Survey of Partitioning Techniques in Silicon Compilation

by

Allen C. H. Wu

Technical Report 91-15

Information and Computer Science Department
University of California, Irvine
Irvine, CA. 92717

Abstract

In the silicon compilation design process, partitioning is usually the first problem to be investigated because partitioning algorithms form the backbone of many algorithms including: system synthesis, processor synthesis, floorplanning, and placement. In this survey, several partitioning techniques will be examined. In addition, this paper will review the partitioning algorithms used by synthesis systems at different design levels.

TABLE OF CONTENTS

LIST OF FIGURES

1. Introduction

Silicon compilation was introduced by Dave Johannsen [Joha79] for assembling parameterized modules of layout. The term **silicon compilation** can be extended to define a translation process that starts with a behavioral description and ends with a set of fabrication masks.

To represent the design process of silicon compilation, Gajski and Kuhn [Gajs83] introduced a Y-chart representation of silicon compilation (Figure 1). In the Y-chart, the design process is classified into three different dimensions: (i) **behavioral**, (ii) **structural**, and (iii) **geometrical**. The translation from behavior to structure is called synthesis. There are four levels in the synthesis processes: (i) **system**, (ii) **processor**, (iii) **module**, and (iv) **cell** [Gajs88]. In system synthesis, the program is decomposed into a set of communicating processes. In processor synthesis, each process is decomposed into a set of microarchitecture modules. In module synthesis, each module is then generated into a set of cells. Finally, in module synthesis, each cell is decomposed into gates, transistors, and fabrication masks. To obtain information about the physical domain of a design, each synthesis is followed by a translation from the structure into geometry. This translation is called floorplanning, placement and routing, or cell generation.

In the silicon compilation design process, partitioning is the first problem to be investigated because partitioning algorithms form the backbone of many algorithms, including those for system synthesis, processor synthesis, floorplanning, and placement. For example, in system synthesis, designers partition the design into a set of chips according to constraints such as power dissipation, number of pins per chip, chip size, and speed. In processor synthesis, designers decompose chips into a set of datapaths

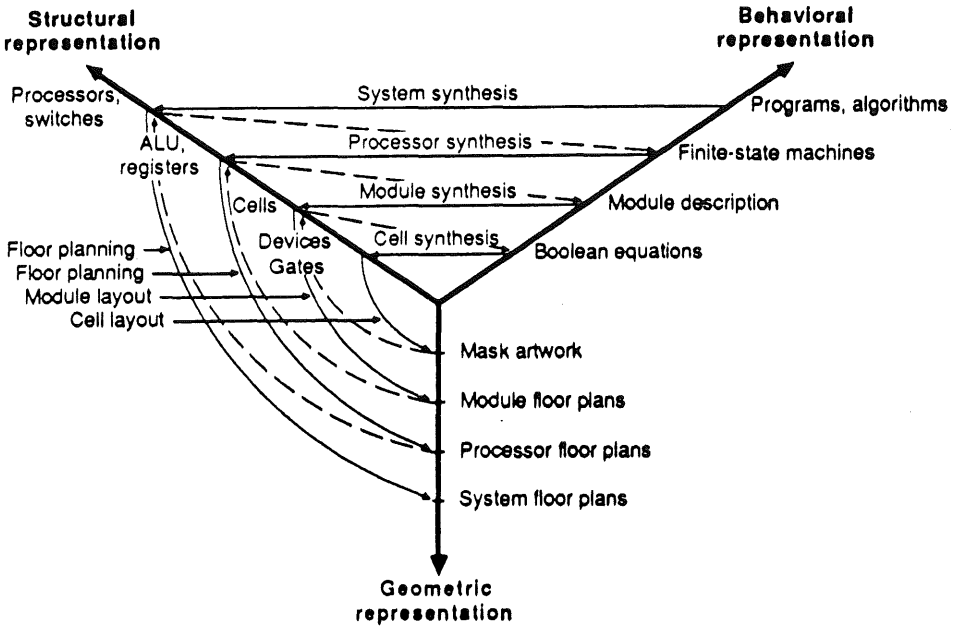


Figure 1. The Y-chart representation of silicon compilation.

and control units according to different layout architectures and constraints. At the placement level, partitioning is used to decompose thousands of objects (gates) into manageable clusters that can decrease the complexity of placement. If the partitioning is done effectively, the design process is simplified without sacrificing the overall performance.

In this survey, several partitioning techniques will be examined. In addition, this paper will review the partitioning algorithms used by synthesis systems at different design levels. Section 2 describes the definition of the partitioning problem. Section 3 presents basic partitioning techniques and algorithms. Section 4 describes the

algorithms that have been used in different synthesis systems. Section 5 contains the conclusion.

2. Definition of Problem

Partitioning is the task of decomposing a given set of objects into subsets so that (i) each subset contains objects that equal to or less than the given size constraints and (ii) the wire connections crossing subsets are minimized.

Let

$\mathbf{G} = (\mathbf{V},\mathbf{E})$ where $\mathbf{V}=\{\mathbf{v}_i\}$, $\mathbf{i}=1..\mathbf{n}$, be a set of nodes and $\mathbf{E}=\{\mathbf{e}_j\}$, $\mathbf{j}=1..\mathbf{m}$, be a set of edges;

$V_p=\{\mathbf{v}_i\}$ be a subset p of nodes;

$\mathbf{S}(V_p)$ be the number of nodes in subset p and $\mathbf{S}(\mathbf{V})$ be the total number of nodes in $\mathbf{V}$;

$\mathbf{C}(V_p)$ be the size constraint of V_p in terms of number of nodes;

$\mathbf{e}_{ni,mj}$ is the net that connects node $\mathbf{n}$ in V_i to node $\mathbf{m}$ in V_j.

The problem of a $\mathbf{k}$-way partition of $\mathbf{G}$ is to partition $\mathbf{V}$ into $\mathbf{k}$ subsets, $\{V_1,V_2,\ldots,V_k\}$ which can be formulated as follows:

$$\sum_{i=1}^{k} \mathbf{S}(V_i) = \mathbf{S}(\mathbf{V}),\ V_i \bigcap V_j = \phi,\ i \neq j, \textbf{ subject to } \mathbf{C}(V_p) \leq \mathbf{S}(V_p)$$

and **minimizing** $\sum_{n \epsilon V_i,\ m \epsilon V_j} \mathbf{e}_{ni,mj}$

The partitioning problem was shown to be NP-complete by [Gare79]. Consider the problem of partitioning graph $\mathbf{G}$ of $\mathbf{n}$ nodes into $\mathbf{k}$ subsets of equal size $\mathbf{m}$, where $\mathbf{km=n}$. There are $\binom{n}{m}$ ways of choosing the first subset, $\binom{n-m}{m}$ ways for the second, and so on. The number of choices for such a partition is:

$$\frac{1}{k!}\binom{n}{m}\binom{n-m}{m}\cdots\binom{2m}{m}\binom{m}{m}$$

For example, for n=30, m=10, and k=3, 10^{12} computations are required to perform exhaustive search. Because the computation time grows at an exponential rate (with the size of partitioning problem), it is impractical to perform an exhaustive search to find the optimal partition. Over the years, numerous heuristic methods have been proposed for solving the partitioning problem. Often in practical applications, heuristic methods can produce good results in a reasonable amount of time.

3. Techniques and Algorithms

3.1. Overview of Partitioning Methods

There are two basic methodologies: (i) **The constructive method** and (ii) **The iterative improvement method** [Sang87, Dona88, PrKa88]. The constructive method uses a clustering (aggregation) strategy [Ever74,Spat80,Rome84] to assign one node at a time to a partition. The cluster growth process is based on closeness measurements of nodes, such as functions and interconnections. Several clustering approaches are reported in [Kurt65, HaKu72, Kodr72, ScUl72, Schw76, CoCa80, Kang83, OdHa87]. A variation of the traditional clustering algorithm is hierarchical clustering [Snea57,John67]. Hierarchical clustering uses different closeness measurements to group the objects into clusters at different levels. Using hierarchical clustering, a cluster tree is formed for further analysis.

The iterative improvement method starts with some initial partition, then improves the results by moving nodes between partitions. There are three major iterative improvement algorithms: (i) Pairwise interchange, (ii) Group migration, and (iii) Simulated annealing.

3.2. Constructive Methods

3.2.1. Cluster Growth

Cluster growth is a constructive method that does not need a given initial partition. This approach starts with a nonpartitioned set and operates by selecting unplaced objects and adding them to the proper clusters. Consider partitioning a set S of **n** nodes into two subsets $\{S_1, S_2\}$ such that $|S_1| = |S_2| = \frac{n}{2}$. Assume that the sizes of nodes and weights of nets are equal. The cost function according to a given partition of $\{S_1, S_2\}$ is defined as:

$$C(S) = \sum_{\substack{n \epsilon S_1 \\ m \epsilon S_2}} e_{n1,m2}$$

where $e_{n1,m2}$ is the net that connects node **n** in S_1 to node **m** in S_2

The first step of cluster growth is to select the seed of the partitions. The seed nodes may be chosen randomly, by the user, or determined algorithmically [Schw76,WuGa90] in order to guide the partitioning process. Next, unplaced nodes are selected according to cost function **C(n)**, where **n** is an unplaced node. The internal cost **I(n)** denotes the number of nets between node **n** in S_1 and other nodes in S_1. The external cost **E(n)** denotes the the number of nets between node **n** in S_1 and other nodes in S_2. **C(n)** is the difference between external and internal costs of node **n** such that **C(n) = E(n) - I(n)**. The unplaced node with minimal **C(n)** is chosen and clustered into S_1. The process is repeated until the number of nodes assigned to S_1 reaches its size constraint. Finally, all of the unplaced nodes are assigned to S_2.

The cluster growth algorithm is easy to implement and fast. However, each clustering decision must be made based on the current existing cluster information without taking into account the global consideration. Therefore, it often produces poor results. The cluster growth algorithm is mostly used as an initial partition for an iterative improvement method.

3.2.2. Hierarchical Clustering

In hierarchical clustering [Snea57,John67], the objects are clustered into groups. The process will be executed repeatedly at different levels to form a tree. Hierarchical techniques can be divided into two methods: (i) agglomerative and (ii) divisive.

The agglomerative method proceeds by successively fusing objects into groups. In general, cluster analysis consists of three steps: (i) computing the closeness matrix, (ii) executing the clustering method, and (iii) rearranging the closeness matrix. First, a closeness matrix $\mathbf{X} = (x_{ij})$ is formed, $i=j=1..n$, where x_{ij} is the closeness coefficient between object **i** and **j** (Figure 2(a)). Since the closeness between two objects is symmetric, only the lower-left half of the matrix contains values. Second, the procedure fuses individuals or groups of objects which are most similar or closest (Figure 2(b)). Third, the closeness matrix is rearranged according to the new cluster configuration. Steps (ii) and (iii) are executed repeatedly until no more clusters can be merged (Figure 2(c)). Using this method, the clustering tree is formed in a bottom-up fashion.

The divisive method partitions the set of objects into clusters. The first task of the divisive method is to split the initial set of objects into two sub-sets. For a set of **n** objects, there are $2^{n-1}-1$ possible ways to divide **n** objects into two sub-sets. It is impractical to perform exhaustive procedure to find the partitions. One of the most

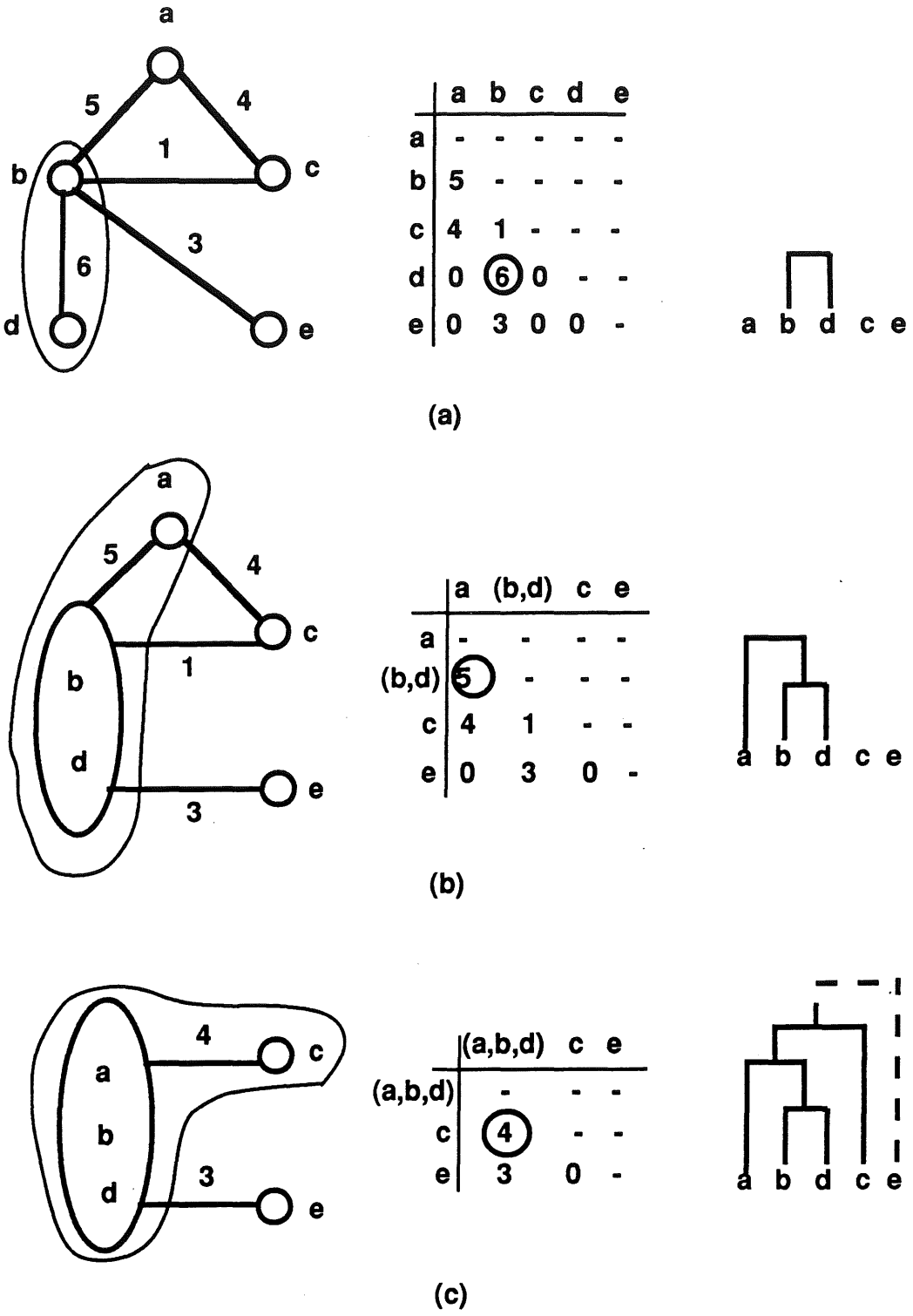


Figure 2. The cluster tree formation

feasible of divisive techniques was proposed by MacNaughton-Smith et al. [MacNa64]. MacNaughton-Smith introduced a technique of dissimilarity analysis which measures the dissimilarity between each object and the other objects in the group.

Consider a closeness matrix $D=\{d_{ij} \mid i=1..n, j=1..n\}$ where **n** is the number of objects and d_{ij} is the closeness between object **i** and **j**. For example in Figure 3(a), there are 7 objects to be divided into two groups according to the dissimilarity analysis. The initial partition is based on the average closeness between each object with the remaining objects. For example, the average closeness between groups {1} and {2,3,4,5,6,7} is calculated as (10+7+30+29+38+42)/6=26. The average closeness between objects 1, 2, 3, 4, 5, 6, and 7, and {2,3,4,5,6,7}, {1,3,4,5,6,7},..,{1,2,3,4,5,6}, are 26, 22.5, 20.7, 17.3, 18.5, 22.2, and 25.5 respectively. Thus, the initial two groups are {1} and {2,3,4,5,6,7}. Next, the average closeness of objects in the two groups are calculated as shown in Figure 3(b). For example, consider object #2 at row 1 in Figure 3(b), the average closeness between {1} and {2} is 10 and between {2} and {3,4,5,6,7} is (7+23+25+34+36)/5=25. The difference of merging {2} with {1} and {3,4,5,6,7} is 15. After calculating the average closeness, the maximum difference is 16.4 for object 3. Therefore, object 3 is merged with object 1. The new groups are {1,2} and {3,4,5,6,7}. Repeating the analysis gives the result based on groups {1,3,2} and {4,5,6,7} as shown in Figure 3(c). As all the differences are now negative, the partitioning into two sub-sets is completed. The process continues until no more objects can be split. Using this method, the clustering tree is formed in a top-down fashion.

A variation of hierarchical clustering is multi-stage clustering [DiTh89]. Multi-stage clustering was proposed by Dirkes Lagnese and Thomas for solving the large scale

$$D = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 0 & 10 & 7 & 30 & 29 & 38 & 42 \\ 10 & 0 & 7 & 23 & 25 & 34 & 36 \\ 7 & 7 & 0 & 21 & 22 & 31 & 36 \\ 30 & 23 & 21 & 0 & 7 & 10 & 13 \\ 29 & 25 & 22 & 7 & 0 & 11 & 17 \\ 38 & 34 & 31 & 10 & 11 & 0 & 9 \\ 42 & 36 & 36 & 13 & 17 & 9 & 0 \end{bmatrix}$$

(a)

Individual	Average closeness (1)	(2)	(2-1)
2	10.0	25.0	15.0
3	7.0	23.4	16.4
4	30.0	14.8	-15.2
5	29.0	16.4	-12.6
6	38.0	19.0	-19.0
7	42.0	22.2	-19.8

(b)

Individual	Average closeness (1)	(2)	(2-1)
4	24.3	10.0	-14.3
5	25.3	11.7	-13.6
6	34.3	10.0	-24.3
7	38.0	13.0	-25.0

(c)

Figure 3. Divisive clustering method.

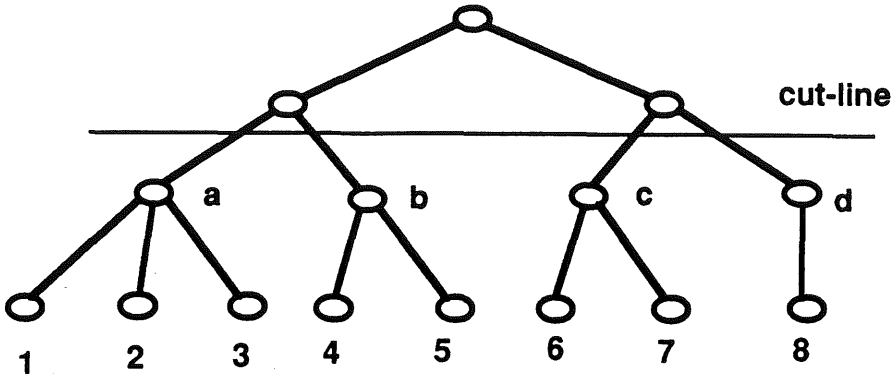
problem of architectural partitioning. This approach performs clustering processes in several stages. Each clustering stage is allowed to use a different closeness criteria. The clustering stages are consecutive and each stage builds on the results of the previous stage. A two-stage clustering example is shown in Figure 4. In the first clustering stage, the objects are clustered together according to their closeness measurements on criterion **A**. As a result, it produces 4 clusters indicated as **a**, **b**, **c**, and **d** in Figure 4(a). In the second stage, the clusters **a**, **b**, **c**, and **d** cut from the first stage are clustered further according to closeness criterion **B** (Figure 4(b)).

The multi-stage clustering approach has two advantages over the traditional clustering approach. The first advantage is that this approach decouples the clustering criteria over several stages. Hence, it provides a hierarchy of criteria that can apply the most important criterion first to ensure that the constraints are satisfied. However, it is difficult to determine the proper weighting for the various criteria to ensure good clustering results. The second advantage is that this approach allows objects to be considered as groups rather than as individual objects. Thus, this approach can cluster objects using more global considerations.

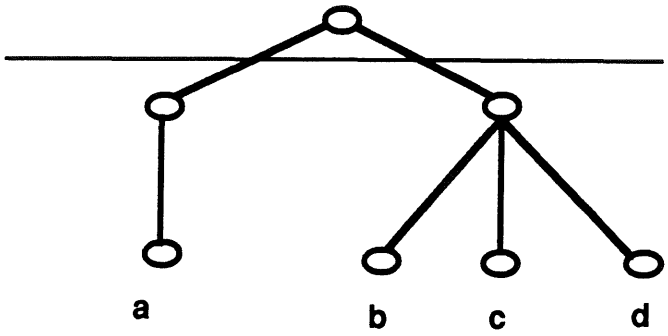
3.3. Iterative Improved Methods

3.3.1. Pairwise interchange

In the pairwise interchange algorithm, a pair of nodes are selected from different partitions. A cost function determines the effect of interchanging the two nodes. If the partitioning cost function is improved, then these two nodes are interchanged. Otherwise, the nodes remain in their previous partitions. This algorithm results in n(n-1)/2 trial exchanges which contributes $O(n^2)$ complexity, where **n** is the number of



(a). Clustering with criterion A



(b). Clustering with criterion B

Figure 4. Multi-stage clustering.

nodes. Several layout systems [HaWo76, Schw76, IoKi83] have used a pairwise interchange algorithm for placement. Pairwise interchange is a simple heuristic method, and it is not guaranteed to find even a local optimal solution.

3.3.2. Group migration

The group migration algorithm is also known as the Kernighan-Lin algorithm [KeLi70]. The Kernighan-Lin algorithm uses the solution of two-way uniform partitions as the basis for solving general partitioning problems. The basic idea is to interchange the group of nodes that contribute the maximal partitioning improvement between two groups. The Kernighan-Lin algorithm is guaranteed to find a local optimal solution. Several group migration algorithms have been reported in [FiMa82, Kris84, ScKe72, BhHi88, SaRa90]. Because group migration algorithms can produce excellent results using a small amount of CPU time (linear complexity), this algorithm is widely used in many applications.

Another approach used to solve the partitioning problem is network flow algorithms [ChKu84, WeCh89]. This algorithm is based on the Ford and Fulkerson maximum flow minimum cut algorithm [FoFu62] for finding a minimum cut between two partitions in a network. The major difficulty of using this algorithm is the inability to constrain the cut-set sizes. In practice, this algorithm usually generates subsets with greatly uneven sizes and hence its applications are limited. However, it does find the minimal cost for unconstrained (without size constraints for subsets) two-way partitions that can be used as lower bound for solutions produced by any partitioning method.

3.3.2.1. The Kernighan-Lin Min-Cut Algorithm

The original Kernighan-Lin algorithm finds a minimal-cost partition of a given graph of **n** nodes connected by edges into two equal subsets of **n/2** nodes (Figure 5). This two-way uniform partitioning algorithm uses heuristics to obtain the minimal uniform partition. It interchanges the nodes in cut-set A and cut-set B, and then

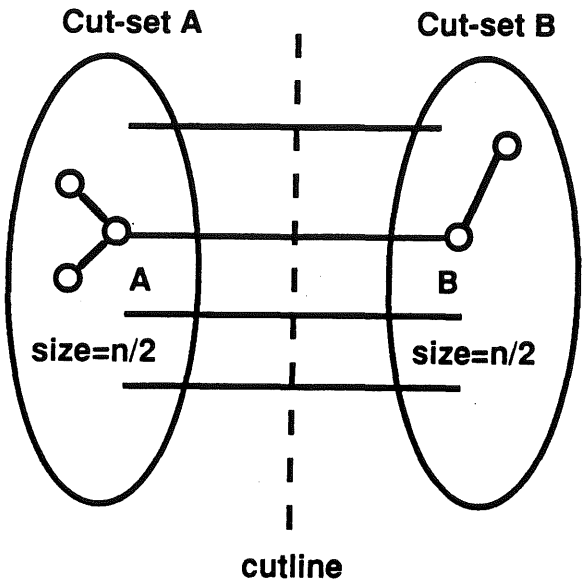


Figure 5. Two-way partition.

performs a local search to find a sequence of favorable swaps of nodes from the two subsets.

Consider the partitioning of a set S of size **n** into two subsets $\{S_1, S_2\}$ with equal size of **n**/2. e_{ab} denotes the net connected between node a and node b. Let us define the external cost E_a where $a \in S_1$ by

$$E_a = \sum_{x \epsilon S_2} e_{ax}$$

and the internal cost by

$$I_a = \sum_{y \epsilon S_1} e_{ay}$$

Similarly, define E_b, I_b for each $b \in S_2$. Let

$$D_i = E_i - I_i$$

be the difference between external and internal cost for all $i \in S$.

Let C_{ab} denote a double counting correction coefficient if node **a** and node **b** connect to the same net. The partition gain ,$gain_{ab}$, by interchanging nodes **a** and **b** is

$$gain_{ab} = D_a + D_b - 2C_{ab}$$

The idea of group migration is to search for a favorable group of swaps rather than to search for one favorable swap. The Kernighan-Lin algorithm first generates a sequence of gains as follows:

(1) Calculate D_i for all nodes $\in$ S.

(2) Choose the pair (a, b), $a \in S_1$ and $b \in S_2$, that generates the maximal gain.

(3) Swap **a** and **b** and recompute the **D** values for all unswapped nodes.

(4) Repeat steps 2 and 3, obtaining a sequence of swapped pairs $\{a_1,b_1\},...,\{a_n, b_n\}$. Once a pair is swapped, this pair will be locked and can not be considered for swapping again.

As a result, the algorithm generates a sequence of gains $gain_1,...,gain_n$. The total gain from interchanging the set $A = \{a_1,...,a_k\}$ with $B = \{b_1,...,b_k\}$ is

$$GAIN(k) = \sum_{i=1}^{k} gain_i$$

Next, the algorithm uses local search to choose a **k** that maximizes the partial sum of **GAIN(k)**. If **GAIN(k)**>0, the algorithm interchanges the corresponding sets **A** and **B** and starts the process over again from step 1. If **GAIN(k)**≤ 0, the algorithm stops. Based on the sequence of $gain_i$, the algorithm produces the swapping-gain function

f(GAIN(k)) as shown in Figure 6. In this example, the peak of the total gain is **k** = 7. Thus the algorithm will interchange the first 7 nodes into sets **A** and **B**. This local search strategy allows the algorithm to climb out of the local minima.

Kernighan and Lin also extended their two-way partitioning technique to perform multi-way partitions. Consider the problem of partitioning a set **S** into **k** subsets such that $|S| = \sum_{i=1}^{k} S_i$. The multi-way partitioning algorithm executes two-way partitioning repeatedly based on the cut-set size of (C_A, C_B) where $C_A = S_i$ and $C_B = \sum_{j=i+1}^{n} S_j$ for $1 \leq i$

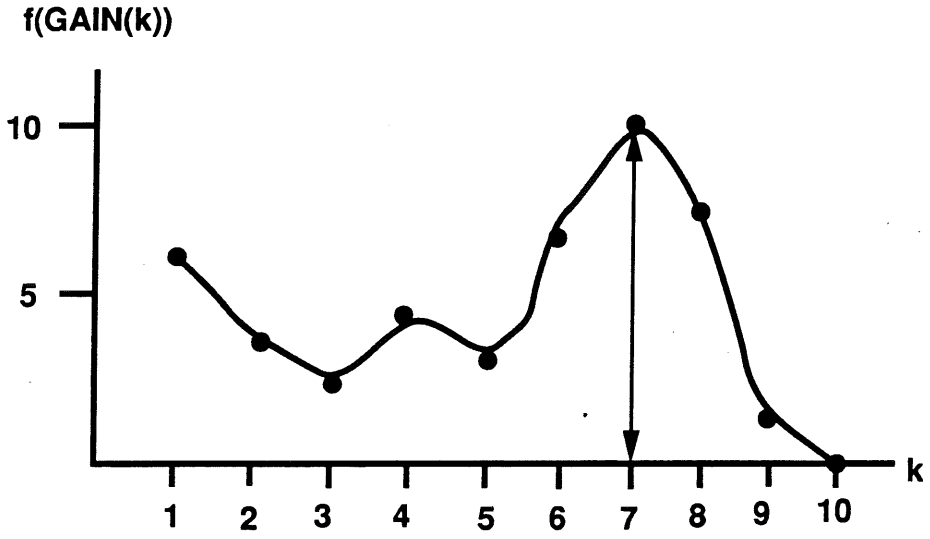


Figure 6. Local search strategy.

$\leq$ n–1. Frequently, C_A and C_B are not equal; therefore, a set of dummy nodes are added to the original set to allow unbalanced partitioning.

The complexity of the original Kernighan and Lin two-way uniform partitioning algorithm is $O(n^2 \log n)$. However, Fiduccia and Mattheyses [FiMa82] used a clever implementation to achieve linear complexity in terms of the number of pins. Dunlop and Kernighan [DuKe85] have compared the Kernighan/Lin and Fiduccia/Mattheyses algorithms. They found that the results of Fiduccia and Mattheyses are not quite as good as those of Kernighan and Lin but that the execution time is substantially reduced. In addition, Krishnamurthy [Kris84] has proposed a look-ahead strategy to guide the heuristics achieving more optimal partitions.

3.3.3. Simulated annealing

Simulated annealing was proposed by Kirkpatrick [KiGe83] for solving combinatorial optimization problems. Using this technique, it is possible to extricate from a local optimal solution and move to a global optimal solution. In physics, an annealing process starts by melting a solid and then slowly lowering the temperature to find the minimal energy state where a crystal is formed. This same idea can be applied to combinatorial optimization. A simulated annealing algorithm can generate moves randomly and calculate the new configuration cost Δc_{ij} for a move from configuration i to j. If $\Delta c_{ij} < 0$ then a lower energy level is achieved and the move is accepted. If $\Delta c_{ij} \geq 0$ then the move is accepted with probability $e^{\frac{-\Delta c_{ij}}{t}}$. As the simulated temperature t decreases, the probability of move acceptances decreases.

Theoretical studies [RoSa85] have shown that simulated annealing can climb out of local minima and find the globally optimal solution. However, it is impractical to find the optimal solution by performing an infinite number of iterations at each temperature. Several heuristics [KiGe83, RoSa84, HuRo86] have been developed to reduce run time. Simulated annealing usually produces very good results; however, it suffers from very long run times.

4. Partitioning Algorithms in Silicon Compilation

4.1. System Synthesis

APARTY [DiTh89] is an architectural partitioner. It uses a multi-stage clustering algorithm to extract high level structural information from the behavioral description. The high level structure reflects physical design considerations such as interconnects. APARTY attempts to examine physical considerations in the early design stage so that the synthesis tools can choose a better design in terms of area.

The multi-stage clustering consists of three major clustering stages: (i) control clustering, (ii) data clustering, and (iii) schedule clustering.

Control clustering. Control clustering groups the operators in the same control path together so that the control flow passing between two clusters can be reduced. Control flow between clusters is measured based on the probability that control will be passed from one operator to another. The control closeness between two operators **a** and **b** is:

$$C_{C_{a,b}} = P(OP_b \mid OP_a)$$

where $P(OP_b \mid OP_a)$ denotes the probability that OP_b will be activated by OP_a. For example, if operators **a** and **b** are in the same path without branches then $P(OP_b \mid$

OP_a)=1. On the other hand, if operators **a** and **b** are on the different paths then $P(OP_b \mid OP_a)=0$.

Data clustering. Data clustering considers data similarities between individual operators. The goal of this stage is to reduce the amount of data passed between two clusters. The closeness of the number of common values between two operator clusters is:

$$C_{D_{a,b}} = \frac{Common(a,b)}{V(a)+V(b)}$$

where $Common(a,b)$ measures the number of common values between cluster **a** and cluster **b**. $V(a)$ is the number of values flowing to and from **a**. For example in Figure 7, the "+" and "-" operators each have 2 out of 3 connections (B and C) in common.

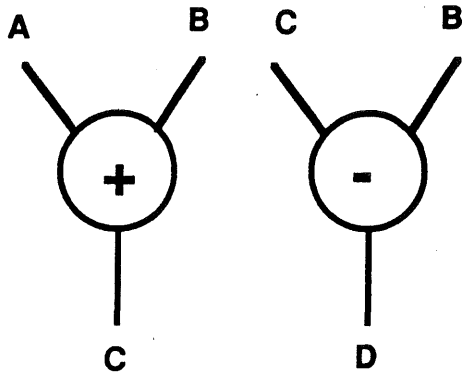


Figure 7. Data closeness calculation.

Thus the data similarity is:

$$C_{D_{+,-}} = \frac{2+2}{3+3}$$

Schedule clustering. Since the data clustering merges operators according to data similarities only, it may prevent some operators that share hardware from being scheduled simultaneously. As a result, this leads to a poor schedule. Schedule clustering considers the potential low level parallelism to ensure a reasonable schedule with minimum hardware. The schedule closeness calculation for two clusters **a** and **b** is:

$$C_{S_{a,b}} = C_{D_{a,b}} \times (1-INC_{a,b})$$

where $INC_{a,b}$ is the incompatibility for the clusters **a** and **b**. $INC_{a,b}$ measures the incompatibility of all the operators in **a** as compared with the operators in **b**. If any two operators are incompatible, the penalty is calculated as the excessive hardware for putting two incompatible operators into same partitions for maintaining the same schedule. The data clustering tends to group operators that share data. In the mean time, the incompatible measurement tends to push incompatible operators into different partitions.

Based on the information obtained from these clustering stages, the area and delay of different designs can be estimated for guiding scheduling, datapath allocation, and the selection of busses. The choice of which stages and the order to be run is decided by the user.

4.2. Processor Synthesis

BUD [McFa86, McKo90] uses a bottom-up analysis of the synthesis process in two ways. First, it obtains the physical and logical information about the primitives

available for use in the design from a database [Wolf86]. Second, the data operations are partitioned into clusters [McFa83], using a metric that takes into account functional unit sharing, interconnect, and parallelism. Each cluster represents a portion of the chip. A leaf cluster contains one or more function units. Nonterminal clusters contain a set of leaf clusters and nonterminal clusters with the interconnects among them. The database offers size and timing information for the individual module. For example, the floorplan of Figure 8(a) is shown in Figure 8(b). Clusters 1 and 2 are leaf clusters which consist of a functional unit and storage interconnected by busses. Clusters 3 and 4 are nonterminal clusters which consist of lower level clusters plus the wires interconnecting them.

BUD groups operations using a matrix that measures the closeness of putting two operations into same cluster. The closeness between operations depends on three factors: their common functionality, degree of interconnection, and potential parallelism. The closeness between operations x and y is defined as follows:

$$\mathbf{closeness(x,y)} = -\mathbf{S_1} \times \mathbf{fprox(x,y)} - \mathbf{S_2} \times \mathbf{cprox(x,y)} + \mathbf{N} \times \mathbf{S_3} \times \mathbf{par(x,y)}$$

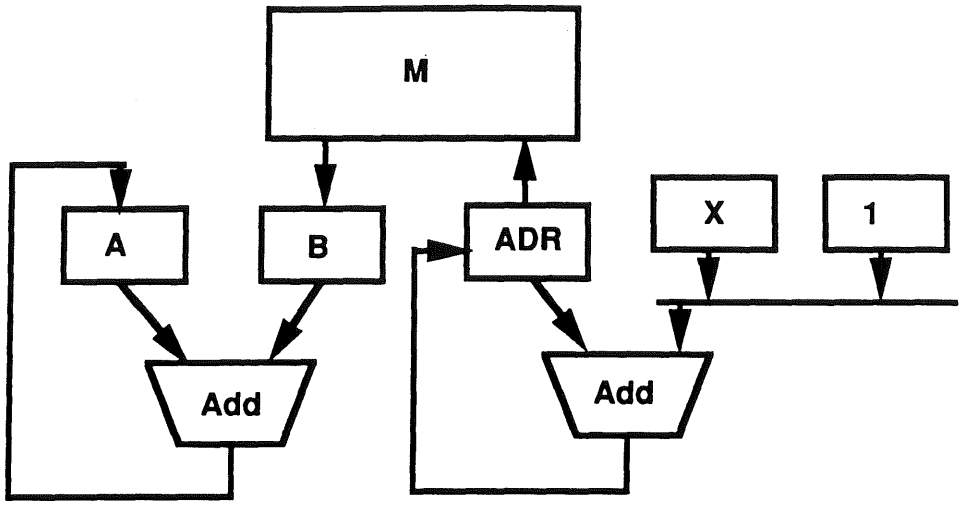
where

$$\mathbf{fprox(x,y)} = \frac{fcost(x)+fcost(y)-fcost(x,y)}{fcost(x,y)}$$

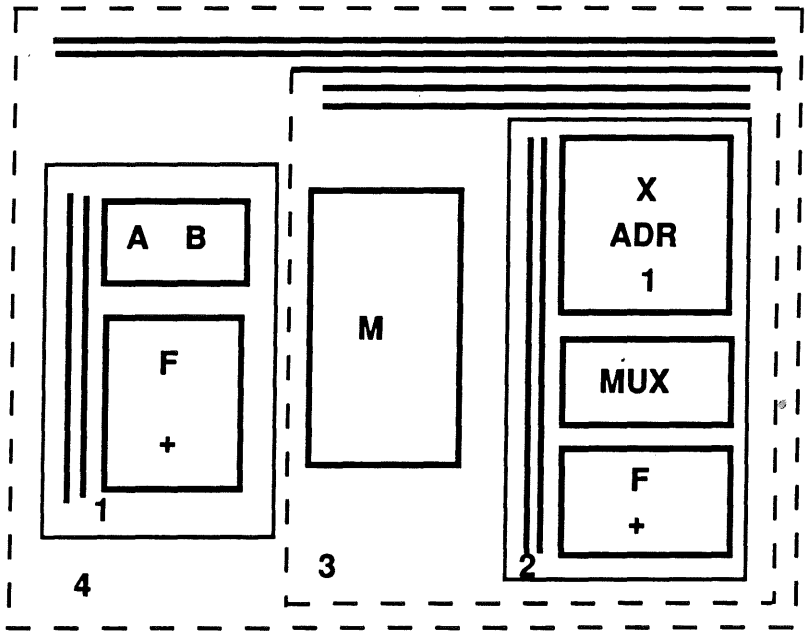
$$\mathbf{cprox(x,y)} = \frac{commconn(x,y)}{totalconn(x,y)}$$

par(x,y) = 1 if x and y can be done in parallel, and 0 otherwise

Here $fcost(x,y,z,..)$ denotes the minimum number of function units required to perform all the operations x,y,z,.. in the list. **fprox(x,y)** denotes the ratio of the shared functionality of x and y. **cprox(x,y)** is the ratio of the dataflow connections shared by x



(a)



(b)

Figure 8. An example of cluster.

and y. The S_1 and S_2 factors are the ratio of the area of the function unit required to do operation x_1 and x_2 to the total area of the design. S_3 is the probability of either x_1 or x_2 being executed in one major cycle. **N** denotes the relative weight given by the user to speed.

Based on this closeness function, a closeness matrix is computed. Then, a hierarchical clustering tree is formed from these closeness data. Different configurations are formed by cutting the tree at different levels. Each configuration represents a particular hardware configuration. The cluster tree guides the search of the design space. The cut line starts at the root and moves toward the leaves. Each time a new cut line is formed, a new design configuration is evaluated in terms of area and delay. The configuration that best meets the design objectives is chosen as the final design. This partitioning approach leads to a simple method for systematically exploring the space of possible designs to find the optimal design.

4.3. Floorplanning

Floorplanning is the first step of VLSI chip design. Designers first partition the chip into macro modules. Next they determine the areas, relative positions, aspect ratios, and I/O pin locations of these modules and try to optimize the overall area utilization, power dissipations, and delays along critical paths.

Many partitioning approaches have been proposed for solving floorplanning problems. These approaches can be divided into three groups: (i) cluster growth, (ii) connectivity clustering, and (iii) partitioning and slicing.

Cluster growth. The cluster growth floorplanning method operates in a bottom-up fashion. Preas [Prva79] used a clustering method to estimate and define cell shapes.

Horng and Lie [HoLi81] build the floorplan by starting in the lower left corner and clustering cells toward the upper right corner. The cluster growth floorplanning method is easy to implement. However, the layout quality is not as good as other methods.

Connectivity clustering. Dai and Kuh [DaKu86,DaEs89] introduced a connectivity clustering method which provides a simultaneous solution of floorplanning and global routing. Their approach consists of two steps: bottom-up clustering and top-down space allocation.

In the bottom-up phase, modules are hierarchically clustered according to their size and connectivity. The cluster size for each level is limited to five for two reasons: (i) five is the minimal number of elements necessary to form a non-slicing floorplan topology and (ii) the number of different floorplans for five components is 92 which can be examined exhaustively. During the clustering stage, the optimal shape, aspect ratio, and the information about connectivity among clusters are passed up along the cluster tree. At the final cluster level, the chip has at most five components which contain clusters formed at previous steps.

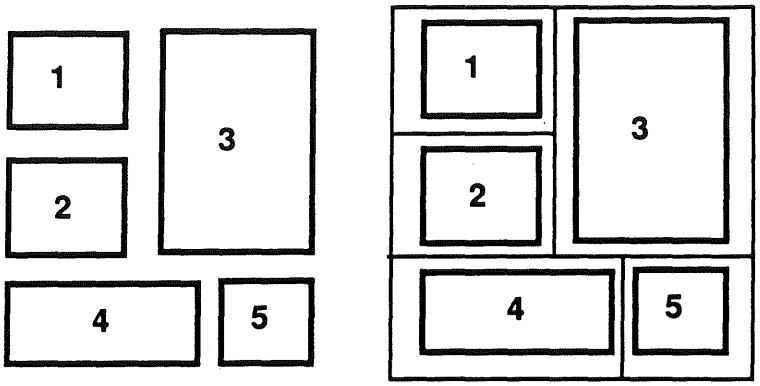
In the top-down phase, all different floorplans are evaluated starting at the top level of the hierarchy. Since the number of components in each cluster level is limited to five, all possible floorplans can be exhaustively examined. The floorplan with minimal total area is chosen as the final design. This approach demonstrates that hierarchical decomposition can simplify the floorplanning problem and produce high quality results.

Partitioning and slicing. Lauther [Laut79] first applied the min-cut partitioning approach to place general cells. Later, LaPotin and Director [LaDi86] applied the min-

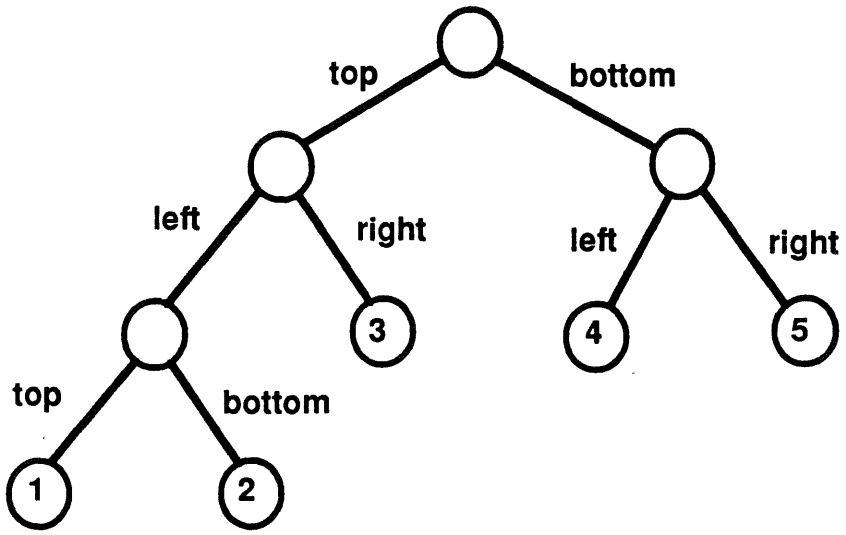
cut method to solve the floorplanning problem. Using a min-cut method, the rectangular chip area is first decomposed to form a slicing tree. To take interconnectivity into account, LaPotin and Director proposed an in-place partitioning method that is identical to the terminal propagation algorithm [DuKe85] which will be described in the next section. The purpose of forming the slicing tree is to represent the partitioning hierarchy. A min-cut partitioning and slicing tree formation example of five modules is shown in Figure 9. The slicing tree determines the relative positions of modules. After forming the slicing tree, a two-phase traversal is performed to determine the absolute position of the modules. In the first phase, a postorder traversal is used to determine a set of possible floorplan dimensions. In the second phase, preorder traversal is performed to determine the aspect ratio and location of each module in the slicing tree.

Another alternative partitioning approach for floorplanning is: **Multi-Terrain Partitioning** [LuDe89a,LuDe89b]. There are three types of terrains for a datapath chip: random logic, datapath stack, and large macros. The multi-terrain partitioning approach uses a min-cut algorithm to partition the objects into terrains. Then, it evaluates all possible terrain configurations and selects the optimal floorplan.

Another approach is termed **Capacity-Based Partitioning** [WuGa90]. This approach dissects the layout area into area blocks according to the given constraints. The algorithm estimates the transistor capacity for each area block, then uses a seed-based multiway partitioning strategy to assign glue-logic components into area blocks. The algorithm runs iteratively and selects the partition with the minimum total area as the final floorplan.



(a) . Min-cut partitioning



(b). Slicing tree

Figure 9. The slicing tree formation.

4.4. Placement

The goal of placement is to determine the positions of components on a layout. To place hundreds or thousands of components and successfully satisfy a set of given constraints is a very complex problem. To reduce the complexity of placement, partitioning approaches are widely used for solving placement problems. Kernighan and Lin [KeLi70] developed a two-way min-cut partitioning scheme for general graph partitioning. Schweikert and Kernighan [ScKe72] extended the min-cut algorithm to take into account special properties of electrical circuits such as multi-net connections. Based on this min-cut partitioning foundation, many algorithms [Schw76,Breu77,Corr79,Laut79,Burs82,DuKe85,SuKe87,BhHi88,Hill88] have been reported for solving cell placement problems.

The basic concept of min-cut placement is to partition components into two clusters so that the number of interconnections crossing the cut is minimized. The min-cut algorithm is executed recursively until each cluster contains only a few cells. Using min-cut partitioning, it is not adequate to simply partition components into clusters without considering the external connections. For instance, by swapping node X and node Y (Figure 10), the partitioning cost increases by 1 if the external connections between the nodes in the block A and the node X in block B are not taken into account. However, the actual partitioning cost decreases by 1 when the external connections are taken into account.

To solve this problem, Dunlop and Kernighan [DuKe85] have proposed a modification of the Kernighan and Lin min-cut algorithm. They introduced a **terminal propagation** strategy to take the external connections into account more accurately.

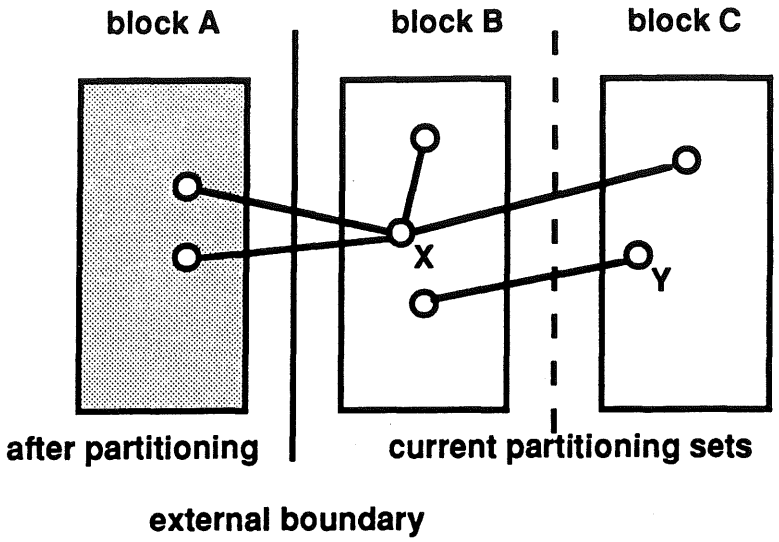


Figure 10. Partitioning with external consideration.

For example, a penalty external cost $\mathbf{PE}_{cost}$ can be added to calculate the partitioning cost. $\mathbf{PE}_{cost}$ can be (i) zero, (ii) negative, or (iii) positive. In case (i) (Figure 11(a)), block B and block C are adjacent to block A. Thus, $\mathbf{PE}_{cost}$ can be set to zero by swapping node X and node Y. In case (ii) (Figure 11(b)), if node X connects to block A and block D is not adjacent to block A, $\mathbf{PE}_{cost}$ will be made negative by swapping node X and node Y (it needs one more extra vertical routing track). In case (iii) (Figure 11(c)), if element X connects to block B and block D is adjacent to block A, $\mathbf{PE}_{cost}$ will be made positive by swapping node X and node Y (it reduces one vertical routing

track).

one improvement to min-cut placement is: (i) **Quadrisection** [SuKe87]. Instead of using a bi-partitioning approach, quadrisection partitions the given set along horizontal and vertical division lines into four partitions simultaneously. This approach obtains results comparable to the simulated annealing approach but with a much shorter run time and (ii) The **Min-Cut Shuffle** [BhHi88] approach that takes into account of the

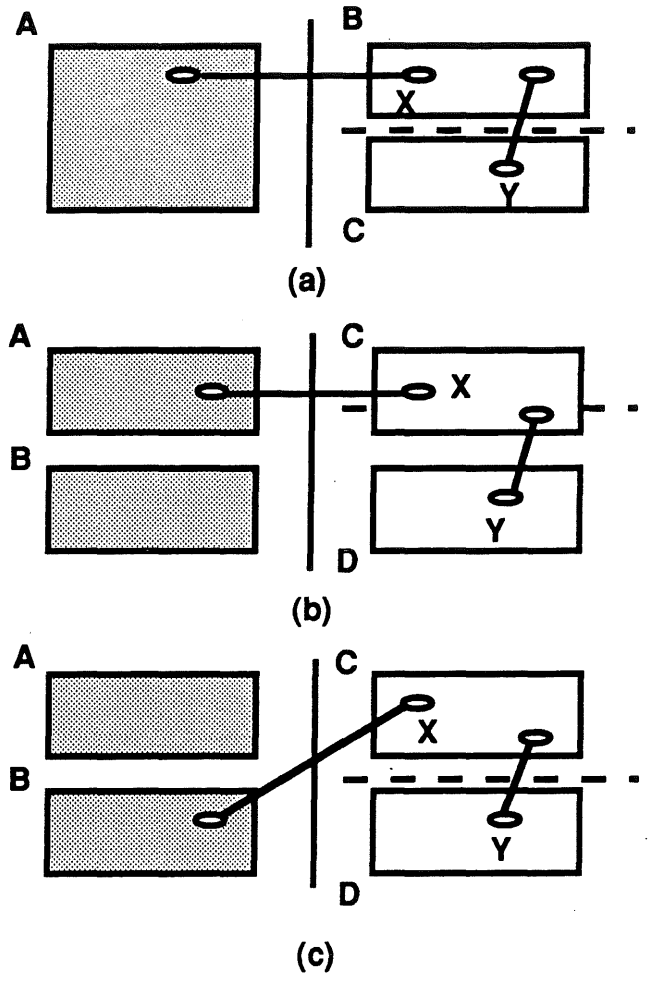


Figure 11. Terminal propagation.

order of partitioning to achieve a solution with global considerations.

5. Conclusion

This survey paper has presented the partitioning techniques used in different VLSI design processes. From the variety of partitioning implementations, it demonstrates that partitioning methods are suitable for solving large scale problems. In the past years, partitioning techniques have been widely used in layout synthesis. However, the usage of partitioning techniques in processor and system synthesis levels is still in the germinant stage. Different approaches for system and processor partitioning need to be investigated further. Thus, system and processor partitioning will become one of the most active research areas in the years to come.

6. References

[BhHi88] Bhandarl, I., Hirsch, M., and Siewlorek, D., "The Min-Cut Shuffle: Toward a Solution for the Global Effect Problem of Min-Cut Placement," Proc. 25th DAC, pp.681-685, 1988.

[Breu77] Breuer, M. A., "A Class of Min-Cut Placement Algorithms," Proc. of the 14th DAC., pp.284-290, 1977.

[Burs82] Burstein, M., "Partitioning of VLSI Networks," Proc. 19th DAC, 1979.

[ChKu84] Chen, C. K., and Kuh, E. S., "Module Placement Based on Resistive Network Optimization," IEEE Trans. on CAD, Vol. CAD3, No. 3,pp.218-225, 1984.

[CoCa80] Cox, G. W., and Carroll, B. D., "The Standard Transistor Array (star), part II: Automatic Cell Placement Techniques," Proc. 17th DAC, pp.451-457, 1980.

[Corr79] Corrigan, L. I., "A Placement Capability Based on Partitioning," Proc. 16th DAC, 1979.

[DaEs89] Dai, W. M., Eschermann, B., Kuh, E. S., and Pedram, M., "Hierarchical Placement and Floorplanning in BEAR," IEEE Trans. on CAD, Vol. 8, No. 12, pp.1335-1349, 1989.

[DiTh89] Dirkes Lagnese, E., and Thomas, D. E., "Architectural Partitioning for System Level Design," Proc. 26th DAC, pp.62-67, 1989.

[Dona88] Donath, W. E., "Logic Partitioning" in *Physical Design Automation of VLSI Systems* (Preas, B. T., and Lorenzetti, M editors), Benjamin/Cumming, 1988.

[DuKe85] Dunlop, A. E., and Kernighan B. W., "A Procedure for Placement of Standard-Cell VLSI Circuits," IEEE Trans. on CAD, Vol. CAD-4, No. 1, pp. 92-98, January, 1985.

[Ever74] Everitt, B., *Cluster Analysis*, Heinemann Educational Books Ltd., 1974.

[FiMa82] Fiduccia, C. M., and Mattheyses, R. M., "A Linear-Time Heuristic for Improving Network Partitions," Proc. 19th DAC., pp.175-181, 1982.

[FoFu62] Ford, L. R., and Fulkerson, D.R., "Flows in Networks," Princeton University Press, 1962.

[Gajs88] Gajski, D. D., *Silicon Compilation*, Addison-Wesley Publishing Company, 1988.

[Gajs83] Gajski, D. D. and Kuhn, R. H., "New VLSI Tools," *Computer*, Vol. 16, no. 12, pp.11-14, December, 1983.

[Gare79] Garey, M. R., and Johnson, D. S., *Computers and Intractability, A Guide to the Theory of NP Completeness*, W. H. Freeman and Co., San Francisco, Califonia, pp.209-210, 1979.

[HaKu72] Hanan, M., and Kurtzberg, J. M., "Placement Techniques" in *Design Automation of Digital Systems* (Breuer, M. A. editor), Prentice-Hall, Inc., Englewood Cliffs, New Jersey, pp.213-282, 1972.

[HaWo76] Hanan, M., and Wolff, Sr., P. K., and Agule, B. J., "Some Experimental Results on Placement Techniques," Proc. 13th DAC, pp.214-224, 1976.

[Hill88] Hill, D. D., "Alternative Strategies for Applying Min-Cut to VLSI Placement," Proc. ICCD, pp.440-444, 1988.

[HoLi81] Horng, C., and Lie, M., "An Automatic/Interactive Layout Planning System for Arbitrarily-Sized Rectangular Building Blocks," Proc. 18th DAC, pp.293-300, 1981.

[HuRo86] Huang, M. D., Romeo, F., and Sangiovanni-Vincentelli, A., "An Efficient General Cooling Schedule for Simulated Annealing," Proc. ICCAD,pp. 381-384, 1986.

[IoKi83] Iosupovici, A., King, C., and Breuer, M. A., "A Module Interchange Placement Machine," Proc. 20th DAC, pp.171-174, 1983.

[Joha79] Johannsen, D., "Bristle Blocks: A Silicon Compiler," Proc. 16th DAC, pp.310-313, 1979.

[John67] Johnson, S. C., "Hierarchical Clustering Schemes," Psychometrika, pp.241-254, September, 1967.

[Kang83] Hang, S., "Linear Ordering and Application to Placement," Proc. 20th DAC, pp.457-464, 1983.

[KeLi70] Kernighan, K. H., and Lin, S., "An Efficient Heuristic Procedure for Partitioning Graphs," Bell System Technical Journal, vol. 49, no. 2, pp.291-307, February, 1970.

[KiGe83] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P., "Optimization by Simulated Annealing," *Science*, vol. 220, no. 4598, pp.671-680, 1983.

[Kodr72] Kodres, U. R., "Partitioning and Card Selection" in *Design Automation of Digital Systems* (Breuer, M. A. editor), Prentice-Hall, Inc., Englewood Cliffs, New Jersey, pp.173-212, 1972.

[Kris84] Krishnamurthy, B., "An Improved Min-Cut Algorithm for Partitioning VLSI Networks," IEEE Trans. on CAD, vol. C-33, pp.438-446, May, 1984.

[Kurt65] Kurtzberg, J. M., "Algorithms for Backplane Formation" in *Microelectronics in Large Systems*, Spartan Books, pp.51-76, 1965.

[LaDi86] La Potin, D. P., and Director, S. W., "Mason: A Global Floorplanning Approach for VLSI Design," IEEE Trans. on CAD, vol. CAD-5, no. 4, pp.477-489, October, 1986.

[Laut79] Lauther, U., "A Min-Cut Placement Algorithm for General Cell Assemblies Based on A Graph Representation," Proc. 16th DAC., pp.1-10, 1979.

[LuDe89a] Luk, W. K., and Dean, A. A., "Multi-Stack Optimization for Data-Path Chip (Microprocessor) Layout," Proc. 26th DAC, pp.110-115, 1989.

[LuDe89a] Luk, W. K., Dean, A. A., and Mathews, J. W., "Multi-Terrain Partitioning and Floorplanning for Data-Path Chip (Microprocessor) Layout," Proc. ICCAD'89, pp.492-495, 1989.

[McFa83] McFarland, S.J. M. C., "Computer-Aided Partitioning of Behavioral Hardware Descriptions," Proc. 20th DAC, pp.472-478, 1983.

[McFa86] McFarland, S.J. M. C., "Using Bottom-Up Design Techniques in the Synthesis of Digital Hardware from Abstract Behavioral Descriptions," Proc. 23rd DAC, pp.330-336, 1988.

[McKo90] McFarland, S.J. M. C. and Kowalski, T. J., "Incorporating Bottom-Up Design into Hardware Synthesis," IEEE Trans. on CAD, vol. 9, no. 9, pp.938-950, September, 1990.

[MacNa64] MacNaughton-Smith, P., Williams, W. T., Dale, N. B., and Mockett, L. G., "Dissimilarity Analysis," *Nature*, Lond., 202, 1034-1035, 1964.

[OdHa87] Odawara, G., Hamuro, T., Iijima, K., Yoshino, T., and Dai, Y., "A Rule-based Placement System for Printed Wiring Boards," Proc. 22rd DAC, pp.777-785, 1987.

[PrKa88] Preas, B. T., and Karger, P. G., "Placement, Assignment and Floorplanning" in *Physical Design Automation of VLSI Systems* (Preas, B. T., and Lorenzetti, M editors), Benjamin/Cumming, 1988.

[Prva79] Preas, B. T., and vanCleemput, W. M., "Placement Algorithms for Arbitrarily Shaped Blocks," Proc. 16th DAC, pp.474-480, 1979.

[Rome84] Romesburg, H. C., *Cluster Analysis for Researchers*, Wadsworth, Inc., 1984.

[RoSa84] Romeo, F., Sangiovanni-Vincentelli, A., and Sechen, C., "Research on Simulated Annealing at Berkeley," Proc. of the Intl. Conf. on Computer Design, pp.652-657, 1984.

[RoSa85] Romeo, F., and Sangiovanni-Vincentelli, A., "Probabilistic Hill Climbing Algorithms: Properties and applications," Proc. of the 1985 Chapel Hill Conf. on VLSI, PP.393-417, 1985.

[Sang87] Sangiovanni-Vincentelli, A., "Automatic Layout of Integrated Circuits" in *Design Systems for VLSI Circuits*: *Logic Synthesis and Silicon Compilation* (DeMicheli, Sangiovanni-Vincentelli, Antognetti, editors), Kluwer Academic Publishers, 1987.

[SaRa90] Saab, Y. G., and Rao, V. B., "Fast Effective Heuristics for the Graph Bisectioning Problem," IEEE on CAD, vol.9, no.1, pp.91-98, January, 1990.

[Schw76] Schweikert, D. G., "A 2-Dimensional Placement Algorithm for The Layout of Electrical Circuits," Proc. 13th DAC, pp.408-416, 1976.

[ScKe72] Schweikert, D. G., and Kernighan, B. W., "A Proper Model for the Partitioning of Electrical Circuits," Proc. 9th DAC, pp.56-62, 1972.

[ScUl72] Schuler, D. M., and Ulrich, E. G., "Clustering and Linear Placement," Proc. 9th DAC, pp.475-481, 1982.

[Snea57] Sneath, P.H.A., "The Application of Computers to Taxonomy," J. gen, Microbiol., 17, 201-226, 1957.

[Spat80] Spath, H., *Cluster Analysis Algorithms*, Ellis Horwood Ltd., 1980.

[SuKe87] Suaris, P. R., and Kedem, G., "Quadrisection: A New Approach to Standard Cell Layout," Proc. ICCAD'87, pp.474-477, 1987.

[WeCh89] Wei, Y-C., and Cheng, C-K., "Towards Efficient Hierarchical Designs by Ratio Cut Partitioning," Proc. ICCAD'89, pp.298-301, 1989.

[Wolf86] Wolf. W., "An Object-Oriented, Procedural Database for VLSI Chip Planning," Proc. 23rd DAC, 1986.

[WuGa90] Wu, A. C. H. and Gajski, D., "Partitioning Algorithms for Layout Synthesis from Register-Transfer Netlists," Proc. ICCAD'90, 1990.