Computer Science Department Technical Report
University of California
Los Angeles, CA 90024-1596

A BOTTOM-UP CLUSTERING ALGORITHM WITH APPLICATIONS TO CIRCUIT PARTITIONING IN VLSI DESIGNS

Jason Cong
M'Lissa Smith

October 1992
CSD-920055

A Bottom-up Clustering Algorithm with Applications to Circuit Partitioning in VLSI Design

Jason Cong and M'Lissa Smith
Department of Computer Science
University of California, Los Angeles
Los Angeles, CA 90024

Abstract

In this paper, we present a bottom-up clustering algorithm based on recursive collapsing of small cliques in a graph. The sizes of the small cliques are derived using random graph theory. This clustering algorithm leads to a natural parallel implementation in which multiple processors are used to identify clusters simultaneously. We also present a cluster-based partitioning method in which our clustering algorithm is used as a preprocessing step to both the bisection algorithm by Fiduccia and Mattheyses [FM82] and a ratio-cut algorithm by Wei and Cheng [WC89]. Our results show that cluster-based partitioning obtains cut sizes up to 41.3% smaller than the bisection algorithm, and obtains ratio cut sizes up to 89.6% smaller than the ratio-cut algorithm. Moreover, we show that cluster-based partitioning produces much stabler results than direct partitioning.

1 Introduction

1.1 Motivation

A *cluster* is group of strongly connected components in a circuit. The goal of clustering algorithms is to identify the clusters in a circuit. In VLSI layout design, clustering algorithms can be used to construct the natural hierarchy of the circuit. Many existing layout algorithms generate a circuit hierarchy based on recursive top-down partitioning [Len90]. Not only does the time and space required by partitioning algorithms increase as circuit sizes increase, but also the stability and quality of their results deteriorate. For example, iterative improvement based partitioning algorithms [KL70, FM82, WC89] do not perform well on very large circuits. These algorithms try to avoid local optima in the solution space by allowing the cut size to temporarily increase. However, as circuit sizes increase, the number of local optima becomes very large and these algorithms often fail to discover a cut size close to the global optimum. A poor result early in the top-down partitioning process imposes an unnatural circuit hierarchy and will likely lead to a suboptimal solution.

Bottom-up clustering algorithms provide a solution to the problems encountered when partitioning very large circuits. A bottom-up clustering algorithm can be integrated into the partitioning process by using clustering as a preprocessing step to partitioning. First, clustering is performed on the circuit to obtain a clustered circuit in which each cluster of components has been collapsed to form a single component. Partitioning is then performed on the clustered circuit instead of the original circuit. Since the number of components in a clustered circuit is usually much smaller than that of the original circuit, the time and space required by the partitioning algorithm is reduced significantly. Moreover, our study shows that partitioning the clustered circuit leads to better results than direct partitioning, since strongly connected components in each cluster are not separated during the partitioning process.

1.2 Basic Concepts and Terminology

1.2.1 Definitions

Partitioning. The *partitioning problem* is to divide a circuit into two or more subcircuits of (roughly) equal size while minimizing the cut size. The *cut size* is the number of nets connected to components in more than one subcircuit. Due to the inherent complexity of the partitioning problem, many heuristic algorithms have been proposed to obtain efficient solutions [KL70, FM82, KJV83, WC89, HK91, CHK92].

Clique. An *r-clique* is a complete graph with r nodes. The number of edges in an r-clique is $\binom{r}{2}$.

1.2.2 Graph Representations of Netlists

A netlist is best represented by a hypergraph with each component being represented by a node and each net represented by a hyperedge. However, many clustering and partitioning algorithms [BHJL89, GPS90, KL70, HK91, CHK92], including the one presented in this paper, use a graph representation of the netlist rather than a hypergraph representation. An r-terminal net is represented by an r-clique in the graph representation. The edges of the clique are usually weighted according to the size of the net. Several weighting functions have been proposed, including $\frac{2}{r-1}$, $\frac{2}{r}$, or $\frac{4}{r^2-mod(r,2)}$ [CP68, HK72, Don88]. In essence, these weighting functions assign smaller weights to edges in larger nets. Our clustering algorithm uses the weighting function $\frac{2}{r}$ for an r-terminal net.

1.2.3 Clustering Metrics

Once a cluster has been formed it is useful to know whether the cluster is *good*, i.e. whether the nodes in the cluster are strongly connected. Several clustering metrics have been proposed as follows.

Cluster Density. The *density* of a cluster is the ratio of the number of edges in the cluster to the maximum number of edges that could be in the cluster. In particular, given a cluster of c nodes, the cluster density is $\frac{E}{M_c}$, where $M_c = \binom{c}{2}$ and E may be either the total number of edges in the cluster or the total weight of those edges. Clusters with a higher density are considered to be of higher quality. Although, the density metric is perhaps the simplest and most intuitive of the three metrics, this metric is biased toward small clusters since the value of M_c increases rapidly as c increases.

k-l-connectedness. Another metric is based on the notion of k-l-connectedness [GPS90]. In a graph, two nodes are *k-l-connected* if and only if there exist k edge-disjoint paths connecting them such that each path has length at most l. The idea is that if two nodes are connected by many separate paths, they are strongly connected. However, it is not obvious what values should be assigned to k and l for any given circuit. This metric is more suitable for determining whether two nodes should be in the same cluster than for comparing the quality of clusters.

Degree/Separation. A more recent metric for determining the quality of clusters is the degree/separation (DS) metric [CHK91, HK92]. The cluster *degree* is the average number of nets incident to each component in the cluster. The cluster *separation* is the average length of a shortest path between two components in the cluster. Clusters with a higher DS value are of higher quality. Since this metric considers the global connectivity information, it is a very robust measurement. However, in general, it is costly to compute the DS value for large circuits since computing cluster separation requires $O(n^3)$ time, where n is the number of components in the cluster.

In this paper, we choose to use the density metric due to its simplicity. Certain adjustments will be taken into consideration to correct the bias of the density metric toward small clusters.

1.3 Previous Work in Bottom-up Clustering

In this section, we summarize previous work in bottom-up clustering. In contrast to the top-down clustering approaches which are based on recursive partitioning, these bottom-up clustering algorithms repeatedly find locally strongly connected clusters.

The Compaction Algorithm. This method [BHJL89] was developed to improve the results of partitioning algorithms such as the Kernighan-Lin algorithm [KL70] and simulated annealing [KJV83]. Those partitioning algorithms tend to perform poorly on graphs with average degree less than or equal to three [BHJL89]. The compaction heuristic increases the average degree of a graph by finding a maximal random matching on the graph. Each edge in the matching represents a cluster, and the two nodes connected by a matching edge are collapsed to form a single node in the compacted graph. Partitioning is performed on the compacted graph and the result is used as the starting point for the partitioning algorithm to compute the partition of the original graph. This clustering method can dramatically improve the results of Kernighan-Lin and simulated annealing and can even decrease the time required to find a partition. However, there is little or no improvement when the original graph has a high average degree. Also, this method does not attempt to find natural clusters which are very useful in VLSI design.

The k-l-connectedness Algorithm. This is a constructive algorithm that forms clusters defined by the transitive closure of the k-l-connectedness relation [GPS90]. For arbitrary k, the complexity of this algorithm is $O(d^{2l-1}n)$ where d is the maximum degree of the nodes. Although this approach is more likely to find natural clusters than the compaction heuristic, it is not obvious how to choose k and l for any given netlist. Moreover, when k and l are large, the computational complexity of the algorithm becomes prohibitive for large circuits.

The Random Walk Algorithms. Two bottom-up clustering algorithms that depend on random walks have been developed [CHK91, HK92]. A *random walk* begins at one node in the graph and takes a predetermined number of steps through the graph. At each step, the walk extends to a node selected randomly among the nodes adjacent to the current node. Both random walk algorithms begin by performing a random walk of n^2 steps on the graph. The clusters are based on cycles in the node sequence of the random walk. The first algorithm, RW1 [CHK91], finds the maximum cycle $C(v)$ for each node v in the graph. $C(v)$ is the longest sequence of *distinct* nodes that begins and ends with v. Two nodes v_i and v_j are in the same cluster if $v_i \in C(v_j)$ and $v_j \in C(v_i)$. The second random-walk based clustering algorithm, named RW-ST [HK92], is based on the concept of the "sameness" of two nodes. The *sameness* of two nodes reflects the size of the intersection of the cycles of the two nodes. Nodes with a sameness value greater than zero are placed in the same cluster. Using RW-ST as a preprocessing step to the Fiduccia-Mattheyses (FM) partitioning algorithm [FM82] resulted in cut sizes up to 17% lower than using the FM algorithm alone. A disadvantage of both RW1 and RW-ST is that they have a time complexity of $O(n^3)$, where n is the total number of nodes in the circuit.

1.4 Overview of the Paper

In this paper, we present a bottom-up clustering algorithm in which clusters are formed by recursively collapsing 5-cliques, 4-cliques, and 3-cliques. Once a clique is found, if it satisfies the size and density thresholds, the clique is collapsed to make a single node that represents the cluster. The collapsed node may be further clustered allowing clusters of arbitrary size to be formed. The parallel version of our algorithm allows cliques to be found simultaneously by multiple processors which reduces the time required for clustering. When combined with the FM partitioning algorithm [FM82] and a ratio-cut partitioning algorithm [WC89], significantly better results are obtained than when applying these partitioning algorithms directly.

The remainder of this paper is organized as follows. Section 2 begins with a discussion of the theoretical background of our clustering algorithm. Then, we describe the basic algorithm and its parallel implementation. Section 3 presents our cluster-based partitioning method and the partitioning results obtained by our method. Section 4 concludes this paper and presents a possible extension of our clustering algorithm to a cluster-based placement method.

2 The Clustering Algorithm

2.1 Theoretical Background

In our clustering algorithm, clusters are based on recursive collapsing of *5*-cliques, *4*-cliques, and *3*-cliques. The choice of these clique sizes can be explained as follows. In a random graph of n nodes with edge probability p (i.e. p is the probability that there is an edge connecting two nodes), the expected number of r-cliques is

$$X_r = \binom{n}{r} p^{\binom{r}{2}}$$

[Bol85]. For most values of n, there exists an integer r_0 such that X_{r_0} is much larger than one and X_{r_0+1} is less than one. The formula for computing this value is

$$r_0 = 2\log_b n - 2\log_b \log_b n + 2\log_b \frac{\mathrm{e}}{2} + 1 + o(1)$$

where $b = \frac{1}{p}$ [Bol85]. In other words, the value of r_0 is an approximation of the size of the largest clique in the graph.

We applied the formulas for X_r and r_0 to 17 test circuits taken from the MCNC Layout Synthesis Workshop (see Table 1). The probability p was replaced by the density of the graph representation of each circuit used by our clustering algorithm, where the density was computed as $p = \frac{E}{M_n}$ where E is the total number of edges with weight greater than 0.2 and $M_n = \binom{n}{2}$. Table 1 shows the values obtained for X_5, X_4, X_3, and r_0. As can be seen in the table, r_0 is usually around 4 or 5. Moreover, only one circuit is expected to contain any 5-cliques and only three

test circuit	total cells	total pads	total nets	total pins	X_r			r_0
					$r=5$	$r=4$	$r=3$	
IC67	52	15	138	474	3.658	107.535	567.422	5.689
IC116	101	14	329	876	0.000	2.021	133.486	4.521
IC151	136	15	419	987	0.000	0.285	65.844	4.192
8870	469	33	494	1541	0.000	0.000	38.194	3.806
bm1	752	131	902	2908	0.000	0.000	94.718	3.878
PrimGA1	752	81	902	2908	0.000	0.000	112.633	3.918
PrimSC1	752	81	902	2908	0.000	0.000	112.633	3.918
5655	801	120	760	2967	0.000	0.000	80.963	3.844
Test04	1489	26	1658	5975	0.000	0.000	89.628	3.791
Test03	1550	57	1618	5807	0.000	0.000	56.544	3.715
Test02	1602	61	1721	6135	0.000	0.000	2.248	3.644
Test06	1691	61	1674	6671	0.000	0.000	10.350	3.693
Test05	2540	55	2751	10,077	0.000	0.000	0.819	3.653
19ks	2684	161	3282	10,547	0.000	0.000	2.317	3.595
PrimGA2	2907	107	3029	11,219	0.000	0.000	8.077	3.771
PrimSC2	2907	107	3029	11,219	0.000	0.000	8.077	3.771
industry2	12,142	0	12,949	47,193	0.000	0.000	0.078	3.524

Table 1: Expected Number of r-cliques (X_r) and Approximate Size of Largest Clique (r_0)

circuits are expected to contain any 4-cliques. Most circuits are expected to contain a number of 3-cliques. However, since the formulas for X_r and r_0 are derived for random graphs and real circuits are usually more structured, our clustering algorithm always starts with searching for 5-cliques. Indeed, in all cases our algorithm successfully found a number of 5-cliques.

2.2 The Algorithm

The clustering algorithm consists of five major steps. First, the original netlist is converted to a graph representation. Then, the next three steps search for and form clusters from 5-cliques, 4-cliques, and 3-cliques in turn. Note that an r-clique ($3 \leq r \leq 5$) does not automatically become a cluster. It has to meet the size and density thresholds as discussed later. Finally, a post-processing step is performed to further reduce the number of *unclustered* nodes. Details of the algorithm are described in the following subsections.

2.2.1 Searching for Clusters

An iteration of one search algorithm constitutes a single pass through the entire graph. The search algorithms traverse a list of the edges and a list of the nodes in the graph. The node list contains both single nodes and clusters formed in previous iterations.

The 3-clique Search. For each edge e in the edge list, the node list is traversed until a node that is connected to both ends of e is encountered. The complexity of this algorithm is $O(m \cdot n)$ where m is the number of edges in the edge list and n is the number of nodes in the node list.

The 4-clique Search. For each edge e_1 in the edge list, the edge list is traversed until we find another edge e_2 disjoint from e_1 such that there are four edges connecting the four end nodes of e_1 and e_2. The complexity of this algorithm is $O(m^2)$.

The 5-clique Search. For each edge e_1 in the edge list, the edge list is traversed until another edge e_2 is found that forms a 4-clique with e_1 as in the 4-clique search algorithm. Then, the node list is traversed until we find a node that is connected to all four nodes in the 4-clique. The complexity of this algorithm is $O(m^2 \cdot n)$.

In practice, the degree of each node in the graph is bounded by a small constant, which is the number of pins in the component. Therefore, $m = O(n)$, and the complexity of the 3-clique, 4-clique, and 5-clique search algorithms is bounded by $O(n^2)$, $O(n^2)$, and $O(n^3)$, respectively. The runtimes of these algorithms can be reduced significantly using the parallel implementation presented in the next section.

The 5-clique search algorithm is always executed first, followed by the 4-clique search algorithm, and then the 3-clique search algorithm. Each search algorithm is repeatedly executed until it does not produce a sufficient number of clusters. Also, if after some iteration of the 5-clique search algorithm, the value of r_0 is less than five, then the 4-clique search algorithm is executed next to avoid unsuccessful searches for 5-cliques. (Recall that r_0 is an approximation of the size of the largest clique in the graph and is defined in Section 2.1.)

2.2.2 Cluster Thresholds

A set of nodes in an r-clique ($3 \leq r \leq 5$) does not necessarily form a cluster. In order to qualify as a cluster, the nodes and edges in an r-clique must satisfy two criteria: the area and size thresholds and the density threshold.

Area and Size Thresholds. The purpose of the area and size thresholds is to keep each cluster from becoming too large. The *area* of a cluster is the sum of the areas of the single nodes it contains. The *size* of a cluster is the total number of single nodes it contains. The *area threshold* is a percentage of the total area of the original graph, and the *size threshold* is a percentage of the total number of nodes in the original graph. When a clique is encountered, in order to become a cluster, its area must not exceed the area threshold, and its size must not exceed the size threshold. The area threshold used in our implementation was 25% of the total area of the original graph, and the size threshold was 33% of the number of nodes in the original graph.

Density Threshold. The purpose of the density threshold is to further ensure that the nodes in a cluster are strongly connected. It also prevents cliques introduced by multi-terminal nets from becoming clusters. The density of a cluster must be greater than or equal to the density threshold to be accepted. The *density threshold* is $\alpha_n \cdot D$ where α_n is a predetermined factor and D is the

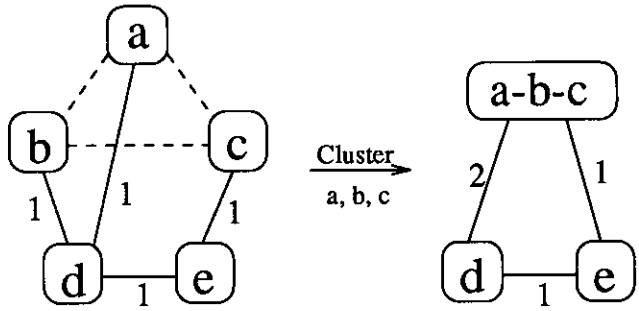

Figure 1: Clustering Nodes a, b, and c in a Graph

density of the graph representation of the original netlist. D is the ratio of the total edge weight to $\binom{n}{2}$ as described in Section 1.2.3 where n is the number of nodes in the graph representation of the original netlist. The value of α_n determines how much higher the density of the clusters must be than the density of the original graph. Since the density metric for measuring cluster quality is biased toward small clusters, higher values of α_n were used for the smaller test circuits and lower values α_n were used for the larger test circuits. In our implementation, $\alpha_n = 4.75$ for the four smallest test circuits ($n < 550$), $\alpha_n = 4.5$ for the eight middle-sized test circuits ($550 \leq n < 2000$), and $\alpha_n = 4.25$ for the five largest test circuits ($n \geq 2000$). Since large clusters tend to have lower density, the density threshold also helps to control the size of clusters.

2.2.3 Construction of Clusters

If a clique satisfies the area, size, and density thresholds, the nodes in the clique are collapsed to form a single cluster node. The edges that are internal to the clique are removed. For any node v outside of the cluster, all edges that connect v to nodes in the cluster are *bundled* together to form a new edge which connects the node v to the newly formed cluster node. In Figure 1, nodes a, b, and c are going to be clustered. The dotted edges are internal to the clique. After being clustered, a, b, and c have been collapsed into a single node, and the edges internal to the cluster are no longer present. The weight of the edge between the new cluster node and node d is 2 because node d was connected to both nodes a and b by edges with weight 1.

2.2.4 Post Processing

After clustering, a post-processing step is executed on the clustered graph to reduce the number of single, unclustered nodes. This helps to balance the sizes of the clusters and further reduce the number of nodes in the clustered graph. In this step, a weighted matching is performed on the clustered graph, and each qualified pair of matched nodes is collapsed into a single node in exactly the same way as a clique. To be qualified, the pair must satisfy the same area, size, and density thresholds as a clique. However, instead of searching for cliques one by one, the matching-based clustering collapses many pairs of nodes simultaneously. Like the clique search algorithms, the

weighted matching algorithm is repeatedly executed until an insufficient number of clusters are produced.

In our experimentation, there were two versions of the post-processing step. The first version considers the entire clustered graph. The second version considers only the single nodes in the clustered graph. In the second version, the first matching is performed only on the single nodes in the clustered graph, and the subsequent matchings are performed on both the single nodes and clusters that resulted from the previous matchings. In general, the first version results in fewer single nodes in the final clustered graph. In the remainder of this paper, we refer to clustering followed by the first version of the post-processing step as $C1$ and clustering followed by the second version as $C2$. The results of C1 and C2 are given in the next subsection.

The weighted matching algorithm used for the post-processing step is based on the $O(n^3)$ weighted matching algorithm of Gabow [Gab73] and was implemented by Ed Rothberg.

2.2.5 Clustering Results

Table 2 shows the clustering results obtained by C1 and C2 for 17 test circuits from the MCNC Layout Synthesis Workshop (see Table 1). In particular, the number of single, unclustered nodes (*singles*), the number of clusters (*clstrs*), the average cluster size in nodes (*avg*), and the size in nodes of the largest cluster (*lgest*) in the clustered graph are given in the table. Single nodes were not included when computing the average cluster size.

C1 consistently produces fewer single nodes, larger clusters, and smaller graphs. On average, C1 produces clustered graphs that are 6-13 times smaller than the original graphs, and C2 produces clustered graphs that are 2-7 times smaller than the original graphs. C1 was selected when implementing the parallel version of the clustering algorithm and when implementing cluster-based partitioning since it lead to slightly lower net cut sizes and ratio cut sizes.

2.3 The Parallel Clustering Algorithm

We have developed a parallel version of the clustering algorithm to reduce the runtime for large circuits. The basic idea of the parallel algorithm is to divide the graph among multiple processors. Each processor searches for and forms clusters in its portion of the graph. The processors occasionally swap part of their data to allow cliques that are divided among processors to be found. As the size of the graph is reduced by the clustering process, the number of processors involved decreases until there is only one processor. The coordination of the processors is controlled by a driver. The parallel algorithm is described in more detail below.

2.3.1 Parallel Algorithm Description

The driver converts the netlist into the graph representation and divides this graph evenly among the processors. After the processors have received all of the nodes and edges from the driver, they

test circuit	C1				C2			
	singles	clstrs	avg	lgest	singles	clstrs	avg	lgest
IC67	5	6	10.3	16	13	11	4.9	9
IC116	4	6	18.5	28	12	15	6.9	21
IC151	0	18	8.4	37	3	28	5.3	31
8870	55	17	26.3	154	75	43	9.9	106
bm1	85	37	21.5	222	107	59	13.1	161
PrimGA1	77	24	31.5	272	94	42	17.6	162
PrimSC1	77	24	31.5	272	94	42	17.6	162
5655	72	51	16.7	228	99	76	10.8	187
Test04	150	54	25.3	499	198	94	14.0	274
Test03	189	67	21.2	526	218	97	14.3	322
Test02	93	40	39.3	548	144	132	11.5	269
Test06	144	30	53.6	578	180	93	16.9	294
Test05	221	70	33.9	856	279	166	14.0	427
19ks	326	109	23.1	580	372	154	16.1	512
PrimGA2	280	127	21.5	985	295	160	17.0	705
PrimSC2	278	127	21.5	985	295	160	17.0	705
industry2	636	278	41.4	3035	1294	782	13.9	2256

Table 2: Clustering Results of C1 and C2

begin forming clusters. As in the sequential version, the processors search for and form clusters from cliques of size five, followed by cliques of size four, and then cliques of size three. Once a processor produces an insufficient number of clusters from cliques of the current size, it notifies the driver.

Once all active processors have notified the driver that they are finished clustering, a *swap* takes place. During the swap, the driver randomly pairs the processors. The driver directs each pair of processors to perform one of two types of data swaps. The first type is a *normal swap* in which each processor sends half of its nodes and the corresponding edges to the other processor. The second type of swap is a *collapsing swap* in which the processor with fewer nodes sends all of it nodes and edges to the other and becomes inactive. The type of swap to be performed depends on the number of nodes the pair of processors contains and the amount by which each processor has reduced its nodes since the time of the previous swap, i.e. the number of clusters the processor produced. In our implementation, a collapsing swap was performed if the pair of processors contained fewer than $\beta \times \frac{N}{P}$ nodes where β is a percentage, N is the number of nodes in the original graph, and P is the initial number of processors.

After a swap, the active processors resume clustering in their own subgraphs. The driver repeats the process of coordinating swaps until there is only one active processor left. The driver allows this processor to finish clustering and to perform the post-processing step.

2.3.2 Implementation

The parallel clustering algorithm was implemented using Maisie [BL90, BL91], a C-based parallel language that enables the algorithm to execute in parallel in a multi-processor environment. The *driver* and each *processor* are executed as processes by Maisie. A process is represented by an **entity** which is similar to a function in C. There is one *driver* **entity** and one *processor* **entity**. Execution begins in the *driver* **entity**. Then, the *processor* processes are started in the *driver* **entity** by executing the *processor* **entity** P times to start P *processor* processes.

Sending data from the driver to the processors and exchanging data among the processors require large amounts of data to be sent at one time. Sending large amounts of data at one time can be very slow due to the memory requirements of the message queue. During our experimentation, it was observed that sending the data in smaller pieces and allowing a number of pieces to be received (processed) before sending more data helped reduce communication time. For this reason, a limit was placed on the size of a message (40 nodes or edges) and on the number of messages that can be sent before the sender has to wait for an acknowledgement (25 messages).

2.3.3 Parallel Clustering Results

Table 3 shows the clustering results and computation times obtained using one, two, and four processors in the clustering algorithm for the 17 test circuits. The table gives the average number of nodes in the clusters, the average density of the clusters in the clustered graph, and the total computation time in seconds. Single, unclustered nodes are not included in the averages.

Table 3 shows that the average cluster sizes and densities obtained by the parallel algorithm are fairly similar to those obtained by the sequential version. The average cluster densities obtained by the 1, 2, and 4 processor versions for 13 of the 17 test circuits differ by less than 0.1. The sequential version of the clustering algorithm is deterministic. However, due to random selection of the nodes to send during a normal swap, the parallel version is non-deterministic. The fact that the parallel results do not vary greatly from the sequential results would seem to imply that the parallel version obtains clusterings that are as good as those obtained by the sequential version.

The computation times were recorded when executing on a network of Sun workstations connected by an ethernet. As the *driver* does not do much computation, it resides on the same Sun workstation as one of the *processor* processes. The computation times are in seconds and include time spent executing in both the user and system modes. The times do not include communication time, i.e. any time a process was sleeping while waiting to receive a message. The total computation time for all test circuits is give at the bottom of the table.

In general, the computation time for clustering is reduced significantly as the number of processors increases. For most of the circuits, there is a larger decrease in computation time when the number of processors is increased from one to two processors than when the number of processors is increased from two to four processors. This is due to the fact that as the number of processors increases the number of swaps increases. When there is a larger number of processors,

test circuit name	Parallel Clustering								
	1 processor			2 processors			4 processors		
	size	density	time	size	density	time	size	density	time
IC67	10.3	0.632	6.77	12.2	0.608	1.55	12.2	0.596	0.97
IC116	16.3	0.597	2.88	22.0	0.295	1.89	21.8	0.303	2.23
IC151	8.4	1.319	5.27	8.3	1.422	2.51	9.1	1.404	1.73
8870	32.4	0.533	21.30	26.3	0.622	11.85	38.8	0.509	9.94
bm1	21.5	0.686	96.30	26.4	0.709	40.98	22.7	0.705	26.84
PrimGA1	31.5	0.499	85.24	37.8	0.565	35.94	44.8	0.520	24.34
PrimSC1	31.5	0.499	85.49	39.6	0.526	36.86	44.7	0.529	25.37
5655	18.4	0.489	113.29	15.3	0.563	47.32	14.1	0.586	43.80
Test04	30.4	0.494	230.46	37.1	0.486	91.95	26.3	0.511	81.75
Test03	20.3	0.749	216.78	22.0	0.714	167.28	19.3	0.750	108.82
Test02	43.4	0.533	293.38	38.9	0.464	185.22	36.8	0.496	160.49
Test06	49.5	0.491	193.27	42.2	0.610	122.84	35.0	0.586	99.83
Test05	37.5	0.509	479.38	34.3	0.555	326.07	39.4	0.566	190.79
19ks	29.6	0.750	1110.40	24.6	0.722	375.48	29.3	0.795	386.36
PrimGA2	18.6	0.700	971.21	21.6	0.728	676.48	30.9	0.667	361.30
PrimSC2	18.6	0.700	971.66	22.2	0.730	623.82	26.3	0.681	459.68
industry2	41.4	0.670	24,387.66	35.7	0.633	20,069.30	32.9	0.613	11,110.77
Total Computation Time			29,270.74			22,817.34			13,095.01

Table 3: Parallel Clustering: Average Cluster Sizes and Densities and Total Computation Times

the processors find fewer clusters because each of them has less data. In this case, the processors have to swap data more often to allow more clusters to be found. In some extreme cases, such as the "19ks" circuit, the computation time of the 4-processor version is larger than that of the 2-processor version.

Aside from the fact that communication time is difficult to compute, the reason that communication time is not included in the table is that it is topology dependent. Our implementation uses a network of Suns connected by an ethernet, which is probably the worst-case topology since all message passing shares the same communication line, and the bandwidth of an ethernet is rather limited. Other topologies, such as a hypercube or a butterfly network, are designed to support more efficient communication between processors. During our experimentation, the total elapsed time (computation and communication time) for clustering was usually least when two processors were used and usually greatest when only one processor was used. The elapsed time was not least when using four processors due to the communication time for swaps over the network. We expect that the elapsed time would decrease considerably for the parallel algorithm if a multi-processor computer (such as a Sun SPARC-10) were used since the communication overhead would be reduced significantly.

3 Cluster-based Partitioning

3.1 The Basic Approach

The cluster-based partitioning algorithm uses the clustering algorithm as a preprocessing step for partitioning. Clustering is performed on the original graph, and then partitioning is performed on the clustered graph instead of the original graph. Afterwards, the partitioned clustered graph is *unclustered* without changing the partition. The areas of the two subsets formed by partitioning may not be as close to equal as desired due to the existence of large clusters in the clustered graph. Therefore, after unclustering it is usually necessary to refine the partition in order to further balance the areas of the subsets.

Instead of completely unclustering the clusters in one step, our cluster-based partitioning method gradually unclusters the clusters following the cluster hierarchy. After partitioning the clustered graph into subsets V_1 and V_2, we replace each cluster node by the r nodes (clusters) in the r-clique ($2 \leq r \leq 5$) used to form that cluster. The resulting subsets V_1' and V_2' are used as an initial partition for partitioning the next level of the cluster hierarchy. This process of gradual unclustering and partitioning is repeated for a predetermined number of times. Then, refinement is performed on the completely unclustered netlist. As the clusters gradually become smaller, the areas of the two partitions gradually become more balanced.

3.2 Applications to Existing Partitioning Algorithms

We have incorporated two existing partitioning algorithms into our cluster-based partitioning method. One is the bisection algorithm developed by Fiduccia and Mattheyses (the FM algorithm) [FM82], and the other is a ratio-cut algorithm developed by Wei and Cheng (the RFM algorithm) [WC89]. Both of these are iterative improvement algorithms.

3.2.1 The FM Algorithm

The FM algorithm is an improvement of the partitioning algorithm by Kernighan and Lin [KL70]. The FM algorithm starts with a balanced initial partition. At each step, one component is selected to move to the other side of the partition. Once a component is moved, it is *locked* for the remainder of the current pass. A component is infeasible for moving if moving it to the other side of the partition violates the balance constraint. The component to move is selected from among the *unlocked* feasible components with the highest gain, where the *gain* of a component is defined to be the amount that the cut size would be reduced by moving that component to the other side of the partition. If moving a component would increase the cut size, the gain of that component is negative. Moving components with negative gain is allowed by the algorithm in order to avoid stopping at local minima in the solution space. When all components are either locked or infeasible for moving, the current pass is complete and the best partition encountered during the pass is saved as the initial partition for the next pass. When a pass makes no improvement

to the partitioning solution, the algorithm stops.

3.2.2 The RFM Algorithm

The RFM algorithm minimizes the ratio-cut metric and is based on the FM algorithm [WC89]. The ratio-cut metric combines the goals of minimizing the cut size and balancing the areas into a single objective function. The *ratio cut size* is the ratio of the cut size to the product of the two subset areas, i.e. $RC(V_1, V_2) = \frac{c(V_1,V_2)}{area(V_1) \cdot area(V_2)}$. The RFM algorithm does not require the areas of the two subsets to satisfy any balance constraints. As in the FM algorithm, at each step, the RFM algorithm selects the component to move from among the unlocked components with the highest gain. If there is a tie, the RFM algorithm selects the component that would cause the greatest decrease in the current ratio cut size.

3.3 Partitioning Results

The clustering portion of the cluster-based partitioning method was implemented in Maisie a C-based parallel language [BL90, BL91] as described in Section 2.3.2. The FM and RFM partitioning algorithms were implemented in C as described in [FM82] and [WC89]. Experiments were executed on a network of Sun workstations.

We compare the results obtained by our cluster-based partitioning method to the results obtained using the partitioning algorithms directly on the 17 test circuits. *FM* and *RFM* refer to the FM and the RFM algorithms when applied directly (without clustering) to the original netlist. *FMC* and *RFMC* refer to the corresponding cluster-based partitioning.

Table 4 compares the best and average cut sizes for FM and FMC and compares the best and average ratio cut sizes for RFM and RFMC. The best and average (ratio) cut sizes are obtained from 10 executions. On average, the best cut size for FMC is 18.2% lower than the best cut size for FM, and the average cut size for FMC is 24.7% lower than the average cut size for FM. In fact, for 12 of the 17 test circuits, the *average* cut size produced by FMC is lower than the *best* cut size produced by FM. On average, the best ratio cut size for RFMC is 29.7% lower than the best ratio cut size for RFM, and the average ratio cut size for RFMC is 44.3% lower than the average ratio cut size for RFM. And again, for 12 of the 17 test circuits, the *average* ratio cut size produced by RFMC is lower than the *best* ratio cut size produced by RFM. These results suggest that our cluster-based partitioning algorithm produces much more stable results than the FM and RFM algorithms.

Tables 5 and 6 show the partitioning results obtained when using the parallel clustering algorithm with up to four processors. For FMC, the cut sizes produced by the 1-processor, 2-processor, and 4-processor implementations are on average lower than the cut sizes for FM by 18.2%, 18.4%, and 16.9%, respectively. For RFMC, the ratio cut sizes produced by the 1-processor, 2-processor, and 4-processor implementations are on average lower than the ratio cut sizes for RFM by 29.7%, 32.8%, and 26.7%, respectively. These results further confirm that our parallel clustering algo-

test circuit	Best Cut		Average Cut		Best Ratio Cut		Average Ratio Cut	
	FM	FMC	FM	FMC	RFM	RFMC	RFM	RFMC
IC67	37	33	39.4	34.5	1.54E-02	1.52E-02	2.64E-02	1.52E-02
IC116	28	28	28.0	28.0	8.55E-03	8.58E-03	8.55E-03	8.58E-03
IC151	50	49	51.0	50.2	6.71E-03	6.71E-03	7.63E-03	6.71E-03
8870	15	15	23.1	15.8	5.98E-05	2.50E-05	8.80E-05	3.40E-05
bm1	73	51	86.2	67.4	1.75E-05	6.20E-06	2.58E-05	7.49E-06
PrimGA1	72	63	79.5	63.0	2.31E-05	1.29E-05	2.72E-05	1.29E-05
PrimSC1	70	54	83.3	62.1	3.22E-05	2.05E-05	4.03E-05	2.05E-05
5655	62	53	67.6	56.8	5.97E-06	8.50E-06	9.88E-06	8.50E-06
Test04	49	42	63.3	43.4	9.96E-08	9.57E-08	1.03E-07	9.92E-08
Test03	102	63	111.8	70.9	8.74E-07	3.78E-07	1.05E-06	4.75E-07
Test02	134	81	159.5	87.0	5.68E-08	5.68E-08	1.28E-07	7.18E-08
Test06	59	65	76.3	67.6	1.49E-06	9.79E-07	1.75E-06	1.04E-06
Test05	75	44	111.4	46.0	3.25E-08	3.79E-08	4.94E-08	4.20E-08
19ks	157	173	192.6	182.6	5.22E-06	3.43E-06	7.45E-06	3.43E-06
PrimGA2	215	136	220.7	175.7	1.47E-05	5.20E-06	2.10E-05	5.20E-06
PrimSC2	200	137	275.3	181.4	2.62E-05	6.08E-06	2.62E-05	6.08E-06
industry2	432	281	749.4	395.2	1.36E-05	1.41E-06	2.19E-05	2.76E-06
Avg. Reduction	18.2%		24.7%		29.7%		44.3%	

Table 4: Best and Average Cut Sizes and Ratio Cut Sizes

rithm produces high-quality clusterings.

4 Conclusion

We have presented a bottom-up clustering algorithm that forms clusters by recursive collapsing of cliques in a graph. Forming clusters based on cliques combined with the use of the density threshold produces high-quality clusters composed of strongly-connected components. Our clustering algorithm reduces the size of a circuit significantly so that existing layout algorithms can be applied to obtain efficient solutions. We have also presented a parallel version of the clustering algorithm which allows multiple processors to form clusters in different parts of the netlist simultaneously. Our results show that increasing the number of processors used in our clustering algorithm reduces the required computation time and memory space without affecting the clustering quality.

Finally, we have presented a cluster-based partitioning method which combines our clustering algorithm with the FM and RFM algorithms. Our cluster-based partitioning method obtained cut sizes that were on average 18% better than those obtained by the FM algorithm, and obtained ratio cut sizes that were on average 30% better than those obtained by the RFM algorithm. Moreover, our cluster-based partitioning method produces results that are much more consistent than those of the two partitioning algorithms. When multiple runs were performed for each algorithm, our

test circuit name	FM	FMC					
		1		2		4	
	cut	cut	% red	cut	% red	cut	% red
IC67	37	33	10.8	34	8.1	33	10.8
IC116	28	28	0.0	28	0.0	28	0.0
IC151	50	49	2.0	49	2.0	50	0.0
8870	15	15	0.0	15	0.0	17	-13.3
bm1	73	51	30.1	68	6.9	46	37.0
PrimGA1	72	63	12.5	66	8.3	56	22.2
PrimSC1	70	54	22.9	52	25.7	50	28.6
5655	62	53	14.5	52	16.1	58	6.5
Test04	49	42	14.3	44	10.2	44	10.2
Test03	102	63	38.2	72	29.4	64	40.2
Test02	134	81	39.6	81	39.6	87	37.3
Test06	59	65	-10.2	55	6.8	90	-11.9
Test05	75	44	41.3	45	40.0	45	41.3
19ks	157	173	-10.2	133	15.3	141	3.2
PrimGA2	215	136	36.7	138	35.8	146	34.9
PrimSC2	200	137	31.5	145	27.5	143	18.5
industry2	432	281	35.0	256	40.7	339	21.5
Average Reduction			18.2%		18.4%		16.9%

Table 5: Cut Sizes with Parallel Clustering

method obtained average cut sizes up to 58.7% better than those of the FM algorithm, and obtained average ratio cut sizes up to 87.4% better than those of the RFM algorithm. In fact, for many of the test circuits the *average* cluster-based partitioning result was better than the *best* direct partitioning result.

Extension to Placement. Our clustering algorithm can also be applied to the large scale placement problem. The basic approach of cluster-based placement would be similar to that of cluster-based partitioning. After clustering, the clusters are first placed on the layout surface. Then, as in cluster-based partitioning, the clusters are gradually unclustered, and placement is performed within each cluster on the sub-clusters that result from one level of unclustering. After placing the sub-clusters, placement is performed within the sub-clusters. This process can be repeated a number of times until every component has been placed. Placement within the clusters could take place in parallel. Finally, a global refinement can be performed to further improve the placement solution. We believe that this cluster-based placement method can handle designs of very high complexity and produce high-quality placement solutions.

test circuit name	RFM	RFMC					
		1		2		4	
	ratio	ratio	% red	ratio	% red	ratio	% red
IC67	1.54E-02	1.52E-02	1.5	1.52E-02	1.5	1.52E-02	1.5
IC116	8.55E-03	8.58E-03	-0.4	8.58E-03	-0.4	8.58E-03	-0.4
IC151	6.71E-03	6.71E-03	0.0	6.71E-03	0.0	6.71E-03	0.0
8870	5.98E-05	2.50E-05	58.3	2.50E-05	58.3	3.55E-05	40.7
bm1	1.75E-05	6.20E-06	64.6	6.20E-06	64.6	6.20E-06	64.6
PrimGA1	2.31E-05	1.29E-05	44.3	1.17E-05	49.5	9.68E-06	58.1
PrimSC1	3.22E-05	2.05E-05	36.3	1.10E-05	65.8	1.54E-05	52.2
5655	5.97E-06	8.50E-06	-42.4	8.50E-06	-42.4	8.50E-06	-42.4
Test04	9.96E-08	9.57E-08	3.9	9.54E-08	4.2	9.54E-08	4.2
Test03	8.74E-07	3.78E-07	56.7	3.78E-07	56.7	5.67E-07	35.1
Test02	5.68E-08	5.68E-08	0.0	5.68E-08	0.0	7.66E-08	-35.0
Test06	1.49E-06	9.79E-07	34.2	9.51E-07	36.1	9.87E-07	33.6
Test05	3.25E-08	3.79E-08	-16.5	3.56E-08	-9.4	3.56E-08	-9.5
19ks	5.22E-06	3.43E-06	34.2	2.61E-06	50.0	3.43E-06	34.2
PrimGA2	1.47E-05	5.20E-06	64.7	5.73E-06	61.1	6.08E-06	58.7
PrimSC2	2.62E-05	6.08E-06	76.8	7.18E-06	72.6	6.98E-06	73.4
industry2	1.36E-05	1.41E-06	89.6	1.41E-06	89.6	2.02E-06	85.1
Average Reduction			29.7%		32.8%		26.7%

Table 6: Ratio Cut Sizes with Parallel Clustering

5 Acknowledgements

The Maisie programming language was developed by R. Bagrodia and W. Liao at UCLA [BL90, BL91]. We are grateful for the funding from the Graduate Opportunity Fellowship and the Graduate Student Research Assistantship provided by UCLA. This work is partially supported by a grant from Cadence under the California MICRO program.

References

[BHJL89] T. Bui, C. Heigham, C. Jones, and T. Leighton. Improving the Performance of the Kernighan-Lin and Simulated Annealing Graph Bisection Algorithms. *26th ACM/IEEE DAC*, pages 775–778, 1989.

[BL90] R.L. Bagrodia and W. Liao. Maisie: A language and optimizing environment for distributed simulation. In *Proc. of 1990 SCS Multiconference on Distributed Simulation*, pages 205–210, San Diego, CA, Jan. 1990.

[BL91] R.L. Bagrodia and W. Liao. Maisie User Manual. Technical report, Computer Science Department, UCLA, Los Angeles, CA 90024, Oct. 1991.

[Bol85] B. Bollobas. *Random Graphs.* Academic Press, London, 1985.

[CHK91] J. Cong, L. Hagen, and A. B. Kahng. Random Walks for Circuit Clustering. In *Proc. IEEE Intl. Conf. on ASIC*, pages 14.2.1–14.2.4, June 1991.

[CHK92] J. Cong, L. Hagen, and A. B. Kahng. Net Partitions Yeild Better Module Partitions. In *Proc. ACM/IEEE Design Automation Conf.*, 1992.

[CP68] H.R. Charney and D.L. Plato. Efficient Partitioning of Components. In *Proc. of the 5th Annual Design Automation Workshop*, pages 16–0 to 16–21, 1968.

[Don88] W. E. Donath. Logic Partitioning. In *Physical Design Automation of VLSI Systems*, B. Preas and M. Lorenzetti, editors, pages 65–86. Benjamin/Cummings, 1988.

[FM82] C.M. Fiduccia and R.M. Mattheyses. A Linear-Time Heuristic for Improving Network Partitions. In *Proc. 19th Design Automation Conference*, pages 175–181, 1982.

[Gab73] H. Gabow. *Implementation of Algorithms for Maximum Matching on Nonbipartite Graphs.* PhD thesis, Stanford University, 1973.

[GPS90] J. Garbers, H.J. Promel, and A. Steger. Finding Clusters in VLSI Circuits. *ICCAD'90*, pages 520–523, 1990.

[HK72] M. Hanan and J.M. Kurtzberg. A Review of the Placement and Quadratic Assignment Problems. *SIAM Review*, 14:324–342, 1972.

[HK91] L. Hagen and A. B. Kahng. Fast Spectral Methods for Ratio Cut Partitioning and Clustering. In *Proc. IEEE Intl. Conf. on Computer-Aided Design*, pages 10–13, 1991.

[HK92] L. Hagen and A. B. Kahng. A New Approach to Effective Circuit Clustering. In *Proc. IEEE Intl. Conf. on Computer-Aided Design*, Santa Clara, Nov. 1992.

[KJV83] S. Kirkpatrick, C. Gelatt Jr., and M. Vecchi. Optimization by Simulated Annealing. *Science*, 220:671–680, May 13 1983.

[KL70] B.W. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. *Bell System Technical Journal*, 49:291–307, Feb. 1970.

[Len90] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout.* John Wiley & Sons, Chichester, 1990.

[WC89] Y.C. Wei and C.K. Cheng. Towards Efficient Hierarchical Designs by Ratio Cut Partitioning. In *Proc. IEEE Intl. Conf. on Computer-Aided Design*, pages 298–301, 1989.