DISTRIBUTED ADVANCED SEARCH TECHNIQUES FOR CIRCUIT PARTITIONING

Shawki Areibi[1] Anthony Vannelli [2]

[1] Ryerson Polytechnic University, Canada
sareibi@ee.ryerson.ca
[2] University of Waterloo, Canada
vannelli@cheetah.vlsi.uwaterloo.ca

ABSTRACT

Parallel and distributed computing systems offer the promise of a quantum leap in the computing power that can be brought to bear on many important problems. The potential for distributed processing exists whenever there are several computers interconnected in some fashion so that a program or procedure running on one machine can transfer control to a procedure running on another. In such an environment we wish to assign optimally the modules of a program to specific processors. The main objective in optimizing is twofold, minimizing the running time of the program and improving the efficiency of the algorithm. Our main task in this work is to develop an environment that allows easy parallelization of the existing sequential algorithms, in which the potential parallelism fits easily into the sequential algorithm. This paper discusses techniques to parallelize advanced search heuristics [1] used to solve the circuit partitioning problem.

Keywords — **Circuit Partitioning, Tabu Search, Genetic Algorithms, Distributed Processing, VLSI Circuit Layout.**

1. INTRODUCTION

Parallelism may be applied in several ways to increase the processing power available to the execution of a program. These approaches can be broadly categorized into two groups, namely, closely coupled or synchronized processors, and loosely coupled or distributed systems. Closely coupled systems have traditionally been more popular since they can be used to speed existing algorithms and programs. Developers of applications for parallel processing must choose between implementing their ideas in special-purpose or general-purpose hardware. The primary trade-off is performance gain versus flexibility in implementing applications. Special hardware (such as a network of transputers) usually yields higher performance for specific applications whereas general-purpose (i.e, network of workstations) can be programmed for a variety of applications. Both approaches have been pursued in the past by researchers in the VLSI design automation community. A major limitation with almost all previous work is that the parallel algorithms have been targeted to run on specific machines like an Intel iPSC^{TM} hypercube-based message passing distributed memory multicomputer or an Encore Multimax^{TM} shared memory multiprocessor. Such work, although interesting, is not usable by the rest of the VLSI CAD community since the algorithms are not portable to other machines.

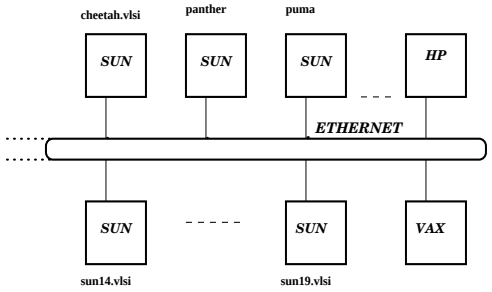

Figure 1. Computing facilities

The most promising algorithms used for circuit partitioning and placement are based on iterative improvement methods. Work by [2] and [3] showed that parallel implementation of these pairwise interchange methods can be accelerated in two ways, by performing several moves in parallel and by performing the subtasks for each move in parallel. The amount of parallelism within a move is limited, and good speedup can be achieved only by performing several moves in parallel. However, if moves are performed in parallel, the system can get into an inconsistent state unless the processors are synchronized after every set of noninteracting moves is calculated in parallel. There is a tradeoff between the communication costs and the need to broadcast information after each accepted move. Typically, several moves are accepted before a broadcast, which leads to some errors. Since communication costs are high even on a tightly coupled system, it is not desirable to speed up interchange methods on a distributed system (by performing several moves in parallel) where communication time is in the order of several milliseconds. A typical CAD environment consists of a number of workstations as seen in Figure 1 connected together by a high speed network that allows a team of designers to access the design database. Most individual CAD problems are typically solved on a single workstation. Our primary strategy is to utilize the network of workstations (portability) in such a way to minimize the interaction between processors, and at the same time utilize the same sequential powerful heuristics developed with minimum modifications (preserve optimality) on the distributed platform. An attempt will first be made to test (synchronization, communication) the distributed environment. This is carried out by im-

[0] 0-7803-4314-X/98/$10.001998 IEEE

plementing a simple distributed scheme where each processor (workstation) starts from a different initial point in the solution space, iteratively improve it, and ultimately report the final solution to the master processor. Next, means to parellize a Genetic Alogorithm [4] and a hybrid based on Tabu Search [5] will be presented.

2. A NETWORK MULTIPROCESSOR ENVIRONMENT

The Network Multiprocessing Environment is a package intended to make distributed, concurrent applications in BSD UNIX socket network easier to write [6, 7]. It sets up a multiprocessor like structure using processes on a UNIX network. The structure of the interprocess communication paths is an arbitrarily interconnected graph. For distributed applications of the client/server model, the Sun Remote Procedure Call (RPC) package provides a good interface to the socket level of BSD UNIX. RPC, however is not well suited to multiprocessor simulations or applications of a more concurrent nature. NMP provides procedure calls that allow easy parallelization of existing sequential algorithms. The NMP environment is implemented using a set of standard NMP routines that allow the creation of virtual multiprocessors with arbitrary interconnection structures. These standard routines are in turn built onto the UNIX networking primitives. Those primitives allow processes to communicate with a variety of protocols and connection strategies.

The execution environment consists of a collection of virtual processors (UNIX processes) whose interconnections (UNIX stream-socket connections over an Ethernet) are created on initialization. In the NMP applications, nodes are referred to by their node id (an integer). The node id is simply the cardinal number of the node description line in the configuration file, starting with the root as node 0. The four basic NMP support routines are: **NodeInit, SendNode, RecvNode** and **NodeClose**. These four routines form the core of the system and programs must use (some form) of these routines. To get started a node (process) initialization function must be invoked, as follows: neighbors = NodeInit(Type, ConfigFile), NodeInit initializes the node by reading a ConfigFile and creates other nodes (processes) as specified in that file.

All messages passed through the NMP are sent using: SendNode(NodeID, Message, Length). The data in memory at the location pointed to by Message are sent to the node NodeId. The number of bytes to send is given by the Length parameter. SendNode returns the number of bytes successfully sent. To receive information sent from SendNode, the following is used: RecvNode(NodeId, Message, Length). RecvNode will only return with a partial message if the sender terminated before completing transmission, otherwise, it will always block until the entire message is read.

3. PARALLEL PARTITIONING

The MCNC benchmarks [8] used throughout this paper are shown in Table 1.

3.1. A Distributed Iterative Heuristic

Iterative improvement techniques that make local changes to an initial partition are still the most successful algorithms used in practice. The advantage of these heuristics is that they are quite robust. But in general, node interchange are greedy or local in nature and get easily trapped in local minima. The performance

Circuit	Nodes	Nets	Pins	Node Degree	Net Size
Chip1	300	294	845	6	14
Chip2	274	239	671	5	7
Prim1	832	901	2906	9	18
Ind1	2271	2192	7743	10	318
Prim2	3014	3029	11219	9	37
Bio	6417	5711	20912	6	860
Ind2	12142	12949	47193	12	584
Ind3	15057	21808	65416	12	325

Table 1. Benchmarks used as test cases

of these heuristics depend on the initial solution used. As the number of initial solutions increase, this allows the heuristic to diversify the search effectively.

Root (Master):
- Read configuration file (Initial Structure)
- Setup connection with all processors.
- Confirm connections with all processors.
- Broadcast arguments to all processors.
- Poll All processors for final result.
- Determine the best solution.
- Report solution quality and timing information

Servers (Slaves):
- Confirm connection to root
- Receive arguments for execution
- Read the circuit and set data structures
- Obtain different initial solution for partitioning
- Perform interchange algorithm
- Send results to Root
- Close Node

Figure 2. A distributed partitioning heuristic

Figure 2 shows the overall algorithm used to implement the simple distributed partitioning scheme. Figure 3 shows the execution mechanism using the sequential algorithm versus the parallel implementation. It is important to notice that one should not expect a linear speedup using the distributed network based on the above mentioned scheme. The main reason is that some runs take more time for execution than others. Therefore the distributed version best time depends on the longest run and the communication of solutions among processors (Figure 3). Initially the Master processor sets up the connection to all processors that are involved in the configuration file. The slaves in turn would confirm the connections. The master would then broadcast the necessary arguments that are needed to execute the program. Some of these arguments are: the program name, the circuit name, the number of blocks and other important

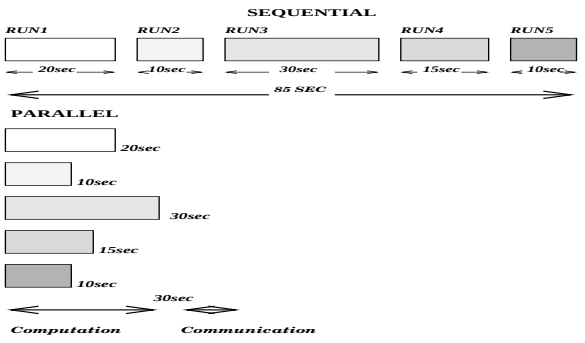

Figure 3. Sequential vs parallel computation

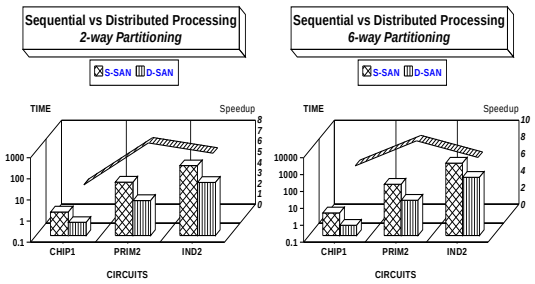

Figure 4. Sanchis distributed heuristic

parameters that are used by the main program. Once these arguments are received the Slaves would then read the input circuit, obtain an initial solution for the circuit partitioning and perform local optimization based on the Sanchis multi-way partitioning scheme. The Master in turn would poll all processors for the final result and report the best solution and timing information.

Tables 2, 3 and Figure 4 present results using the distributed Sanchis heuristic. Results indicate that on average a speedup of 7 can be obtained using the above mentioned heuristic.

Circuit	SEQUENTIAL		PARALLEL (10)		Xs
	Cuts	Time	Cuts	Time	
Chip1	20	1.3	20	0.42	**3**
Chip2	15	0.9	15	0.2	**4**
Prim1	75	5.8	75	1.1	**5**
Prim2	243	33.9	243	4.6	**7**
bio	143	75	143	10.7	**7**
ind1	50	18.7	50	3.1	**6**
ind2	593	206	593	33.1	**6**
ind3	519	199	519	28.4	**7**

Table 2. 2-way Distributed Local Search

Circuit	SEQUENTIAL		PARALLEL (10)		Xs
	Cuts	Time	Cuts	Time	
Chip1	77	2.4	77	0.5	**5**
Chip2	66	2.2	66	0.4	**6**
Prim1	181	9.2	181	1.1	**8**
Prim2	773	110	773	12.3	**9**
Bio	821	350	821	60.1	**6**
Ind1	364	60.5	364	7.4	**8**
Ind2	2486	1967	2486	280	**7**
Ind3	2689	747	2689	103	**7**

Table 3. 6-way Distributed Local Search

3.2. A Distributed Genetic Search Heuristic

There are many simple avenues to parallelize a sequential Genetic Algorithm (assuming a global shared memory) [9] e.g., selecting and crossing over pairs of solutions in parallel, and mutating solutions in parallel. However, such avenue results in only a simple hardware accelerator, and will not be suitable for the local memory, distributed model of computation. Therefore, the attention is focused on distributing the population of solutions among the processors and allowing the migration of these subpopulations for mating and producing excellent offsprings. This method provides a suitable paradigm to map Genetic Algorithms onto a distributed system. The Genetic Algorithm code used by each processor is shown in Figure 5. Tables 4, 5 and Figure 6 show the results of the distributed Genetic Algorithm. The tables clearly shows that the distributed Genetic heuristic achieves good speedup in execution time. The tables also show that there is a difference between the sequential and distributed version in solution quality. In most cases the heuristic achieves speedup with the same or better quality of solutions. Even though the population size is distributed among processors, the infeasible solutions of these subpopulations are repaired using an efficient repair heuristic, thus optimized to have a better affect. Since the subpopulations used are small compared to the population used for the sequential heuristic, we can afford to use an efficient repair heuristic. Another reason for the quality of solutions is due to the diversification of initial solutions generated on different

```
Parallel Genetic Algorithm:
        Read Circuit
        For E iterations do
          For Each Processor i do
            Run GA for G Generations
          End For
          For Each Processor i do
            Send a set of solutions to Master
          End For
        End For

Genetic Algorithm within each Processor:
        For G Iterations do
          While offsprings created ≤ X
            Select two solutions
            Crossover to obtain offsprings
          EndWhile
          add offsprings to subpopulation
          Calculate Fitness
          Select Population of n elements
        End For
```

Figure 5. A distributed Genetic Algorithm

processors. It is worth noting that the parallel implementation for the Genetic Algorithm gives better results than the sequential heuristic used.

3.3. A Distributed GA-Tabu Search Heuristic

[1] introduced a Tabu Search-Genetic Algorithm Hybrid based on the intercommunicating hybrid model. In this model, the independent processing modules in the form of Tabu Search, GRASP, Genetic Algorithms and simple local search exchanged information and performed separate functions to generate near optimal solutions. The main task of the hybrid algorithm was to generate good initial starting points, locally fine-tune the search and finally diversify and intensify the search in the solution space. The distributed Tabu Search Genetic Algorithm (GA-TS) code used is shown in Figure 7. Table 6 presents the results obtained for the distributed GA-TS hybrid.

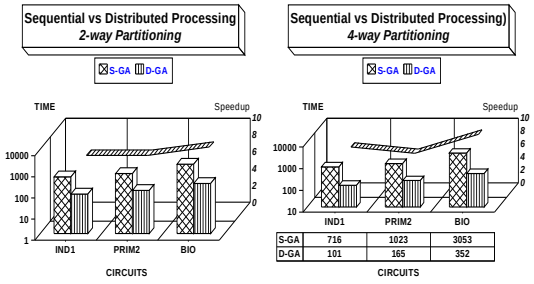

Figure 6. Distributed GA heuristic

Even though the speedups obtained are not linear (as expected), we still benefit from using the distributed NMP approach, since the execution time is faster by at least a factor of three.

Circuit	SEQUENTIAL		PARALLEL (10)		Xs
	Cuts	Time	Cuts	Time	
Chip1	20	68.7	20	7.6	9
Chip2	15	56.2	16	6.3	8.9
Prim1	76	185	75	22.4	8
ind1	67	484	66	74	6.5
Prim2	168	697	171	111	6.2
Bio	282	1982	284	237	8.3
ind2	333	2976	328	559	5.3
ind3	649	4007	645	504	7.9

Table 4. 2-way Distributed Genetic Search

Circuit	SEQUENTIAL		PARALLEL (10)		Xs
	Cuts	Time	Cuts	Time	
Chip1	77	121	76	14.7	8.2
Chip2	66	104	67	17.1	6.1
Prim1	162	356	172	47.9	7.5
ind1	228	991	263	149	6.6
Prim2	564	1350	592	213	6.3
Bio	1549	4065	1158	478	8.5
ind2	2698	6789	2837	1136	5.9
ind3	2536	8417	2742	1033	8.1

Table 5. 6-way Distributed Genetic Search

Master Processor:
Receive Solutions from Servers
Distribute Solutions for Diversification
Distribute Solutions for Intensification
Report Best overall Solution
Servers (Slaves):
G=Generation Size; PS=Population Size
P=Total Processors; P_i = Processor i
For Each Processor P_i (SLAVE):
Read Circuit
Initialize PS/P population of solutions
For G/P Generations
Crossover;Mutate;Select;.
Produce New Solutions.
End For
Initialize Parameter of TS
Improve Initial Solutions using TS
Report Solutions Back to Master
End For
Invoke Diversification or Intensification Phase.
Report solution to Master.

Figure 7. A distributed GA-TS heuristic

4. SUMMARY

In this paper, we introduced a network multiprocessing environment that was suitable to parallelize the circuit partitioning heuristics used in this paper. The main goals of speeding up the execution time and improving the efficiency of the existing heuristics was established by careful design of the parallel heuristics. Results obtained in Section 3. indicate that this environment is effective to parallelize the heuristics and consequently to speedup the execution times. The main importance of this work lies in the portability of these heuristics to many sites for experimentation. Another goal that has been achieved is increasing the processing power available to the execution of most heuristics developed thus far.

Distributed GA-TS Heuristic					
Blks	Prim2 Circuit				
	SEQ		DIS		Xs
	C	T	C	T	
2	161	632	161	211	3
4	376	495	366	126	4
6	501	563	501	177	3
Blks	Ind2 Circuit				
	SEQ		DIS		Xs
	C	T	C	T	
2	273	3422	271	1721	2
4	943	5087	946	1681	3
6	1322	6600	1322	2221	3
Blks	Ind3 Circuit				
	SEQ		DIS		Xs
	C	T	C	T	
2	454	3005	454	1550	2
4	1045	4074	1045	1011	4
6	1900	7850	1900	2556	3

Table 6. GA-TS heuristic

REFERENCES

[1] S. Areibi and A. Vannelli. Advanced Search Techniques for Circuit Partitioning. In DIMACS Series in Discrete Mathematics and Theoretical Computer Science, pages 77–98, Rutgers State University, New Jersey, 1994.

[2] S.H. Bokhari. Partitioning problems in Parallel, Pipelined and Distributed Computing. IEEE Transaction on Computers, 37(1):48,57, 1988.

[3] B. Nandy and W.M. Loucks. Parallel Partitioning for Conservative Parallel Simulation Onto Multicomputers. In 7th Workshop on Parallel and Distributed Simulation, pages 43–51, San Diego, California, May 1993.

[4] D.E. Goldberg. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley Publishing Company, Inc, Reading, Massachusetts, 1989.

[5] F. Glover. Tabu Search Part I. ORSA Journal on Computing, 1(3):190–206, 1990.

[6] T. Breitkreutz. NMP Applications Development. Technical report, University of Alberta, University of Alberta, January 1989.

[7] W.R. Stevens. UNIX Network Programming. Prentice Hall, Englewood Cliffs, New Jersey, 1990.

[8] K. Roberts and B. Preas. Physical Design Workshop 1987. Technical report, MCNC, Marriott's Hilton Head Resort,South Carolina, April 1987.

[9] J.P. Cohoon, S.U. Hegde, W.N. Martin, and D.S Richards. Distributed Genetic Algorithms for The Floorplan Design Problem. IEEE Transaction on Computer Aided Design, 10(4):483–492, 1991.