Multi-Resource Aware Partitioning Algorithms for FPGAs with Heterogeneous Resources *

Navaratnasothie Selvakkumaran
Department of Computer Science and Engineering, Digital Technology Center and Army HPC Research Center, University of Minnesota, Minneapolis
selva@cs.umn.edu

Abhishek Ranjan
HierDesign Inc
ranjan@hierdesign.com

Salil Raje
HierDesign Inc
salil@hierdesign.com

George Karypis
Department of Computer Science and Engineering, Digital Technology Center and Army HPC Research Center, University of Minnesota, Minneapolis
karypis@cs.umn.edu

ABSTRACT

As FPGA densities increase, partitioning-based FPGA placement approaches are becoming increasingly important as they can be used to provide high-quality and computationally scalable solutions. However, modern FPGA architectures incorporate heterogeneous resources, which place additional requirements on the partitioning algorithms because they now need to not only minimize the cut and balance the partitions, but also they must ensure that none of the resources in each partition is oversubscribed. In this paper, we present a number of multilevel multi-resource partitioning algorithms that are guaranteed to produce solutions that balance the utilization of the different resources across the partitions. We evaluate our algorithms on twelve industrial benchmarks ranging in size from 5,236 to 140,118 vertices and show that they achieve minimal degradation in the min-cut while balancing the various resources. Comparing the quality of the solution produced by some of our algorithms against that produced by hMETIS, we show that our algorithms are capable of balancing the different resources while incurring only a 3.3%–5.7% higher cut.

Categories and Subject Descriptors

B.7.2 [**Integrated Circuits**]: Design Aids

*This work was supported in part by NSF CCR-9972519, EIA-9986042, ACI-9982274, ACI-0133464, and ACI-0312828; the Digital Technology Center at the University of Minnesota; and by the Army High Performance Computing Research Center (AHPCRC) under the auspices of the Department of the Army, Army Research Laboratory (ARL) under Cooperative Agreement number DAAD19-01-2-0014. The content of which does not necessarily reflect the position or the policy of the government, and no official endorsement should be inferred. Access to research and computing facilities was provided by the Digital Technology Center and the Minnesota Supercomputing Institute.

.

General Terms

Algorithms, Experimentation

Keywords

Partitioning, Placement, multi-constraint, multi-resource, FPGA

1. INTRODUCTION

In recent years, due to the development of high-quality multilevel hypergraph partitioning algorithms [9, 2], partitioning-based placement has emerged as a promising approach for placing large designs on ASICs. These methods have been shown to be computationally scalable, capable of leading to high-quality solutions, and scale to very large designs [13, 1]. Moreover, as FPGA densities increase, the characteristics of this placement methodology are becoming increasingly important for placing large designs on FPGAs, as well [12].

However, unlike ASICs that are in general homogeneous, and as such, the only constraint that they impose on the partitioning algorithm is that of balancing the area of the cells assigned to the different partitions, modern FPGA architectures incorporate heterogeneous resources (*e.g.*, CLBs, Multipliers, RAM blocks, IP Cores [16], *etc*). This places additional constraints on the type of partitionings that need to be computed, as the partitioning algorithm must now ensure that the resources used in each partition can be accommodated by the resources provided at the different regions of the FPGA. For example, a partitioning solution that places most of the FFs on one side of the bisection and most of the RAM blocks on the other side of the bisection, even if it is balanced in terms of the total number of cells on either side of the cut, it is not very useful for FPGA placement as it may over-subscribe these two resource types.

As a result, existing partitioning algorithms [9, 2, 4, 14, 7] can not be used to develop partitioning-based placement methods for FPGAs with heterogeneous resources, as they can lead to partitionings that have highly unbalanced resource requirements. To illustrate this, we used a multilevel hypergraph partitioning algorithm (hMETIS [10]) to bisect twelve different circuits synthesized for the Xilinx Vertex II architecture, which contain cells that map to different resources. Various statistics measuring the balance of the different resource types are shown in Table 1. These results show that even though the bisection, in terms of the number of cells assigned to each partition, achieves a balance of 49%-51%, in general, individual resources are considerably more unbalanced.

In this paper, we present a new class of *multi-resource hypergraph bisectioning algorithms* that are capable of producing a parti-

		value of $\leq$ 2.0 required			
	# types	min ub	max ub	avg ub	# viol
ind1	11	0.4	10.3	4.4	6
ind2	9	0.6	9.5	4.8	6
ind3	11	0.9	27.1	6.4	7
ind4	12	0.8	81.5	10.6	9
ind5	11	0.8	16.6	5.8	7
ind6	11	0.5	13.8	4.3	5
ind7	11	0.7	11.0	3.0	3
ind8	12	0.7	7.6	2.6	4
ind9	11	0.9	33.2	5.3	6
ind10	5	0.8	3.1	1.6	1
ind11	11	0.8	11.1	3.3	4
ind12	11	1.2	30.9	5.6	8

Table 1: The distribution of unbalance factors of different types of cells, for 49%-51% bisection. For the partition to be feasible, unbalance factor of each cell-type must be below 2.0. The column "min ub" shows the minimum unbalance factor, "max ub" shows the maximum unbalance factor, "avg ub" shows average unbalance factor, and "# viol" shows the number of cell-types in violation by exceeding the unbalance factor of 2.0.

tioning solution that simultaneously balance the different resources assigned to each one of the partitions, and thus can be used to power partitioning-based placement methodologies for emerging FPGA architectures. Specifically, we present five different multi-resource partitioning algorithms that are based on the multilevel hypergraph partitioning paradigm. Three of these algorithms solve the problem by balancing the different resources at the same time that they compute the bisection, while the other two are used to post-process a high-quality but potentially unbalanced solution to enforce the multiple balancing constraints. We experimentally evaluated the performance of these algorithms on twelve different industrial circuits containing up to 140,118 cells. Our results show that each one of these algorithms is capable of producing solutions that satisfy the multiple balancing constraints and achieve different time-quality trade-offs. Moreover, comparing the quality of the solution produced by some of our algorithms against that produced by hMETIS, we show that our algorithms are capable of balancing the different resources while incurring only a 3.3%–5.7% higher cut.

The rest of the paper is organized as follows. Section 2 defines various concepts and terms that are used in the paper and present a brief overview of the multilevel partitioning paradigm. Section 3 provides a formal definition of the multi-resource partitioning problem. Section 4 describes the various multi-resource partitioning algorithms that we developed. Section 5 present a comprehensive experimental evaluation of these algorithms. Finally, Section 6 provides some concluding remarks.

2. NOTATION AND BACKGROUND

A *hypergraph* $G = (V, E)$ is a set of vertices V and a set of hyperedges E. Each hyperedge is a subset of the set of vertices V. The *size* of a hyperedge is the cardinality of this subset. A vertex v is said to be *incident* on a hyperedge e, if $v \in e$. Each vertex v and hyperedge e has a weight associated with them and they are denoted by $w(v)$ and $w(e)$, respectively. A circuit/netlist consisting of a set of cells and a set of nets can be directly represented via a hypergraph, whose vertices corresponds to the cells and whose hyperedges corresponds to the nets. Due to this one-to-one correspondence between hypergraphs and netlists we will use the terms vertices/cells and hyperedges/nets interchangeably throughout this paper.

A bisection of V is denoted by a vector P such that $P[i]$ indicates the partition number that vertex i belongs to. The *cut* of the bisection is equal to the sum of the weight of the hyperedges that connect vertices belonging to different partitions. We say that a bisection P of V *satisfies* a *single balancing constraint* specified by $[l, u]$, where $l < u$, iff $l \leq \sum_{v \in V_i} w(v) \leq u$, for each partition V_i. A bisection that satisfies the constraint is called *feasible*, otherwise it is *infeasible*. Given these definitions, the hypergraph bisection problem is formally defined as follows: *Given a hypergraph $G(= V, E)$ and a balancing constraint $[l, u]$, find a feasible bisection P of G that minimizes the cut.* Since there is only a single balancing requirement, this formulation is usually referred to as the single-constraint bisectioning problem [5].

3. PROBLEM DEFINITION

Historically, FPGA devices contained single type of resource (CLBs for example) that were uniformly distributed throughout the chip. However, taking advantage of ever-increasing silicon densities, modern FPGA devices contain multiple types of resources, which allow them to efficiently implement complex and high performance designs. One such example is the recently introduced Virtex II architecture from Xilinx that contains specialized resources such as multiplier and RAM blocks interspersed among CLBs. As a result, designs created for such modern FPGAs try to pro actively make use of these specialized resources in order to obtain better performance and versatility.

For partitioning driven placement to succeed in utilizing these different resource types, the partitioning algorithms need to take them into account and balance each type of cells across the cut lines. Motivated by this observation we focus on multi-resource aware partitioning, which can be formally defined as follows. Consider an FPGA architecture with m distinct resource types and let ${cl_i}^j$ denote the minimum number of resources of type i allowed in partition j, and ${cu_i}^j$ be the maximum number of resources of type i allowed in partition j. Then the multi-resource bisection P of G seeks to minimize the cut subject to:

$$ {cl_i}^j \leq \sum_{\forall v \in V : P[v]=1 \ and \ t(v)=i} 1 \leq {cu_i}^j $$

for $j = 1, 2$, $i = 1, 2, \ldots, m$, and $t(v)$ is the resource type required by cell v. Note that this is a general definition of the multi-resource bisection and only the upper bound is usually needed in most cases. Furthermore, when the number of cells of a certain type are small and an odd number, it sometimes makes it impossible to satisfy the balance constraint. In such cases the balance constraint needs to be relaxed. For example, if there are only 3 cells of a certain type present, then balance constraint of 49%-51% is impossible to satisfy and needs to be relaxed to 33% - 67% to accomodate them.

4. MULTI-RESOURCE PARTITIONING ALGORITHMS FOR FPGAS

To solve the multi-resource bisectioning problem we developed two classes of multi-resource partitioning algorithms. The first class, computes the overall solution by constructing a bisection that simultaneously balances the multiple resources, whereas the second class, achieves the desired balance by modifying a bisection that was initially obtained using a traditional single-constraint bisectioning algorithm. We will refer to the first class as the *native multi-resource partitioning* algorithms and to the second class as the *multi-resource enforcement* algorithms. The details of the various algorithms in each of these classes are provided in the rest of this section.

4.1 Native Multi-Resource Partitioning Algorithms

We developed three different algorithms, called *multi-phase*, *multi-constraint*, and *multi-phase–multi-constraint* that are capable of directly computing a partitioning that balances the different resources. These algorithms were motivated by recently developed graph partitioning algorithms for partitioning finite element meshes arising in multi-phase and multi-physics scientific numerical simulations [11, 3]. Specifically, our multi-phase algorithm is based on the graph partitioning algorithm proposed in [3], our *multi-constraint* algorithm is based on the graph-partitioning algorithm proposed in [11], whereas the multi-phase–multi-constraint algorithm combines elements from both of these approaches. Details on these algorithms are provided in the remainder of this section.

4.1.1 Multi-Phase Bisection (MP)

The basic idea of this algorithm is very simple. First we construct a series of hypergraphs containing cells of type 1 (H_1), cells of type 1 and 2(H_2), cells of type 1,2 and 3 (H_3), and so on. The hyperedges for these sub hypergraphs are reconstructed based on the information from the original hypergraph. After that, hMETIS is used to obtain a partition of H_1. Now using the partition information of H_1, we can easily assign partitions for cells of type 1 in H_2. To obtain the bisection of type 2 cells of H_2, we fix the cells of type 1 (also set the area as zero) and use hMETIS as usual which generates the partition information for cells of type 2. Now partition information for cells of type 1 and cells of type 2 are available. This partitioning also satisfies the balance constraints for both types due to the fact the balance constraint of type 1 was preserved since they were fixed vertices and the balance constraint of the type 2 cells were satisfied hMETIS. (because area of type 1 cells were set to zero). We continue this process by influencing the partitioning of H_3 by incorporating partition information of cell types 1 and 2 from H_2. Next, we handle H_4 by using partition information from H_3 and so on.

Since it is easier to influence the bisection of smaller subset of cells from the partition information of larger subset of cells, we reorder the types such that the number of cells of type 1 are the most, type 2 second most and so on.

4.1.2 Multi-Constraint Bisection (MC)

The multi-resource partitioning problem can be naturally solved using the multi-constraint partitioning problem initially developed in the context of graphs. Specifically, using the general framework introduced in [11], we extend the hypergraph model so that each vertex v has a weight vector $\boldsymbol{w}(v)$ of size m associated with it. The ith component of this vector $w_i(v)$ corresponds to the weight associated with the ith constraint. This model assumes, without loss of generality, that the weight vectors of the vertices satisfy the property that $\sum_{\forall v \in V} w_i(v) = 1.0$ for $i = 1, 2, \ldots, m$. Using a framework analogous to that used for single-constraint problems, we allow for m lower- and upper-bound constraints on the size of each partition (l_i, u_i) for $i = 1, 2, \ldots, m$, such that $0 < l_i < u_i$ and $l_i + u_i = 1$. Given these definitions, the multi-constraint hypergraph bisection problem is formally defined as follows:

Compute a bisection P of V that minimizes the sum of the weight of the hyperedges that span multiple partitions subject to the constraint that

$$l_i \leq \sum_{\forall v \in V : P[v] = j} w_i(v) \leq u_i,$$

where $j = 1, 2$ and $i = 1, 2, \ldots, m$ represent the different vertex weights. This multi-constraint partitioning problem tries to find a bisection such that each weight is individually balanced within the specified lower- and upper-bound tolerances.

Using this multi-constraint partitioning problem formulation the multi-resource partitioning problem can be formulated as follows. Given a multi-resource hypergraph $G = (V, E)$ with m different vertex types, then each vertex $v \in V$ is assigned a vector of m vertex weights $\boldsymbol{w}(v)$, such that $w_{t(v)}[v] = 1$ and $\forall i \neq t(v) w_i(v) = 0$. It is easy to see that a feasible multi-constraint solution of this hypergraph will correspond to a feasible solution for the multi-resource partitioning problem, as well.

We have developed a multi-constraint hypergraph partitioning algorithm that follows the traditional structure of the multilevel partitioning paradigm. Specifically, we developed algorithms for the coarsening, initial partitioning, and uncoarsening phases that combine elements of the single-constraint hypergraph partitioning algorithms in hMETIS with the multi-constraint extensions, initially introduced for graph partitioning [11]. Due to space constraints, in this paper we will only describe the multi-constraint partitioning refinement algorithm used during the uncoarsening phase as it is an integral part in many of the approaches presented in this paper. The interested readers should refer to [11, 8, 5] for further details.

Multi-constraint Refinement (MC-FM). We developed a multi-constraint bisection refinement algorithm, called MC-FM, which is based on the widely used single-constraint FM algorithm [6] and operates as follows. For each one of the two partitions, it maintains m priority queues, where m is the number of weights. A vertex belongs to only a single priority queue depending on the relative order of the weights in its weight vector. In particular, a vertex v with weight vector $(w_1(v), w_2(v), \ldots, w_m(v))$, belongs to the jth queue if $w_j(v) = \max_i(w_i(v))$. Given these $2m$ queues, the algorithm starts by initially inserting all the vertices to the appropriate queues according to their gains. Then, it proceeds by selecting one of these $2m$ queues, picking the highest gain vertex from this queue, and moving it to the other partition. The queue is selected as follows. If the current bisection represents a feasible solution, then the queue that contains the highest gain vertex among the $2m$ vertices at the top of the priority queues is selected. On the other hand, if the current bisection is infeasible, then the queue is selected depending on the relative weights of the two partitions. Specifically, if A and B are the two partitions, then the algorithm selects the queue corresponding to the largest $w_i(x)$ with $x \in \{A, B\}$ and $i = 1, 2, \ldots, m$. If it happens that the selected queue is empty, then the algorithm selects a vertex from the non-empty queue corresponding to the next heaviest weight *of the same partition*. For example, if $m = 3$, $(w_1(A), w_2(A), w_3(A)) = (.43, .60, .52)$, and $(w_1(B), w_2(B), w_3(B)) = (.57, .4, .48)$, the algorithm will select the second queue of partition A. If this queue is empty, it will then try the third queue of A, followed by the first queue of A. Note that we give preference to the third queue of A as opposed to the first queue of B, even though B has more of the first weight than A does of the third. This is because our goal is to reduce the second weight of A. If the second queue of A is non-empty, we will select the highest gain vertex from that queue and move it to B. However, if this queue is empty, we still will like to decrease the second weight of A, and the only way to do that is to move a node from A to B. This is why when our first-choice queue is empty, we then select the most promising node from the same partition that this first-queue belongs to.

4.1.3 Multi-Phase Multi-Constraint (MPMC)

This algorithm incorporates the features of both multi-phase bisection and multi-constraint bisection. The general structure is similar to that of Section 4.1.1, but when constructing the sub hypergraphs (H_1, H_2 .. H_m), it also incorporates pseudo hyperedges to retain the information of the original hypergraph more accurately and also to prevent these sub hypergraphs from becoming sparser and result in disconnected segments. This problem is especially severe when numerous constraints are present and results in highly disconnected H_1. Bisection of this trivial hypergraph H_1 may not correspond well with min-cut bisection of the original hypergraph.

Adding pseudo hyperedges is done in the following way. When a vertex is removed, its neighbors are analyzed to determine how closely each neighbor is connected to the removed vertex. If the connectivity is larger than 10% of average hyperedge weight, then these neighbors are considered to be connected to the removed vertex and are connected by a light weight pseudo hyperedge. The connectivity to neighbors is estimated by representing each hyperedge by a clique of edges each with the weight of $w(e)/(|e| - 1)$ and by summing the weights of edges common to each neighbor and the removed vertex. The pseudo hyperedges introduced do not participate in estimating connectivity. These settings work very well for our purpose as evident in Section 5 but may require fine tuning depending on the application.

In addition to the above process, we also apply MC-FM for each of the sub hypergraphs containing more than one type ($H_2..H_m$). This allows previously fixed cells to become free and move, which often results in substantial improvement.

4.2 Multi-Resource Enforcement Algorithms

In analyzing the characteristics of the various multi-resource circuits we discovered that the different types of vertices are reasonably well-distributed throughout the underlying hypergraph. This suggests that the bisections produced by single-constraint partitioning algorithms, even though they will not be perfectly balanced, they will not be arbitrarily unbalanced either. Moreover, since these partitionings can be computed using state-of-the-art multilevel schemes, they will have small cuts. Motivated by this observation, we developed two schemes that take as input a min-cut single constraint partitioning and try to enforce the various multi-resource balanced constraints.

4.2.1 Single-Constraint Direct-Balancing (SCDB)

In this method, we use the multilevel single-constraint partitioner hMETIS to seed the initial bisection. Then we use an explicit balancing algorithm to balance the multiple resources in a single step. This multi-constraint balancing algorithm operates very similar to MC-FM (described in Section 4.1.2), except that it gives priority to finding a balanced bisection rather than minimizing cut. This balancing step tends to increase the cut, especially when the number of constraints is large. Hence, it is imperative to apply multi-constraint refinement algorithms after obtaining a feasible bisection. Therefore, a single iteration of MC-FM is applied in an effort to improve the cut quality after obtaining a feasible bisection.

4.2.2 Single-Constraint Multi-Phase Balancing (SCMB)

As in the previous algorithm (Section 4.2.1), we use hMETIS to obtain an initial solution and then fix all the cells of the types that satisfy the balancing constraints. For the unbalanced types, we order them from least unbalanced to most unbalanced, and then bisect each of them in the way described in Section 4.1.1. After each unbalanced type is balanced we also apply an iteration of MC-FM to capitalize on the perturbation caused during balancing.

4.3 Additional Improvements

After the bisection of the original hypergraph has been computed, it is possible to further improve the cut by applying a multi-constraint V-cycle. Multi-Constraint V-cycle consists of two components, *restricted multi-constraint coarsening* and multi-constraint refinement. The restricted multi-constraint coarsening step differs from regular multi-constraint coarsening by the presence of an additional requirement that any two vertices that are collapsed together belong to the same partition. The information regarding the partitioning is thus preserved during the creation of successive approximate hypergraphs. This coarsening scheme is a multi-constraint version of restricted coarsening presented in [9]. The second component is same as the multi-constraint refinement presented in Section 4.1.2.

	# cells	# nets	# types	No. of cells of various types min	max	avg
ind1	18160	17689	11	1	8138	1651
ind2	5236	4874	9	3	2584	582
ind3	15783	16272	11	14	5889	1435
ind4	58571	60734	12	6	22193	4881
ind5	89697	91925	11	9	45305	8154
ind6	56462	57674	11	3	26759	5133
ind7	119407	121822	11	5	55873	10855
ind8	136539	139147	12	1	73106	11378
ind9	109115	111776	11	4	54377	9920
ind10	72130	49594	5	58	42789	14426
ind11	92778	93184	11	1	46577	8434
ind12	140118	141505	11	4	76887	12738

Table 2: The characteristics of netlists used for evaluating algorithms

5. EXPERIMENTS

We experimentally evaluated our multi-resource aware partitioning algorithms on an industrial benchmark suite consisting of twelve large designs synthesized for Virtex II architecture [15]. The types of cells consist of sub CLB elements such as LUTs, FFs, MUXes, control gates and non CLB elements such as RAM Blocks, DCMs, IOBs etc. The details of these benchmarks are listed in Table 2. The column labeled as "# types" shows the number of distinct types of cells available on that particular benchmark. The columns labeled as "min" shows minimum number of cells of any type for that benchmark, and similarly the "max" and "avg" columns provide the details of distribution of number of cells in each hypergraph.

To evaluate the quality of the solutions obtained by the various multi-resource partitioning algorithms, we used hMETIS (version 1.5.3 [10]) to obtain single-constraint bisections of the different hypergraphs. These solutions were obtained using hMETIS's default parameters (including V-cycle at the end). Furthermore, to make such quality comparisons easier, we computed the Average Ratio of Quality (ARQ) of each algorithm against that obtained by hMETIS. To ensure the meaningful averaging of these ratios, we first took the $\log_2$-values of these ratios, then calculated their mean μ, and then used 2^μ as their average. This method ensures that ratios corresponding to comparable degradations or improvements (*i.e.*, ratios that are less than or greater than one) are given equal importance. The ARQ number larger than 1.0 indicates degradation in quality.

To ensure the statistical significance of our experimental results, for both hMETIS and each one of the five multi-resource partitioning algorithms we report average min-cut of ten runs.

5.1 Comparison of Native Algorithms

Tables 3 and 4 show the results obtained by the various native multi-resource partitioning algorithms (described in Section 4.1)

	hMETIS	Without V-cycle			With V-cycle		
		MP	MC	MPMC	MP	MC	MPMC
ind1	246	987	378	403	426	346	388
ind2	149	349	181	149	144	173	129
ind3	101	908	224	169	908	224	169
ind4	153	4012	405	446	508	376	336
ind5	717	2188	1133	1053	1221	1058	1039
ind6	809	2615	1649	1038	2548	1649	1038
ind7	1021	4126	1187	1234	957	1081	1151
ind8	400	4076	682	921	707	568	734
ind9	1392	4937	1577	1832	1651	1491	1656
ind10	480	719	528	550	505	498	528
ind11	373	1311	545	582	730	504	570
ind12	409	1300	636	533	744	576	531
ARQ	1.000	4.406	1.554	1.500	1.882	1.448	1.386
Time	1.000	0.230	0.577	2.496	2.360	1.760	5.206

Table 3: Performance of algorithms as an average of 10 runs for 49%-51% balance constraint.

	hMETIS	Without V-cycle			With V-cycle		
		MP	MC	MPMC	MP	MC	MPMC
ind1	213	940	261	375	337	243	355
ind2	147	316	152	123	103	141	114
ind3	85	922	126	177	128	110	110
ind4	127	3910	217	241	184	171	149
ind5	634	2242	779	943	813	739	883
ind6	822	2390	924	1022	841	871	932
ind7	917	4376	983	1167	849	873	1059
ind8	430	3781	558	711	431	502	425
ind9	1289	4052	1449	1454	1371	1367	1326
ind10	360	543	429	391	376	399	377
ind11	193	1053	271	237	240	247	236
ind12	307	1334	375	440	366	361	413
ARQ	1.000	4.811	1.246	1.383	1.141	1.136	1.165
Time	1.000	0.255	0.636	2.667	1.863	1.806	5.015

Table 4: Performance of algorithms as an average of 10 runs for 45%-55% balance constraint.

for 49%–51% and 45%–55% balance, respectively. Each of these tables shows the average minimum cuts obtained by the MP, MC, and MPMC multi-resource partitioning algorithms under two different scenarios. In the first scenario, the solution obtained by these algorithms was kept as it was, whereas in the second scenario, the solution was further refined by performing a V-cycle refinement step (as discussed in Section 4.3).

The columns labeled "hMETIS" show the average min-cut obtained by hMETIS for either 49%–51% or 45%–55% balance. Note that hMETIS's bisections will not necessarily solve the multi-resource problem, as they do not account for the different vertex types.

Finally, the rows labeled "ARQ" provides the average ratio of quality of each algorithm to hMETIS's results (computed using the scheme described in the previous section), and the rows labeled "Time" shows the amount of time required by the multi-resource partitioning algorithms relative to that required by hMETIS. Numbers less than one represent runtimes that are smaller than that of hMETIS, whereas numbers greater than one represent higher runtimes.

Comparing the results in these tables we can see that all schemes produce solutions whose cuts are worse than those produced by hMETIS. This should not be surprising, as hMETIS solves the single-constraint bisectioning problem which, in general, does not solve the multi-resource partitioning problem.

Comparing the solutions produced by the various multi-resource partitioning algorithms we can see that there is a considerable amount of variability on the quality of the final solutions. In particular, in

	hMETIS	Without V-cycle		With V-cycle	
		SCDB	SCMB	SCDB	SCMB
ind1	246	265	251	260	238
ind2	149	161	165	160	162
ind3	101	125	124	125	124
ind4	153	230	251	226	251
ind5	717	1340	868	799	864
ind6	809	880	827	879	827
ind7	1021	998	1056	997	1048
ind8	400	488	411	472	394
ind9	1392	1463	1439	1456	1438
ind10	480	491	488	489	486
ind11	373	414	374	403	213
ind12	409	499	503	494	503
ARQ	1.000	1.184	1.119	1.123	1.057
Time	1.000	1.075	1.845	1.898	2.945

Table 5: Performance of algorithms combined with multi-constraint V-cycle as an average 10 runs for 49%-51% balance factor.

the absence of V-cycle refinement, the quality of the solutions produced by MP are significantly worse than those produced by either MC or MPMC. On the average, the 49%–51% cuts produced by MP are 4.4 times worse than those produced by the single-constraint hMETIS, whereas the cuts produced by MC and MPMC are only 55.4% and 50% worse than hMETIS's cuts, respectively. Similar trends can be also observed for the 45%–55% cuts, as well. These results illustrate that the multi-constraint algorithm (MC) and the modifications to the multi-phase partitioning algorithm implemented in the MPMC algorithm, lead to superior solutions.

Comparing the results without and with V-cycle refinement we see that the overall quality of all three algorithms improves by using V-cycle refinement. However, the overall rate of improvement is different for different schemes. The MP algorithm gains the most, whereas the MPMC algorithm gains the least. We believe that the reason for that is the fact that the solutions of MC and MPMC are already of reasonable high quality, and thus, there is relatively little room for improvement. However, because MP's initial solution is considerably worse, by applying a V-cycle refinement, we can achieve dramatic quality improvements. As a result, the 49%–51% solution for MP now becomes only 88.2% worse than that of hMETIS.

Finally, comparing MC with MPMC we can see that the latter leads to consistently better solutions, which are on the average 5%–10% better than those obtained by MC. However, this quality advantage comes at the expense of higher computational requirements. In general, MPMC requires 2.5 to 5.0 times more time than that required by MC. Note that the reason that the runtimes of MP and MC without V-cycle are in general smaller than that of hMETIS is because hMETIS does perform a V-cycle refinement at the end.

5.2 Comparison of Enforcement Algorithms

Tables 5 and 6 show the results obtained by the various enforcement-based multi-resource partitioning algorithms (described in Section 4.2) for 49%–51% and 45%–55% balance, respectively. Each of these tables shows the average minimum cuts obtained by the SCDB and SCMB partitioning algorithms without and with V-cycle refinement. In addition, the columns labeled "hMETIS" show the results obtained by hMETIS (which are identical to those shown in Tables 3 and 4), the rows labeled "ARQ" provides the average ratio of quality of each algorithm to hMETIS's results, and the rows labeled "Time" shows the amount of time required by the multi-resource partitioning algorithms relative to that required by hMETIS.

Comparing the solutions produced by the two sets of enforcement-

	hMETIS	Without V-cycle SCDB	SCMB	With V-cycle SCDB	SCMB
ind1	213	218	213	216	204
ind2	147	149	150	149	150
ind3	85	99	96	98	95
ind4	127	167	159	149	155
ind5	634	675	665	669	652
ind6	822	848	832	846	831
ind7	917	928	922	902	905
ind8	430	479	430	425	427
ind9	1289	1334	1335	1320	1332
ind10	360	368	364	363	364
ind11	193	212	193	211	192
ind12	307	375	327	363	322
ARQ	1.000	1.088	1.046	1.058	1.033
Time	1.000	1.034	1.278	1.945	2.035

Table 6: Performance of algorithms combined with multi-constraint V-cycle as an average 10 runs for 45%-55% balance factor.

based multi-resource partitioning algorithms we can see that, unlike the native algorithms, there is relatively little variation between the performance achieved by them. Specifically, the performance difference between the two schemes is less that 7%, on the average. However, the SCMB algorithm is consistently better than SCDB, leading to better solutions in 31 out of the 48 different experimental data-points. Comparing the results without and with V-cycle refinement we see that as it was the case with the native algorithms, the overall quality of the two algorithms improves, as well. However, those improvements are relatively small, ranging on the average between 2% and 5%. Finally, comparing the amount of time required by these algorithms we can see that SCMB is slower than SCDB, but in most cases the difference is small.

5.3 Overall Comparisons

Comparing the performance achieved by the various multi-resource partitioning algorithms we can see that in almost all the cases, the enforcement-based algorithms lead to solutions that have lower cut than those obtained by the native multi-resource partitioning algorithms. For example, the best-performing enforcement-based scheme SCMB outperforms the best-performing native scheme in 41 out 48 data-points. Moreover, the cut differences are considerable, and on the average SCMB leads to cuts that are 13%–32% better than that of MPMC. However, this performance advantage is also data-set dependent, and the relative performance of the various schemes can change for different benchmarks.

Finally, comparing the performance achieved by SCMB against that achieved by the single-constraint hMETIS, we can see that the overall increase in the cut resulting by solving the multi-resource partitioning problem, is quite small. For example, if we consider SCMB's results with V-cycle refinement we can see that on the average the cut increase by only 5.7% and 3.3% for the 49%–51% and 45%–55% balance constraints, respectively.

6. CONCLUSION

In this paper we presented two classes of multi-resource aware partitioning algorithms for enabling partitioning-based placement methods for FPGA architectures with heterogeneous devices. These algorithms are very effective in minimizing the cut while satisfying multiple balancing requirements with acceptable computational effort. The average cut of the most effective algorithm is only 5.7% and 3.3% worse than that of the state-of-the-art partitioning tool hMETIS [10] for 49%–51% and 45%–55% balance constraints, respectively. Moreover, their additional computational requirements are small, requiring only two to three times more time than hMETIS.

These results indicate that high-quality partitionings are feasible for designs with multiple resource requirements, suggesting that partitioning-based placement methods can be used for placing such designs on modern FPGA architectures.

7. REFERENCES

[1] A. A.E.Caldwell and I.L.Markov. Can recursive bisection alone produce routable placements? In *Proceedings of Design Automation Conference*, pages 477–482, 2000.

[2] C. J. Alpert, J. H. Huang, and A. B. Kahng. Multilevel circuit partitioning. In *Proc. of DAC*, 1997.

[3] M. C. Walshaw and K. McManus. Multiphase mesh partitioning. *Applied Mathematical Modelling*, 25:123–140, 2000.

[4] J. Cong, H. P. Li, S. K. Lim, T. Shibuya, and D. Xu. Large scale circuit partitioning with loose/stable net removal and signal flow based clustering. In *Proc. of the Design Automation Conference*, pages 441–446, 1997.

[5] J. Cong(Editor) and J. Shinnerl(Editor). Chapter3: Multilevel hypergraph partitioning. In *Multilevel Optimization in VLSICAD*, 2003.

[6] C. M. Fiduccia and R. M. Mattheyses. A linear time heuristic for improving network partitions. In *In Proc. 19th IEEE Design Automation Conference*, pages 175–181, 1982.

[7] J.Cong and S.K.Lim. Edge separability based circuit clustering with application to circuit partitioning. In *Proc. Asia and South Pacific Design Automation Conf.*, pages 429–434, 2000.

[8] G. Karypis. Multilevel algorithms for multi-constraint hypergraph partitioning. Technical Report TR 99-034, Department of Computer Science, University of Minnesota, 1999. Also available on WWW at URL http://www.cs.umn.edu/~karypis.

[9] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning: Application in vlsi domain. *IEEE Transactions on VLSI Systems*, 20(1), 1999. A short version appears in the proceedings of DAC 1997.

[10] G. Karypis and V. Kumar. hMETIS 1.5: A hypergraph partitioning package. Technical report, Department of Computer Science, University of Minnesota, 1998. Available on the WWW at URL *http://www.cs.umn.edu/~metis*.

[11] G. Karypis and V. Kumar. Multilevel algorithms for multi-constraint graph partitioning. In *Proceedings of Supercomputing*, 1998. Also available on WWW at URL http://www.cs.umn.edu/~karypis.

[12] A. R. M. Wang and S.Raje. Multi-million gate FPGA physical design challenges. In *To Appear on ICCAD 2003*, 2003.

[13] X. M.Wang and M.Sarrafzadeh. Dragon2000: Standard-cell placement tool for large industry circuits. In *ICCAD*, pages 160–163, 2000.

[14] S. Wichlund and E. J. Aas. On Multilevel Circuit Partitioning. In *Proc. of ICCAD*, 1998.

[15] www.xilinx.com.

[16] P. Zuchowski, C. Reynolds, R. Grupp, S.G.Davis, B.Cremen, and B.Troxel. A hybrid asic and fpga architecture. In *Proc. of ICCAD*, pages 187–194, 2002.