PHIGURE: A PARALLEL HIERARCHICAL GLOBAL ROUTER

Randall J. Brouwer and Prithviraj Banerjee

Center for Reliable and High Performance Computing
University of Illinois at Urbana-Champaign
1101 W. Springfield Ave.
Urbana, IL 61801

ABSTRACT

A new Parallel HIerarchical algorithm for Global Routing (*PHIGURE*) is presented in this paper. The router is based on the work of Burstein and Pelavin [1], but has many extensions for general global routing and parallel execution. Main features of the algorithm include structured hierarchical decomposition into separate independent tasks which are suitable for parallel execution and adaptive simplex solution for adding feedthroughs and adjusting channel heights for row-based layout. The algorithm is described and results are presented for a shared-memory multiprocessor implementation.

I. INTRODUCTION

During the last few years, there has been an increased interest in the use of parallel processing machines for Computer-Aided Design (CAD) applications[2] due to the advantages of *decreased design times*, ability to solve *larger problems sizes*, the achieving of *high quality results*, and the availability of *low cost multiprocessors*.

In this paper, we present a new parallel algorithm for global routing called *PHIGURE*, a Parallel Hierarchical Global Router.

The task of global routing is to take a netlist, a list of pin positions, and a description of the available routing resources and determine the connections and macro paths for each net. Some criteria used to evaluate the quality of the routing include: total net length, total chip area, the number of tracks required (row-based routing), the number of feedthroughs used, and the number of vias required. For row-based layout, the output of the global router is used to set up the channels to be routed by a channel router.

Previous work done in uniprocessor global routing can be basically divided into these categories: minimum spanning tree solutions [3, 4], maze routing [5, 6], hierarchical routing [7, 8], and ripup and reroute methods [9]. Other work may be identified as a combination of these approaches.

In the past, there have been a number of different approaches to the task of parallelizing the global routing problem. One approach was to develop a maze routing algorithm suitable for a special purpose hardware routing machine, made up of a 2-D array of microprocessors [10]. Similarly, a maze router was implemented on the AAP-1 2-D array processor [11]. Two other algorithms for maze routing have been developed, specifically for the hypercube [5, 6]. A different approach, developed by Rose for shared-memory multiprocessors [12], determines the best of possible two-bend routes for each two-pin subnet of each net.

One major underlying problem inherent in the above techniques is the dependence of the final result on the order in which nets are analyzed and routed. Typically this problem is ameliorated by ripping up and rerouting nets a fixed number of times. Another problem was the deterioration of the quality of results with parallel processing. The top-down hierarchical routing model provides for the routing of all nets at the same time, along with techniques for handling very large routing problems. We have retained the basic ideas of Burstein's hierarchical router, but have made some adjustments to facilitate parallel global routing. Our objective in this work was to generate high quality routes with no degradation in quality with the use of parallel processing. Hence, we applied Burstein and Pelavin's hierarchical model [1].

In Section II of this paper we discuss our global routing model, and the hierarchical routing technique. In Section III we provide an overview of the decomposition strategies applied to subdivide and solve the problem, along with other issues in the algorithm design. Implementation details and parallel processor results for the algorithm are given in Section IV.

Acknowledgment: This research was supported by the National Aeronautics and Space Administration under contract NASA NAG 1-613.

II. THE GLOBAL ROUTING FORMULATION

2.1. Global Routing Model

The global routing model we are using is similar to that of Burstein and Pelavin [1]. The entire layout area (including pads) is divided into a two-dimensional array of routing cells. Each routing cell is assigned routing capacity information for each of its four boundaries, based on the physical dimensions of the routing cell. This provides constraints on the number of nets that can be routed through the edges of the routing cell, as in Figure 1(a).

At each level of the hierarchical decomposition, the current set of routing cells is divided into four regions, forming a two-by-two array of supercells. Thus, each supercell will encompass a subregion of the layout area. These supercells are further divided at later stages of the decomposition. Each net is cast into one of 15 net types, based on the presence of pins in each of the four supercells. The net types consisting of two or more pins are shown in Figure 2, along with the possible routings for each. Such a formulation was proposed by Burstein and Pelavin [1].

A linear (integer) programming formulation of the problem (LP) is defined such that

$$\text{For all } x,\ MAX\ (px)$$
$$\text{subject to } Ax \le a \text{ and } Bx = b,$$

where x represents the variable space, p represents the objective function, A and a represent the inequality constraints, and B and b represent any equality constraints. In our problem, the set of variables represent the 28 possible net routings and the set of 15 constraints is based on the available routing capacities and the types

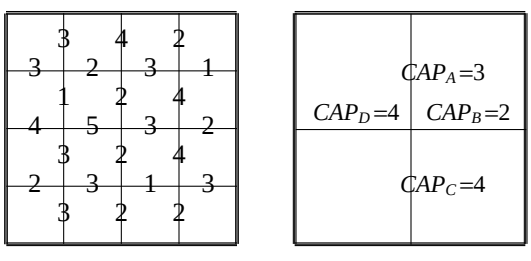

Figure 1. Routing Cell Model.

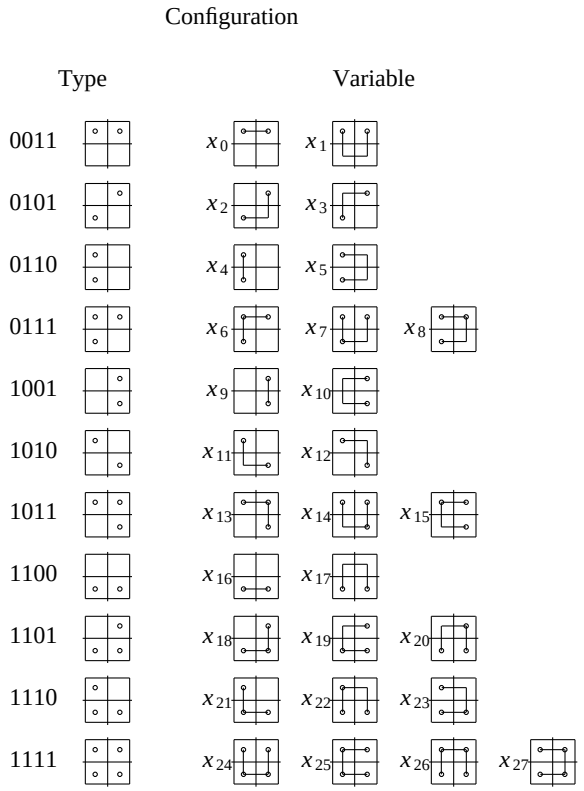

Figure 2. Net Types and Possible Routings.

of nets being routed. The cost function is designed to minimize interconnection lengths of the nets. The resulting values of the variables x_i represent the number of nets routed in the particular pattern which the variable represents. After a solution to the LP is found, the nets are assigned to the appropriate configuration.

2.2. Estimating Routing Capacities

In the routing capacity model, it is sufficient for each routing cell to maintain capacity information for only two of its four shared edges (for example the top and right edges). Denote the vertical capacity for a routing cell in row r and column c as $v_{r,c}$ (across the top edge), and the horizontal capacity as $h_{r,c}$ (across the right edge). Let L, R, T, and B be the left, right, top, and bottom edges (rows and columns) of the region being solved. Let X and Y be the locations of the vertical (Y) and horizontal (X) axes respectively of the two-by-two supercell array. Let CAP_i, $i \in A,B,C,D$ represent the capacities of the four axis segments in clockwise order around the two-by-two supercell array, as shown in Figure 3. Then

$$CAP_A = \sum_{i=Y}^{T} min\,(h_{i,X-1}, h_{i,X}, h_{i,X+1})$$

$$CAP_B = \sum_{i=X}^{R} min\,(v_{Y-1,i}, v_{Y,i}, v_{Y+1,i})$$

$$CAP_C = \sum_{i=B}^{Y} min\,(h_{i,X-1}, h_{i,X}, h_{i,X+1})$$

$$CAP_D = \sum_{i=L}^{X} min\,(v_{Y-1,i}, v_{Y,i}, v_{Y+1,i}).$$

This scheme quickly estimates the capacity of the axes with little chance of overestimating by concentrating on the regions closest to the axis. Cases in which the cell capacities are nonuniform near an axis are handled as well. See Figure 1(b) for an example.

III. PARALLEL ALGORITHM OVERVIEW

In this section we give an overview of our parallel hierarchical global routing algorithm that is based on the linear programming model described in the previous section. Many important aspects of our algorithm will be discussed in this section, including the hierarchical methods of global routing decomposition, automatic feedthrough insertion, automatic channel width adjustment, and details of the parallelisms inherent in the algorithm.

3.1. Hierarchical Decomposition

As mentioned earlier, we are applying two-dimensional hierarchical decomposition methods to the global routing problem. At each stage of the hierarchy we are attempting to divide a larger problem into four smaller subproblems (divide and conquer). Deciding how to separate the subproblems so that they are independent of each other is very important, and there are a few options to consider. These options primarily have to do with how net-crossing locations along the boundaries between the subproblems are determined and locked in place.

3.1.1. Minimal Boundary Determination

Figure 4 shows two stages in the hierarchical decomposition for this strategy. The top-most two-by-two problem is solved, followed by the quick approximations of the crossings of nets. The four subproblems are then solved completely in parallel in the second stage, with no need for periodic global data updating between processors. These steps are repeated recursively until the

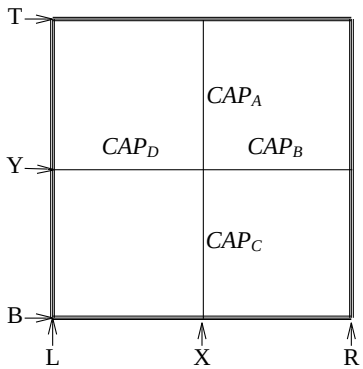

Figure 3. Axes Capacities of 2x2 Supercell.

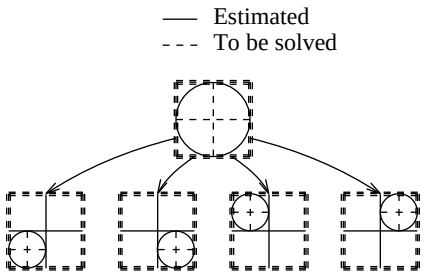

Figure 4. Minimal Boundary Determination.

routing cell level (supercell = routing cell) is reached. The circled regions of the figure show the locations of the current two-by-two routing problems.

The strategy of the minimal determination of the boundary lines is by far the fastest; however, there is a trade-off in the expected quality of the solution for computation speed. The problem arises because it is unknown exactly where along the boundary each net should cross. Some approximations may be used to estimate the crossing based on the pin locations of each net, but unless the boundaries are well predicted, the quality of the routing will be severely degraded from the top-most two-by-two solution.

3.1.2. Maximal Boundary Determination

The second strategy completely determines the net crossing locations by recursively decomposing along the axes of interest down to the routing cell level. This strategy is computationally more costly than the previous method, but it is able to handle the net crossing interface problems directly in a well defined manner. Figure 5 shows two stages in the decomposition for this strategy. The top-most two-by-two solution is followed first by the recursive subdivision and solution of the Y axis (5a), down to the level of individual routing cells, and second by the recursive subdivision and solution of the X axis (5b). After completing these steps, the net crossings have been completely determined and locked into place along both axes of the two-by-two supercell problem, and the four sub-problems are completely independent of each other (5c). This sequence of steps is then recursively repeated until the net crossings across all routing cell edges have been determined.

3.2. Feedthrough Insertion

In row-based layout, feedthroughs must be inserted into the rows to make connections when no built-in feedthroughs or equivalent pins are available when connections must be made from row_i to row_{i+2}. *PHIGURE* handles the problem through the simplex computations. After the problem has been set up, if sufficient routing facilities are available, a solution will be found, else the simplex algorithm will terminate with an infeasible initial problem. By analyzing the simplex state and the given routing problem, adjustments to certain capacities will provide a feasible initial problem for the simplex algorithm. Adjustments to CAP_A and CAP_C are equivalent to increasing the channel width. Adjustments to CAP_B and CAP_D are equivalent to inserting feedthroughs in the row along the X axis. This technique has proven very effective in *PHIGURE*.

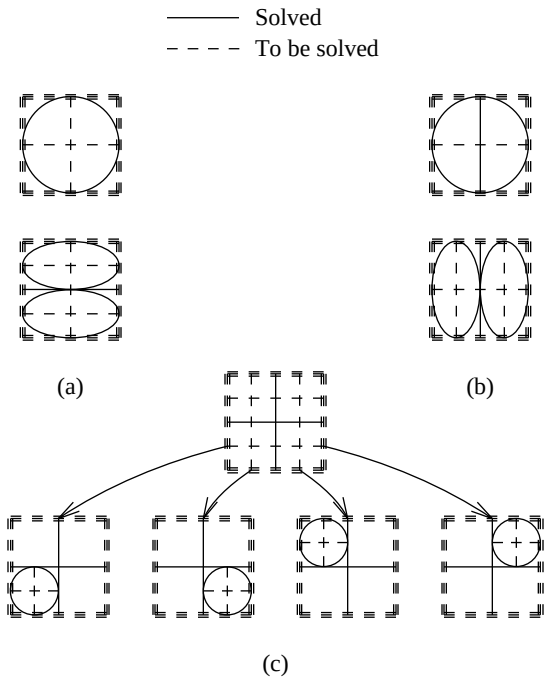

Figure 5. Maximal Decomposition (a)Y Axis Eval. (b)X Axis Eval. (c)Stage 1 to Stage 2.

3.3. Exploitation of Fine-Grained Parallelism

During the LP setup, the type for each net of the current two-by-two problem is determined. Since each net is independent, the nets may be divided between available processors and be evaluated in parallel. The exchange operations required to solve the linear/integer program may also be divided up between available processors for parallel execution. These areas of parallelism are orthogonal, but are of a very fine grain. Because of the size of the simplex problem and the number of available processors, we determined that is was unnecessary to extract these parallelisms at this time. Future research will focus in on evaluating the benefits of including these fine-grained parallelisms.

3.4. Exploitation of Coarse-Grained Parallelism

In *PHIGURE*, we focused on the coarse-grain parallelism of the algorithm. The parallel execution of a binary tree is a well known paradigm, and the hierarchical routing execution in *PHIGURE* takes the form of a binary tree in which the nodes of the tree represent the LP set-up, the LP solution, and the net assignments for a single two-by-two routing problem. Each node of the tree that is currently being evaluated is completely independent of all other active nodes. The local information for the current subproblem is derived from its parent node's data structures and global pin location information which is strictly read-only. The solution of the routing subproblem causes the executing process to write the results to a global (shared) output data structure. Since the tasks are spatially independent, there are no contention or race hazards as a process writes out its results.

After writing the results, the process creates two child routing subproblems. One child subproblem is assigned to the first idle and waiting process. The second child subproblem is then executed by the parent itself. If no processes are waiting, the parent will proceed to execute the first subproblem, followed by the second. The number of processes created and initially available for subproblem solution is set equal to the number of processors available to the user.

IV. IMPLEMENTATION AND RESULTS

PHIGURE was implemented using approximately 5000 lines of C code on an eight processor Encore Multimax 510 (shared memory multiprocessor), in which each processor is a National 50310 CPU. Experiments were performed on a few of the placement and routing benchmarks from the MCNC Workshop on Placement and Routing, along with a number of other circuits. Testing was done on a single processor, two processors, four processors, and eight processors.

Table 1 compares the routing results of *PHIGURE* to actual runs of the TimberWolf 5.4 global router (TW) [3], and some of the recently published results for the UTMC router (UT) [3], a router by Cong and Preas (CP) [4], and Locusroute (LR) [12], This table shows that *PHIGURE* performs well within the range of some recently published routers. Table 2 compares the uniprocessor runtimes for the TimberWolf 5.4 router with those of *PHIGURE*. These timings were executed on a Sun 3/110 workstation.

Table 1. Routing Quality Comparison

	Number of Trks			
Circuit	PHIGURE	TW	UT	CP
Primary1	177	163	177	190
Primary2	404	432	447	449

Table 2. Runtime Comparison

	Runtime (secs, Sun 3-110)	
Circuit	*PHIGURE*	TimberWolf 5.4
Primary1	207	812
Primary2	863	3883

Table 3 shows the results for two of the Placement and Routing Workshop benchmark circuits and two other standard cell circuits. For each circuit, the table gives the number of tracks used as estimated by the maximum channel density and the average execution times in seconds (real time, including process creation) on one, two, four, and eight processors for the Minimal and Maximal decomposition strategies. Cell placements for all of the circuits were performed by TimberWolf 5.4. As is clear from the table, there is no degradation in routing quality when going from a single processor to many processors, and very good speedups were achieved (>6 for 8 processors). Since the hierarchical decomposition creates a large number of jobs, our algorithm is scalable for a large numbers of processors.

V. CONCLUSIONS

In this paper we have presented a new parallel global router, *PHIGURE*, which applies hierarchical routing and decomposition techniques to create independent subproblems which can be evaluated in parallel. Results were presented which compare two strategies for decomposing the routing problem and show that high quality routings are attainable for one strategy. Most importantly, the routing quality is not degraded by decomposing in parallel. We are currently investigating alternative methods of decomposition, improved net assignment techniques, incorporating fine grain parallelism, implementation on a message-passing multiprocessor, and strategies for adaptive rerouting of problem nets.

ACKNOWLEDGEMENT

The authors would like to express their appreciation to Ralph Kling for his assistance in generating the benchmark placements and Krishna P. Belkhale for his assistance in the coding.

Table 3. *PHIGURE* Decomposition Comparison

Circuit		Min Decomp		Max Decomp	
(Nets)	P	Trks	Time(sec)	Trks	Time(sec)
Primary1	1	348	33	177	66
(1185)	2	348	17	177	34
	4	348	9	177	21
	8	348	6	177	14
Primary2	1	817	187	404	257
(3710)	2	817	97	404	131
	4	817	52	404	74
	8	817	30	404	42
Circuit X1	1	641	189	416	202
(1979)	2	641	92	416	101
	4	641	47	416	59
	8	641	29	416	36
Circuit X2	1	709	254	515	284
(3013)	2	709	139	515	145
	4	709	74	515	79
	8	709	44	515	44

REFERENCES

[1] M. Burstein and R. Pelavin, ‘‘Hierarchical wire routing,’’ *IEEE Trans. Computer-Aided Design*, vol. CAD-2, no. 4, pp. 223-234, Oct. 1983.

[2] P. Banerjee, ‘‘The use of parallel processing in VLSI computer-aided design applications,’’ ICCAD-88 Tutorial, also Tech. Rep. no. CSG-104, Coordinated Science Laboratory, Univ. of Illinois, Urbana, IL, May 1988.

[3] K. W. Lee and C. Sechen, ‘‘A new global router for row-based layout,’’ *Proc. Int. Conf. Computer-Aided Design*, pp. 180-183, Nov. 1988.

[4] J. Cong and B. Preas, ‘‘A new algorithm for standard cell global routing,’’ *Proc. Int. Conf. Computer-Aided Design*, pp. 176-179, Nov. 1988.

[5] O. A. Olukotun and T. N. Mudge, ‘‘A preliminary investigation into parallel routing on a hypercube computer,’’ *Proc. 24th Design Automat. Conf.*, pp. 814-820, June 1987.

[6] Y. Won and S. Sahni, ‘‘Maze routing on a hypercube multiprocessor computer,’’ *Proc. Int. Conf. Parallel Process.*, pp. 630-637, Aug. 1987.

[7] M. Marek-Sadowska, ‘‘Global router for gate array,’’ *Proc. Int. Conf. Computer Design*, pp. 332-337, Oct. 1984.

[8] W. K. Luk, D. T. Tang, and C. K. Wong, ‘‘Hierarchical global wiring for custom chip design,’’ *Proc. 23rd Design Automat. Conf.*, pp. 481-489, June 1986.

[9] P. S. Tzeng and C. H. Sequin, ‘‘Codar: A Congestion-Directed General Area Router,’’ *Proc. Int. Conf. Computer-Aided Design*, pp. 30-33, Nov. 1988.

[10] R. Nair, S. J. Hong, S. Liles, and R. Villani, ‘‘Global wiring on a wire routing machine,’’ *Proc. 19th Design Automat. Conf.*, pp. 224-231, June 1982.

[11] T. Watanabe, H. Kitazawa, and Y. Sugiyama, ‘‘A parallel adaptable routing algorithm and its implementation on a two-dimensional array processor,’’ *IEEE Trans. Computer-Aided Design*, vol. CAD-6, No 2, pp. 241-250, Mar. 1987.

[12] Jonathan Rose, ‘‘LocusRoute: A parallel global router for standard cells,’’ *Proc. 25th Design Automat. Conf.*, pp. 189-195, June 1988.