International Conference on Parallel Processing (ICPP) 1994

Parallel Logic Synthesis using Partitioning *

Kaushik De
LSI Logic Corporation
1551 McCarthy Blvd., MS E-192
Milpitas, CA 95035, USA
Email: kaushik@lsil.com

Prithviraj Banerjee
Center for Reliable & High-Perf. Computing
Coord. Sci. Lab., 1308 W. Main Street,
Urbana, IL 61801, USA
Email: banerjee@crhc.uiuc.edu

Abstract

In this paper, we present a partitioning approach of parallel logic synthesis, which is different from the previous approaches which involved parallelization of individual operations within the synthesis algorithm. We partition the given logic circuits and distribute the partitions to different processors for synthesis. For good load balancing, partitioning algorithm is tuned so that the estimated synthesis times of individual partitions are equal. To improve the quality of synthesized circuits, we propose a novel iterative repartitioning and resynthesis approach to parallel logic synthesis. Experimental evaluation in several large circuits are shown on a network of workstations, and results are compared with MIS.

1 Introduction

Combinational logic synthesis deals with the optimization of logic to realize a specific combinational function and many efficient algorithms have been developed recently [1, 2, 3, 4]. However, it is computationally very expensive and several researchers have investigated into parallel algorithms for logic synthesis [5, 6, 7, 8, 9] to reduce the computational time. Recently, some work has been reported which developed portable parallel algorithms for logic synthesis for the Transduction method, and the important feature of the parallel algorithms is that they use asynchronous, message driven model of computation [10, 11]

Very large circuits, however, cannot be handled as a whole by any synthesis algorithm, sequential or parallel alike, due to prohibitive runtimes and memory requirements. As a result, a very large circuit must be partitioned and each partition must be synthesized separately. Since partitions are synthesized separately, global optimization is not possible and the quality of the synthesized circuit will not be optimal. Hence, in order to obtain a good quality circuit by using this partitioning approach, the primary objective of the logic partitioning algorithm needs to partition a given circuit in such a fashion that the potential of sharing common terms among the nodes in a partition is maximal.

Since we plan to synthesize the partitions in parallel, we need to consider one more property of the partitioned circuit during the partitioning process. The completion time of the synthesis procedure in parallel is bounded by the largest completion time among all the partitions. Hence, the secondary objective of this approach is to partition in such a fashion such that the largest synthesis time among all the partitions is minimized.

In this paper, we will describe a parallel logic system using the partitioning approach, called ProperPART. We will describe a new partitioning algorithm which is suitable for the partitioning approach to parallel logic synthesis. We will also describe an iterative approach by which we can improve the quality of the synthesized circuit modestly.

2 The Logic Partitioning Algorithm

2.1 Previous Work in Partitioning

The optimum graph partitioning problem is known to be a NP-complete problem [12]. Efficient heuristics for partitioning based on the group migration method have been proposed by Kernighan and Lin to reduce the total cost of the cut between two partition [13]. A partitioning approach called BEAT_NP based on the seed clustering method has been reported in [14]. This method generates seeds for each partition and the remaining nodes are clustered around the seeds. A recent work has been reported on a circuit partitioning method based on the analysis of reconvergent fanout [15]. Another approach

*Acknowledgement: This research was supported in part by Semiconductor Research Corporartion under grant SRC 92-DP-109 and in part by the Joint Services Electronics Program under contract N00014-90-J-1270

has been presented recently where a probabilistic scheme was used to estimate the size of the don't care sets across the partitions and that estimate was used to minimize the cost of partitioning and improve the testability of the synthesized circuit [16].

2.2 Objectives of Partitioning

The primary objective of partitioning is to retain the logic minimization potential as much as possible. In order to achieve that, the partitions need to capture the gross structural features of the given circuit. Hence, a variant of the clustering method used in BEAT_NP partitioner will be used to make effective use of the information regarding the structure of the given circuit.

We have a secondary objective during partitioning a logic circuit. We plan to synthesize the partitions in parallel. The completion time of the parallel synthesis procedure is bounded by the largest completion among all the partitions. Hence, in order to reduce the completion time of the parallel synthesis procedure, one needs to minimize the largest completion time for synthesis among all the partitions.

2.3 Size of Circuit and Synthesis Time

Since we want to minimize the maximum completion time for synthesis among all the partitions, we need to have some estimate of the completion time for synthesis during partitioning. But the synthesis time of a circuit depends on many factors like the size of the circuit, the synthesis algorithm used, number of primary inputs and outputs, the complexity of the logic expressions of nodes in the circuit, etc. To generate a complete mathematical model for the completion time for synthesis for any give circuit is a very complex task and is beyond the scope of this research.

Hence, we simplified our model considerably. We plan to use MIS [2] to synthesize each partition, so the synthesis algorithm is not a variable in the model. We assume that the synthesis time is a function of the size of the circuit alone. The size of the circuit is measured by the initial literal count of the circuit. We assume that the synthesis time (T) is proportional to some power of the size of the circuit (S) as given in Equation 1.

$$T \; = \; \beta \star S^{\alpha} \tag{1}$$

By applying natural logarithm to both sides of Equation 1, we obtain

$$\log T \; = \; \alpha \star \log S \; + \; \log \beta \tag{2}$$

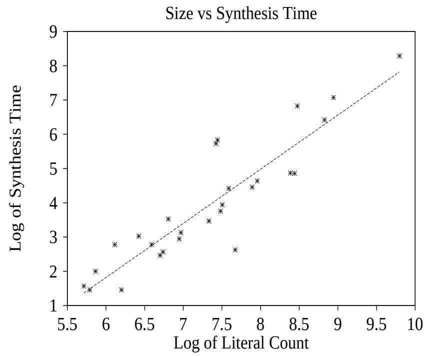

Figure 1: Variation of synthesis time with the size of the original circuit in terms of literal count

To determine the values of α and β empirically, we performed an experiment. We performed synthesis using MIS-II on 27 benchmark circuits with various sizes and collected the runtimes for synthesis and the original sizes of the circuit in terms of the literal count. That data, scatter plotted on a log scale, is presented in Figure 1. Using a statistical method of *least-square line fitting* on that data, we computed the value of α to be 1.58 and the value of β to be 0.00047. Hence, the empirical equation relating the runtime of the synthesis procedure (T) to the size of the circuit in terms of the literal count (S) is given in Equation 3.

$$T \; = \; 0.00047 \star S^{1.58} \tag{3}$$

2.4 Cost Function

A cost function is used to guide the partitioning process, and discern the best move among all possible moves. We mentioned our two objectives of partitioning earlier in this paper. We have modified the cost function given in [14] to suit our purpose. We compute the average size of a partition, AS, as follows.

$$AS \; = \sum_{for\ all\ nodes\ in\ circuit} S(node) \, / \, N$$

where

$$S(node) \; = \; 0.00047 \star (literal\_count(node))^{1.58}$$

and N is the number of partitions. Let us denote AI to be the average number of inputs to each partition. Since AI can not be exactly determined a priori, it is approximated as

$$AI \; = \; number\, of\, primary\, inputs \, / \, 2$$

Table 1: Comparison on Literal counts and the runtimes (1, 2 and 4 processors) between our partitioning algorithm (ProperPART) and BEAT_NP for 4 partitions

CKT	ProperPART				BEAT_NP			
	Lit	Run Time (sec)			Lit	Run Time (sec)		
	Cnt	1 P	2 P	4 P	Cnt	1 P	2 P	4 P
seq	2369	687.55	417.52	236.30	1936	800.14	524.26	476.84
des	4526	1718.01	1157.29	717.70	4202	2074.81	1106.38	903.99
k2	1611	2397.37	2210.11	2123.63	-	-	-	-
C7552	2948	443.29	335.99	231.78	3469	936.62	785.40	738.00
C6288	3673	1091.57	602.62	365.58	3998	379.35	260.61	204.08
C5315	2284	234.48	168.51	132.56	2438	228.43	172.01	129.79
duke2_berger	826	162.60	130.39	99.44	-	-	-	-

Let us consider a node η we want to put in a partition π. Let us denote DI_π to be the change in the number of inputs of the block π caused by moving η into π, and $PS(\pi)$ to be the size of the partition π prior to moving the node η to π. Then the cost of moving the node η to the partition π is expressed as follows:

$$\begin{aligned} cost(\eta, \pi) \;&=\; C_1 \star (DI_\pi / AI) \;-\; (1 - C_1)\star \\ &(S(\eta)/AS) \star SIGN(AS - PS(\pi) - S(\eta)) \end{aligned} \tag{4}$$

where $SIGN(val)$ = -1.0 if val < 0, otherwise it is 1.0.

The cost function given in Equation 4 has two parts. The first part penalizes a move if it introduces a lot of additional inputs to the block. Hence, that part encourages the acceptance of a node which forms a good cluster. The second part of the cost function encourages a move of a large size node into the block as long as the block size does not exceed AS after the node is moved. On the other hand, if the size is going to exceed AS, it penalizes that move. This part of the cost function encourages the formation of equal size partitions.

The BEAT_NP partitioning algorithm [14] implicitly assumes all the nodes to be of equal sizes; hence, it gives the same weight to all the individual nodes. In our partitioning algorithm, we used the literal count of a node as a weight of that node. We performed experiments to observe the effectiveness of that decision. In Table 1, we compare the final literal counts and the runtimes (on 1, 2 and 4 processors) obtained by applying the one-pass approach (described in a later section of this paper) with two partitioning algorithms: 1) our proposed partitioning algorithm, ProperPART, and 2) our implementation of the BEAT_NP algorithm. This experiment was performed by partitioning the given circuits into 4 parts. One can observe from the data presented in Table 1 that for most of the circuits, the runtimes were much higher for the BEAT_NP algorithm for 1 processor compared to those with our partitioner. For two circuits, *k2* and *duke2_berger*, synthesis could not be completed when BEAT_NP was used to partition them. Another point to be observed is that the speedups obtained with BEAT_NP were poor compared to those obtained with our partitioner, as we went for multiple processors. This shows that the load balancing is not good with the BEAT_NP algorithm.

2.5 Methodology

The partitioning procedure starts by generating N seeds for N partitions. The seed generation method is similar to the approach given in [14]. The seeds are generated such that they are maximally away from the primary inputs and outputs as well as themselves. Then the other nodes are placed one by one in different partitions. The procedure starts by selecting the partition π which has the minimum size in terms of the literal count. It checks all the neighbors of the partition π, and the node ν which has the minimum cost to move into π according to Equation 4 is chosen and placed in π. If no such suitable neighbor is found, a new seed is generated by using the procedure described in the last paragraph and is placed in π. This process is repeated until all the nodes are placed in one of the N partitions.

3 One Pass Approach of Synthesis

3.1 Methodology

In this section, we will describe the overall synthesis methodology using the one-pass approach. The entire system is developed as a part of the ProperCAD

Table 2: Comparison of quality (literal count in sum-of-products form) and runtime in a single processor (in sec) obtained by applying ProperPART with one-pass approach with that obtained by applying MIS 2.2 on the entire circuit

CKT	Init Lit Cnt	MIS 2.2		ProperPART (One pass)			
				4 Partitions		8 Partitions	
		Lit	Time	Lit	Time	Lit	Time
seq	17973	1367	7907.40	2369	687.55	2974	503.08
des	7657	4024	2784.00	4526	1718.01	4892	1374.40
k2	3063	-	-	1611	2397.37	2003	372.84
C7552	6144	2691	652.60	2948	443.29	2980	334.27
C6288	4800	3729	1037.20	3711	428.37	3695	315.18
C5315	4386	2010	241.70	2284	234.48	2431	199.29
duke2_berger	1314	-	-	826	162.60	920	72.38

project [17], based on the CHARM runtime system [18] and is named as ProperPART. This system is portable across a variety of parallel machines, but we will report results on only a network of workstations. Given a combinational circuit, it first partitions the circuit into N partitions, using the partitioning algorithm described in the last section. The partitioning is performed on a single processor. We have not looked into the problem of parallelizing the partitioning algorithm because it is beyond the scope of this research; also, the partitioning time forms a small fraction of the total synthesis time. If a suitable parallel partitioning algorithm is available, that algorithm can be applied to partition the circuit in parallel using multiple processors.

After the partitioning is performed, individual partitions are distributed to different processors by the CHARM runtime system. When a partition is picked up by a processor, that partition is synthesized by a combinational synthesis algorithm. We have used the MIS algorithm [2] to synthesize the individual partitions, but we could have used any other synthesis algorithm like the Transduction method [3] as well. After the completion of synthesis on all the partitions, all the synthesized partitions are merged to form the synthesized circuit.

3.2 Experimental Results

In this subsection, we compare the experimental results obtained by applying the one-pass approach on various *ISCAS* and *MCNC* benchmark circuits. In Table 2, we compare the literal counts (in sum-of-products form) and the runtimes (on a uniprocessor SUN4 workstation) obtained by running MIS 2.2 with those obtained by running the one-pass approach of ProperPART on the benchmark circuits. The runtime for ProperPART for the one-pass approach on any circuit includes the initial partitioning time, parallel synthesis time for various partitions (on uniprocessor, partitions were synthesized one by one) and the final merge time. A '-' in any table means it either ran out of memory or it could not finish in 40 hours.

One can observe that the quality of the final synthesized circuit obtained by ProperPART is not as good as that obtained by running MIS 2.2 on the entire circuit. It is also clear that the quality of the synthesized circuit goes down as the number of partitions increases. But on some circuits, MIS 2.2 could not be run on the entire circuit because either it ran out of memory or it could not finish after running for a long time. Those circuits can only be synthesized by this partitioning approach. One can also observe that the runtime for one-pass approach of ProperPART for a large circuit is much smaller than that for MIS 2.2 on the same circuit and it becomes smaller as the number of partitions increases.

We will now present the speedup results for the one-pass approach of ProperPART on a network of SUN4 workstations. The results for 4 partitions is presented in Table 3 and the results for 8 partitions is presented in Table 4. One can observe that the speedup results are reasonably good for most of the circuits. Only the circuit $k2$ performed poorly for 4 partitions in terms of the speedup result. It is because one of the partitions became much larger than the others and the runtimes were dominated by the synthesis time of that partition.

Table 3: Runtime(speedup) results obtained by applying ProperPART with one-pass approach on a network of SUN4 workstations for 4 partitions

CKT	1 Proc. Sec(spd)	2 Proc. Sec(spd)	4 Proc. Sec(spd)
seq	687.55(1.0)	417.22(1.7)	236.20(2.9)
des	1718.01(1.0)	1157.29(1.5)	717.70(2.4)
k2	2397.37(1.0)	2210.11(1.1)	2123.63(1.1)
C7552	443.29(1.0)	335.99(1.3)	231.78(1.9)
C6288	428.37(1.0)	262.79(1.6)	184.28(2.3)
C5315	234.48(1.0)	168.51(1.4)	132.56(1.8)
duke2_berger	162.60(1.0)	130.39(1.3)	99.44(1.6)

Table 4: Runtime(speedup) results obtained by applying ProperPART with one-pass approach on a network of SUN4 workstations for 8 partitions

CKT	1 Proc. Sec(spd)	2 Proc. Sec(spd)	4 Proc. Sec(spd)	8 Proc. Sec(spd)
seq	503.08(1.0)	279.77(1.8)	137.15(3.7)	126.49(4.0)
des	1374.40(1.0)	812.14(1.7)	485.39(2.8)	393.50(3.5)
k2	372.84(1.0)	285.70(1.3)	142.48(2.6)	125.18(3.0)
C7552	334.27(1.0)	242.24(1.4)	135.47(2.5)	113.85(3.0)
C6288	315.18(1.0)	188.80(1.7)	96.95(3.3)	68.32(4.6)
C5315	199.29(1.0)	140.75(1.4)	79.73(2.5)	65.44(3.1)
duke2_berger	72.38(1.0)	49.29(1.5)	28.89(2.5)	22.37(3.2)

4 Iterative Approach of Synthesis

4.1 Methodology

The major limitation of the one-pass approach described in the last section is that the quality of the circuit is not optimal because the synthesis is performed on only one partition at a time. There will be no sharing of common logic among the nodes which are in different partitions. That can potentially degrade the quality of the resultant synthesized circuit. Also, very large circuits cannot be resynthesized to improve the quality because of prohibitive runtimes and memory requirements. Hence, we have devised an iterative procedure to improve the quality of the circuit. The main idea is to allow synthesis among certain constrained sets of nodes which are in different partitions at one time, and this procedure is repeated a certain number of times. This iterative procedure is explained with an example in Figure 2 with 4 partitions. The partitions are numbered from 1 to 4 in the figure.

Figure 2(I) shows the first phase of the iteration. This is the same as the one pass approach described in the last subsection, i.e., each partition is synthesized independently. In this phase, a node η in a particular partition can share logic with only the nodes in the same partition as η. After the first phase, we obtain the synthesized version of the four partitions of the circuit. In the second phase, shown in Figure 2(II), we bi-partition each of the four partitions obtained in the last phase and mark them as A and B. Then we merge the partitions $1A$ and $2A$ to form the new partition 1 and merge $1B$ and $2B$ to form the new partition 2. Similarly, we merge the partitions $3A$ and $4A$ to form the new partition 3 and merge $3B$ and $4B$ to form the new partition 4. Now these new partitions (1 to 4) are synthesized independently. In this phase, one half of the nodes of partition 1 are synthesized with one half of the nodes of partition 2, and the other half of the nodes of partition 1 is synthesized with the other half of the nodes of partition 2. This will allow some logic sharing among the nodes in partitions 1 and 2. The same is true for partitions 3 and 4. This can potentially improve the quality of the circuit, but will never degrade the quality. In the third phase, as shown in

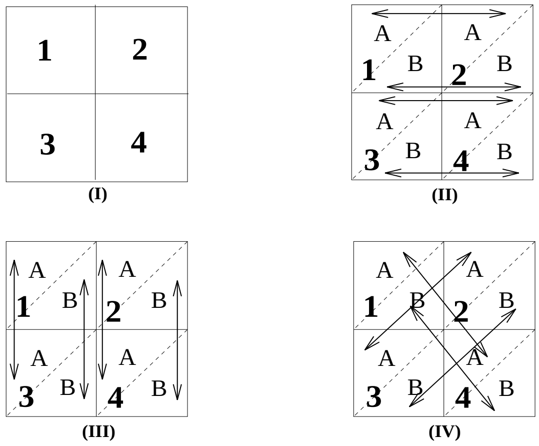

Figure 2: An example of iterative approach of synthesis using the partitioning approach with 4 partitions

Figure 2(III), each partition is bi-partitioned again. But this time, partitions 1A and 3A (1B and 3B) are paired and partitions 2A and 4A (2B and 4B) are paired and the same procedure is repeated. In the fourth phase, as shown in Figure 2(IV), partitions 1A and 4A (1B and 4B) are paired and partitions 2A and 3A (2B and 3B) are paired and the same procedure is repeated.

We need to generate the pairing of different partitions for different phases of this iterative approach. We will assume that the number of partitions, N, is a power of 2, i.e., N = 2^k where k is a positive integer. We need to generate the pairing in such a way that each partition is paired with different partitions in different phases of this iterative approach. Also, in any phase, any particular partition is involved in only one pairing. Then, it is obvious that the number of phases is the same as the number of partitions, N. Also, the number of pairings in any phase is N. For example, for 4 partitions, the pairings at different phases are given as

Phase 2: [(1A, 2A), (1B, 2B), (3A, 4A), (3B, 4B)]
Phase 3: [(1A, 3A), (1B, 3B), (2A, 4A), (2B, 4B)]
Phase 4: [(1A, 4A), (1B, 4B), (2A, 3A), (2B, 3B)]

The phase 1 (whose pairing can be listed as [(1A, 1B), (2A, 2B), (3A, 3B), (4A, 4B)]) is the same as the one pass approach, i.e., all the individual partitions are synthesized independently. The procedure for generating pairing for all phases is omitted due to lack of space.

It can be observed that in the iterative approach, the very first partitioning and the very last merging are performed by one processor. During the other phases, N partitions are bi-partitioned and then they are merged to form N new partitions according to the pairings listed for that phase. Those operations can be performed in parallel by distributing those jobs to different processors. Also, the synthesis on different partitions can be performed in parallel by distributing them to the different processors. Another important feature of this iterative approach is that the size of the partitions handled by the synthesis algorithm remains approximately the same at different phases of this iterative approach, the partition sizes do not grow. This is because we are bi-partitioning and merging in different combinations in different phases, but we are not merging two existing partitions to form a bigger partition. Hence, it is possible to apply this approach to the large circuits which can not be handled as a whole by the synthesis algorithms.

4.2 Experimental Results

In Table 5, we compare the literal counts (in sum-of-products form) and the runtimes (on a uniprocessor SUN4 workstation) obtained by running MIS 2.2 with those obtained by running the iterative approach of ProperPART on the benchmark circuits. The runtime for ProperPART for the iterative approach on any circuit includes the initial partitioning time, parallel partitioning-merge-synthesis times at different phases (on uniprocessor, done one by one sequentially) and the final merge time. As mentioned earlier, a '-' in any table means it either ran out of memory or it could not finish in 40 hours.

One can observe that the quality of the synthesized circuit obtained by the iterative approach is always better compared to the quality obtained by the one pass approach. But the quality is not as good as that obtained by applying MIS 2.2 on the entire circuit, whenever it is possible to run MIS 2.2 on the entire circuit. But for two circuits, *k2* and *duke2_berger*, MIS 2.2 could not run on the whole circuit. It can be also observed that the runtime for the iterative approach increases as the number of partitions increases. This is due to the fact that the number phases for the iterative approach increases as the number of partitions increases, which in turn increases the runtime.

We will now present the speedup result for the iterative approach of ProperPART on a network of SUN4 workstations. The runtimes and speedup results with 4 partitions are presented in Table 6 and the results for 8 partition are presented in Table 7. The speedup results are very good for most of the

Table 5: Comparison of quality (literal count in sum-of-products form) and runtime in single processor (in sec) obtained by applying ProperPART with one pass approach and iterative approach with that obtained by applying MIS 2.2 on the entire circuit

CKT	Init Lit Cnt	MIS 2.2		ProperPART (One Pass)				ProperPART (Iterative)			
				4 Partitions		8 Partitions		4 Partitions		8 Partitions	
		Lit	Time	Lit	Time	Lit	Time	Lit	Time	Lit	Time
des	7657	4024	2784.00	4526	1718.01	4892	1374.40	4383	2707.40	4659	3185.37
k2	3063	-	-	1611	2397.37	2003	372.84	1500	2655.66	1692	989.44
C7552	6144	2691	652.60	2948	443.29	2980	334.27	2824	834.20	2893	1225.26
C6288	4800	3729	1037.20	3711	428.37	3695	315.18	3673	1091.37	3652	1566.78
C5315	4386	2010	241.70	2284	234.48	2431	199.29	2179	534.61	2320	884.69
duke2_berger	1314	-	-	826	162.60	920	72.38	774	276.08	840	370.63

circuits. Another point to be observed is that the speedup results presented for the iterative approach is much better than those obtained for the one-pass approach (presented in the last section). This is because the fraction of all the works which can be performed in parallel is much more for the iterative approach than for the one-pass approach. In each phase of the iterative approach, N partitioning, merging and synthesis are performed, where N is the number of partitions. Those operations can be performed in parallel. Also there are N phases in the iterative approach. As a result, the speedup results are better with larger number of partitions, as it is obvious from the results in Table 6 and 7.

5 Conclusions

In this paper, we have presented a parallel logic system using partitioning. Given a combinational circuit, the circuit is partitioned into N partitions, those partitions are synthesized in parallel by using multiple processors, and then the synthesized partitions are merged to form the synthesized circuit. This approach is specially suitable for the very large circuits which cannot be handled as a whole by any synthesis algorithm due to prohibitive runtimes or memory requirements. In this paper, we have presented a new partitioning algorithm suitable for this approach.

Since in this approach the partitions are synthesized independently, in most of the cases the quality of the synthesized circuit will not be as good as it would be if the entire circuit as a whole is synthesized (whenever it is possible to synthesize the entire circuit as a whole). Hence, we have devised an iterative approach to improve the quality of the synthesized circuit by performing synthesis at different phases. At each phase, only certain sets of nodes are allowed to perform synthesis together. The results show that the quality the synthesized circuit improves modestly by using this iterative approach over that obtained by the one-pass approach.

References

[1] R. K. Brayton and et al., "ESPRESSO-II: A New Logic Minimizer for Programmable Logic Arrays," *CICC*, pp. 370–376, June 1984.

[2] R. Brayton, R. Ruddel, A. Sangiovanni-Vincentelli, and A. Wang, "MIS: A Multiple-level Logic Optimization System," *IEEE Transactions on Computer-Aided Design*, pp. 1062–1081, November 1987.

[3] X. Xiang, *Multilevel Logic Network Synthesis Systems, SYLON-XTRANS*. PhD thesis, Univ. of Illinois, 1990.

[4] K. A. Barlett, D. Bostick, G. Hachtel, R. Jacoby, and M. Lightner, "BOLD: A Muliple-level Logic Optimization System," *International Conference on Computer Aided Design*, 1987.

[5] R. Galivanche and S. M. Reddy, "A Parallel PLA Minimization Program," *Design Automation Conference*, pp. 600–607, 1987.

[6] G. D. Hachtel and P. H. Moceyunas, "Parallel Algorithms for Boolean Tautology Checking," *ICCAD*, pp. 422–425, 1987.

[7] H. T. Ma, S. Devadas, and A. S. Vincentelli, "Logic Verification Algorithms and their Parallel Implementations," *24th DAC*, 1987.

Table 6: Runtime(speedup) results obtained by applying ProperPART with iterative approach on a network of SUN4 workstations for 4 partitions

CKT	1 Proc. Sec(spd)	2 Proc. Sec(spd)	4 Proc. Sec(spd)
des	2707.40(1.0)	1672.64(1.6)	1014.64(2.7)
k2	2655.66(1.0)	2348.56(1.1)	2194.99(1.2)
C7552	834.20(1.0)	536.34(1.6)	391.40(2.1)
C6288	1091.57(1.0)	602.62(1.8)	365.58(3.0)
C5315	534.61(1.0)	323.98(1.6)	216.44(2.5)
duke2_berger	276.08(1.0)	193.44(1.4)	133.39(2.0)

Table 7: Runtime(speedup) results obtained by applying ProperPART with iterative approach on a network of SUN4 workstations for 8 partitions

CKT	1 Proc. Sec(spd)	2 Proc. Sec(spd)	4 Proc. Sec(spd)	8 Proc. Sec(spd)
des	3185.37(1.0)	1721.43(1.8)	962.44(3.3)	652.44(4.9)
k2	989.44(1.0)	605.44(1.6)	308.06(3.2)	210.80(4.7)
C7552	1225.26(1.0)	685.30(1.8)	365.83(3.4)	241.66(5.1)
C6288	1566.78(1.0)	822.84(1.9)	416.21(3.8)	240.07(6.5)
C5315	884.69(1.0)	494.14(1.8)	265.15(3.3)	168.14(5.3)
duke2_berger	370.63(1.0)	213.25(1.7)	116.35(3.2)	77.44(4.8)

[8] C. F. Lim, P. Banerjee, K. De, and S. Muroga, "A Shared Memory Parallel Algorithm for Logic Synthesis," *The Sixth International Conference on VLSI Design*, January 1993.

[9] G. Zipfel, "Parallel Algorithm for Algebraic Factorization with Application to Multi-Level Logic Synthesis," Master's thesis, Univ. of Illinois, 1991.

[10] K. De, B. Ramkumar, and P. Banerjee, "ProperSYN: A Portable Parallel Algorithm for Logic Synthesis," *International Conference in Computer-Aided Design*, pp. 412–416, 1992.

[11] K. De, *Parallel Algorithms for Logic Synthesis.* PhD thesis, Univ. of Illinois, 1993.

[12] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and co., San Fransisco, California, 1979.

[13] B. W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs," *Bell System Technical Journal*, vol. 49, pp. 291–307, 1970.

[14] H. Cho, G. Hachtel, M. Nash, and L. Setiono, "BEAT_NP: A Tool for Partitioning Boolean Networks," *Proc. International Cinference of Computer Aided Design*, pp. 10–13, 1988.

[15] S. Dey, F. Berglez, and G. Kedem, "Corolla Based Circuit Partitioning and Resynthesis," *27th Design Automation Conference*, pp. 607–612, 1990.

[16] K. De and P. Banerjee, "PREST: A System for Logic Partitioning and Resynthesis for Testability," *IEEE Transactions on VLSI Systems*, pp. 514–525, December 1993.

[17] B. Ramkumar and P. Banerjee, "ProperCAD: A Portable Object-oriented Parallel Environment for VLSI CAD," *International Conference in Computer Design*, 1992.

[18] L. V. Kale, "The Chare Kernel Parallel Programming System," *International Conference on Parallel Processing*, August 1990.