ProperSYN: A Portable Parallel Algorithm for Logic Synthesis *

Kaushik De[†] **Balkrishna Ramkumar**[‡] **Prithviraj Banerjee**[†]

[†] Center for Reliable & High-Perf. Computing
Coordinated Science Laboratory
University of Illinois
Urbana, IL 61801

[‡] Dept. of Electrical & Computer Engg.
3406 Engineering Building
University of Iowa
Iowa City, IA 52242

Abstract

Parallel processing is fast becoming an attractive solution to reduce the computation time of CAD applications. Much of the work in parallel algorithms for CAD reported to date, however, suffers from a major limitation. The parallel algorithms proposed for the CAD applications are designed with a specific underlying parallel architecture in mind. We have developed a portable parallel algorithm based on the Transduction method [1], called ProperSYN. The same algorithm runs on a variety of parallel machines. Experimental results on various parallel machines are presented.

1 Introduction

Combinational logic synthesis deals with the optimization of logic to realize a specific combinational function [2], [1]. Logic synthesis for large circuits have tremendous computing times and memory requirements. Parallel processing offers an attractive solution to this problem, hence researchers have started to investigate parallel algorithms for logic synthesis and verification [3] [4] [5].

Much of the work in parallel algorithms for CAD, however, suffers from a major limitation of being designed for a specific underlying parallel architecture. Hence, those parallel algorithms need to be developed afresh for every target parallel architecture. The ongoing ProperCAD (PoRtable Object-oriented Parallel EnvioRment for CAD) project offers a novel solution to that problem by addressing it in two fronts. [1] First, the ProperCAD environment offers a portable platform to develop parallel algorithms which will run in various MIMD machines, shared and distributed memory alike, *unchanged*. Second, it allows the development of parallel algorithms *around* existing sequential algorithm with a strict interface such that the parallel algorithm can benefit from future improvements in sequential algorithm.

In this paper, we present a parallel algorithm for logic synthesis using the Transduction Method [1] and the system implemented in the ProperCAD environment is named as ProperSYN. The Transduction method is known to produce very high quality logic, but it requires large execution time. Our parallel algorithm attempts to make the execution time manageably small. The parallel algorithms in ProperSYN system is novel and different from that described by Lim [5]. Our algorithm uses an asynchronous message driven computing model with no synchronizing barriers, hence it is scalable to larger number of processors. Also, our algorithm is portable across a wide variety parallel machines. Finally, our algorithm is built around a well defined sequential algorithm interface such that we can benefit from future expansion of the sequential algorithm.

2 Overview of the Transduction Method

2.1 Terminology

In this paper, we will consider loop-free multi-level combinational networks consisting of simple gates such as AND, OR NAND, NOR and NOT gates. However, it is conceptually trivial to extend the methods discussed in this paper to handle more complex gates like XOR and XNOR.

A multilevel network consists of gates and connections. A network can be viewed as a directed acyclic graph, consisting of the gates as the nodes in the graph and the connections as the edges in graph. The Transduction method is based on the concept of *sets of permissible functions* [1]. The output function of a gate can be any member of its set of permissible functions without changing the functionalities of the primary outputs of the network. There are two types of permissible functions, the *maximum set of permissible functions (MSPF)* and the *compatible set of permissible functions (CSPF)*. The MSPF is more powerful, but it requires very large computation time. For our parallel algorithms reported in this paper we used the

*Acknowledgement: This research was supported in part by Semiconductor Research Corporartion under grant SRC 91-DP-109, in part by NASA under grant NAG 1-613, and in part by the National Science Foundation PYI grant NSF MIP 87-57563

[1] This environment is built on top of a platform for portable parallel programming called the Chare kernel [6].

CSPF, but we could have used MSPF as well without any conceptual change of our algorithms.

2.2 Types of Transformation

A suite of transformations and optimizations can be applied to the network with the help of the CSPFs of the gates and the connections. The Transduction method consists of four main transformations, namely, *gate substitution*, *gate merging*, *generalized gate substitution* and *gate input reduction*. It also *prunes* redundant connections in the network.

In *gate substitution*, a gate v_i can be substituted by another gate v_j if the output function of v_j is a member of the CSPF of v_i. Then v_i can be deleted from the network. In *gate merging*, two gates v_i and v_j whose CSPFs intersect are tried to be merged by synthesizing a new gate whose output function is a member of the intersection of CSPFs of two gates. The connection to the new gate is from a minimal set of existing gates in the network. Then both v_i and v_j can be removed from the network. A more general version of gate substitution is *generalized gate substitution*, where each output connection of a gate v_i is substituted by an existing gate in the network. Then v_i can be removed from the network. Finally, in *gate input reduction*, one tries to reduce the number of input connections to a gate v_i by trying to synthesize a new gate whose output function is a member of the CSPF of v_i such that the number of inputs to the new gate is less than that of v_i. Then the new gate can replace v_i, and v_i can be removed.

3 Parallel Algorithms for Transduction Method

3.1 The ProperCAD Parallel Algorithm Framework

In the ProperCAD approach to the design of parallel CAD algorithms, a parallel algorithm is designed around an existing sequential algorithm by identifying modules in the sequential code and designing a well-defined interface between the parallel and sequential code. Parallel execution follows a coarse-grained dataflow model and is message driven. Conceptually, a pool of messages is created where each message may either request the creation of a new *object* or be destined for an existing *object*. An *object* can only create new messages or be waken up by messages sent to it at designated entry points. Processes are not allowed to block, or explicitly make receive requests from other processes. Every message thus represents some work. The machine specific code is responsible for dynamically distributing the messages across the available processors. The environment also provides one process per processor for data abstraction, called *work manager*. All the processes in a processor can access the data using access routines. Further details on the ProperCAD environment can be found in [7].

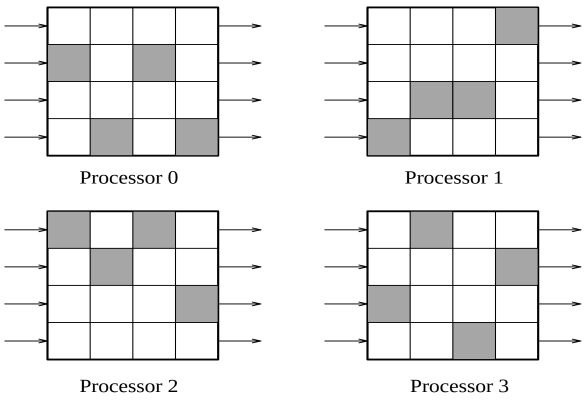

Figure 1: Partitions of a Network and the "ownership" of each processor

3.2 Parallel Synthesis Algorithm Overview

An obvious method of parallelism for logic synthesis is to divide the network into non-overlapping partitions and synthesize each partition in parallel. But in that process, one looses in quality as well as in testability. So, we have not used that approach. Instead, we synthesize the entire network as a whole. In our ProperSYN algorithm, we divide the logic network into a set of non-overlapping partitions. The partitioning is for *parallelization purposes only*, the transformations and the optimizations are performed on the entire network.

For each partition, one *object* is created. Processes that perform various operations on these objects are distributed to different processors by the ProperCAD system using some load balancing methods within the ProperCAD framework, as depicted in Figure 1. We typically create more partitions than the number of processors for better load balancing, but without adding a lot of communication overhead. We will denote the object for a partition p as *partition_object* p. Each processor has a copy of the network, but has the responsibility for the shaded regions of the network. Effectively, we can say that each processor "owns" some parts of the network.

3.3 Parallel Evaluation of Output Function and CSPF

To support the transformations and optimizations in the transduction method, we need to evaluate the output functions and CSPFs at each gate and connection in the network. The output functions need to be expressed in terms of the primary inputs of the network. We compute output functions and CSPFs using the binary decision diagrams (BDD).

Due to the fact that different processors will be

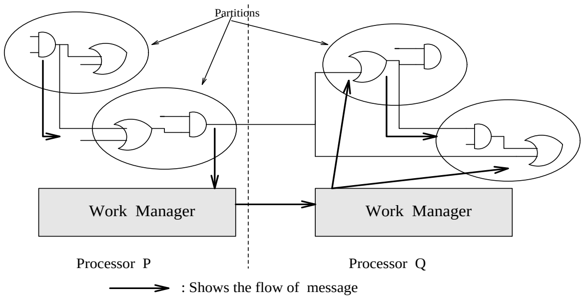

Figure 2: Evaluation of Output Functions and the Flow of Messages

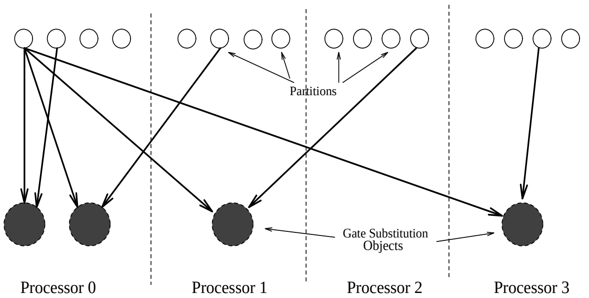

Figure 3: Creation of objects for gate substitution

performing different logic transformations simultaneously, the output functions and the CSPFs will keep changing. To maintain coherence among various applications, we introduced the concept of 'version' for each BDD. If the version of a BDD is not 'current', then it is not a "valid" BDD.

The evaluation of output functions start from the primary inputs of the network. The output function of a gate can be evaluated if "valid" output functions of its fanins are available. After the output function of a gate ζ is evaluated in a partition, it is checked if its output function is needed in any other partition. If the other partition that needs the output function is on the same processor, that *partition_object* is informed about the availability of the output function of ζ. But if it is on a different processor, then the BDD is transmitted to the other processor through the *work managers* of the corresponding processors. When the output function of a gate reaches a processor, partitions which need that output function will be informed.

The flow of evaluation of CSPF moves from primary outputs to the primary inputs. The procedure is almost similar to that of output function. The details are discussed in [8].

3.4 Parallel Gate Substitution

Gate substitution is one of the transformations provided in the Transduction method. The main idea of the gate substitution is to search a given network for pair of gates such that in each pair, one of the gates can substitute for the other. However, as the network needs to be loop-free, the substitute gate must not be a successor of the gate to be substituted.

Let us denote p as the number of *partition_objects* created. Then the *partition_object* x will create another p objects, called *gate_sub_objects*. They are denoted as the pair (x, y), x = 1, ..., p, y = 1, ..., p. The *gate_sub_object* denoted by (i, j) is responsible for substitution of any gate in the *partition_object* i by any gate in the *partition_object* j. Note that there will be another *gate_sub_object* (j, i) will be created whose job is just the opposite of the job of (i, j). Hence, the number of *gate_sub_objects* created will be p^2 as shown in Figure 3.

Now, consider the *gate_sub_object* (i, j). A gate g_1 in the partition i can be substituted by a gate g_2 in the partition j if the output function of the gate g_2 is a subset of the CSPF of the gate g_1. So the *gate_sub_object* (i, j) will need "valid" CSPF of all the gates in the partition i and output functions of the gates in the partition j.

When the *gate_sub_object* denoted by (i, j) is awakened by a message, it starts by collecting the output functions and the CSPFs that it needs. Some output functions and CSPFs may not be available that time, so it requests the *work manager* in its processor to get the BDD and inform it when that is available. After this *gate_sub_object* has collected all the available "valid" BDDs, it does pairwise comparisons to find out any possible gate substitution. If any gate substitution is possible, it asks permission from the *master* processor (a designated processor). When it is finished with all comparisons, it asks *partition_object* i to inform it if the CSPF of any gate in the partition i has changed and it asks *partition_object* j to inform it if the output function of any gate in the partition j changes.

Whenever an output function or CSPF arrives from other processor, or the current version gets computed in this processor, the *work manager* checks its list to find out if any object has asked for this BDD. If so, it sends a message to the corresponding objects. Whenever a *gate_sub_object* (i, j) receives a message about availability of an output function or CSPF, it checks for some other possible gate substitution.

When the *master* processor receives the request for permission for a gate substitution, it checks if the gates exist and if the gate substitution has not become invalid due to some other transformations. If "valid", it then informs other processors about this gate substitution and then performs the substitution. It is through

Table 1: Runtime results in various parallel machines

CKT	Encore Multimax		Intel/860		Network of Workstations	
	1 Proc. (sec)	8 Proc. (sec)	1 Proc. (sec)	8 Proc. (sec)	1 Proc. (sec)	4 Proc. (sec)
5xp1	109.4	18.3	29.2	3.5	64.8	26.2
9symml	5027.7	675.4	1339.7	307.5	2977.2	976.4
b9	165.9	32.6	43.2	10.2	96.5	23.2
bw	594.9	67.8	158.6	32.2	352.5	80.4
f51m	232.7	48.2	62.0	13.2	137.9	38.2
misex1	51.9	7.7	13.8	5.3	30.7	12.9
misex2	162.9	29.7	43.4	10.0	96.5	27.9
rd73	688.5	90.1	182.4	21.6	547.2	172.4
rd84	6406.3	791.2	1704.7	206.3	3795.0	1207.3
sao2	839.4	107.6	223.7	30.0	497.2	107.5
vg2	833.4	107.1	222.7	32.2	439.9	140.3
duke2	32107.3	4479.2	8545.3	1241.2	-	-
misex3c	45484.6	6421.7	12147.6	1953.4	-	-

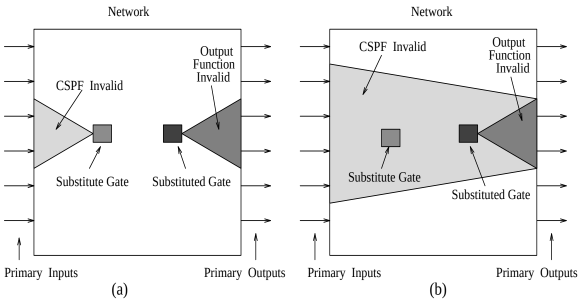

Figure 4: Invalidation of Output Function and CSPF due to gate substitution

this centralized update, that coherence is maintained.

After the substitution is performed, the output functions of all the *successors* of the substituted gate will be invalid as well as the CSPFs of the *predecessors* of the substitute gate. This is shown in Figure 4(a). But if the output functions are reevaluated in the region where they are marked, then the CSPFs of all the gates in the transitive fanins of that region will get invalidated. Since in our approach, we do not have any barrier synchronization to decide when to start reevaluation of the output functions and the CSPFs, the reevaluation can start any time in any order. So we mark the invalidation of output function and the CSPFs as shown in the Figure 4(b). After the invalidation of the output functions and the CSPFs, the corresponding *partition_objects* are sent messages to reevaluate the BDDs. After they have reevaluated the BDDs, they will inform the corresponding *gate_sub_objects* to check for more gate substitutions as the BDDs have changed.

The algorithms for parallel gate merging, parallel generalized gate substitution, parallel gate input reduction and parallel pruning are not discussed due to lack of space. Those are discussed in detail in [8].

4 Results

The parallel algorithms developed in the last section have been implemented on the ProperCAD environment and are part of a synthesis system called ProperSYN. Of the four transformations described, parallel *gate substitution* and parallel *gate merging* have already been implemented. The two other transformations, namely *generalized gate substitution* and *gate input reduction* are under implementation and will be completed shortly. All the results presented for ProperSYN in this paper are with respect to *gate substitution* and *gate merging* only. The quality of the resultant circuits are expected to improve after inclusion of those transformations. The system contains approximately 9000 lines of C code.

We would like to reiterate that to run ProperSYN in different machines, *no* change in the parallel program is necessary. One has to compile the program in that machine and link to the ProperCAD library and it becomes ready to execute. Table 1 presents results for runtimes on various parallel machines. Encore Multimax is a *shared memory MIMD* machine with 8 processors. Intel/860 machine is a *distributed memory MIMD* machine with 8 processors connected by a hypercube configuration. Network of workstations environment behaves as a *distributed processing* system. There is a *host* workstation, which distributes work to other workstations. The workstations used in this experiments are SUN4 workstations with SPARC 1 processor in them.

Table 2: Comparison of Quality of Network Produced by ProperSYN, the Transduction Method and MIS-II

CKT	Initial (g/l)	MIS (g/l)	XTRAN (g/l)	ProperSYN 1 Proc (g/l)	ProperSYN 8 Proc (g/l)
5xp1	126/276	99/216	51/121	59/130	61/135
9symml	154/397	134/308	112/314	113/340	122/345
b9	65/227	68/209	-	48/191	48/191
bw	202/480	136/276	105/234	94/267	98/267
f51m	126/269	100/214	48/115	58/132	61/138
misex1	62/154	41/91	31/70	31/88	31/91
misex2	51/221	58/166	65/148	41/193	41/191
rd73	129/322	56/127	63/152	78/210	90/240
rd84	291/671	131/275	95/253	117/301	131/337
sao2	119/307	94/221	79/185	84/228	84/231
vg2	140/390	39/139	52/128	45/168	50/176
duke2	446/1206	331/711	245/643	285/733	293/748
misex3c	459/1202	379/792	256/649	316/820	320/842

ProperSYN and the Transduction method needs the initial network in terms of simple gates like AND, OR, NAND, NOR and NOT, hence we first perform *technology mapping* to a simple gate library, apply transformations and then apply technology mapping again. In case of MIS 2.2, we perform synthesis first using the boolean script [9], then we apply technology mapping. The result is presented in terms of 'g/l', which represents gate and literal count. For ProperSYN, the results obtained with 1 and 8 processors are presented to show that the quality of the circuit does not change significantly with large number of processors. The results are shown in Table 2.

5 Conclusions

In this paper, we have presented a portable parallel algorithm of logic synthesis, called ProperSYN. ProperSYN has been implemented on top of the ProperCAD environment [7] and it runs on variety of parallel machines like Sequent Symmetry, Encore Multimax, Intel/860 Hypercube and network of workstations. The algorithm developed for ProperSYN uses an asynchronous message driven data-flow model of computation. No explicit synchronizing barrier is allowed in the algorithm, hence our algorithm is developed to scale well with larger number of processors. we have applied our algorithm to *MCNC* synthesis benchmark examples and obtained excellent results.

6 Acknowledgement

We are grateful to C. F. Lim and Professor Saburo Muroga for various discussions on the transduction method of logic synthesis. We are also thankful to Argonne National Laboratory for giving us the access to their parallel machines.

References

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

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

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

[5] C.-F. Lim, "Parallel Algorithm for Multi-Level Logic Synthesis Using the Transduction Method," Master's thesis, Univ. of Illinois, 1991.

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

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

[8] K. De, B. Ramkumar, and P. Banerjee, "A Portable Parallel Algorithm for Logic Synthesis using Transduction," *submitted to IEEE Transactions of CAD*, 1992.

[9] H. Savoj, H.-Y. Wang, and R. K. Brayton, "Improved Scripts in MIS-II for Logic Minimization of Combinational Circuits," *International Workshop on Logic Synthesis*, 1991.