International Conference on Computer-Aided Design, 1997.

Replication for Logic Bipartitioning

Morgan Enos, Scott Hauck, Majid Sarrafzadeh
Department of Electrical and Computer Engineering, Northwestern University, Evanston, IL 60208
{morgan, hauck, majid}@ece.nwu.edu

Abstract

Logic replication, the duplication of logic in order to limit communication between partitions, is an effective part of a complete partitioning solution. In this paper we seek a better understanding of the important issues in logic replication. By developing new optimizations to existing algorithms we are able to significantly improve the quality of these techniques, achieving up to 12.5% better results than the best existing replication techniques. When integrated into our already state-of-the-art partitioner, we improve overall cutsizes by 37.8%, while requiring the duplication of at most 7% of the logic.

1 Introduction

Logic partitioning is one of the critical issues in CAD for digital logic. For most applications the goal is to minimize the communication between partitions while ensuring that each partition is no larger than the capacity of the target device. While it is possible to solve the case of unbounded partition sizes exactly [3], the case of balanced partition sizes is NP-complete [9]. As a result, numerous heuristic algorithms have been proposed [1].

We have already demonstrated that a careful integration of existing bipartitioning techniques, along with some new approaches and optimizations, can yield a bipartitioning algorithm significantly better than the current state-of-the-art [12]. Our algorithm achieved a 8% improvement over PROP [5], 16% improvement over FBB [19], 22% improvement over Paraboli [18] and MELO [2], 50% improvement over Fiduccia-Mattheyses [7], and a 58% improvement over EIG1 [10] [2], some of the best current bipartitioning algorithms.

In this paper, we seek to understand the critical issues in logic replication, the selective duplication of logic to reduce the resulting cutset. We examine most of the current logic replication literature, integrating it into our already extremely efficient partitioning system. We also develop techniques to significantly improve the quality of their results, yielding much smaller cutsets. This will lead to a complete logic bipartitioning algorithm with replication which delivers superior results.

1.1 Optimized FM

Most of the algorithms for replication use the Fiduccia-Mattheyses (FM) algorithm [7]. FM is a widely used algorithm, and as such a number of ideas have emerged which improve upon the basic design. By a judicious combination of these ideas, improvements in performance can be realized. This is the premise behind Strawman [11] [12], for this partitioner actually consists of many techniques which achieve a synergistic reduction in cut size. As far as we know, this partitioner delivers the best standard bipartitioning results in the literature today. Because of these strong results we have chosen to use this system as a base for our replication work.

Although a complete description of the Strawman partitioner is beyond the scope of this paper, it will be important to understand three of its optimizations.

- **Random initial partition creation.** The partitioner randomly creates an initial partition, and multiple runs of the partitioner use different initial partitions.
- **Hierarchical clustering and iterative unclustering.** The algorithm clusters the circuit recursively, building a hierarchy of clusters. Then it first partitions at the top level of the hierarchy. Once it can find no improvement, the highest level of clustering is removed, and the partitioning resumed. This continues until all clusterings are removed.
- **Higher-level gains.** We use higher-level gains as proposed by Krishnamurthy [15]. This is a tie-breaking mechanism for nodes with identical gains which adds some foresight to the algorithm. A net contributes an nth 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 nth 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.

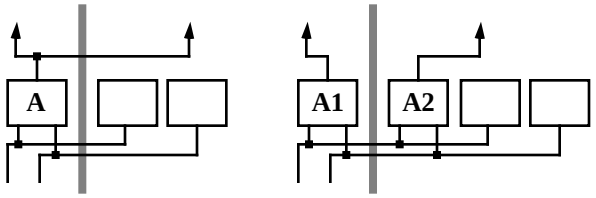

Figure 1. An example of node replication. Node **A** is replicated into **A1** and **A2**, yielding a gain of 1.

1.2 Methodology

Replication for cut minimization seeks out cells that when replicated 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.

As Figure 1 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:

(Eqn. 1) | cut outputs | - | uncut, *unreplicated* inputs |

By *unreplicated* input net we 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.

To determine the effectiveness of the replication algorithms, we run each thirty times on a SPARC5 on the benchmarks in Table 1. Partitions are allowed to grow to at most 53.6% of the circuit size (Strawman normally allows a partition to grow to 51% of total circuit size, and by allowing 5% of the logic to be replicated we reach an upper limit of .51*1.05 0.536). 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. We also require that if a circuit input is replicated, a net must be added to the cutset to carry the input between partitions [6].

The 11 benchmarks used in this paper are given in Table 1. Although this is a subset of the circuits typically used in partitioning research, these were the only circuits available to us that contained information on what node generates a given net, which is necessary for replication.

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
prim2	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 1. Characteristics of benchmark circuits.

In this paper we investigate most of the replication algorithms from the literature, as well as some novel extensions. Throughout this paper, algorithms directly from the literature will be referred to as "Basic".

2 Kring/Newton Replication

The algorithm proposed by Kring and Newton (KN) is a fairly straight-forward adaptation of FM to support cell replication [14]. In the FM algorithm, a node can exist in one of two states dependent upon which partition it is currently located. Kring and Newton introduce a third state – replication, in which a node exists in both partitions simultaneously. Also, nodes now have two potential moves: unreplicated nodes can either be replicated or moved to the other partition, and replicated nodes can be unreplicated into either of the two partitions.

The Kring/Newton algorithm always requires that unreplication move with a positive gain be taken before any normal moves, and disallows replication moves with a gain of less than 1. Our experiments indicate that allowing replication moves with a gain of 0 are useful, and our version of Kring/Newton allows such moves.

2.1 Basic Kring/Newton

The simplest way to integrated KN replication into Strawman is to replace Strawman's final FM run on the unclustered circuit with the KN algorithm. Essentially, this is the KN algorithm performed on an optimized initial partition – a modification proposed by Kring and Newton which was found to be useful for larger benchmarks.

2.2 Higher-Level Gains

In general, if we have two moves that have the same impact on the cutsize, but one is a normal move and the other is a replication move, we should choose the normal move in order to limit the amount of replication. To do this we set all higher-level gains of replication moves to -∞. This achieves a slight improvement over KN with a normal gain function (Table 2), and thus all future KN variants will be performed with higher-level gains.

Bench.	Straw	Basic KN	KN HLG	KN Clust.	KN Grad.
s5378	60	41	39	48	43
s9234	45	30	32	33	32
s13207	67	53	48	49	43
s15850	52	43	43	39	41
C6288	50	33	33	34	33
biomed	161	136	137	126	130
prim2	133	114	114	110	114
struct	33	33	33	33	33
s35932	46	39	39	28	23
s38584	50	35	35	46	32
s38417	55	51	49	47	49
Mean	60.8	48.2	47.7	47.6	44.7
Time	17.4	17.3	17.6	17.1	19.7
Rep %	0.0%	4.2%	4.5%	6.2%	3.5%

Table 2. Comparison of different optimizations to KN replication. "HLG" is higher-level gains. Each algorithm is executed thirty times, and the best cutsize is kept. "Time" is the geometric mean total CPU minutes for all 30 runs for a given benchmark. "Rep %" is the average percentage of the logic that is replicated to achieve the best cut for each benchmark.

2.3 Extension to Clustering

It was hoped that by earlier application of KN, better results might be achieved via increased opportunity for replication. So rather than perform KN style replication only after complete unclustering, KN 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 KN would be allowed to expand the circuit by one percent at each clustering level. Such expansion is cumulative.

2.4 Gradient Method

As the FM algorithm progresses the partitions get steadily better. Early in the algorithm there may be many high gain moves, and the partitioner radically changes the current partitioning. However, in later iterations the current partitioning is close to the final result, and the partitioner makes relatively minor alterations to the partitioning. When we add replication to this process, it can create a significant amount of duplication early in the partitioning process, limiting the partitioner's ability to change the current partitioning. Thus, replicating too early can degrade the final results. To deal with this, in Gradient KN we only attempt further replication when the cut sizes between successive inner-loop iterations of the Strawman FM algorithm change by less than ten percent. The circuit is expanded as in the previous section.

2.5 Kring/Newton Results

The results for the various optimizations to the KN algorithm are given in Table 2. As can be seen, basic KN produces results 20.7% better than the non-replicating algorithm. Higher-level gains improve this by an additional 0.8%, hierarchical partitioning yields an additional 0.2%, and the Gradient approach yields an additional 4.8%. This produces an overall 26.5% improvement over the basic Strawman algorithm.

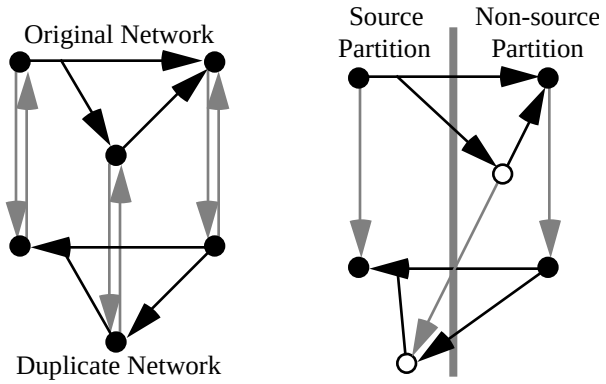

Figure 2. Hypothetical duplicated graph with infinite edges in both directions between original and duplicated nodes (left). Gray arrows are infinite weight edges. The replication graph has infinite edges only from original to duplicate (right). A duplicate node in the source partition with it's original in the non-source partition is considered duplicated (such as the node shown in white).

3 DFRG

Another move based approach is the Directed Fiduccia-Mattheyses on a Replication Graph, or DFRG [17]. DFRG uses a directed version of FM, where one partition is designated the "source" partition, and a net is considered cut only if one of its sources is in the source partition, and one of its sinks is in the other partition.

Imagine that we duplicate the circuit graph , and in the duplicate we reverse the signal flow direction (sources become sinks, sinks become sources). We connect each node to its duplicate with two infinite weight edges (edges that should never be cut), one directed from original to duplicate, another from duplicate to original (see Figure 2 left). This means that a node and its duplicate will always been in the same partition. Assuming a partitioning does not cut an infinite weight edge, it can be shown that the directed cutsize of this graph is the same as the undirected cutsize of the nets in the original graph. Thus, a directed partitioning of this graph is equivalent to an undirected partitioning of the original graph.

In DFRG partitioning we use the same graph, but remove the infinite weight edges from duplicate to original (the other edge remains). In this graph we can now have an original node in the non-source partition and its duplicate in the source partition (Figure 2 right). All such nodes will be replicated, and the directed cutsize of this graph accurately reflects this. Specifically, it can be shown that all input nets to a replicated node will be in the directed cutset unless their source is replicated, and the outputs of a replicated node are never in the directed cutset. Thus, performing directed partitioning of this replication graph allows partitioning and replication of the original graph to be performed simultaneously.

For basic DFRG we obtain an initial partition via Strawman and use DFRG as a post-processor. This is essentially DFRG performed upon a good initial partition.

Bench.	Straw	Basic DFRG	DFRG HLG	DFRG Clust.
s5378	60	40	41	41
s9234	45	34	34	32
s13207	67	54	47	63
s15850	52	45	44	45
C6288	50	33	33	33
biomed	161	136	136	107
prim2	133	114	116	121
struct	33	33	33	33
s35932	46	39	39	24
s38584	50	37	35	29
s38417	55	48	48	45
Mean	60.8	48.9	48.2	44.9
Time	17.4	27.8	31.6	48.2
Rep %	0.0%	3.5%	3.5%	5.1%

Table 3. Comparison of DFRG optimizations.

3.1 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 uncutting the net via source movements. Since source and sink motion operates independently in determining the state of a net, we use the following higher-level gains formulation: a cut net contributes +1 to the nth 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, and no locked sources, in the "source" partition; a cut net contributes a -1 to the nth level gain of a source node S if S is in the "non-source" partition, and there are n-1 unlocked sources, and no locked sources, in the source partition; Higher-level gains for sink nodes are defined similarly. For ease of implementation, only cut nets contribute to the higher level gains of a node.

3.2 Extension to Clustering

Because DFRG more than doubles the size of the network, we thought that it would benefit from reducing the problem size via clustering. Thus, we perform DFRG at each clustering level in place of the Strawman FM, and gradually expand the maximum circuit size as in the KN 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.

3.3 DFRG Results

The results for the various optimizations to the DFRG algorithm are given in Table 3. As can be seen, basic DFRG produces results 19.6% better than the non-replicating algorithm. Higher-level gains improve this by an additional 1.1% and DFRG hierarchical partitioning yields an additional 5.5%. Combined together, these techniques produce an overall 26.2% improvement over the basic Strawman algorithm.

4 Flow Based Techniques

The final 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 definitions [4] [8]:

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

A *flow* in G is a function f: V x V → (where " " is the set of reals) such that

(1) $f(u,v) \le c(u,v)$, u,v V, where c(u,v) is the capacity of the edge (u,v) E

(2) $f(u,v) = -f(v,u)$, u,v V

(3) $\sum_{v} f(u,v) = 0$, u V - {S,T}

Ford-Fulkerson max-flow/min-cut theorem [8]:

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

4.1 The Optimal Algorithm for Unconstrained Replication Size [13, 20]

Given an initial partitioning of the vertices we select a source partition arbitrarily. We form a flow network by creating a node S, which will be the source of the network, and clustering with S nodes in the source partition which we would like to act as a sources of flow in the network (see Figure 3 left). We also create a node T, which will be the sink of the network, and cluster with T all nodes which are in the non-source (sink) partition, and are attached to edges which have their source in the source partition. All edges which have their source in the sink partition are removed from the network, and all remaining edges are given a capacity of one.

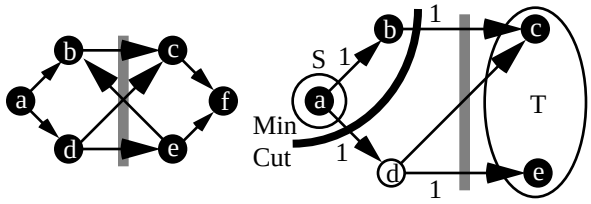

Figure 3. A directed graph (left) is turned into a flow graph. Once a maximum flow is found, which induces a minimum cut, nodes between the min cut and the original partitioning are replicated (right). The replicated node is shown in white.

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 in partition X on the sink side of the minimum cut is replicated. By replicating these nodes we place all edges in the minimum cut into the cut set, and guarantee that any other edge which has its source in the source partition is not in the cut set. The original graph is then reestablished; we select the opposite partition as the source partition, and we repeat the above. By finding the replication subset which minimizes the cut edges which have their source in each partition we find the replication set which minimizes the number of cut edges for this partitioning.

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.

4.2 Hyper-MAMC (Min-Area Min-Cut) [20]

To use the network flow formulation we must reduce hyperedges to edges. This is done as shown in Figure 4. For each hyperedge we create a hypernode with an edge of weight 1 from the source to the hypernode, and an edge

of infinite weight from the hypernode to each sink. In this way, we restrict each hyperedge to unit flow, and ensure that the hyperedge can only be cut in the edge from the source to the hypernode. This guarantees that each hyperedge contributes at most once to the minimum cut derived from the maximum flow.

Figure 4. Reduction of hyper-edges to normal edges. White node is the hyper-node.

Hyper-MAMC proceeds similarly to the optimal algorithm, pushing flow through the network and finding nodes to be replicated. However, before replicating these nodes it first checks if this replication would exceed the partition size limits. 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. The hyper-MAMC algorithm is then rerun. By doing this the chosen sink cannot be replicated, and it now acts as a source of flow. This can increase the flow, and reduce the size of the replication set (see Figure 5). This process of incrementing the flow is repeated until the the replication set size is acceptable. Once we have a sufficiently small set for replication, we dismantle the flow network, select the opposite partition, and repeat.

A peculiarity of this algorithm is that we may exceed partition size limits, yet there may be no node available to cluster with the source. This is caused by large numbers 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.

4.2.1 Node Choice for Source Clustering

When creating the flow network we must choose nodes to cluster together to create the network source. The most obvious candidates for source clustering are circuit inputs (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.

Bench.	Hyper-MAMC					
	IBUF	RANDOM				
		2.0	1.0	0.5	0.25	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
prim2	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	43.2	44.5	42.5	41.8	41.2	42.0
Time	24.7	21.7	22.5	23.4	24.3	21.9

Table 4. Effect of source choice on cut size for Hyper-MAMC. Initial partitions are generated by Strawman with a maximum partition size of 51%.

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. Table 4 details the various source selection strategies. The IBUF column uses all IBUFs in the source partition as flow sources. The other columns randomly select a number of sources equal to X * directed_cut_size,

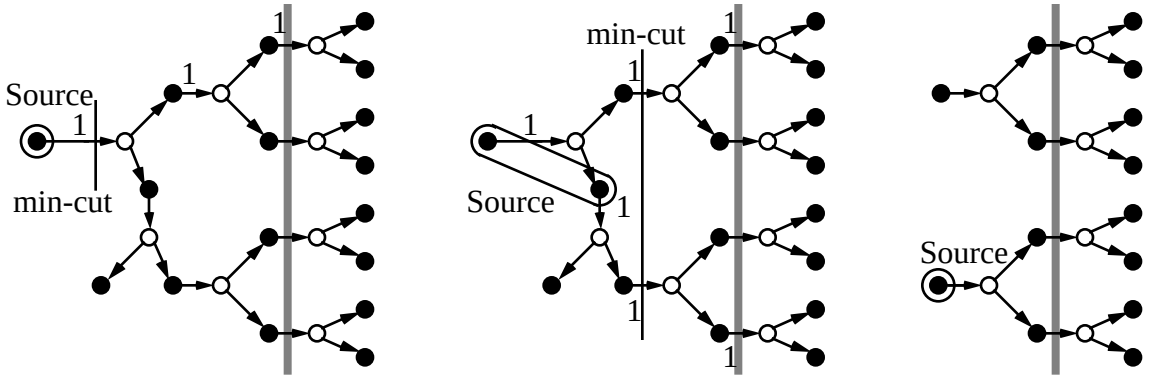

Figure 5. Given an initial flow network that replicates too much logic (left), incrementing the flow reduces the replication set and increases the cut size (center). However, nodes which cannot be reached from the source, such as those at the top of the figure at right, will always be replicated.

where X is the number listed in the column heading. Randomly selecting 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.

Bench.	Straw	Initial Partition size limit		
		49%/51%	48%/52%	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
prim2	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	60.8	41.2	39.1	43.1
Time	17.4	24.3	24.3	21.8

Table 5. Effect of initial partition size limit on Hyper-MAMC.

4.2.2 Initial Partition Variation

In move-based algorithms, size limits were expressed as maximum partition sizes, limiting both partition size imbalance and replication. Thus, with a limit of 53.6% of the logic, one partition could contain 53.6% of the logic an have no replication, or both partitions could contain 46.4% unreplicated logic and 7.2% replicated logic. This ability to trade size imbalance with replication yields an added optimization opportunity. Hyper-MAMC is a post-processing algorithm, where the partitioning of the circuit, and thus the partition size imbalance, must be fixed before replication begins. In Table 5 we explore how the limits applied to partition size imbalance affect the final results. 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 hyper-MAMC variants in this paper will use an initial partition size limit of 52%/48% circuit size.

4.2.3 Node Choice for Incrementing Flow

When faced with an large replication set, the hyper-MAMC algorithm will increment the flow by clustering a node with the network source. We can attempt to find the best node by testing each node, and determining which produces the best result. Unfortunately this exhaustive search is too time consuming, and therefore the hyper-MAMC algorithm includes a threshold parameter for use with node choice [19]. 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 use that node which produced the best results (least increase in max-flow, with ties broken by the largest decrease in total size of nodes replicated). If we have more than a threshold number of nodes to choose from we choose arbitrarily.

As shown in Table 6, small threshold values have little impact on partition quality. Apparently there are few occasions where the node choice falls below the threshold value on these benchmarks. Unfortunately, a higher threshold value tends to greatly slow the algorithm. To avoid this performance penalty, we decided to select the initial partition (produced by Strawman) which produced the best final cut size after performing hyper-MAMC with a zero size threshold, and then run a high threshold hyper-MAMC on only this initial partition. This amortizes the time cost of a high threshold hyper-MAMC run over all thirty iterations of hyper-MAMC.

Bench.	Straw	Threshold		Amortized			Random	MC-Rep
		0	25	50	100	200	5	
s5378	60	40	40	40	40	40	39	44
s9234	45	31	31	31	28	28	28	37
s13207	67	44	44	44	44	44	42	61
s15850	52	38	38	38	38	38	38	50
C6288	50	33	33	33	33	33	33	33
biomed	161	61	61	61	58	58	58	127
prim2	133	113	113	113	108	108	106	121
struct	33	33	33	33	33	33	33	33
s35932	46	23	23	23	23	23	23	23
s38584	50	25	25	25	25	25	25	25
s38417	55	36	36	36	36	36	34	44
Mean	60.8	39.1	39.1	39.1	38.4	38.4	37.8	46.3
Time	17.4	21.9	37.6	26.3	28.1	32.0	59.7	21.7

Table 6. Effect of node choice for incrementing flow on partition quality. Threshold columns try all node choices when the number of choices is less than the threshold, and otherwise randomly pick a single node. Amortized columns run threshold 0, pick the best of 30 runs, and then uses the listed threshold on just that best partitioning. Random 5 always randomly tries 5 nodes each time the flow must be incremented. MC-Rep results are also included.

Replication Algorithm	Cutsize	Improvement over Straw	Improvement over Basic Alg.	Time (minutes, 30 runs)
Strawman	60.8	---	---	17.4
KN	48.2	20.7%	---	17.3
higher-level gains	47.7	21.5%	1.0%	17.6
clustering	47.6	21.7%	1.2%	17.1
gradient	44.7	26.5%	7.3%	19.7
DFRG	48.9	19.6%	---	27.8
higher-level gains	48.2	20.7%	1.4%	31.6
clustering	44.9	26.2%	8.2%	48.2
Hyper-MAMC	43.2	28.9%	---	24.7
flow source node choice	41.2	32.2%	4.6%	24.3
partition variation	39.1	35.7%	9.5%	24.3
flow incr. node choice	37.8	37.8%	12.5%	59.7
MC-Rep	46.3	23.8%	---	21.7

Table 7. Summary of optimizations. Basic algorithms are shown in bold, followed by the new optimizations in this paper. "Improvement over Strawman" represents how much each replication technique improves over a non-replicated bipartitioning. "Improvement over Basic Algorithm" is how much better the optimized version does as compared to the unoptimized replication technique (i.e. the version taken directly from the literature). All algorithms are allowed to generate partitions containing at most 53.6% of the logic through both bipartitioning imbalance and replication.

An alternative to a threshold function is to instead always randomly choose a small number of nodes to test whenever we must increment the flow. In this way we always make a (somewhat) informed decision about what node to add to the source set. Although this greatly increases the processing time, we reduce this time by also amortizing the cost, only optimizing the best of the 30 runs as determined by threshold 0 runs. This techniques is used in the "Random 5" column of Table 6.

4.2.4 Hyper-MAMC Replication Results

The results for the various optimizations to the Hyper-MAMC algorithm are given in Table 4 - Table 6. As can be seen, the choice of flow sources can make a significant difference, with randomly picking 1/4 as many sources as the cutsize of the initial partitioning producing a 4.6% improvement over using the circuit's IBUFs. Balancing the amount of flexibility given to the partitioner verses the amount of replication allowed is also an issue, providing up to a 9.3% difference. Finally, always evaluating five random nodes to add to the source when incrementing the flow is important, providing a 3.3% improvement over just randomly adding nodes. However, this does almost triple the runtime, even though this technique is only applied to the most promising of the 30 partitioning attempts. Combining these approaches produces cutsizes 37.8% smaller than the unreplicated version, significantly better than any other technique.

4.3 MC-Rep [13]

MC-Rep is a simpler version of Hyper-MAMC. Just as in Hyper-MAMC it uses the minimum cut in a flow network to form the replication set. However, unlike Hyper-MAMC, if the the replication set is too large we 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. We only allow replication and unreplication moves, and only by nodes which were in the selected partition. In this way, a replication set is derived that meets the size limit. We then select the opposite partition and repeat.

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. As shown in Table 6, MC-Rep performs significantly worse than Hyper-MAMC, which is able to stay closer to the basic network flow formulation for unbounded size replication.

6 Conclusion

We have investigated replication, seeking to develop the best possible bipartitioning results. In the process we were able to reduce cutsizes by 37.8% over Strawman's already impressive results. Also, these improvements are obtained with relatively little logic duplication, with most techniques replicating less than 5% of the logic.

The replication techniques in the literature are not so much partitioning algorithms in their own right, but are optimizations to be added into an existing partitioner. In order to perform a comparison we have implemented most of the replication algorithms from the literature within our Strawman system. Thus, we can directly compare all of the algorithms on the same benchmarks, with the same size constraints, using the same partitioner. The only difference is the replication techniques. This comparison is shown in Table 7, which includes all of the algorithms discussed in this paper. Results are included for comparing the optimized replication techniques against non-replicating Strawman, as well as the new optimized replication techniques proposed in this paper

against the basic versions from the literature. As can be seen, our optimized algorithms achieve a 12.5% to 13.5% improvement over the existing techniques even factoring out the benefits of our Strawman bipartitioner.

Achieving these high-quality results has required innovative optimizations to the individual algorithms. To improve the KN algorithm we extended it to take advantage of higher-level gains and recursive clustering. We also developed the Gradient method to delay replication until the latter stages of partitioning, avoiding ignorant replication moves. Combined together, this improves the results by 7.3%. To improve the DFRG algorithm we developed a directed graph partitioning formulation for higher-level gains and extended the algorithm to work under recursive clustering. This achieved a 8.2% improvement over the basic DFRG algorithm. Finally, we improved the Hyper-MAMC algorithm by carefully considering which nodes should be used as flow network sources, by varying the amount of size imbalance allowed for bipartitioning verses replication, and by developing methods for deciding which nodes to add as flow sources, and thus reduce the replication set. This provides a 12.5% improvement over the basic Hyper-MAMC algorithm, and generates the best results for any single replication technique.

Our investigations of Functional Replication [16] do not appear here because of space limitations. It produces only a 6% improvement over Strawman, making it less interesting than the other approaches.

As has been demonstrated in this paper, achieving the greatest possible partitioning quality requires the careful consideration not only of individual techniques, but also how these techniques fit together. Optimizations such as higher-level gains, multiple random initializations, and recursive clustering may be key to achieving the best standard bipartitioning results, but integrating replication techniques into such a system can cause significant added concerns. Also, subtle issues such as node choice for flow sources and incrementing flows can easily be overlooked, yet can provide significant quality improvements. Thus we believe that as we continue to add to the repertoire of replication techniques, we must also consider how these techniques can be added to what is already known to produce the best overall algorithm.

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] C. J. Alpert, S.-Z. Yao, "Spectral Partitioning: The More Eigenvectors, The Better", *Design Automation Conference*, pp. 195-200, 1995.

[3] C. K. Cheng, T. C. Hu, "Maximum Concurrent Flow and Minimum Ratio-cut", *Technical Report CS88-141*, University of California, San Diego, December, 1988.

[4] T. H. Cormen, C.E. Leiserson, R.L. Rivest, *Introduction to Algorithms*, MIT Press/McGraw-Hill, Cambridge, MA, 1990.

[5] S. Dutt, W. Deng, "A Probability-Based Approach to VLSI Circuit Partitioning", *Design Automation Conference*, pp. 100-105, 1996.

[6] M. Enos, M.S., "Replication for Logic Bipartitioning", Master's Thesis, Northwestern University, Department of ECE, 1996.

[7] C. M. Fiduccia, R. M. Mattheyses, "A Linear-Time Heuristic for Improved Network Partitions", *Design Automation Conference*, pp. 241-247, 1982.

[8] L.R. Ford, D.R. Fulkerson, *Flows in Networks*, Princeton Univ. Press, Princeton, NJ, 1962.

[9] M. Garey, D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, San Francisco, CA: Freeman, 1979.

[10] 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.

[11] S. A. Hauck, *Multi-FPGA Systems*, Ph.D. dissertation, University of Washington, Dept. of CSE., pp. 131-168, 1995.

[12] S. Hauck, G. Borriello, "An Evaluation of Bipartitioning Techniques", *Chapel Hill Conference on Advanced Research in VLSI*, pp. 383-402, March, 1995.

[13] L. J. Hwang, A. El Gamal, "Min-Cut Replication in Partitioned Networks", *IEEE Transactions on Computer-Aided Design*, Vol. 14, No. 1, pp. 96-106, Jan 1995.

[14] C. Kring, A. R. Newton, "A Cell-Replicating Approach to Mincut-Based Circuit Partitioning", *International Conference on Computer-Aided Design*, pp. 2-5, 1991.

[15] 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.

[16] 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.

[17] L. Liu, M. Kuo, C. K. 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.

[18] G. M. Riess, K. Doll, F. M. Johannes, "Partitioning Very Large Circuits Using Analytical Placement Techniques", *Design Automation Conference*, pp. 646-651, 1994.

[19] H. H. Yang, D. F. Wong, "Efficient Network Flow Based Min-Cut Balanced Partitioning", *International Conference on Computer-Aided Design*, pp. 50-55, 1994.

[20] 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.