SRC TR 88-25

Parallel Algorithms for Wiring Module Pins to Frame Pads

by

Shing-Chong Chang
and
Joseph Ja'Ja'

Parallel Algorithms for Wiring Module Pins to Frame Pads[1]

Shing-Chong Chang
Department of Electrical Engineering
Systems Research Center
University of Maryland
College Park, MD. 20742

Joseph JáJá
Department of Electrical Engineering
Institute For Advanced Computer Studies
Systems Reseach Center
University of Maryland
College Park, MD 20742

Abstract

We present fast and efficient parallel algorithms for several problems related to wiring a set of pins on a module to a set of pads lying on the boundary of a chip. The one-layer model is used to perform the wiring. Our basic model of parallel processing is the CREW-PRAM model which is characterized by the presence of an unlimited number processors sharing the main memory. Concurrent reads are allowed while concurrent writes are not. All our algorithms use $O(n)$ processors, where n is the input length. Our algorithms have fast implementations on other parallel models such as the mesh or the hypercube.

[1]Supported in part by NSA Contract No. MDA-904-85H-0015, NSF Grant No. DCR-86-00378 and by the Systems Research Center Contract No. OIR-85-00108.

1 Introduction

We consider the problem of connecting a set of pins on a module to a set of pads lying on the boundary of a chip. This problem has been addressed in the sequential context by Baker and Pinter ([BP]). The boundary of the module is assumed to be an arbitrary rectilinear polygon and the chip is assumed to be a rectangle containing the module. The module pins and the chip pads are fixed. Our goal is to find out whether a *one-layer* routing exists within the given area and if it does to determine the wiring of such a routing. We also address the problem of changing a given wiring so that the resulting wiring is of minimum length.

The above problems are considered in the CREW-PRAM model, which is characterized by the presence of an unlimited number of processors which can access a shared memory unit. Concurrent read is allowed while concurrent write is not. We are aiming for fast algorithms that are also efficient, i.e., the number of processors is $O(n)$ if the input is of length n. In the rest of the paper, we assume that the reader is familiar with some of the basic parallel techniques such as path doubling, parallel prefix, and the Euler tour technique. As we will mention in the last section, our algorithms can be mapped efficiently into fixed-interconnection parallel architectures such as the array architecture.

The known algorithms to solve the above problems seem to be inherently sequential. We develop new algorithms that possess fast and efficient parallel implementations in addition to their suitability for serial implementations.

The rest of the paper is organized as follows. Several basic definitions and basic strategies for river routing problems will be introduced in the next section. Section 3 will contain a new algorithm for routability testing, while the following section will show how to obtain a minimal length wiring from an arbitrary wiring. The last section mentions the suitability of these algorithms on different parallel architectures.

2 Preliminaries

The general class of *river routing* problems can be viewed as the planar interconnection of two ordered sequences of terminals. The simplest version assumes that the two ordered sequences belong to two parallel rows forming a channel. An understanding of the solution to this case is essential to the understanding of the methods used to solve several less restricted versions. Therefore a review of the basic river routing strategy is in order.

Let $\{N_i =< b_i, t_i >\}$ be a set of nets, where b_i belongs to the bottom row and t_i belongs to the top row of a channel. The rows of the channel are assumed to be horizontal. b_i and t_i will also denote the horizontal integer displacements of these terminals relative to an arbitrary origin. N_i is a *right* net if $b_i < t_i$. If $b_i > t_i$, then N_i is called a *left* net. Otherwise, it is a *vertical* net. The given set of nets can be decomposed into maximal left, right or vertical *blocks*. Each of these blocks can be routed independently. A *greedy* algorithm can be used to route each such block. For example, a right block can be wired by wiring its leftmost net as close to the upper row as possible and then wiring the next leftmost net as close to the upper row while avoiding the wires of the previous net, and so on. The *characteristic* bend points introduced in [CJ] are those bendpoints closest to one row, say the bottom row. For example, A_{11}, B_{11}, A_{81} and B_{81} are charactersitic bend points induced by nets N_1 and N_8 in Figure 1.

We now turn to the main problem of this paper. An instance is given by a triplet $< \mathcal{M}, \mathcal{F}, \mathcal{N} >$, where $\mathcal{M}$ is an arbitrary rectilinear polygon representing a module, $\mathcal{F}$ is a rectangle representing a frame (assuming a horizontal bottom edge), and $\mathcal{N}$ is a set of two-terminal nets such that one terminal is on $\mathcal{M}$ and the other is on $\mathcal{F}$. We assume that $\mathcal{F}$ contains $\mathcal{M}$ and that each boundary segment of $\mathcal{M}$ is parallel to a frame edge. We are supposed to determine a one-layer routing of $\mathcal{N}$ whenever it exists. We borrow some definitions from [BP]. A *spoke* is a line segment perpendicular to a frame edge and which extends from the frame edge to the closest module edge. It is shown in [BP] that if a wiring exists, then there is a routing such that no wire crosses any spoke more than twice and that

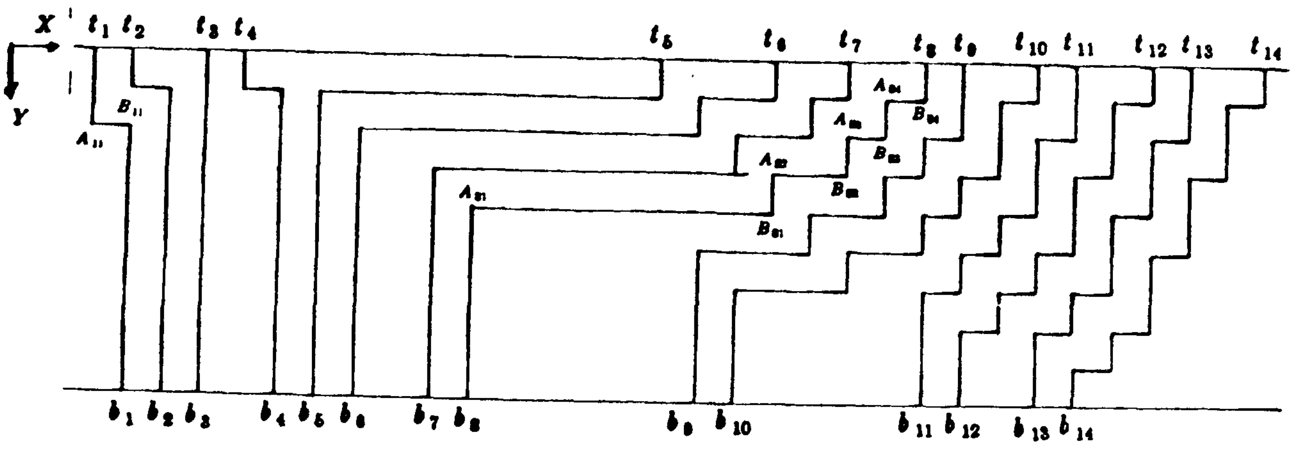

X
Y
t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{10} t_{11} t_{12} t_{13} t_{14}
B_{11}
A_{11}
A_{41}
A_{51}
B_{51}
A_{52}
A_{53}
A_{54}
B_{52}
B_{53}
B_{54}
b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8 b_9 b_{10} b_{11} b_{12} b_{13} b_{14}

Figure 1: A River Routing Example

all crossings of a spoke are in the same direction. If the crossings of one spoke for one net are specified, then the crossings of any other net are uniquely defined. Therefore in the rest of this paper we will assume that such information is provided as part of the input since there are five such possibilities and the algorithm could be run on each one of these separately.

The routing strategy used in [BP] consists of combining the outputs generated by two greedy algorithms. One algorithm (*greedy-in*) begins at the module terminals and routes as close to the module boundaries as possible. The other (*greedy-out*) begins at the frame terminals and stays as close to the frame edges as possible.

Let n be the length of the input (number of nets and boundary segments of the module). The algorithm presented in [BP] runs in time $O(n^2)$ and finds a detailed routing of all the nets whenever such a routing exists. This algorithm is optimal since the number of bend points in the detailed routing could be $\Omega(n^2)$. We address three problems in the parallel context:

- Given an instance of our problem, can we route all nets within the space provided?
- Given an instance of our problem, determine a detailed routing of all nets whenever such a routing exists.

- Given a preliminary routing for an instance of our problem, derive a routing with the minimum total wire length.

We will present fast parallel algorithms for all the three problems. Our algorithms are efficient in the sense that the product *time* $\times$ *number of processors* is within some power of $\log n$ factor of the best possible sequential algorithm. As a byproduct, we obtain a new $\Theta(n)$ serial algorithm for the first problem. We present this algorithm in the next section.

3 Routability Testing

Consider the set of nets whose frame terminals are on the *bottom* frame edge. We will first examine the possibility of routing these nets within the space provided. Recall that we are assuming that the crossings of each net at a given spoke are given as a part of the input. Since we cannot route all the nets fast with a linear number of processors, we have to extract enough information about the wiring of certain critical nets to perform the routability testing. We accomplish this by decomposing the nets into groups such that the routability testing can be done by examining the "contours" of these groups. Before presenting a more formal description of our overall strategy we introduce the following definitions.

Let the *module-bottom line* be the gridline parallel to the bottom frame edge and which passes through points on the module boundary closest to the bottom frame edge. This line will be partitioned into segments, say $A_1B_1, A_2B_2, ..., A_kB_k$, by the module boundary. Figure 2 shows an example whose module-bottom line goes through A_1B_1, A_2B_2 and A_3B_3. The *bottom boundary* consists of the set module segments from A_1 to B_k in a couterclockwise direction. We define the *type* of a net $< a, b >$, $a \in \mathcal{M}$, $b \in \mathcal{F}$, as follows.

- *Type 1* if the direction of the wiring from a to b is clockwise and a is not on the bottom boundary of the module. Nets N_1, N_2, N_3, N_4, N_5 and N_6 in Figure 2 are type 1 nets[2].

[2]From now on, we will use the notation $N_i =< a_i, b_i >$.

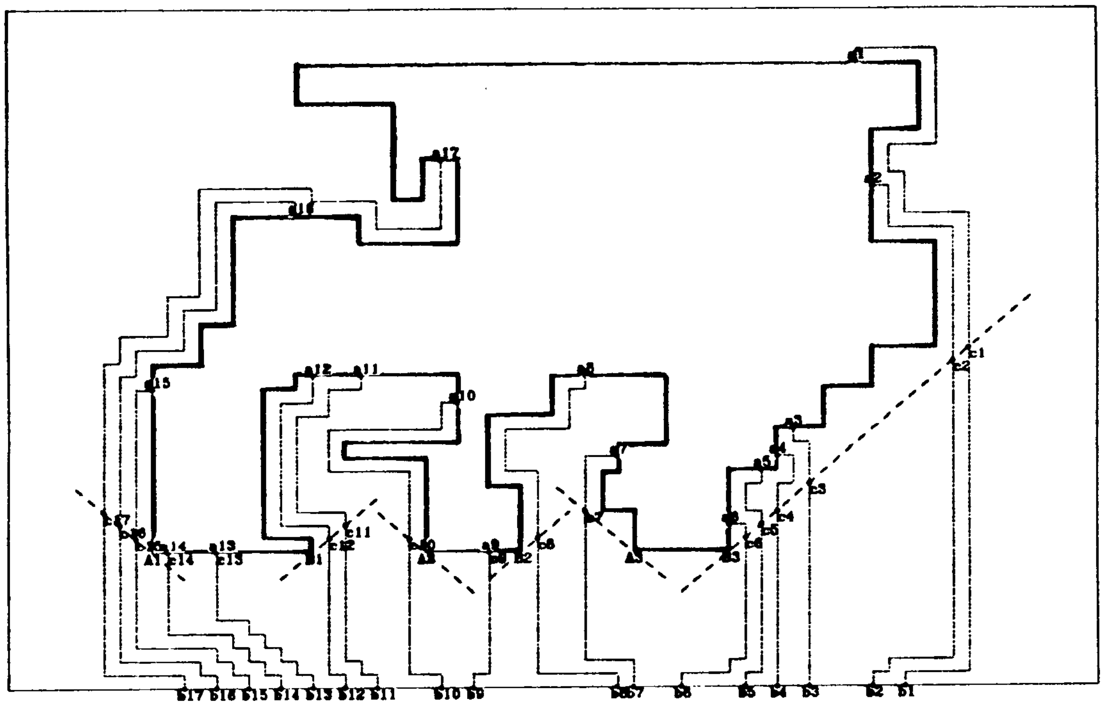


Figure 2: Illustration of Different Types of Nets

- *Type 2* if a is on the bottom boundary between B_j and A_{j+1}, for some j, such that the distance (number of module edges) between a and A_{j+1} is greater than or equal the distance between a and B_j. Nets N_8, N_{11} and N_{12} in Figure 2 are of type 2.

- *Type 3* if the direction of the wiring is counterclockwise and a is not on the bottom boundary of the module. Nets N_{15}, N_{16} and N_{17} in Figure 2 are type 3 nets.

- *Type 4* same as type 2 but distance is less than. In Figure 2, nets N_7 and N_{10} are of type 4.

- *Type 5* if none of the above, i.e., a is on some segment A_iB_i of the module-bottom line. Nets N_9, N_{13} and N_{14} are type 5 nets.

A *consecutive* set S of nets (with one terminal on the bottom frame edge) consists of nets of the same type such that there is no net of a different type whose module terminal lies between two terminals of nets in S. A ***group*** is a consecutive set of nets S such that either (1) the convex corner C closest to the bottom frame edge has the property that all convex corners between C and any terminal in S (counterclockwise direction for types 1 and 2, otherwise clockwise) are above the 45 degree diagonal drawn through C and the diagonal has a positive slope (relative to the bottom frame edge) for types 1 and 2 and negative slope otherwise, or (2) all the terminals of S are on the same module segment. For example, the groups of Figure 2 are given by: $G_1 = \{N_1, N_2, N_3, N_4, N_5, N_6\}$, $G_2 = \{N_7\}$, $G_3 = \{N_8\}$, $G_4 = \{N_{10}\}$, $G_5 = \{N_{11}, N_{12}\}$, $G_6 = \{N_{15}, N_{16}, N_{17}\}$, $G_7 = \{N_9\}$ and $G_8 = \{N_{13}, N_{14}\}$.

We are ready to give an outline of our routability testing algorithm.

Algorithm Routability Testing

1. Partition the nets with one terminal on the bottom frame edge into groups and identify the corresponding corners and diagonals.

2. Determine the outer contour of each group as well as the intersection points (*intermediate terminals*) of the routing with the corresponding diagonal assuming a greedy strategy as close to the module as possible (greedy-in).

3. Move the intermediate points vertically to a horizontal line L such that the separation distance is enough to solve the corresponding river routing problem. Find the characteristic bend points of nets corresponding to the induced river routing problem.

4. Determine if there is any intersection between the wirings of any two different nets or between the wiring of any net and the module or frame boundary.

Notice that if we use the above strategy to produce a *detailed* routing, then the wiring obtained will in general be different than the one generated by the method of [BP].

In the rest of this section we will present fast and efficient parallel algorithms to perform each of the steps outlined in the routability strategy.

It is not hard to see that step 1 can be easily done in time $O(\log n)$ time on a CREW-PRAM with $O(n)$ processors. The details will be left to the reader. Let G_i be a group of nets with a 45 degree diagonal. Each net of this group has one terminal on the module boundary and another on the bottom frame edge. Assume that these nets are of type 1 or 2. Similar definitions and algorithms can be developed for the other types of nets. Let $N =< a, b >$ be a net of this group such that a is furthest from the corner of this group. Then N is called the *representative* net of G_i. In Figure 2, N_1, N_7, N_8, N_{10}, N_{11} and N_{17} are the representative nets of G_1, G_2, G_3, G_4, G_5 and G_6 respectively. Notice that the wiring of a representative net will determine the contour of the wirings of all nets in G_i obtained by a greedy-in strategy. Our next goal is to show how to determine this contour and the intermediate terminals (i.e., points of intersections of the wiring with the diagonal).

Each convex corner lying between two terminals of two nets in a group G may introduce a corner in the wiring of a certain net. If the wiring of a certain net intersects the diagonal of the group *before* such a corner is introduced, we say that the intersection takes place *near* that corner. For example, Nets N_1 and N_2 intersect the diagonal near E in Figure 3.

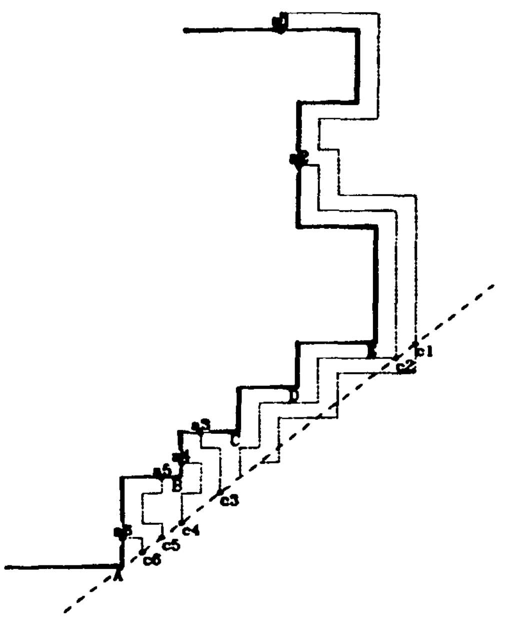
Algorithm Intermediate Terminals

Input: Corners of module boundary, module terminals of a group of nets (type 1 or 2), and the corner and the diagonal of the group.

Output: The intermediate terminal of each net in the group.

1. Rank each terminal of the group according to its order counterclockwise around the boundary. Call the corresponding rank of a net *sequence number*.

2. Calculate the distance between each convex corner and the diagonal of the group. For each convex corner C, determine an integer k such that



	a1	a2	a3	a4	a5	a6
Sequence Number :	6	5	4	3	2	1

	A	B	C	D	E
Bounding Value :	1	4	6	6	5

	N1	N2	N3	N4	N5	N6
Bounding Point :	E	E	B	A	A	A

Figure 3: Determination of Intermediate Terminals

each net with sequence number $\geq k$ intersects the diagonal near C. k will be called the *bounding value* of C.

3. For each net N, determine the closest convex corner Q with bounding value less than or equal its sequence number. Q is called the *bounding point* of N. From this information, determine the intermediate terminals of all the nets in the group.

Figure 3 shows the lists obtained by the above algorithm for a group of type 1 nets.

Lemma1: The above algorithm correctly determines the intermediate terminals of all the nets in the group. It can be implemented to run on the CREW-PRAM in $O(\log n)$ parallel time with $O(n)$ processors.

Proof: The correctness of the algorithm is easy to establish and will be left to the reader. As for the time complexity, it can be estimated as follows.

Step 1 can be done by sorting and a parallel prefix computation.

Step 2 requires few constant time operations.

Step 3 can be done as follows:

- sort corners countercloskwise and rank them.

- sort corners and terminals in increasing order with sequence number or bounding value as the primary key and the ranks obtained from 1 as secondary key.
- For each terminal, find the predecessor which has the maximum rank.

Finding the contour of each group routed by a greedy-in strategy can be done by using the methods of [CJ].

We can now move all the intermediate terminals vertically to a horizontal line L (parallel to the bottom frame edge) such that the separation distance is enough to solve the corresponding river routing problem. By the method developed in [CJ], we can find the characteristic bend points as well as the separation needed. This gives enough information to complete the routability testing (Step 4 of Algorithm Routability Testing).

Lemma2: If the given problem is routable, then the intersection of wiring of each net produced by the greedy-in and the greedy-out strategies is a point below the intermediate terminal of the net and has the same x-coordinate as the intermediate terminal.

Proof Sketch: If the intersection is above the intermediate terminal, then there exists a net N whose greedy-out wiring intersects a corner C before intersecting the wiring generated by the greedy-in strategy. Therefore the given instance is unroutable.

The information obtained above about the wiring of each net can be generated regardless of whether the given instance routable or not. As a matter of fact, a routing exists if and only if there is no intersection between the wirings of any two different nets or between the wiring of a net and the module or the boundary frame. We now discuss how to test for such an intersection.

For each frame edge, the nets whose terminals are the extreme terminals on that edge are called *boundary* nets. There are eight such nets which can be partitioned into four *adjacent* pairs. For example, the right boundary

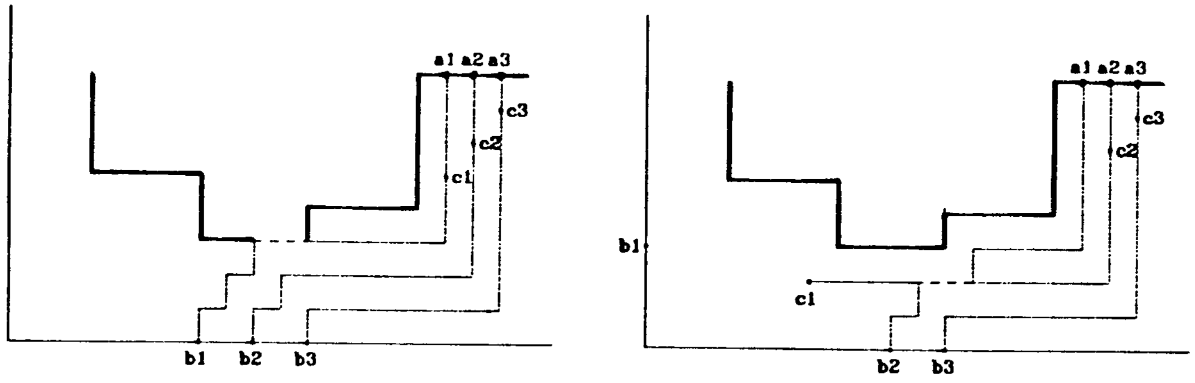


Figure 4: Case 1. Intersections

net of the bottom frame edge is adjacent to the bottom boundary net of the right frame edge.

For a given placement, the nets may be unroutable because: (i) the area between module boundary and frame edge is not enough, or (ii) the area between module edges is not enough. Case (i) can be detected by the following intersections:

1. Intersection between (a) wiring generated by greedy-out strategy of those nets with one terminal in a fixed frame edge and (b) module boundary or wiring generated by greedy-in strategy of those nets with one terminal in another frame edge. See Figure 4 for an example.

2. Intersection between the wiring generated by greedy-in strategy and frame boundary. See figure 5 for an example.

3. Intersection between the wiring of adjacent boundary nets. See Figure 6 for an example.

Each of the cases above will be reduced to testing the intesection between two set of line segments. Consider the case of the wiring generated by the greedy-out strategy applied to those nets with one terminal on the

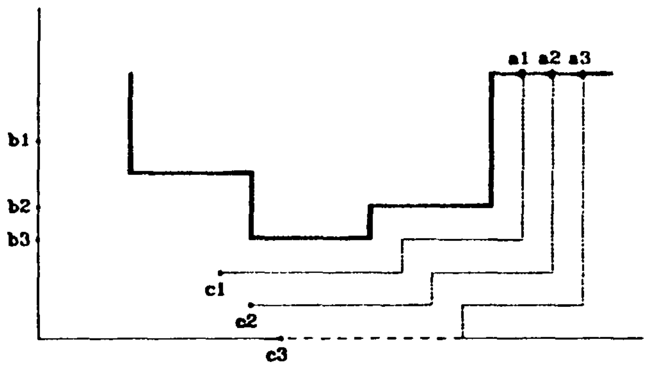


Figure 5: Case 2. Intersections

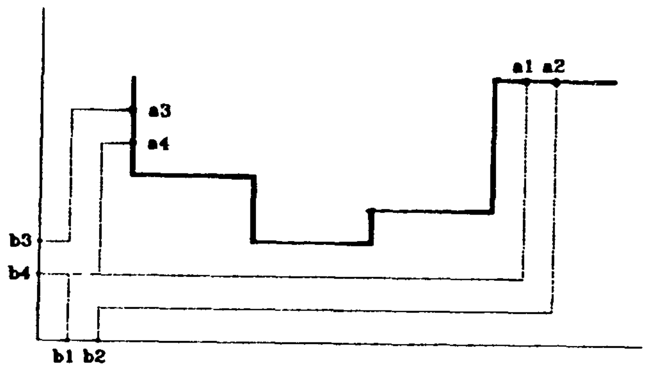


Figure 6: Case 3. Intersections

bottom frame edge. Similar arguments can be used for all the other cases. Let S_1 be the set of horizontal segments given by:

1. All horizontal segments between two characteristic bend points of each net.
2. All horizontal wire segments of vertical blocks.
3. All the horizontal wire segments of the right most net of each left block.
4. All the horizontal wire segments of the left most net of each right block.

Let S_2 be the set of all vertical wiring segments generated by a greedy-in strategy of the outermost net with no terminal on the bottom frame edge and all the vertical line segments of the module boundary.

Lemma3: Case 1 intersection occurs if and only if there exists an intersection between S_1 and S_2. The number of line segments involved is $O(n)$ and hence the testing can be done in time $O(\log^2 n)$ on the CREW-PRAM with $O(n)$ processors.

Proof: From the river routing algorithms described in [CJ], we know that the number of line segments in S_1 is less than $2n$. Moreover, it is obvious that the number of line segments in S_2 is less than $2n$. Therefore the total number of line segments involved is $O(n)$. Intersection can be determined by the methods of [MS] with corresponding time complexity of $O(\log^2 n)$.

Similarily for case 2, define S_1 as the set of all the wire segments of the representative nets of the groups and S_2 as the set consisting of the four frame edges. Again it can be checked that case 2 intersections can be detected by determining whether or not S_1 and S_2 intersect. We leave the third case to the reader.

We now address the problem of whether there is enough area to do the wiring between the module edges. Let $< a_i, b_i >$ and $< a_j, b_j >$ be extreme nets (an extreme net of a group is a net whose module terminal is the first

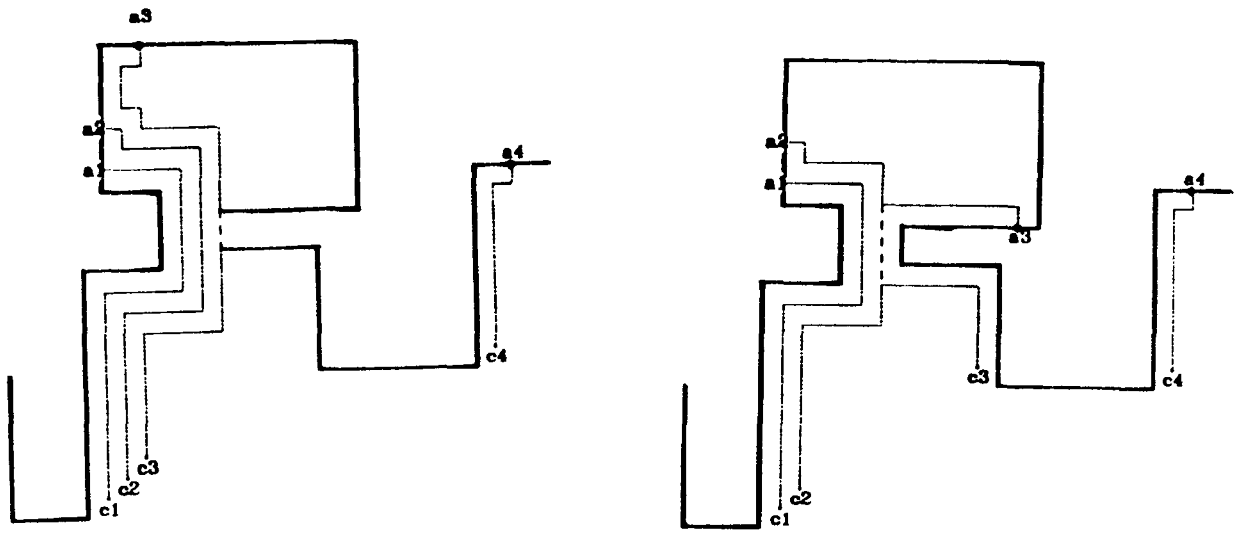


Figure 7: Intersections Determined by Test Pairs

or last when the terminals are sorted in a clockwise direction) of two groups G_k and G_l such that a_i and a_j are adjacent along the module boundary. This pair of nets is called a ***test pair***. Figure 7 shows how to use such test pairs.

Lemma4: The intersection between the wirings generated by the greedy-in strategy can be tested by examining the intersection of test pairs The total number of line segments involved is $O(n)$ and hence this can be done in $O(\log^2 n)$ time on the CREW-PRAM with $O(n)$ processors.

The intersection between a greedy-in wiring and a module segment can be easily determined by the method of ([MS]). Therefore we have the following.

Theorem1: Given an instance of the routability testing problem, we can test whether a solution exists in $O(\log^2 n)$ time with $O(n)$ processors on a CREW-PRAM, where n is the length of the input.

Notice that the above strategy for routability testing can be used to obtain a detailed routing in $O(\log^2 n)$ parallel time. However, the number of processors required is $O(n^2)$ since $\Omega(n^2)$ bend points may have to be determined.

4 Minimizing Wire Length

Suppose that in addition to the input $< \mathcal{M}, \mathcal{F}, \mathcal{N} >$, a wiring of all the nets in $\mathcal{N}$ is also provided. Our problem is to modify the given wiring in such a way that the total wire length is minimized. The strategy of deleting *empty U's* outlined in [BP] minimizes the total wire length. In this section, we will develop a fast and efficient parallel algorithm which minimizes the total wire length.

A *U-Wire* is a sequence of three successive segments resulting from two successive 90 degree turn clockwise or counterclockwise. A U-wire is *reducible* (empty in the terminology of [BP]) if the line segment one unit from the base is not occupied by another wire or module edge, or is occupied by the base of a reducible U-wire. It is shown in [BP] that a routing with no reducible U's achieves the minimum total wire length. However their algorithm is inherently sequential.

It is clear that shapes more complicated than reducible U's have to be considered if a fast parallel algorithm is desired. We will assign *types* to each segment of the given module and wiring as follows. Trace the module boundary clockwise starting from an arbitrary point. Each horizontal segment traversed from left to right is of *type 1*, otherwise it is of *type 2*. A vertical segment is of *type 1* if it is traversed top down; otherwise, it is of *type 2*. We now extend this classification to each segment of the wiring. If we traverse a wire from its module terminal to its frame pad, then a horizontal segment is of *type 1* if it is traversed from left to right, otherwise it is of *type 2*. We can similarily extend the definition to vertical segments. A *horizontal well* is a maximal consecutive sequence of wire segments $e_1, e_2, \ldots, e_k$, such that e_1 and e_k are horizontal with nonempty vertical projections, and $e_1, e_3, \ldots, e_{2t+1}$ are of one type (for some t) and the rest of the horizontal segments are of the other type. Notice that we can have *left* horizontal wells (Figure 8(a)) and *right* horizontal wells (Figure 8(b)). In a similar fashion, we can define *vertical wells*. Given a net N with a well W, an *obstacle* of W is a set S of consecutive module segments or wire segments of a net of *different type* such that S lies inside W. For example, in Figure 2 net N_{17} has a vertical well with a set of module segments as an

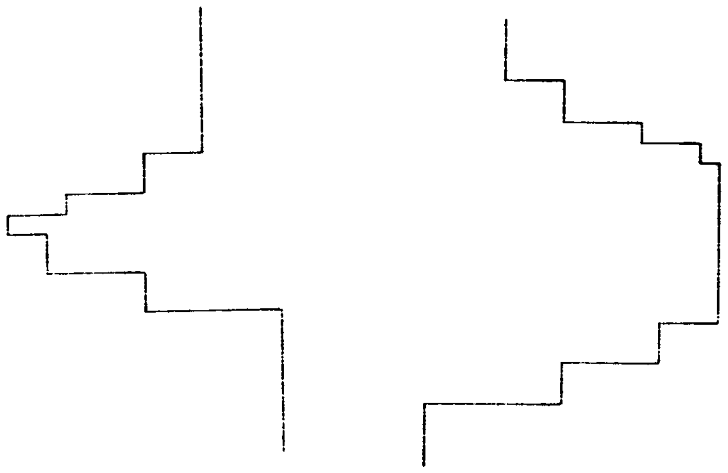


Figure 8: Left and Right Wells

obstacle while net N_{11} has a horizontal well with a set of wire segments as an obstacle.

We can shrink wells whenever possible as follows. Let W be a horizontal well with intial segment $e_1 = (B_1, A_1)$ and last segment $e_2 = (B_2, A_2)$ with (say) the x-coordinate of B_1 less than or equal the x-coordinate of B_2. In addition, suppose there is no obstacle inside W. Then we can apply the transformation shown in Figure 9 to shorten W. If W has an obstacle inside it, then we find a maximal set of wells with the same obstacle and apply the transformation shown in Figure 10 . We now show the following.

Lemma5: Suppose there are k reducible wells for a given wiring. Then after applying the transformations described above, the number of reducible wells will be $\leq \frac{k}{2}$.

Proof: At most one well will be created between two previous wells after the reduction step.

It follows from the above lemma that if n is the number of wire segments given as input then after $\log n$ iterations of the above transformations, no reducible wells will remain and therefore the resulting wiring is as short as possible.

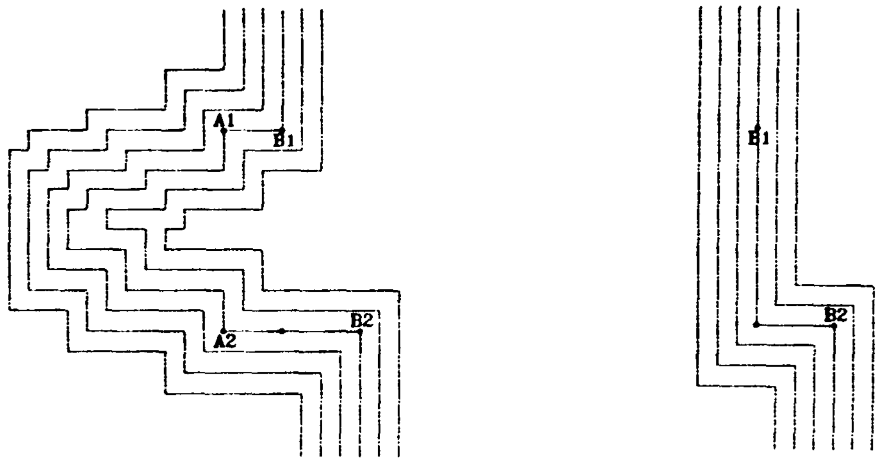


Figure 9: Transformation of Wells with no Obstacles

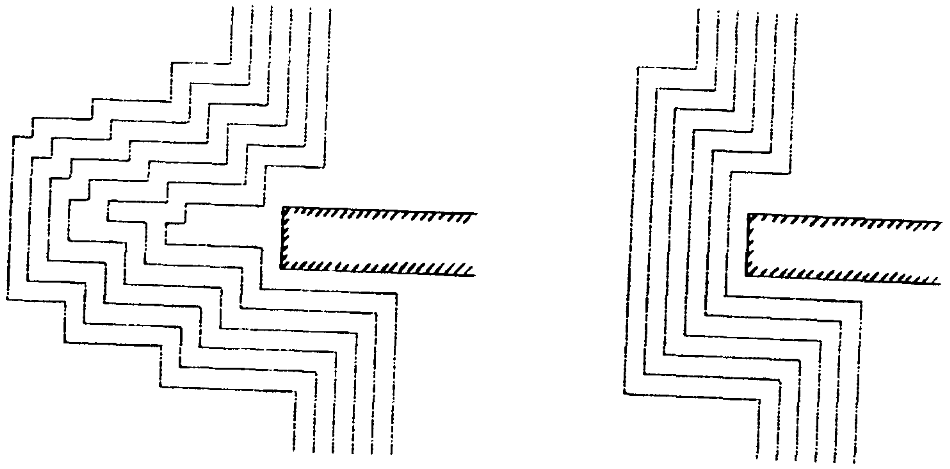


Figure 10: Transformation of Wells with an Obstacle

Theorem2: Given an intial wiring, we can change this wiring so that the resulting wiring is of minimum total length in time $O(\log^2 n)$ with $O(n)$ processors on the CREW-PRAM model, where n is the length of the input.

Proof: The algorithm consists of identifying horizontal and vertical wells and reducing them whenever possible. This process has to be repeated $O(\log n)$ times as implied by the previous lemma. Since there will be no reducible U-wires, the resulting wiring is of minimum length. What remains to be shown is that each iteration can be implemented in $O(\log n)$ time with $O(n)$ processors, where n is the input length. One can verrify that determining the types of the segments takes $O(\log n)$ parallel time by using essentially sorting and that identifying the wells can be done in $O(\log n)$ parallel time by using path doubling and few other constant time operations. Once the wells are identified, applying the above transformations can be done with few simple operations in $O(1)$ time.

5 Other Parallel Models

A careful look at the algorithms presented in the previous sections will reveal that the basic operations used are sorting, path doubling (or shorcutting), prefix computations and few other simple operations. Each of these operations can be implemented efficiently on a mesh-connected processor or on the hypercube. For example, testing whether a given instance of our problem is routable can be done on the mesh in $O(\sqrt{n})$ time and on the hypercube in $O(\log^2 n)$ time, where n is the length of the input. On the other hand, the algorithm presented in the previous section uses $O(\log n)$ iterations. However, the size of the relevant data involved decreases by a factor of 2 after each iteration and hence has an $O(\sqrt{n})$ implementation on the mesh.

6 References

[BP] B. Baker and R. Pinter, "An Algorithm for the Optimal Placement and Routing of a Circuit Within a Ring of Pads," Proceedings of the 24th Symposuim on Foundations of Computer Science, Nov. 1983, pp. 360-370, Tucson, Az.

[CJ] S. Chang and J. JáJá, "Parallel Algorithms For River Routing," submitted for publication.

[D et al] D. Dolev, K. Karplus, A. Seigel, A. Strong and J. Ullman, "Optimal wiring between rectangles," Proceedings 13th Annual ACM Symposuim STOC, May 1981, pp. 312-317.

[J] D. Johannsen, "Bristle blocks: a silicon compiler," Proceedings 16th Design Automation Conference, June 1979, pp. 310-313.

[MS] R. Miller and Q. Stout, "Mesh Computer Algorithms for Computational Geometry," Technical Report 86-18, Dept. Computer Science, State University of New York at Buffalo, July 1986.