Parallel Hierarchical Global Routing for General Cell Layout

Sanjay Khanna , Shaodi Gao and K. Thulasiraman
Department of Electrical and Computer Engineering
Concordia University
Montreal, Quebec H3G 1M8, Canada

Abstract

In this paper we present a parallel global routing algorithm for general cell layout. The algorithm applies a hierarchical decomposition strategy that recursively divides routing problems into simple, independent subproblems for parallel processing. The solution of each subproblem is based on integer programming and network flow optimization. The algorithm is implemented on a shared-memory machine and experiment results on different examples show relative speedup between 4 and 5 for 8 processors. The speedup is achieved without compromising the quality of the routing results.

1 Introduction

The recent advances in the VLSI technology allow the fabrication of highly complex circuits on single chips. Time-consuming software tools are needed to successfully design such chips. Parallel processing has become an essential approach to handle the rapid growing computational complexity. During the past few years, many parallel algorithms have been developed for various VLSI physical design problems. In this paper we present a parallel algorithm for global routing in general cell layouts.

Most physical design systems divide the whole process into three phases: Floorplanning (or placement), global routing and detailed routing. Floorplanning generates a relative position for all cells based on the connectivity and the dimensions of the cells. Global routing then determines the rough topology of the interconnections (nets) among the cells so as to achieve a uniform and low wiring density. Finally, detailed routing transforms the cells and rough wires into a geometric layout. The methods used for global routing depend greatly on the layout structure. In standard cell or gate array layouts, cells are placed in rows or in a matrix pattern; while in general cell layouts, cells with arbitrary sizes are placed freely in the layout area. Given the many degrees of freedom, the routing problem for general cell layout is more difficult to solve.

The first generation of parallel routing algorithms were designed for special purpose machines, such as 2-dimensional array processors [9, 12]. The basic scheme is that each processor routes one net at a time. The same scheme was also applied to algorithms designed for hypercubes [10, 13] and shared-memory machines [11]. This "one at a time" scheme has *data dependency* problem: the cost of a route depends on the routes already chosen by other nets. In order to relax this dependency, one has to rely on some inaccurate data, which can cause deterioration of the routing quality.

Hierarchical approaches evaluate all nets simultaneously and take into account the global information to decompose the routing problem into smaller subproblems. Hierarchical decomposition can be done in a way such that subproblems have independent routing areas and netlists. Therefore, this approach offers a natural way for parallelization. Hierarchical global routing was first proposed by Burstein and Pelavin for gate array layout [1]. Luk et al. [7] extended the method to general cell layout. Lauther [4] and Marek-Sadowska [8] based their algorithms on network flow optimization. Brouwer and Banerjee developed a parallel hierarchical global router for standard cell layout [2]. Our parallel algorithm adopts the method of [7] to reduce each routing problem into up to four subproblems. We also modified the methods of [4, 8] to decompose nets among the subproblems.

Section 2 provides an overview of our hierarchical routing approach. Section 3 discusses the design of the parallel algorithm in a shared-memory environment. Experiment results are given in Section 4. Conclusions are presented in Section 5.

2 Overview of the Hierarchical Approach

The global routing problem is given by a floorplan, which describes the physical locations of cells in the chip, and a netlist, which defines the interconnections among the cells. The floorplan represents a partition of the chip into a set of rectangles with each rectangle containing a cell. It also defines a routing graph: each cell is represented by a vertex, two vertices are connected by an edge if the corresponding cells share a common boundary. Each edge has a capacity which is the maximum number of wires which are allowed to cross the boundary.

Routing a net is to find a Steiner tree connecting all pins of the net in the routing graph. The goal of the global routing problem is to find a set of tree connections for all nets satisfying the constraints, and if possible, minimize the total wire length. The main constraint is that the number of wires allowed to use any edge of the routing graph is limited by its capacity.

For hierarchical routing, we consider a special class of floorplans known as slicing structures. A floorplan is a slicing structure if it is a single rectangle or if it can be cut by a line into two slicing structures. A slicing structure can be considered as the result of a recursive partitioning in a simple pattern, 4-partition, and it degenerate forms. A 4-partition is obtained by first applying a horizontal or vertical cut and then applying a perpendicular second cut on each side of the first cut. There

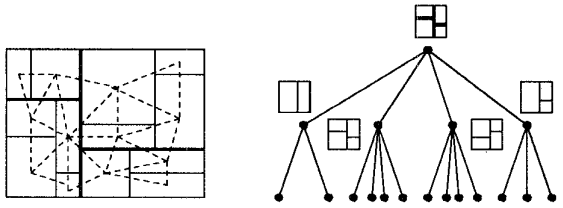

Figure 1: A floorplan, its routing graph and slicing tree.

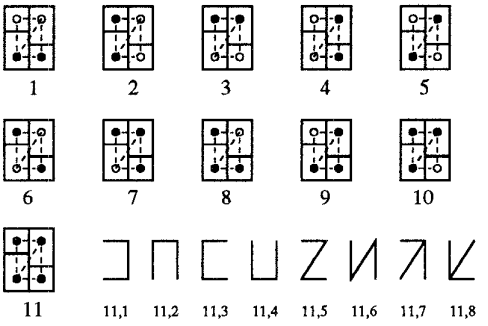

Figure 2: All net types and the route patterns for net type 11.

are two degenerate cases: In 3-partition only one of the two second cuts is applied, while in 2-partition none of the second cuts is applied. This recursive partitioning can be represented by a slicing tree with each node representing a partition (cf. Figure 1). A 4-partition (and its degenerate forms) can be considered as a floorplan with up to 4 rectangles. Cells in each of these rectangles form a super-cell. The corresponding routing graph is called a super-graph. It has up to five super-edges, whose capacities are the sum of the capacities of the corresponding member edges.

Our hierarchical algorithm uses the slicing tree to guide the routing procedure in a top-down manner. At each level of the slicing tree, a node corresponds to a routing task, which consists of (1) routing in the super-graph and (2) decomposition of nets among the super-cells. Routing in a super-graph is also a global routing problem, which, due to the simple graph structure, can be solved efficiently by integer programming. The result of the super-graph routing determines which super-edge each net will take. In order to decompose the nets into independent subnets, nets also have to be assigned to individual edges of the super-edges. We apply a modified min-cost flow algorithm to solve the net assignment problem. After the net decomposition, the global routing problem is divided into up to four totally independent subproblems, each of which is represented by a node at the next level of the slicing tree. The top-down procedure proceeds until the bottom level of the slicing tree is reached and all super-cells become actual cells of the chip.

2.1 Routing in a Super-Graph

Super-graphs only have three different structures: A routing graph of a 2-partition (3-partition) is K_2 (K_3), a clique of two (three) node; while a routing graph of a 4-partition is H_4, which is a clique of four nodes with an edge removed. Here we only present the routing algorithm for H_4 and omit details about the routing for K_2 and K_3, because they can be solved in a similar way.

There are altogether 11 different net types in a H_4-graph according to the number of pins and their position. Figure 2 depicts all these net types. We can also easily enumerate all possible route patterns for each net type. Due to space limitation, Figure 2 only shows the route patterns for net type 11.

Now we describe how to formulate a global routing problem in H_4 as an integer program. Let $T_{i,j}$ be the j-th route pattern for net type i. For each $T_{i,j}$ there is a variable $x_{i,j}$ in the integer program, which indicates the sum of nets of type i that are routed via $T_{i,j}$. For net type i, let n_i be the total number of the nets and x_i the number of the unroutable nets. H_4 has five edges e_k, $1 \le k \le 5$, $c(e_k)$ represents the capacity and $l(e_k)$ the length of edge e_k. Then the integer program contains the following constraints.

$$x_i + \sum_j x_{i,j} = n_i, \qquad 1 \le i \le 11 \qquad (1)$$

$$\sum_{e_k \in T_{i,j}} x_{i,j} \le c(e_k), \qquad 1 \le k \le 5 \qquad (2)$$

Equation set (1) is for completeness constraints, while set (2) is for capacity constraints. The cost function to be minimized is

$$L \sum_{i=1}^{11} x_i + \sum_{1 \le k \le 5} l(e_k) \sum_{e_k \in T_{i,j}} x_{i,j}$$

Here L is chosen large enough such that minimizing the number of unroutable nets has higher priority. We choose L to be the sum of all edge lengths times the number of nets.

The solution of the integer program generates a global routing with the minimum total wire length. Denote each optimal value of $x_{i,j}$ by $x^*_{i,j}$. For each net type i, we have a set of values $x^*_{i,1}, x^*_{i,2}, \cdots, x^*_{i,m}$, where m is the total number of route patterns for net type i. We then assign the nets of type i to the different route patterns.

2.2 Net Decomposition

For every net the route pattern obtained during the super-graph routing determines which super-edges the net will take. We have yet to assign the nets to individual edges contained in the super-edges. Luk et al. [7] proposed a top-down refinement approach, which assigns nets to super-edges of the next level and hence requires to consider the entire routing problem at each level of the hierarchy. Here we follows a different strategy: assign nets directly to the routing graph edges of the original floorplan. Figure 3.1 shows two super-cells separated by a vertical line. The net has two pins (solid circles) on either super-cell. If the net is assigned to

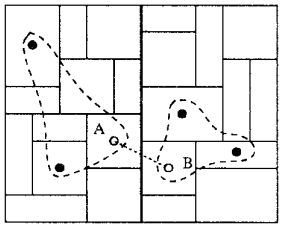

Figure 3: Assignment of a net to a cell boundary.

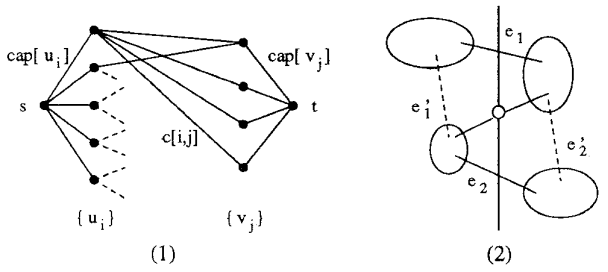

Figure 4: (1) A network formulation for the net assignment problem. (2) Multiple crossings of a net.

the edge that crosses the boundaries between cells A and B, we can create a pseudo-pin on both cells (hollow circles) and divide the net into two subnets, which can be routed independently. Therefore, assigning net directly to individual cell boundaries can divide routing problems into independent subproblems.

The goal of the assignment problem is to minimize the total wire length of the nets without violating the capacity constraints. We follow the ideas by Lauther [4] and Marek-Sadowska [8] to formulate the problem as a min-cost flow problem as follows. We build a network which contains two sets of nodes along with the source s and sink t, one set $V_n = \{u_i\}$ representing nets, the other set $V_b = \{v_j\}$ representing cell boundaries (cf. Figure 4.1). Remember that the cell boundaries are dual to the edges of the original routing graph. Edges (s, u_i) and (v_j, t) have non-zero capacities and zero costs; while edge (u_i, v_j) have non-zero costs $c[i, j]$ and unlimited capacities.

First we assume that each net crosses the super-cell boundary only once. Then the capacity of (s, u_i) is set to 1 and the capacity of (v_j, t) is equal to the capacity of the cell boundary; while $c[i, j]$ is the minimum length to connect the two parts of net u_i via cell boundary v_j. The min-cost maximal flow on this network will deliver an assignment of nets to cell boundaries, which is optimal under the above assumption. The issue of allowing multiple crossings was discussed in [4, 8] without mentioning the problem that increasing crossings for some nets may affect the routability of other nets. Our solution to the problem is as follows. First we solve the assignment problem under the constraint of single crossings. Then we construct for each net a minimal spanning tree connecting its pins and its crossing point. For each edge e of the spanning tree which cross the boundary, replace it by another edge e' which does not crosses the boundary (cf. Figure 4.2). The difference of their lengths is the gain of e. The crossing edges of all nets are sorted based on their gains, and each of them is given a priority number accordingly. Then we update sthe network functions: $cap[u_i]$ is the number of crossing edges of net u_i; $cap[v_j]$ is the free capacity of boundary v_j after the first assignment; $c[i, j]$ is the priority number of the spanning tree edge of net u_i which crosses boundary v_j. The second round of min-cost flow on the network then gives an assignment of nets with multiple crossings without changing the routability.

3 The Parallel Routing Algorithm

The important aspects of our parallel algorithm has already been presented in the last section, such as slicing tree guided task generation and the hierarchical net decomposition. In this section we describe more about the organization of our algorithm and further exploit parallelism in individual routing tasks.

3.1 The Overall Structure

The parallel algorithm takes a top-down approach. For each node in the slicing tree a routing task will be generated. Each task includes performing global routing in a super-graph by means of integer programming and solving the corresponding net assignment problem by finding min-cost flows. The parallel execution of the node tasks follows a breadth-first scheme: Tasks at the same level of the slicing tree will be carried out in parallel by the available processors. Because of our decomposition strategy, these tasks (at the same level) are completely independent from each other. All information exchanges follow through global (shared) data structures. Each processor gets the global data before starting a task and update the global data after completing the task. There is no need for date exchanges among processors during task execution periods. After all tasks at a slicing tree level are completed, a new set of independent tasks corresponding to the next level nodes are generated and ready to be assigned to the available processors for parallel execution.

3.2 Parallelization within a Routing Task

Exploitation parallelism in individual routing tasks is important for further speed-up of the overall routing process. It is particularly effective at higher levels of the slicing tree, where there are not enough tasks for parallel processing. On the other hand, we have to take into consideration the overheads caused by the parallel task generation and management. If the task size is too small, the overheads may outweigh the benefits gained by parallelization. After experimenting with several options, we decided to choose the net assignment for parallel processing, because solving the min-cost flow problem is fairly time-consuming. An H_4 super-graph has five assignment tasks, each for one super-edge. These tasks can be executed in parallel. If the number of processors is not large, we apply this parallelism only at the first few levels of the slicing tree, because at lower levels

there are enough nodes to be assigned to the available processors.

The basic structure of our parallel algorithm is as follows.

Algorithm Parallel Global Routing
Input: A slicing structure with I levels and a netlist
Output: A set of Steiner trees for the netlist.
$i = 0$; /* level 0 of the slicing tree */
while $i < I$
  dopar /* parallel processes for level i nodes */
    find the super-graph routing for the node
    dopar /* parallel processes
          for the super-edges of the node */
      find net assignment for the super-edge
    endpar
  endpar
  $i := i + 1$;
endwhile

4 Experimental Results

We have implemented the parallel routing algorithm on a BBN GP1000 machine under Mach 1000 operating system. The program contains about 7000 lines of C code. Experiments were performed on two MCNC general cell benchmarks, ami33 and ami49, and two randomly generated larger examples. Testing was done on one, four and eight processors. Table 1 summarizes the relative speedups for different problem instances. The speedup by eight processors is around five for problems with large number of cells. The speedup for smaller problems is not so good, because there are few levels in the slicing tree and at most of these levels the power of parallel processing cannot be full utilized.

Chips	p	Time(sec)	Speedup
ami33	1	10.64	1.0
	4	4.21	2.5
	8	3.02	3.5
ami49	1	26.55	1.0
	4	9.72	2.7
	8	6.68	4.0
C1	1	34.23	1.0
	4	10.37	3.3
	8	7.71	4.4
C2	1	87.34	1.0
	4	26.96	3.2
	8	17.26	5.1

Table 1: Parallel speedup

The routing results obtained by a single processor and by multiprocessors are the same. This is because the hierarchical decomposition creates routing tasks which are totally independent from each other. Table 2 shows the example chips and the wirelengths of the global routing results . Our algorithm can always find a route for every net if there exits such a solution, because it always takes the capacity constraints into consideration during minimization of the wirelength.

Chips	Cells	nets	Wirelength
ami33	33	123	183
ami49	49	408	1625
C1	63	500	2533
C2	92	1000	14524

Table 2: Routing Results

5 Conclusions

We have presented a parallel algorithm for global routing in general cell layouts. The algorithm takes a hierarchical approach to decomposed the global routing problem into independent subproblems for parallel processing. It achieves very good speedup without compromising the routing quality. We are currently developing parallel algorithms for integrated floorplanning and global routing. These two design phases have close interaction with each other. Research efforts to perform them simultaneously by hierarchical decomposition have shown very promising results [3, 6]. To achieve the full integration, global routing must be invoked whenever changes to floorplan are made. This approach requires much more intensive computation and therefore parallelization of the process becomes necessary.

References

[1] M. Burstein and R. Pelavin, "Hierarchical wire routing," *IEEE Transactions on Computer-Aided Design*, CAD-2(4), pp. 223-234, 1983.

[2] R. J. Brouwer and P. Banerjee, "PHIGURE: A parallel hierarchical global router," *Proc. 27th Design Automation Conference*, pp. 650-653, 1990.

[3] W. W.-M. Dai and E. S. Kuh, "Simultaneous floorplanning and global routing for hierarchical building block layout," *IEEE Transactions on Computer-Aided Design*, CAD-6(5), pp. 828-837, 1987.

[4] U. Lauther, "Top-down hierarchical global routing for channelless gate arrays based on linear assignment," *Proc. VLSI 87*, pp. 141-151, Elsevier Science Publishers, 1987.

[5] T. Lengauer, *Combinatorial Algorithms for Integrated Circuits Layout*, John Wiley & Sons, 1990.

[6] T. Lengauer and R. Müller, "A robust framework for hierarchical floorplanning with integrated global route," *Proc. Int. Conf. Computer-Aided Design*, pp. 148-151, 1990.

[7] W. K. Luk, P. Sipila, M. Tamminen, D. Tang, L. S. Woo, and C. K. Wong, "A hierarchical global wire routing for custom chip design," *IEEE Transactions on Computer-Aided Design*, CAD-6(4), pp. 518-533, 1987.

[8] M. Marek-Sadowska, "Route planner for custom chip design," *Proc. Int. Conf. Computer-Aided Design*, pp. 246-249, 1986.

[9] R. Nair, S. J. Hong S. Liles, and R. Villani, "Global wiring on a wire routing machine," *Proc. 19th Design Automation Conference*, pp. 224-231, 1982.

[10] O. A. Olukotun and T. N. Mudge, "A preliminary investigation into parallel routing on a hypercube computer," *Proc. 24th Design Automation Conference*, pp. 814-820, 1987.

[11] J. Rose, "Parallel global routing for standard cells," *IEEE Transactions on Computer-Aided Design*, CAD-9(10), pp. 1085-1095, 1990.

[12] T. Watanabe, H. Kitazawa, and Y. Sugiyama, "A parallel adaptable routing algorithm and its implementation on a two-dimensional array processor," *IEEE Transactions on Computer-Aided Design*, CAD-6(2), pp. 241-250, 1987.

[13] Y. Won and S. Sahni, "Maze routing on a hypercube multiprocessor computer," *Proc. Int. Conf. on Parallel Processing*, pp. 630-637, 1987.