A HYBRID MULTILEVEL/GENETIC APPROACH FOR CIRCUIT PARTITIONING

Charles J. Alpert[1] *Lars W. Hagen*[2] *Andrew B. Kahng*[1]

[1]UCLA Computer Science Department, Los Angeles, CA 90095-1596
[2]Cadence Design Systems, San Jose, CA 94135

ABSTRACT

We present a genetic circuit partitioning algorithm that integrates the Metis graph partitioning package [15] originally designed for sparse matrix computations. Metis is an extremely fast iterative partitioner that uses multilevel clustering. We have adapted Metis to partition circuit netlists, and have applied a genetic technique that uses previous Metis solutions to help construct new Metis solutions. Our hybrid technique produces better results than Metis alone, and also produces bipartitionings that are competitive with previous methods [20] [18] [6] while using less CPU time.

1. INTRODUCTION

A netlist hypergraph $H(V,E)$ has n modules $V = \{v_1, v_2, \ldots v_n\}$; a hyperedge (or *net*) $e \in E$ is defined to be a subset of V with size greater than one. A *bipartitioning* $P = \{X, Y\}$ is a pair of disjoint *clusters* (i.e., subsets of V) X and Y such that $X \cup Y = V$. The *cut* of a bipartitioning $P = \{X, Y\}$ is the number of nets which contain modules in both X and Y, i.e., $cut(P) = |\{e \mid e \cap X \neq \emptyset, e \cap Y \neq \emptyset\}|$. Given a balance tolerance r, the *min-cut bipartitioning problem* seeks a solution $P = \{X, Y\}$ that minimizes $cut(P)$ such that $\frac{n(1-r)}{2} \leq |X|, |Y| \leq \frac{n(1+r)}{2}$.

The standard bipartitioning approach is iterative improvement based on the Kernighan-Lin (KL) [16] algorithm, which was later improved by Fiduccia-Mattheyses (FM) [8]. The FM algorithm begins with some initial solution $\{X, Y\}$ and proceeds in a series of *passes*. During a pass, modules are successively moved between X and Y until each module has been moved exactly once. Given a current solution $\{X', Y'\}$, the module $v \in X'$ (or Y') with highest *gain* ($= cut(\{X' - v, Y' + v\}) - cut(\{X, Y\})$) that has not yet been moved is moved from X' to Y'. After each pass, the best solution $\{X', Y'\}$ observed during the pass becomes the initial solution for a new pass, and the passes terminate when a pass does not improve the initial solution. FM has been widely adopted due to its short runtimes and ease of implementation.

One significant improvement to FM addresses the tie-breaking used to choose among alternate moves that have the same gain. Krishnamurthy [17] proposed a lookahead tie-breaking mechanism, and Sanchis [22] extended this approach to multi-way partitioning. Hagen, Huang, and Kahng [9] have shown that a "last-in-first-out" scheme based on the order that modules are moved in FM is significantly better than random or "first-in-first-out" tie-breaking schemes. More recently, Dutt and Deng [7] independently reached the same conclusion. Finally, Saab [21] has also exploited the order in which modules are moved to produce an improved FM variant.

A second significant improvement to FM integrates *clustering* into a "two-phase" methodology. A k-way clustering of $H(V,E)$ is a set of disjoint clusters $P^k = \{C_1, C_2, \ldots C_k\}$ such that $C_1 \cup C_2 \cup \ldots \cup C_k = V$ where k is sufficiently large.[1] We denote the input netlist as $H_0(V_0, E_0)$. A clustering $P^k = \{C_1, C_2, \ldots, C_k\}$ of H_0 induces the *coarser* netlist $H_1(V_1, E_1)$, where $V_1 = \{C_1, C_2, \ldots, C_k\}$ and for every $e \in E_0$, the net e' is a member of E_1 where $e' = \{C_i \mid \exists v \in e$ and $v \in C_i\}$ unless $|e'| = 1$ (i.e., each cluster in e' contains some module that is in e). In two-phase FM, a clustering of H_0 induces the coarser netlist H_1, and then FM is run on $H_1(V_1, E_1)$ to yield the bipartitioning $P_1 = \{X_1, Y_1\}$. This solution then *projects* to the bipartitioning $P_0 = \{X_0, Y_0\}$ of H_0, where $v \in X_0(Y_0)$ if and only if for some $C_h \in V_1$, $v \in C_h$ and $C_h \in X_1(Y_1)$. FM is then run a second time on $H_0(V_0, E_0)$ using P_0 as the initial solution.

Many clustering algorithms for two-phase FM have appeared in the literature (see [2] for an overview of clustering methods and for a general netlist partitioning survey). Bui et al. [5] find a random maximal matching in the netlist and compact the matched pairs of modules into $\frac{n}{2}$ clusters; the matching can then be repeated to generate clusterings of size $\frac{n}{4}, \frac{n}{8}$, etc. Often, two-phase FM (not including the time needed to cluster) is faster than a single FM run because the first FM run is for a smaller netlist and the second FM run starts with a good initial solution, allowing fast convergence to a local minimum.

The "two-phase" approach can be extended to include more phases; such a *multilevel* approach is illustrated in Figure 1 (following [15]). In a multilevel algorithm, a clustering of the initial netlist H_0 induces the coarser netlist H_1, then a clustering of H_1 induces H_2, etc. until the coarsest netlist H_m is constructed ($m = 4$ in the Figure). A partitioning solution $P_m = \{X_m, Y_m\}$ is found for H_m (e.g., via FM) and this solution is projected to $P_{m-1} = \{X_{m-1}, Y_{m-1}\}$.

[1]A *partitioning* and a *clustering* are identical by definition, but the term partitioning is generally used when k is small (e.g., $k \leq 10$), and the term clustering is generally used when k is large (e.g., $k = \Theta(n)$ with constant average cluster size). Although a bipartitioning can also be written as $P^2 = \{C_1, C_2\}$, we use the notation $P = \{X, Y\}$ to better distinguish between partitioning and clustering.

P_{m-1} is then refined, e.g., by using it as an initial solution for FM. In the Figure, each projected solution is indicated by a dotted line and each refined solution is given by a solid dividing line. This *uncoarsening* process continues until a partitioning of the original netlist H_0 is derived.

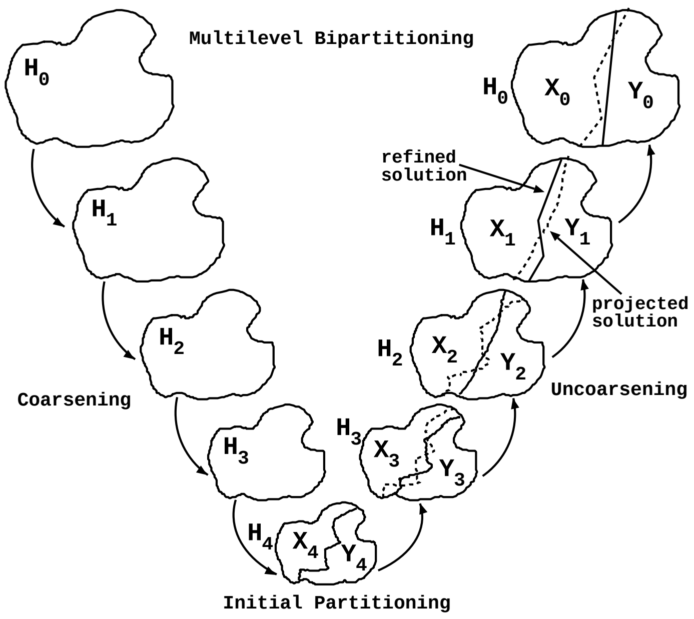

Figure 1. The multilevel bipartitioning paradigm.

Multilevel clustering methods have been virtually unexplored in the physical design literature; the work of Hauck and Borriello [10] is the notable exception. [10] performed a detailed study of multilevel partitioning for FPGAs and found that simple connectivity-based clustering combined with a KL and FM multilevel approach produced excellent solutions. However, multilevel partitioning has been well-studied in the scientific computing community, e.g., Hendrickson and Leland [11] [12] and Karypis and Kumar [13] [14] [15] have respectively developed the Chaco and Metis partitioning packages.

The Metis package of [15] has produced very good partitioning results for finite-element graphs and is extremely efficient, requiring only 2.8 seconds of CPU time on a Sun Sparc 5 to bipartition a graph with more than 15,000 vertices and 91,000 edges. Our initial hypothesis, which our work has verified, was that Metis adapted for circuit netlists is both better and faster than FM. Metis runtimes are so low that we can easily afford to run it over 100 times to generate a partitioning. However, instead of simply calling Metis 100 times, we propose to integrate Metis into a genetic algorithm; our experiments show that this approach produces better average and minimum cuts than Metis alone. Overall, our approach generates bipartitioning solutions that are competitive with the recent approaches of [20] [18] [6] while requiring much less CPU time.

The rest of our paper is as follows. Section 2 reviews the Metis partitioning package and presents our modifications for circuit netlists. Section 3 presents our Metis-based genetic algorithm. Section 4 presents experimental results for 23 ACM/SIGDA benchmarks, and Section 5 concludes with directions for future work.

2. GRAPH PARTITIONING USING METIS

The Metis package of Karypis and Kumar has multiple algorithm options for coarsening, for the initial partitioning step, and for refinement. For example, one can choose among eight different matching-based clustering schemes including random, heavy-edge, light-edge, and heavy-clique matching. The methodology we use follows the general recommendations of [13], even though their algorithm choices are based on extensive empirical studies of finite-element graphs and not circuit netlists. Before multilevel partitioning is performed, the adjacency lists for each module are randomly permuted. The following discussion applies our previous notation to weighted graphs; a weighted graph is simply a hypergraph H_i and $|e| = 2$ for each $e \in E_i$ with a nonnegative weight function w on the edges.

To cluster, Karypis and Kumar suggest *Heavy-Edge Matching* (HEM), which is a variant of the random matching algorithm of [5]. A matching M of H_i is a subset of E_i such that no module is incident to more than one edge in M. Each edge in the matching will be contracted to form a cluster, and the contracted edges should have the highest possible weights since they will not be cut in the graph H_{i+1}. HEM visits the modules in random order; if a module u is unmatched, the edge (u, v) is added to M where $w(u, v)$ is maximum over all unmatched modules v; if u has no unmatched neighbors, it remains unmatched. This greedy algorithm by no means guarantees a maximum sum of edge weights in M, but it runs in $O(|E_i|)$ time. Following [15], our methodology iteratively coarsens until $|V_m| \leq 100$.

An initial bipartitioning for H_m is formed by the Greedy Graph Growing Partitioning (GGGP) algorithm. Initially, one "fixed" module v is in its own cluster X_m and the rest of the modules are in Y_m. Modules with highest gains are greedily moved from Y_m to X_m until $P_m = \{X_m, Y_m\}$ satisfies the cluster size constraints. Since the solution is extremely sensitive to the initial choice of v, the algorithm is run four times with different initial modules, and the best solution observed is retained for the next step. Despite its simplicity, the GGGP heuristic proved at least as effective as other heuristics for partitioning finite element graphs [13].

The refinement steps use the *Boundary Kernighan-Lin Greedy Refinement* (BGKLR) scheme. Despite its name, the heuristic actually uses the FM single-module neighborhood structure. Kumar and Karypis label the KL algorithm "greedy" when only a single pass is performed, and propose a hybrid algorithm which performs "complete" KL when the graph is small (i.e., less than 2000 modules) and greedy KL for larger graphs. They show that greedy KL is only slightly inferior to complete KL, but saves substantial CPU time. A "boundary" scheme is also used for updating gains: initially, only modules that are incident to cut edges (i.e., boundary modules) are stored in the FM bucket data structure and are eligible to be moved; when a module that is not in the data structure becomes incident to a moved module, it is inserted into the bucket data structure only if it has positive gain. The cost of performing the boundary version of KL is small, since only the boundary modules are considered. The overall Metis methodology is presented in Figure 2.

```
The Metis Algorithm
Input:    Graph $H_0(V_0, E_0)$
Output:   Bipartitioning $P_0 = \{X_0, Y_0\}$
1. i = 0; randomly permute the adjacency lists of $H_0$.
2. while $|V_i| < 100$ do
3.    Use HEM to find a matching $M$ of $H_i$
4.    Contract each edge in $M$ to form a clustering.
 !    Construct the coarser graph $H_{i+1}(V_{i+1}, E_{i+1})$.
      Set $i$ to $i + 1$.
5. Let $m = i$. Apply GGGP to $H_m$ to derive $P_m$.
   Refine $P_m$ using BGKLR.
6. for i = m-1 downto 0 do
7.    Project solution $P_{i+1}$ to the new solution $P_i$.
      Refine $P_i$ using BGKLR.
8. return $P_0$.
```

Figure 2. The Metis Algorithm

To run Metis on circuit netlists, we use an efficient hypergraph to graph converter that constructs sparse graphs. The traditional clique net model (which adds an edge to the graph for every pair of modules in a given net) is not a good choice since large nets will destroy sparsity. Since we observed that keeping large nets generally increases the cut size regardless of the net model, we removed all nets with more than T modules (we use $T = 50$). For each net e, our converter picks $F \cdot |e|$ random pairs of modules in e and adds an edge with cost one into the graph for each pair. Here, F is a constant; our experiments show that the value of F is not too significant as long as it is large enough (we use $F = 5$). Table 1 shows how the Metis cuts vary with various values of F and T. Our converter retains the sparsity of the circuit, introduces randomness to allow multiple Metis runs, and is fairly efficient.

3. A GENETIC VERSION OF METIS

Our experiments in Section 4 show that over the course of 100 independent runs Metis generates at least one very good solution, but that its performance is not particularly stable, generating average cuts much higher than minimum cuts. To try to stabilize solution quality and generate superior solutions, we have integrated Metis into a genetic framework.

An *indicator vector* $\vec{p} = \{p_1, p_2, \ldots, p_n\}$ for a bipartitioning $P = \{X, Y\}$ has entry $p_i = 0$ if $v_i \in X$ and entry $p_i = 1$ if $v_i \in Y$, for all $i = 1, 2, \ldots, n$. The *distance* between two bipartitionings P and Q with corresponding indicator vectors $\vec{p}$ and $\vec{q}$ is given by $\sum_{i=1}^{n} |p_i - q_i|$, i.e., by the number of module moves needed to derive solution Q from the initial solution P. Boese et al. [4] showed that the set of local minima generated by multiple FM runs exhibit a "big valley" structure: solutions with smallest distance to the lowest-cost local minima also have low cost, and the best local minima are "central" with respect to the other local minima. Thus, we seek to combine several local minimum solutions generated by Metis into a more "central" solution.

Given a set S of s solutions, the s-digit binary code $C(i)$ for module v_i is generated by concatenating the i^{th} entries of the indicator vectors for the s solutions. We construct a clustering by assigning modules v_i and v_j to the same cluster if $C(i)$ and $C(j)$ are the same code. Our strategy integrates this code-generated clustering into Metis, in that we use HEM clustering and force every clustering generated during coarsening to be a refinement of the code-based clustering.[2] Our Genetic Metis (GMetis) algorithm is shown in Figure 3.

```
The Genetic Metis (GMetis) Algorithm
Input:      Hypergraph $H(V, E)$ with $n$ modules
Output:     Bipartitioning $P = \{X, Y\}$
Variables:  $s$: Number of solutions
            $numgen$: Number of generations
            $C(i)$: $s$-digit code for module $v_i$
            $S$: set of the $s$ best solutions seen
            $G$: graph with $n$ modules
1. Set $C(i) = 00\ldots0$ for $1 \le i \le n$
2. for i = 0 to $numgen - 1$ do
3.    for j = 0 to $s - 1$ do
4.       if $((i \cdot s) + j)$ modulo 10 = 0
           then convert $H$ to graph $G$
5.       P = Metis(G) (HEM based on codes $C(i)$)
6.       if $\exists Q \in S$ such that $Q$ has larger cut than $P$
           then $S = S + P - Q$.
7.       if $i > 0$ and $(s(i-1) + j)$ modulo 5 = 0
           then recompute $C(i)$ for $1 \le i \le n$ using $S$.
8. return $P \in S$ with lowest cut.
```

Figure 3. The Genetic Metis Algorithm

Step 1 initially sets all codes to $00\ldots0$ which causes GMetis to behave just like Metis until s solutions are generated. Steps 2 and 3 are loops which cause *numgen* generations of s solutions to be computed. Next, Step 4 converts the circuit hypergraph into a graph, but this step is performed only once out of every 10 times Metis is called. We perform the conversion with this frequency to reduce runtimes while still allowing a variety of different graph representations; the constant 10 is fairly arbitrary. In Step 5, Metis is called using our version of HEM described above. Step 6 maintains the set of solutions S; our replacement scheme replaces solution $Q \in S$ with solution P if P has smaller cut size than Q; other replacement schemes may work just as well and need to be investigated. Step 7 computes the binary code for each module based on the current solution set, but only after the first generation has completed and five solutions with the previous code-based clustering have been generated. As in Step 4, the constant 5 is fairly arbitrary. Finally, the solution with lowest cut is returned in Step 8.

4. EXPERIMENTAL RESULTS

All of our experiments use a subset of the benchmarks from the ACM/SIGDA suite given in Table 2; hypergraph formats of these circuits are available on the world wide web

[2] A clustering P^k is a *refinement* of Q^l $(k \ge l)$ if some division of clusters in Q^l will yield P^k.

T/F	1	2	3	4	5	6	8	10	12	15
10	289(238)	241(184)	239(185)	238(180)	225(176)	228(184)	230(174)	220(169)	225(178)	227(169)
15	276(231)	224(188)	239(185)	228(184)	222(178)	228(175)	228(165)	215(176)	241(181)	228(174)
20	320(261)	252(202)	259(180)	258(189)	252(173)	253(187)	261(165)	269(190)	265(176)	253(178)
25	321(243)	251(189)	250(174)	254(170)	243(169)	238(162)	255(173)	232(162)	245(176)	266(174)
35	309(243)	248(172)	239(170)	227(168)	249(173)	245(171)	240(166)	247(164)	254(176)	239(169)
50	316(233)	258(190)	250(177)	251(173)	245(169)	240(167)	255(159)	255(178)	232(162)	240(175)
100	310(231)	274(184)	256(173)	260(172)	256(173)	254(175)	248(166)	237(170)	245(180)	252(176)
200	325(252)	265(182)	266(170)	257(174)	288(184)	258(182)	261(187)	260(192)	271(181)	266(186)
500	471(366)	427(333)	418(318)	412(327)	414(294)	429(295)	399(311)	408(321)	414(296)	411(270)

Table 1. Average(minimum) cuts for the avqlarge test case for 50 runs of Metis shown for various values of T (rows) and F (columns).

at http://ballade.cs.ucla.edu/~cheese. Our experiments assume unit module areas, and our code was written in C++ and was compiled with g++ $v2.4$ on a Unix platform. Our experiments were run on an 85 Mhz Sun Sparc 5 and all runtimes reported are for this machine (in seconds) unless otherwise specified. We performed the following studies:

- We compare Metis against standard and two-phase FM, to show the effectiveness of the multilevel approach.
- We show that the GMetis algorithm is more effective than running Metis multiple times.
- Finally, we show that GMetis is competitive with previous approaches while using a fraction of the runtime.

Test Case	# Modules	# Nets	# Pins
balu	801	735	2697
bm1	882	903	2910
primary1	833	902	2908
test04	1515	1658	5975
test03	1607	1618	5807
test02	1663	1720	6134
test06	1752	1541	6638
struct	1952	1920	5471
test05	2595	2750	10076
19ks	2844	3282	10547
primary2	3014	3029	11219
s9234	5866	5844	14065
biomed	6514	5742	21040
s13207	8772	8651	20606
s15850	10470	10383	24712
industry2	12637	13419	48404
industry3	15406	21923	65792
s35932	18148	17828	48145
s38584	20995	20717	55203
avqsmall	21918	22124	76231
s38417	23849	23843	57613
avqlarge	25178	25384	82751
golem3	103048	144949	338419

Table 2. Benchmark circuit characteristics.

4.1. Metis vs. FM and Two-phase FM

Our first set of experiments compares Metis against both FM and two-phase FM. We ran Metis 100 times with balance parameter $r = 0$ (exact bisection) and recorded the minimum cut observed in the second column of Table 3. Since there are many implementations of FM (some of which are better than others), we compare to the best FM results found in the literature.

Test Case	Minimum cut (100 runs)				CPU (s)	
	Metis	FM	FM	2-FM	Metis	FM
		[6]	[9]	[1] [9]		[6]
balu	34	32			23	21
bm1	53	55		52	22	24
primary1	55	57	56	53	22	24
test04	53	86		56	37	41
test03	61	72		60	39	56
test02	99	115		97	43	46
test06	94	71		68	53	50
struct	36	45	36	43	41	46
test05	107	97		93	69	81
19ks	116	142		121	59	115
primary2	158	236	171	182	90	128
s9234	49	53			72	222
biomed	83	83	83	124	134	296
s13207	84	92			111	339
s15850	62	112			123	339
industry2	218	428	275	438	349	727
industry3	292		312	328	399	
avqsmall	175		373	399	293	
avqlarge	171		406	518	355	

Table 3. Comparison of Metis with FM.

Dutt and Deng [6] have implemented very efficient FM code; their exact bisection results for the best of 100 FM runs are given in the third column of Table 3 and the corresponding Sparc 5 run times are given in the last column. Hagen et al. [9] have run FM with an efficient LIFO tie breaking strategy and a new lookahead function that outperforms [17]; their bisection results are reported in the fourth column. Finally, we compare to various two-phase FM strategies. In the fifth column, we give the best two-phase FM results observed for various clustering algorithms as reported in [1] and [9].

Metis does not appear to be faster than FM for circuits with less than two thousand modules, but for larger circuits with five to twelve thousand modules, Metis is 2-3 times faster. In terms of cut sizes, again Metis is indistinguishable from FM for the smaller benchmarks, but Metis cuts are significantly lower for the larger benchmarks. We conclude that multilevel approaches are unnecessary for small circuits, but greatly enhance solution quality for larger circuits. For these circuits, more than two levels of clustering are required for such an iterative approach to be effective.

4.2. Genetic Metis vs. Metis

The next set of experiments compares Metis with the GMetis. We ran GMetis for 10 generations while maintaining $s = 10$ solutions so that both Metis and GMetis considered 100 total solutions. The minimum and average cuts observed, as well as total CPU time, are reported for both algorithms in Table 4.

Test	Metis			GMetis		
Case	min	avg	CPU	min	avg	CPU
balu	34	47	26	32	38	24
bm1	53	65	23	54	59	22
primary1	55	66	23	55	59	21
test04	53	68	37	52	58	37
test03	61	76	42	65	74	39
test02	99	113	44	96	101	42
test06	94	117	59	97	121	55
struct	36	52	41	34	40	39
test05	107	125	70	109	117	69
19ks	116	132	59	112	116	59
primary2	158	195	95	165	174	91
s9234	49	66	71	45	52	68
biomed	83	149	145	83	134	143
s13207	84	90	106	78	89	112
s15850	62	84	126	59	74	125
industry2	218	280	336	204	230	339
industry3	292	408	384	291	313	423
s35932	55	71	257	56	62	265
s38584	55	101	310	53	67	368
avqsmall	175	241	289	148	174	322
s38417	73	110	294	74	104	301
avqlarge	171	248	318	144	181	355
golem3	2196	2520	1592	2196	2648	1928

Table 4. Comparison of Metis with Genetic Metis. The minimum cut, average cut and CPU time for 100 runs of each algorithm are given.

On average, GMetis yields minimum cuts that are 2.7% lower than Metis, and significantly lower average cuts (except for golem3). For the larger benchmarks (seven to twenty-six thousand modules) GMetis cuts are 5.7% lower, with significant improvements for industry2, avqsmall and avqlarge. We believe that GMetis can have the greatest impact for larger circuits. Note that golem3 is the one benchmark in which Metis outperforms GMetis in terms of the average case – the quality of GMetis solutions gradually became worse in subsequent generations instead of converging to single good solutions.

4.3. Genetic Metis vs. Other Approaches

Finally, we compare GMetis to other recent partitioning works in the literature, namely PROP [6], Paraboli [20], and GFM [19], the results of which are quoted from the original sources and presented in Table 5. All these works use $r = 0.1$, i.e., each cluster contains between 45% and 55% of the total number of modules. The CPU times in seconds for PROP, Paraboli, and GFM are respectively reported for a Sun Sparc 5, a DEC 3000 Model 500 AXP, and a Sun Sparc 10. We modified GMetis to handle varying size constraints by allowing the BGKLR algorithm to move modules while satisfying the cluster size constraints. We found that with this implementation, GMetis with $r = 0.1$ was sometimes outperformed by GMetis with $r = 0$ (exact bisection). Hence, in Table 5, we present results for GMetis with $r = 0.1$ and $r = 0$ (given in parentheses).[3] Since for $r = 0.1$, GMetis runtimes sometimes increase by 20-50%, we report runtimes for $r = 0$ in the last column. These experiments used the somewhat arbitrary parameter values of $s = \log_2 n$ ($|V| = n$) solutions and 12 generations.

Observe that GMetis cuts are competitive with the other methods, especially for the larger benchmarks s15850, industry2, and avqsmall. However, the big win for GMetis is the short runtimes: generating a single solution for avqlarge and golem3 respectively takes $417/(12 \log_2 25178) = 2.5$ and $450/(2 \log_2 103048) = 15$ seconds on average. For golem3, we only ran 2 generations since the results do not improve with subsequent generations; the solution with cost 2144 was achieved after only 210 seconds of CPU time.

5. CONCLUSIONS

This work integrates the Metis multilevel partitioning algorithm of [15] into a genetic algorithm. We showed (i) Metis outperforms previous FM-based approaches, (ii) GMetis improves upon Metis alone for large benchmarks, and (iii) GMetis is competitive with previous approaches while using less CPU time. There are many improvements which we are pursuing:

- Find sparser and more reliable hypergraph conversion algorithms.
- Try alternative genetic replacement schemes, instead of simply inserting the current solution into S if it is a better solution.
- Tweak parameters such as F, T, s, and the number of generations in order to generate more stable solutions in fewer iterations.
- Experiment with various schemes to control cluster sizes in the bipartitioning solution. That GMetis frequently finds better 50-50 solutions versus 45-55 solutions is not acceptable.
- Finally, we are integrating our own separate multilevel circuit partitioning code into a new cell placement algorithm.

REFERENCES

[1] C. J. Alpert and A. B. Kahng, "A General Framework for Vertex Orderings, with Applications to Netlist Clustering", to appear in *IEEE Trans. on VLSI.*

[2] C. J. Alpert and A. B. Kahng, "Recent Directions in Netlist Partitioning: A Survey", *Integration, the VLSI Journal*, 19(1-2), pp. 1-81, 1995.

[3] C. J. Alpert and S.-Z. Yao, "Spectral Partitioning: The More Eigenvectors, the Better," *Proc. ACM/IEEE Design Automation Conf.*, 1995, pp. 195-200.

[4] K. D. Boese, A. B. Kahng, and S. Muddu, "A New Adaptive Multi-Start Technique for Combinatorial Global Optimizations", *Operations Research Letters*, 16(20), pp. 101-13, 1994.

[3] We attribute this undesirable behavior to improper modification of the Metis code. We believe that a better implementation should yield $r = 0.1$ results at least as good as those for $r = 0$ without increasing runtimes.

Test Case	cuts				CPU			
	PROP	Paraboli	GFM	GMetis(bal)	PROP	Paraboli	GFM	GMetis
balu	27	41	27	27(32)	16	16	24	14
bm1	50			48(53)	20			12
primary1	47	53	47	47(54)	19	18	16	12
test04	52			49(52)	49			21
test03	59			62(66)	51			23
test02	90			95(96)	64			26
test06	76			94(93)	75			32
struct	33	40	41	33(34)	42	35	80	27
test05	79			104(109)	97			46
19ks	105			106(110)	87			39
primary2	143	146	139	142(158)	139	137	224	53
s9234	41	74	41	43(45)	139	490	672	58
biomed	83	135	84	102(83)	250	711	1440	95
s13207	75	91	66	74(70)	177	2060	1920	102
s15850	65	91	63	53(60)	291	1731	2560	114
industry2	220	193	211	177(204)	867	1367	4320	245
industry3		267	241	243(286)		761	4000	299
s35932		62	41	57(55)		2627	10160	266
s38584		55	47	53(53)		6518	9680	397
avqsmall		224		144(145)		4099		328
s38417		49	81	69(77)		2042	11280	281
avqlarge		139		145(144)		4135		417
golem3		1629		2111(2144)		10823		450

Table 5. Comparison of GMetis with PROP, Paraboli, and GFM for min-cut bipartitioning allowing 10% deviation from bisection. Exact bisection results for GMetis are given in parentheses in the fifth column.

[5] T. Bui, C. Heigham, C. Jones, and T. Leighton, "Improving the Performance of the Kernighan-Lin and Simulated Annealing Graph Bisection Algorithms", *Proc. ACM/IEEE Design Automation Conf.*, pp. 775-778, 1989.

[6] S. Dutt and W. Deng, "A Probability-Based Approach to VLSI Circuit Partitioning", to appear in *Proc. ACM/IEEE Design Automation Conf.*, 1996.

[7] S. Dutt and W. Deng, "VLSI Circuit Partitioning by Cluster-Removal Using Iterative Improvement Techniques", *Technical Report*, Department of Electrical Engineering, University of Minnesota, Nov. 1995.

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

[9] L. W. Hagen, D. J.-H. Huang, and A. B. Kahng, "On Implementation Choices for Iterative Improvement Partitioning Algorithms", to appear in *IEEE Trans. Computer-Aided Design* (see also *Proc. European Design Automation Conf.*, Sept. 1995, pp. 144-149).

[10] S. Hauck and G. Borriello, "An Evaluation of Bipartitioning Techniques", *Proc. Chapel Hill Conf. on Adv. Research in VLSI*, 1995.

[11] B. Hendrickson and R. Leland, "A Multilevel Algorithm for Partitioning Graphs", Technical Report SAND93-1301, Sandia National Laboratories, 1993.

[12] B. Hendrickson and R. Leland, "The Chaco User's Guide", Technical Report SAND93-2339, Sandia National Laboratories, 1993.

[13] G. Karypis and V. Kumar, "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs", *Technical Report* #95-035, Department of Computer Science, University of Minnesota, 1995.

[14] G. Karypis and V. Kumar, "Multilevel k-Way Partitioning Scheme for Irregular Graphs", *Technical Report* #95-035, Department of Computer Science, University of Minnesota, 1995.

[15] G. Karypis and V. Kumar, "Unstructured Graph Partitioning and Sparse Matrix Ordering", *Technical Report*, Department of Computer Science, University of Minnesota, 1995 (see http://www.cs.umn.edu/~kumar for postscript and code).

[16] B. W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs", *Bell Systems Tech. J.*, 49(2), pp. 291-307, 1970.

[17] B. Krishnamurthy, "An Improved Min-Cut Algorithm for Partitioning VLSI Networks", *IEEE Trans. Computers*, 33(5), pp. 438-446, 1984.

[18] J. Li, J. Lillis, and C.-K. Cheng, "Linear Decomposition Algorithm for VLSI Design Applications", *Proc. IEEE Intl. Conf. Computer-Aided Design*, pp. 223-228, 1995.

[19] L.-T. Liu, M.-T. Kuo, S.-C. Huang, and C.-K. Cheng, "A Gradient Method on the Initial Partition of Fiduccia-Mattheyses Algorithm", *Proc. IEEE Intl. Conf. Computer-Aided Design*, pp. 229-234, 1995.

[20] B. M. Riess, K. Doll, and F. M. Johannes, "Partitioning Very Large Circuits Using Analytical Placement Techniques", *Proc. ACM/IEEE Design Automation Conf.*, pp. 646-651, 1994.

[21] Y. Saab, "A Fast and Robust Network Bisection Algorithm", *IEEE Trans. Computers*, 44(7), pp. 903-913, 1995.

[22] L. A. Sanchis, "Multiple-Way Network Partitioning", *IEEE Trans. Computers*, 38(1), pp. 62-81, 1989.