both require good initial partitions or clustering to produce cut sizes which are significantly better than those produced by Strawman. This, to some extent, explains the superior performance of hyper-MAMC. Like the move-based approaches, hyper-MAMC requires a restricted space (induced by a good initial partition) to produce good cut sizes; however, unlike the move based algorithms, hyper-MAMC is based upon a formulation which derives an optimal solution set (without size constraint) given the restricted space, and therefore is better adapted to utilize the subspace effectively.

When integrating a replication algorithm into a partitioner, one must look for ways to restrict the solution space over which the replication algorithm operates. Through an innovative use of clustering, unclustering, and higher-level gains, we have achieved an effective synthesis of the Strawman partitioner with the various replication algorithms. By further experimentation with the parameters associated with the various algorithms, and by making some unique algorithmic adaptations, we have achieved a ten percent improvement in cut size over the standard implementations of the most promising replication algorithms.

Algorithm	Basic Implementation	Improved Implementation	Percent Improvement
Newton/Kring	48.2	44.7	7.3
DFRG	48.9	44.9	8.2
Hyper-MAMC	43.2	37.8	12.5

Table 22. Improvements achieved over basic algorithms.

With these improvements, we can achieve nearly a forty percent cut size reduction over already high-quality partitions through a judicious use of replication. This is particularly dramatic considering the relatively small circuit expansions we are working with, and our conservative convention regarding I/O node replication.

Replication Algorithm	Percent Improvement over Strawman	Percent Replication
Functional Replication	6.0	2.9
MC-Rep	23.8	3.3
DFRG	26.2	5.1
N/K	26.5	3.5
N/K, DFRG	30.9	5.3
Hyper-MAMC	37.8	4.6
Hyper-MAMC, N/K, DFRG	38.8	5.1

Table 23. Effect of replication on cut size. Each algorithm incorporates the best set of optimizations and parameter settings described in this paper.

References

- [1] C. J. Alpert, A. B. Kahng, "Recent Directions in Netlist Partitioning: A Survey", *Integration: the VLSI Journal*, Vol. 19, No. 1-2, pp. 1-81, 1995.
- [2] J. Cong, M. Smith, "A Parallel Bottom-up Clustering Algorithm with Applications to Circuit Partitioning in VLSI Design", *Design Automation Conference*, pp. 755-760, 1993.
- [3] T. H. Cormen, C.E. Leiserson, R.L. Rivest, *Introduction to Algorithms*, MIT Press/McGraw-Hill, Cambridge, MA, 1990.
- [4] C. M. Fiduccia, R. M. Mattheyses, "A Linear-Time Heuristic for Improved Network Partitions", *Design Automation Conference*, pp. 241-247, 1982.
- [5] L.R. Ford, D.R. Fulkerson, *Flows in Networks*, Princeton Univ. Press, Princeton, NJ, 1962.
- [6] S. A. Hauck, *Multi-FPGA Systems*, Ph.D. dissertation, University of Washington, Dept. of CSE., pp. 131-168, 1995.
- [7] L. Hagen, A. B. Kahng, "New Spectral Methods for Ratio Cut Partitioning and Clustering", *IEEE Transactions on Computer-Aided Design*, Vol. 11, No. 9, pp. 1074-1085, Sept. 1992.
- [8] L. J. Hwang, A. El Gamal, "Min-Cut Replication in Partitioned Networks", *IEEE Transactions on Computer-Aided Design*, Vol. 14, No. 1, Jan 1995, pp. 96-106.
- [9] C. Kring, A. R. Newton, "A Cell-Replicating Approach to Mincut-Based Circuit Partitioning", *International Conference on Computer-Aided Design*, pp. 2-5, 1991.
- [10] B. Krishnamurthy, "An Improved Min-Cut Algorithm for Partitioning VLSI Networks", *IEEE Transactions on Computers*, Vol. C-33, No. 5, pp. 438-446, May 1984.
- [11] R. Kuznar, F. Brglez, B. Zajc, "Multi-way Netlist Partitioning into Heterogeneous FPGAs and Minimization of Total Device Cost and Interconnect", *Design Automation Conference*, pp. 238-243, 1994.
- [12] G. M. Riess, K. Doll, F. M. Johannes, "Partitioning Very Large Circuits Using Analytical Placement Techniques", *Design Automation Conference*, pp. 646-651, 1994.
- [13] L. Liu, M. Kuo, C. Cheng, T. C. Hu, "A Replication Cut for Two-Way Partitioning", *IEEE Transactions on Computed-Aided Design*, Vol. 14, No. 5, May 1995, pp. 623-632.
- [14] L. Liu, M. Kuo, S. Huang, C. Cheng, "A Gradient Method on the Initial Partition of Fiduccia-Mattheyses Algorithm", *International Conference on Computer-Aided Design*, pp. 229-234, 1995.
- [15] D. M. Schuler, E. G. Ulrich, "Clustering and Linear Placement", *Design Automation Conference*, pp. 50-56, 1972.
- [16] Xilinx Inc., *The Programmable Gate Array Data Book*, 1992.
- [17] H. H. Yang, D. F. Wong, "Efficient Network Flow Based Min-Cut Balanced Partitioning", *International Conference on Computer-Aided Design*, pp. 50-55, 1994.
- [18] H. H. Yang, D. F. Wong, "New Algorithms for Min-Cut Replication in Partitioned Circuits", *International Conference on Computer-Aided Design*, pp. 216-222, 1995.