An Effective General Connectivity Concept for Clustering

Jianjian Song, Zhaoxuan Shen and Wenjun Zhuang
National Supercomputing Research Center
89 Science Park Drive, Singapore 118261
songjj@nsrc.nus.edu.sg

Abstract

This paper shows how algorithmic techniques and parallel processing can speed up general connectivity computation. A new algorithm, called Concurrent Group Search Algorithm (CGSA), is proposed that divides $\frac{N(N-1)}{2}$ *vertex pairs into N-1 groups. Within each group general connectivities of all pairs can be calculated concurrently. Our experimental results show that this technique can achieve speedup of 12 times for one circuit. In addition, group computations are parallelized on a 16-node IBM SP2 with a speedup of 14 times over its serial counterpart observed. Combining the two approaches could result in a total speedup of up to 170 times, reducing CPU time from over 200 hours to 1.2 hour for one circuit.*

Our new model is better than those without clustering because it characterizes the connection graph more accurately, is faster to compute and produces better results. The best performance improvements are 43% for one circuit and 49% for another.

1. Introduction

Clustering is used extensively in VLSI physical design. In general, the connectivity of each pair of modules is defined and calculated and then those pairs that are strongly connected are clustered [16]. Clusters provide a hierarchical structure of the modules related to their connections. The concept has been applied to placement, [7], [12], [15], [16], [21], [28], partitioning [4], [9], [18], [10], and floorplanning [5], [6].

Clustering has received a great deal of attention in partitioning. [4] shows that partitioning with clusters can obtain for some circuits up to 49.6% smaller cut sizes than those with the bisection algorithm and up to 66.8% smaller ratio cut sizes than those with the ratio-cut algorithm. [18] reports 60% better results for partitioning than the F&M method. [9] finds clustering can reduced the number of nets cut by 46% compared to the standard min-cut approach [1]. The BISECT partitioning algorithm in [15] could be 73 times faster than the F-M. 17% average improvement in bisection width is obtained in [10].

In the area of placement, up to 21.6% reduction is achieved in the number of feed-throughs in [28]. Up to 41% improvement of open nets after channel routing is obtained in [7]. Clustering is also combined with simulated annealing in [12] to obtain 6-17% improvement in the estimated wire length and [20] to reduce chip area by 21% and the total wire length by 8-11%. Another application of clustering is to delay minimization as shown in [14].

One way to find good clusters is to define connectivity and clustering criteria. Two modules (or clusters) that are strongly connected can form a larger cluster. Once the clusters are formed they will not be disbanded. Success of this approach is heavily dependent on connectivity definition.

The conventional connectivity between two modules considers the weighted nets directly connecting the two modules. A number of attempts are made to define the strength of connection between two modules such as the conjunctivity and disjunctivity models in [16] and [18], and simple sum of net weights in [5]. They all use the number of common nets as the measure of how strongly two modules are connected. But different repulsive forces are used such as the total number of nets connected to a module [16] or the total number of nets in the smaller clusters as well as the number of modules in the smaller clusters [18]. Connectivity is defined with Rent's rule in [7] and multiple attributes in [12].

Another way to find good clusters is to form clusters first and then evaluate if they satisfy a quality measure so that they can be preserved. A clique is found and is called a cluster if it satisfies the module size and cluster density thresholds in [4]. Cycles of modules are found by random walk and two modules are clustered if they both appear in at least two cycles in [10]. The more cycles they appear together the stronger they are connected. Another approach tries to generate all clusters at the same time [20], [23].

Zhuang, et al. proposed a steady cluster concept in [24], [25], [26] and later formulated definition of stable and near stable clusters in [28]. It has been shown that cluster quality is related to connectivity estimation in [22]. The concept of general connectivity and a conductance network model for its calculation are described in [22] and [28] and are shown to be effective together with stable cluster concept in reducing the number of feedthroughs by as much as 21.6%. The concept of general connectivity between two modules is that both direct and indirect connections of the two cells should be considered for connectivity estimation.

Zhuang, et al. In [28] have shown that application of the general connectivity to a circuit with 1724 cells could reduce the number of feedthroughs of the resulting placement by 21.6% in comparison to a traditional inside-outside approach although only the second-order connectivity was used. A new general connectivity model is proposed in [19] based on better understanding and analysis of connection graphs and physical characteristics of netlists.

This paper describes further research initiated in [19]. Its main contributions are better understanding of clustering process and a new algorithm for general connectivity calculation and its parallel implementation. The new algorithm is 12 times faster than our old algorithm for one calculation. Our parallel program achieves speedup of up to 14 times on a 16-node IBM SP2 parallel computer. The program reduces CPU time for one calculation from 200 hours to merely 1.2 hours.

2. The General Connectivity Concept and Definition

A netlist can be represented by a weighted hypergraph where each module is a node and each net is one hyper-edge with an associated weight of importance. Let $G(V, E, W)$ be a weighted graph, where $V=\{v_i \mid i = 1, 2, ..., n\}$ is a set of vertices, $E=\{ e_{ij} = (v_i, v_j) \mid 1 \le i, j \le n\}$ is a set of edges, and $W=\{w_e \in R \mid e \in E\}$ is a set of real numbers called the weights defined on E of G. A weighted hypergraph can be mapped to a weighted graph as follows. Each node corresponds to one vertex in G(V, E, W) and each edge is one edge of G(V, E, W). A multi-pin net with r nodes can be mapped to a r-clique i.e., a complete graph with r vertices. G(V, E, W) is called *weighted connection graph.*

Some heuristics can be developed to guide us in defining general connectivity between two vertices on G(V, E, W), which should take into consideration of the effect of vertices indirectly connected with the two vertices. In general we believe that all the edges on paths between the two vertices should have some contribution to their general connectivity.

First, the connectivity between any two vertices should be greater than zero iff there exists a path between them (Nonzero Principle of Path). In other word, any path between two vertices should have nonzero contribution to the general connectivity of the two vertices. Secondly, Any edge in a path between two cells should have nonzero contribution to the connectivity of the two cells (Nonzero Principle of Edge). Assuming an edge in a path has zero contribution, the edge can then be removed from the path without affecting the connectivity of the two cells. But the path does not exist anymore, which is contradictory. Thirdly, The contribution of a path between two cells to the connectivity of the two cells is inversely proportional to the length of the path (Inverse Distance Square Principle). The contribution of an edge in a path between two cells to the connectivity of the two cells is inversely proportional to the square of the path length and proportional to the weight of the edge. Fourthly, If an edge appears in more than one path between two cells, its contribution to the connectivity of the two cells should only be related to the shortest paths of all the paths in which it appears (Shortest Path Principle).

A sequence of vertices $<v_{i_0}, v_{i_1}, ..., v_{i_{k-1}}, v_{i_k}>$ is said to be a k-path (path length equal to k) between two vertices v_{i_0} and v_{i_k}, denoted by $p(i_0, i_k, k)$, iff $e_{i_j i_{j+1}} \in E$ for $j=0,1, ..., k-1$, and $v_{i_j} \ne v_{i_l}$ for any $0 \le j \ne l \le k$. Let $E_{p(i_0,i_k,k)} = \{ e_{i_j i_{j+1}} \mid j = 0, 1, ..., k-1\}$ be the set of all the edges on the path.

Let $P(s, t, k)=\{ \forall p(s,t,k) \}$ be the set all the k-paths between v_s and v_t, and $E_{P(s,t,k)} = \bigcup_{p \in P(s,t,k)} E_p$ be the set of all the edges on paths in $P(s, t, k)$.

$L(s, t, k) = E_{P(s,t,k)} - \bigcup_{i=1}^{k-1} E_{P(s,t,i)}$ is the set of edges that appear on a k-path and not on any shorter path between v_s and v_t.

The kth-order general connectivity between vertices v_s and v_t is defined as

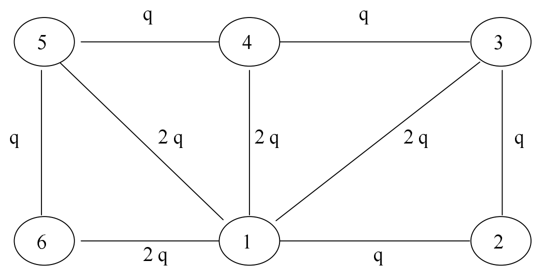


Figure 1. The connection graph for Example 1.

$$GC(s,t,k) = \sum_{i=1}^{k} \frac{\sum_{e \in L(s,t,i)} w_e}{i^2} \qquad (1)$$

Example 1: Given the connection graph in Figure 1, calculate GC(2,6,2) and GC(2,6,3).

L(2,6,1) = ϕ, L(2,6,2) = {(2,1), (1,6)} and L(2,6,3)={(2,3),(3,1),(1,5),(5,6)}. Therefore, GC(2,6,2) = $\frac{2q+q}{2^2} = \frac{3q}{4}$ and

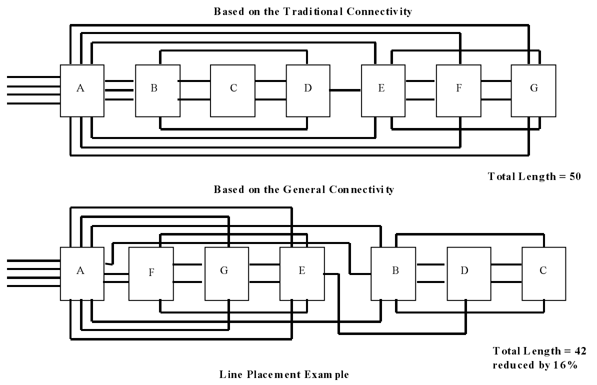
$$\text{GC(2,6,3)} = \frac{2q+q}{2^2} + \frac{q+2q+2q+q}{3^2} = \frac{17q}{12}.$$

Once general connectivities are found, clustering can be done by combining modules with highest connectivities. Our solution to a linear placement problem is given in Figure 2, where the traditional direct connectivity solution results in the total wire length of 50 and ours is 42. A partition example is given in Figure 3 where reduction in the number of cut lines by 45% is observed.

3. The Concurrent Group Search Algorithm

One method to calculate the general connectivity of a connection graph is to search through the graph for the subgraph for one pair of vertices at a time. It can be shown that many redundant searches will be performed and therefore time wasted. For a connection graph with N vertices, a total of $\frac{N(N-1)}{2}$ pairs of general connectivities need to be calculated. Assume that the vertices are numbered with consecutive integers starting from 1. The $\frac{N(N-1)}{2}$ pairs can be divided into N-1 groups: {[1,2], [1,3], ..., [1,N]}, {[2,3], [2,4], ..., [2,N]} ,...., { [N-1,N]}. All general connectivity calculations in each group can be calculated by traversing the search tree once. This method is faster as it only searches all the search trees N times instead of $\frac{N(N-1)}{2}$ times. In addition, each group is independent of the other so that computations can be carried out in parallel. The method is named *Concurrent Group Search Algorithm* (CGSA).

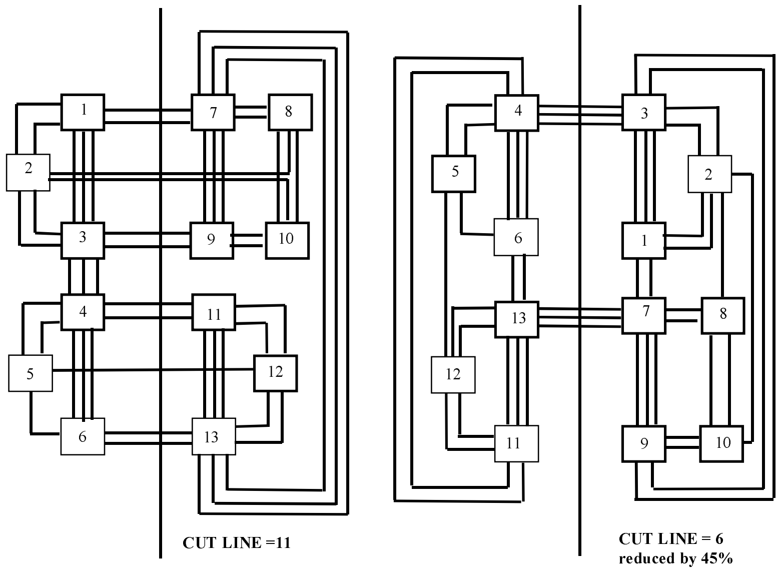
Algorithm 1 describes how the kth-order general connectivities for a layout with N vertices are computed. It basically traverses the connection graph once for every vertex to build a tree with the depth of *k* and one vertex being its root. It will store an edge once only on the shortest path that the edge appears. Once this tree is built, all the kth-order connectivities related to the vertex can be calculated simultaneously. Subroutine Compute GC(s, k) may create up to $(N-1)^k$ paths and N^2 edges may need to be compared to know if the edges on a path are new. Therefore, the worst-case time complexity of the algorithm is $O(N^{k+3})$.



The Connectivities with various order K

Cell-pair	K = 1	K = 2	K = 3	K = 4	K = 5	K = 6	K = 7
A - B	3.0000	3.0000	3.5550	4.3050	4.3850	4.3850	4.3850
A - C		1.2500	2.0270	2.5270	2.6070	2.6070	2.6070
A - D		2.0000	3.3320	3.4570	3.4570	3.4570	3.4570
A - E	2.0000	4.0000	4.8880	5.1380	5.1380	5.1380	5.1380
A - F	2.0000	4.0000	4.2220	4.5960	4.7560	4.7560	4.7560
A - G	2.0000	4.0000	4.2220	4.5960	4.7560	4.7560	4.7560
B - C	2.0000	3.0000	3.0000	3.3740	3.6940	3.7490	3.7490
B - D	2.0000	3.0000	3.6660	4.1660	4.2460	4.2460	4.2460
B - E		2.0000	3.3320	3.4570	3.4570	3.4570	3.4570
B - F		1.2500	2.4710	2.8460	2.8460	2.8460	2.8460
B - G		1.2500	2.4710	2.8460	2.8460	2.8460	2.8460
C - D	2.0000	3.0000	3.0000	3.3740	3.6940	3.7490	3.7490
C - E		0.7500	1.7490	2.2490	2.3290	2.3290	2.3290
C - F			1.3320	1.9570	1.9570	1.9570	1.9570
C - G			1.3320	1.9570	1.9570	1.9570	1.9570
D - E	1.0000	1.0000	1.7770	2.5270	2.6070	2.6070	2.6070
D - F		0.7500	2.1930	2.5680	2.5680	2.5680	2.5680
D - G		0.7500	2.1930	2.5680	2.5680	2.5680	2.5680
E - F	2.0000	4.0000	4.2220	4.5690	4.7560	4.7560	4.7560
E - G	2.0000	4.0000	4.2220	4.5690	4.7560	4.7560	4.7560
F - G	2.0000	4.0000	4.2220	4.2220	4.4620	4.5720	4.5720

Figure 2. Linear placement example using the general connectivity concept.



(a) Partition based on direct connectivity. (b) Partition based on general connectivity.

Figure 3. Partition solutions without and with general connectivity.

Take the graph in Figure 1 as an example. To compute the 2nd-order general connectivities between vertex 2 and each of the other vertices, the search tree starting from vertex 2 can be constructed as shown in Figure 4. All the 2nd-order general connectivities between vertex 2 and other vertices can be calculated by traversing the search tree once. It is clear that the 2nd-order general connectivity between vertices 2 and 3 can be calculated as follows:

L(2,3,1) = {(2,3)} and L(2,3,2) = {(2,1), (1,3)}.

Therefore,

$$CG(2,3,2) = \frac{q}{1} + \frac{q+2q}{2^2} = \frac{7q}{4}.$$

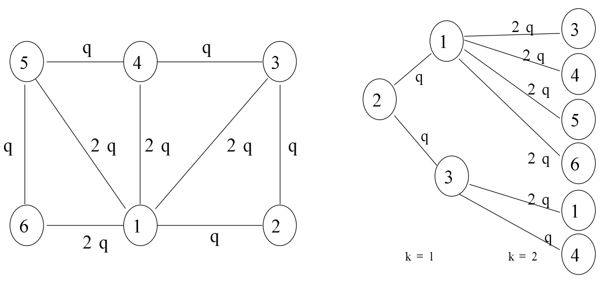


Figure 4. Search tree and concurrent computation for vertex 2.

4. Parallelization of the Concurrent Group Search Algorithm

As explained in Section 3, $\frac{N(N-1)}{2}$ pairs of general connectivities need to be calculated and can be divided into N-1 groups {[1,2], [1,3], ..., [1,N]}, {[2,3], [2,4], ...,

Algorithm 1:

```
For s = 1 to N-1
   Compute GC(s, k).

Compute GC(s, k) {
/* to compute kth-order general connectivities
of [vs, vt],∀ vt and (t > s). */
   For x =2 To k
      For each (x-1)th path p = < vs , ... , vt >
         For each vi ∈ Neighbor(vt)
            If vi is not on path p {
               Create a new kth-path q =
               < vs , ... , vt , vi >.
               For each edge e on path q
                  If e ∉L(s, i, y) add e to L(s, i, x).
            }
   For t = s+1 To N
      Calculate GC(s, t, k) using formula (1).
} /* end of Compute GC(s, k) */
```

[2,N]} ,...., { [N-1,N]}. The groups can be distributed over a number of processors to be calculated in parallel as they are independent of each other. This kind of problem is embarrassingly parallelizable except for load balancing consideration. Notice that the computation loads of groups are not identical. The first group may need more CPU time and the last group may require the least CPU time as they have different number of pairs. A load balancing scheme is proposed to tackle this issue.

Our parallel code is implemented with the Message Passing Interface(MPI) standard library on a 16-node IBM SP2. The master-slave paradigm is adopted where the master process dispatches tasks to slave processes and collects results from the slaves. Static group clustering and dynamic task scheduling are used in order to balance the loads of slave processes as well as minimize communication between the master and slaves. The master process will select a number of groups to form a task so that each task may have more or less the same CPU time requirement, and dynamically dispatch the tasks in the queue to the slaves during run time. For example, 16 groups can be clustered into 4 tasks as follows: {1,16,5,12}, {2, 15,6,11}, {3,14, 7, 10}, {4, 13, 8, 9}.

The master and slave processes are described in ***Algorithm 2***.

Figure 5 shows the speedup of ***Algorithm 2*** over ***Algorithm 1*** with the number of processors ranging from 3 to 16 for the two larger circuits in Table 1. It can be seen that speedup improves with the increasing order of general connectivity. When the order is large enough speedup approaches its theoretical maximum. Our group clustering and dynamic load balancing techniques have reduced communication time and balanced the CPU loads within 10% of each other as shown in Figure 6 for computing the 5th-order general connectivity of HK5851 on 16 processors.

5. Application to placement

One measurement of the performance of a placement is chip area and the other is circuit delay. Chip area can only be minimized by reducing the routing area. The routing area can be reduced by reducing the number of feedthroughs in a placement. Feedthroughs contribute more to circuit delay than other connections of the same length. The reduction of total wire length, especially the number of feedthroughs, will improve the global optimization in circuit delay. Therefore, the number of feedthroughs can be a performance indicator for a placement.

Algorithm 2:
```
Master program{
   Get the number of slaves;
   Cluster groups;
   For each slave{
     Send connection graph;
     Send one task;
   }
   While not completed {
     Receive result from a slave;
     Send a new task to the slave;
}
}/* end of Master */

Slave program{
   Receive initial data;
   While not done{
     Receive a task;
     Carry out the task;
     Send results to Master;
   }
}/* end of Slave */
```

Three commercial circuits were experimented to evaluate ***Algorithm 1***. The larger one, HK5851, has over 28,000 transistors. For completeness, the characteristics of the circuits are copied from [28] as shown in Table 1. The CPU times for general connectivities of up to 5th order are given in Table 2.

The LSIS-II system [27] was used to conduct circuit designs with our general connectivity model. The system carries out automatic layout of designs based on standard and macro cells. It has twelve subsystems for design activities such as schematic editing, logic description and compiling, physical cell editing, cell library management, connectivity verification, cell clustering, automatic placement and global routing, automatic detailed routing, physical design and optimization, and format conversion.

Table 1. Characteristics of three circuits.

Circuit	No. of Transistors	No. of Cells	No. of Net
HK5601	4,000	256	355
HK5852	6,400	608	699
HK5851	28,000	1724	1893

Table 2. CPU times in seconds for general connectivity calculation on IBM RS6K/390.

Order	1	2	3	4	5
Time(Sec) for HK5852	2.5	7.0	68.0	981.5	14711.4
Time(Sec) for HK5851	10. 5	18.2	114.7	1348.8	16804.9

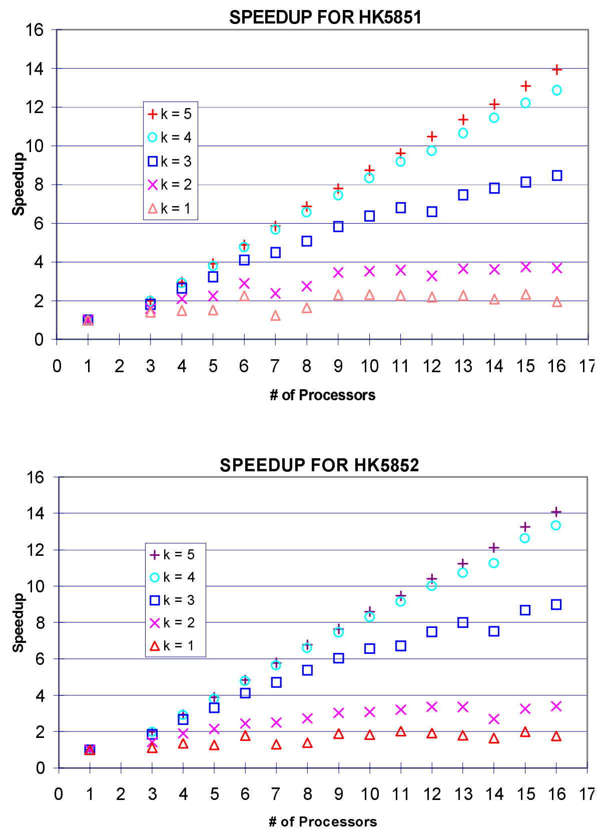


Figure 5. Speedup results for the two circuits.

Its GUI runs on top of the SUN OpenWindow environment.

A program was written to compute general connectivity and to generate clusters. It consists of a new routine to compute general connectivity based on ***Algorithm 1*** and a number of routines from [28] for clustering and cluster stability verification. Once the clusters were obtained, they were inserted into the LSIS-II package so that designs based on the new clusters could be carried.

The program takes its input data from the output of circuit verification of the LSIS-II system and produces an output file that is used by the block initial construction process of the LSIS-II.

Let L be the number of iterations of the clustering process, which is called the height of the clustering tree in [28]. The average time complexity of the clustering process is $O(LNM^{k+1})$, where N is the number of cells, M is the average degree of the cells, and k is the order of the general connectivity.

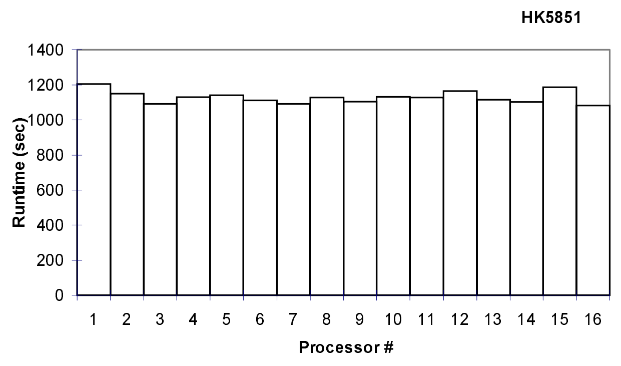


Figure 6. CPU time of each processor in seconds for HK5851.

The three circuits in Table 1 were experimented and the results of two of them are listed in Table 3 and 4.

6. Performance Analysis

The placement results as shown in Tables 3 and 4 are measured by two indicators: maximum block length and the total number of feedthroughs. We define performance improvement percentage of a placement i, η_i, in comparison to a reference as follows.

$$\eta_i = p\frac{\delta \max BL}{\max BL} + \frac{\delta\, feedthrough}{feedthrough}, \qquad (4)$$

where $\delta \max BL$ is the difference of the block length and p is a weight related to the improvement of block length.

Table 3. The experimental results for HK5601.

#Blocks	order	Old	1	2	3	4	5	6	7
9	#feedthroughs	229	249	241	181	223	182	202	213
9	Max Block Length	239	245	239	235	237	234	236	238
9	Max Block Length Difference	6	17	2	8	3	5	7	18
10	#feedthroughs	269	270	259	194	221	189	192	179
10	Max Block Length	219	219	217	215	214	210	213	211
10	Max Block Length Difference	7	5	3	12	8	3	9	9

Table 4 Th experimental results for HK5851.

#Blocks	order	Old	1	2	3	4	5
23	#feedthroughs	1889	1573	1650	1552	1526	1664
23	Max Block Length	588	574	576	569	577	576
23	Max Block Length Difference	13	16	16	12	28	14
24	#feedthroughs	1973	1694	1808	1666	1535	1656
24	Max Block Length	581	555	562	551	549	553
24	Max Block Length Difference	38	19	23	11	40	16

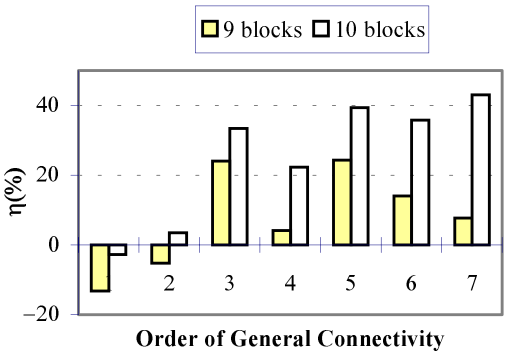


Figure 7. Performance results for HK5601.

The maximum block length reduction may imply the reduction of all the horizontal wire lengths. Therefore, the performance improvement due to its reduction is proportional to the number of blocks in a design. Our experiences indicate that p=0.2B is a good figure to use, where B is the number of blocks in a circuit. Hence the following formula is used to obtain performance values.

$$\eta_{ii} = (0.2B)\frac{m_{ref} - m_i}{m_{ref}} + \frac{f_{ref} - f_i}{f_{ref}}, \qquad (5)$$

where m_i is the maximum block length of placement i, f_i is the number of feedthroughs of placement i, and m_{ref} and f_{ref} are those of the reference placement.

Using Formula (5), we obtained the curves for the two circuits as shown in Figures 8 and 9. The reference point is the results from the placement without clustering.

7. Discussions

The earlier results in [28] and our own experiments have all shown that placement with clustering can result in better performance than that without clustering regardless of the order of general connectivity.

Computation time for general connectivity increases with its order because the depth of the search tree is equal to the order. The time complexity of our algorithm is related to the order of connectivity and the order in practice should not be larger than 5. The parallel version displays its advantage when the order is high.

Our Concurrent Group Search Algorithm is more effective for larger circuits. The combination of the algorithmic techniques and parallel computing results in speedup of 168 times against an early implementation for the largest circuit we have tested with the order of general connectivity equal to 5. This work makes it practical to apply the concept of general connectivity to large industrial circuits.

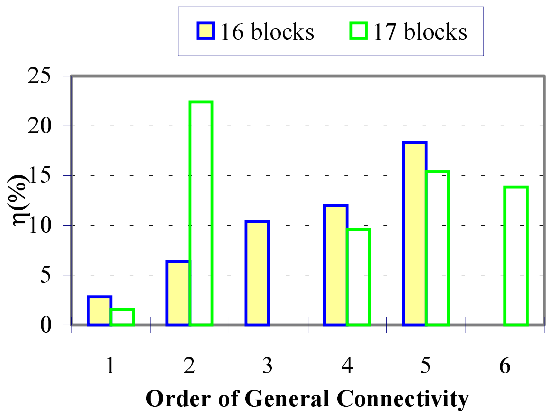


Figure 8. Performance results for HK5852.

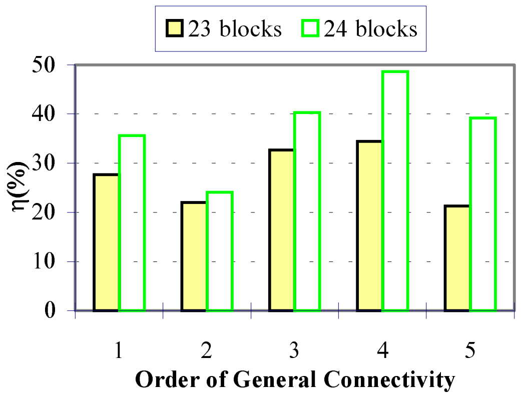


Figure 9. Performance results for HK5851.

The performance improvement with the low order general connectivity does not seem to be stable. The higher order results are more stable and better than the reference ones. It may therefore conclude that the higher order (>3) general connectivity should always be used to ensure stable results, especially for larger circuits.

One of the reasons for the unstable results with the low orders is that “bottom-up” cluster growth may not produce an optimal solution. The reason is that a stable cluster with n cells may not be a subset of any stable cluster with (n+1) cells as shown in Figure 10.

A stable cluster with 4 cells is $\{1,3,4,7\}$. A stable cluster with 5 cells is $\{1,2,3,4,7\}$. As can be seen, $\{1,3,4,7\} \subset \{1,2,3,4,7\}$. The stable cluster with 6 cells is $\{1,3,4,5,6,7\}$. But, $\{1,2,3,4,7\} \not\subset \{1,3,4,5,6,7\}$.

We call this fact “***discontinuity of clustering process***”. Obtaining large clusters from smaller ones may not result in an optimal solution. Therefore, high order

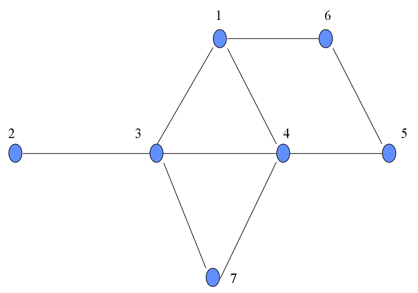


Figure 10. A stable n cell cluster may not be a subset of any stable (n+1) cell cluster.

general connectivity is necessary in order to obtain optimal solutions.

References

[1] M. Brever, “Min-cut placement,” Journal of Design Automation and Fault Tolerant Computing, Vol. 1, No. 4, October 1977, pp.343-362.

[2] P.K. Chan, M.D.F. Schlag, and J.Y. Zien, “Spectral k-way ratio-cut partitioning and clustering,” 30th DAC, 1993, pp.749-754.

[3] K.X. Cheng and W.J. Zhuang, “The placement subsystem of LSIS-II automatic layout system,” Vol. 7, No. 4, 1986, pp.412-417.

[4] J. Cong and M. Smith, “A parallel bottom-up clustering algorithm with application to circuit partitioning in VLSI design,” 30th DAC, 1993, pp.755-760.

[5] W.M. Dai and E.S. Kuh, “Simultaneous floor placement and global routing for hierarchical building block layout,” IEEE Trans. on CAD, Vol. CAD-6, No. 5, 1987, pp.828-837.

[6] W.M. Dai, B. Eschermann, E.S. Kuh, and M. Pedram, “Hierarchical placement and floorplanning in BEAR,” IEEE Trans. on CAD, Vol. CAD-8, No. 12, 1989, pp.1335-1349.

[7] C.L. Ding, C.Y. Ho, M.J. Irwin, “A new optimization driven clustering algorithm for large circuits,” 1993 European DAC, pp.28-32.

[8] W.E. Donath, “Complexity theory and design automation,” Proc. of the 17th DAC, 1980, pp.412-419.

[9] J. Garbers, H.J. Promel and A. Steger, “Finding clusters in VLSI circuits,” 1990 IEEE ICCAD, pp.520-523.

[10] L. Hagen and A.B. Kahng, “A new approach to effective circuit clustering,” 1992 IEEE ICCAD, pp.422-427.

[11] E.Q. Kang, R.B. Lin and E. Shragowitz, “Fuzzy logic approach to VLSI placement,” IEEE Trans. on VLSI Sys., Vol 2., No. 4, Dec. 1994, pp. 489-501.

[12] S. Mallela, et al., “Clustering based simulated annealing for standard cell placement,” 25th DAC., 1988, pp.312-317.

[13] B. Preas and M. Lorenzetti, Ed., Physical Design Automation of VLSI Systems, The Benjamin/Cummings Pub. Co., Inc., 1988.

[14] R. Rajaraman and D.F. Wong, “Optimal clustering for delay minimization,” 30th DAC, 1993, pp.309-314.

[15] Y. Saab, “Post-analysis-based clustering dramatically improves the Fiduccia-Mattheyses algorithm,” 1993 European Design Automation Conference, pp.22-27.

[16] D.M. Schuler and E.G. Ulrich, “Clustering and linear placement,” 1972 DAC, pp50-56.

[17] C. Sechen, et al., “An improved simulated annealing algorithm for tow-based placement,” IEEE ICCAD, 1987, pp.478.

[18] H. Shin and C.H. Kim, “A simple yet effective technique for partitioning,” IEEE Trans. On VLSI Systems, Vol. 1, No. 3, September ‘93, pp.380-386.

[19] J.J. Song, H.K. Choo, and W.J. Zhuang, “A new model for general connectivity and its application to placement,” The 6th Great Lakes Symposium on VLSI, March 22-23, ‘96, pp.60-63.

[20] W. Sun and C. Sechen, “Efficient and effective placement for very large circuits,” IEEE Trans. on CAD, Vol. 14, No. 3, 1995, pp.349-359.

[21] W. Swartz and C. Sechen, “Timing driven placement for large standard cell circuits,” 32nd DAC, 1995, pp.211-215.

[22] N. Yan, H.X. Xia and W.J. Zhuang, “A discussion about connectivity with 3-depth,” Chinese National 6th Conference on Computer Aided Design and Computer Graphics, 1990.

[23] C.W. Yeh, C.K. Cheng and T.T. Y. Lin, “A probabilistic multicommodity-flow solution to circuit clustering problems,” 1992 IEEE ICCAD, pp.422-427.

[24] M.Y. Yu and W.J. Zhuang, “The optimal construction of clusters,” Int. Conf. on Computer Aided Tech., ‘88, pp.456-460.

[25] M.G. Yu, Z.C. Ma, and W.J. Zhuang, “Cluster method and its application to LSI/VLSI layout,” Chinese Journal of Semiconductors, Vol. 10, No. 4, 1989, pp.432-444.

[26] M.Y. Yu, X.L. Hong, Y.E. Lien, Z.Z. Ma, J.G. Bo, and W.J. Zhuang, “A new clustering approach and its application to BBL placement,” European DAC, 1990, pp.665-669.

[27] W.J. Zhuang, K.X. Cheng, et al., “LISI-II automated layout system,” Chinese Journal of Semiconductors, Vol. 8, No. 5, 1987, pp.270-276.

[28] W.J. Zhuang, Y.C. Lim, G. Samudra, and N. Yan, “A new clustering method based on general connectivity,” VLSI Design, Vol. 2, No. 2, 1994, pp. 131-141.