SRC-TR 87-103

Parallel Algorithms for River Routing

by

Joseph Ja'Ja'
Shing-Chong Chang

Parallel Algorithms For River Routing[1]

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

and

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

[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.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
1987		**00-00-1987 to 00-00-1987**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Parallel Algorithms for River Routing	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of Maryland,Electrical Engineering Department,College Park,MD,20742**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
see report

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**30**	

1 Introduction

It is well-known that many of the optimization problems arising in VLSI routing are NP-complete (e.g. [KL],[L],[SB],[S]). One notable exception is the class of *river routing* problems associated with a hierachical layout strategy such as Bristle-Blocks([J]). See ([D et al],[LP],[P],[SD],[T]) for more examples. Efficient serial solutions have already appeared in the literature for most of these problems. In this paper, fast parallel algorithms for several river routing problems are presented. These algorithms are efficient in the sense that the number of processors used is $O(n)$, where n is the size of the input. Most of the known algorithms seem to be inherently sequential and new techniques are developed to obtain these parallel algorithms.

We consider two models of parallel computation: the Concurrent Read Exclusive Write (CREW) PRAM model and the two-dimensional mesh of processors. We are assuming that the reader is familiar with both of these parallel models and the corresponding parallel techniques such as path doubling, prefix computation, sorting and the Euler tour technique. We obtain fast and efficient parallel algorithms for all the problems considered. In particular, $O(\log n)$ or $O(\log^2 n)$ time algorithms with $O(n)$ processors are developed for the separation problem, the minimum offset problem and for the routability problem around a rectilinear polygon ([P]). For the mesh, our algorithms all run in time $O(\sqrt{n})$ except for the offset problem whose algorithm runs in time $O(\sqrt{n}\log n)$. Optimal algorithms are developed for several subproblems that are interesting in their own. One such subproblem is to determine the contours of the unions of sets of contours within a rectilinear polygon.

The rest of the paper is organized as follows. The main problems are introduced in the next section, while section 3 deals with the separation problem. The offset problem is considered in section 4 and the problem of routing around a rectangle is tackled in section 5.

2 Definitions

The class of general river routing problems involves routing between ordered sequences of terminals such that the final layout is planar. One such

problem is the wiring of two ordered sets of terminals $\{b_1, b_2, \ldots, b_n\}$ and $\{t_1, t_2, \ldots, t_n\}$ across a channel between the parallel boundaries of two rectangles. The width of the channel is the vertical distance between the two lines forming the channel. The *separation problem* is to find the minimum width of the channel necessary to wire all nets such that any two wires are separated by a unit distance. We will restrict ourselves to the case where the wires are rectilinear, i.e., there is a grid structure such that each wire consists of a set of grid line segments. Our methods generalize for all the other known variations ([SD],[T]). The *offset problem* is to find the distance to slide one of the rectangles so as to achieve the minimum possible separation between the rectangles.

A more general version of the river routing problem that is known to have an efficient serial algorithm is to perform planar routing where the ports lie on the boundary of a simple rectilinear polygon. In this case, we are interested in whether the routing is possible or not and, if it is possible, we have to provide the detailed routing. Several interesting subproblems such as finding the contour of the union of a set of rectilinear polygons or determining whether a set of nets can be wired within a set "passages" are also tackled.

3 The Separation Problem

Let $\{N_i =< b_i, t_i > \mid 1 \leq i \leq n\}$ be an instance of the channel separation problem. Notice that b_i and t_i will be also used to denote the horizontal coordinates of the terminals relative to an arbitrary origin which is assumed to be on the upper row. A net N_i is a *right going* net if $b_i < t_i$. If $b_i > t_i$, then N_i is a *left going* net. Otherwise, it is a *vertical* net. We can partition the nets into *right going* blocks, *left going* blocks and *vertical* blocks. A set of right going nets $N_i, N_{i+1}, \ldots, N_p$ is a right block if it is a maximal block with the property $b_k < b_{k+1} \leq t_k$, for any $i \leq k < p$. We can similarly define left blocks and vertical blocks.

For any right block $\{N_i, N_{i+1}, \ldots, N_p\}$, define the *rank* of net N_k to be $k - i + 1$, $i \leq k \leq p$. For a left block $\{N_i, N_{i+1}, \ldots, N_p\}$, the *rank* of net N_k is defined to be $p - k + 1$, $i \leq k \leq p$.

The wiring problem is reduced to wiring each block separately. We will

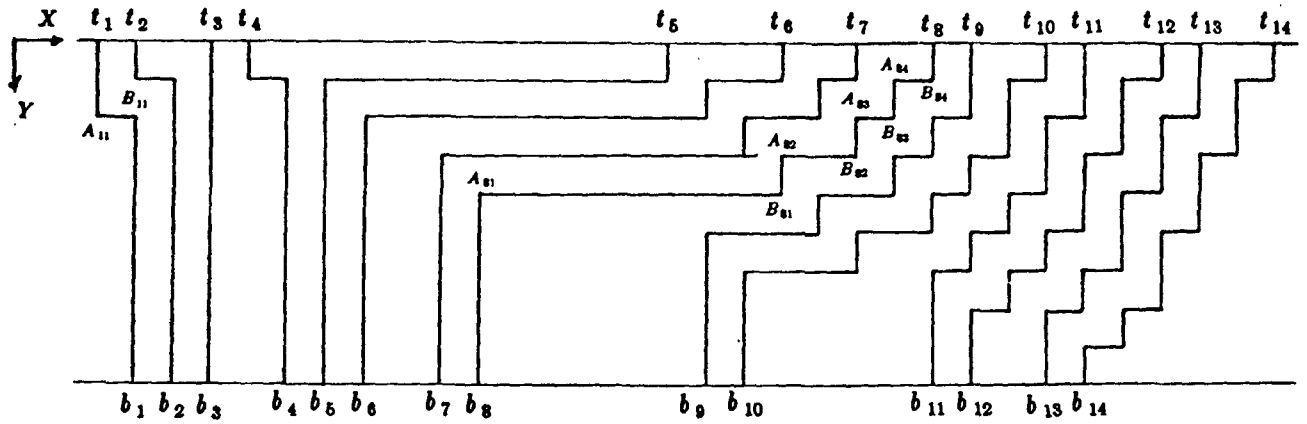

Figure 1: Basic river routing problem

concentrate on the wiring of right blocks. Obvious changes can be made to deduce the corresponding algorithm for left blocks.

The wiring of a net can be specified by the coordinates of its bend points. For example, net N_1 of Figure 1 has the bend points A_{11}, B_{11}. For each net N_i, we have $2k$ bend points, $A_{i1}, A_{i2}, \ldots, A_{ik}$ and $B_{i1}, B_{i2}, \ldots, B_{ik}$, for some k. Not all of these bend points are needed to determine the overall wiring. Let's call A_{i1} and B_{i1} (bend points closest to the bottom row) the *characteristic* bend points and all the others *ordinary* bend points. Notice that the characteristic bend points uniquely define the overall wiring since once we have the wiring of N_{i-1} and the characteristic bend points A_{i1} and B_{i1}, we can determine all the ordinary bend points of N_i very easily. Figure 1 shows an example of a river routing problem and a wiring achieving the minimum separation.

The algorithm to find the minimum separation is based on the following lemma.

Lemma 1 *Let N_i be a net in a right block and let $\hat{j}$ be the minimum $j \leq i$ such that $t_j + (i - j - 1) \geq b_i$. Then the coordinates of the characteristic bend points of N_i are $A_{i1} = (b_i, i - \hat{j} + 1)$ and $B_{i1} = (t_{\hat{j}} + i - \hat{j}, i - \hat{j} + 1)$.*

Proof: Since $\hat{j}$ is the minimum $j \leq i$ such that $t_j + (i-j-1) \geq b_j$, there is no terminal point at $(t_{\hat{j}}-1, 0)$. Hence there is a bend point at $(t_{\hat{j}}, 1)$ for net $N_{\hat{j}}$. The number of vertical grid lines between $t_{\hat{j}}$ and b_i is $b_i - t_{\hat{j}}$ and hence smaller than the number of nets between $N_{\hat{j}}$ and N_i,i.e., $i-\hat{j}+1$ horizontal tracks are needed to route net N_i. A simple argument will show that the coordinates of the characteristic bend points have the values stated in the Lemma.

We now show how to compute in parallel the index $\hat{j}(i)$ for each i. A simple binary search method will yield a PRAM algorithm that runs in $O(\log\ n)$ time with $O(n)$ processors. However this does not seem to yield an optimal algorithm when implemented on the mesh. We use the following procedure.

Algorithm Index

input: A set of nets $< b_i, t_i >$, $1 \leq i \leq n$, forming a right going block.

output: $\hat{j}(i)$ such that $\hat{j}(i)$ is the minimum j such that $b_i - t_j \leq i - j - 1$, for each $1 \leq i \leq n$.

1. Compute $b'_i = b_i - i$ and $t'_j = t_j - j - 1$ for each i and j.
2. Sort the t'_js, say $t_{p_1} \leq t_{p_2} \leq \ldots \leq t_{p_n}$.
3. For each p_i, determine $f(p_i) = min\{p_k | i \leq k \leq n\}$.
4. Sort the b'_is and the t'_js such that if a $b'_i = t'_j$, the b'_i is pushed to the lower rank.
5. For each b'_i, let t'_{p_j} be the closest $t'_{p_k} \geq b'_i$. Then $f(p_j) = \hat{j}(i)$.

The correctness proof of the above algorithm is straightforward and will be left to the reader. It is also clear that the above procedure can implemented on the mesh to run in $O(\sqrt{n})$ time. We now give the algorithm to find the minimum separation as well as the charactersitic bend points of all the nets.

Algorithm Separation

Input: A set of n nets given by $\{N_i =< b_i, t_i > | 1 \leq i \leq n\}$.

Output: The characteristic bend points and the minimum separation.

1. Partition the nets into blocks by creating the following chains.

For each i **do**

If $t_i > b_i$ and $b_i < b_{i+1} \leq t_i$, **then** $Next(N_{i+1}) = N_i$.

If $t_i < b_i$ and $t_i \leq t_{i+1} \leq b_i$, **then** $Next(N_i) = N_{i+1}$

For each net, determine the sink reachable from this net and its corresponding rank.

2. Apply Algorithm Index to get the index $\hat{j}(i)$ for each i. Use Lemma1 to obtain all the characteristic bend points.

3. Let the characteristic bend points be $B_{i1} = (x_{i1}, y_{i1})$, $1 \leq i \leq n$. Then the minimum separation is $d + 1$, where $d = max\{y_{11}, \ldots, y_{n1}\}$.

Theorem 1 *Algorithm separation correctly finds the characteristic bend points of the n input nets as well as the minimum channel separation. This algorithm can be implemented on a* $\sqrt{n} \times \sqrt{n}$ *mesh in* $O(\sqrt{n})$ *time and on an* $O(n)$ *processor PRAM with the corresponding running time of* $O(\log\ n)$.

Proof: The correctness proof follows from the discussion at the beginning of this section (See also [D et al], [LP] and [T]). Step 1 of the algorithm uses the path doubling technique (PRAM) or the technique in [AH]. The rest of the time analysis is straightforward.

From the characteristic bend points we can easily obtain all the bend points of all the nets. However we have to use $O(n^2)$ processors in this case because there might be $\Omega(n^2)$ bend points.

4 The Offset Problem

Given two sequences of terminals $t_1, t_2, \ldots, t_n$ and $b_1, b_2, \ldots, b_n$ to be wired through a channel, the horizontal displacement of the b_i's relative to the

t_i's is called the *offset*. Our goal in this section is to develop a fast parallel algorithm to compute the offset that minimizes the separation needed to wire all the nets. Our algorithm will yield a new *serial* algorithm whose running time is the same as the best known sequantial algorithm [D et al], but it is considerably simpler.

Assume that the t_i's are fixed. The b_i's should be moved to the right or left so as to achieve the smallest possible separation. As before, partition the nets into blocks and let D_r be the largest separation of a right block, say R, and let D_l be the largest separation of a left block, say L. We need the following fact established in ([D et al]).

Lemma 2 *Sliding a left block to the right will not decrease the required separation for wiring this block. Similarly, sliding a right block to the left cannot decrease the width of the channel required by that block. In particular, if $D_l = D_r$, the current offset achieves the minimum possible channel width.*

One can check that for the routing problem in Figure 1, $D_l = 3$ and $D_r = 9$. Figure 2 shows the wiring obtained with an offset of $k = 7$. In this case, $D'_l = D'_r = 5$.

Assume without loss of generality that $D_r > D_l$. Notice that $D_r \leq n+1$. Suppose that we want to find out whether there exists an offset k such that the resulting separation is not greater than d, $D_l \leq d \leq D_r$. Since the t_i's are fixed, each b_i will have to be restricted to a certain fixed interval. Following arguments similar to those used in the previous section, it is not hard to see that b_i has to satisfy the following two conditions.

$$b_i + k - t_{i-(d-1)} \geq d - 1 \tag{1}$$

$$t_{i+d-1} - b_i - k \geq d - 1 \tag{2}$$

It is clear that 1 and 2 are equivalent to

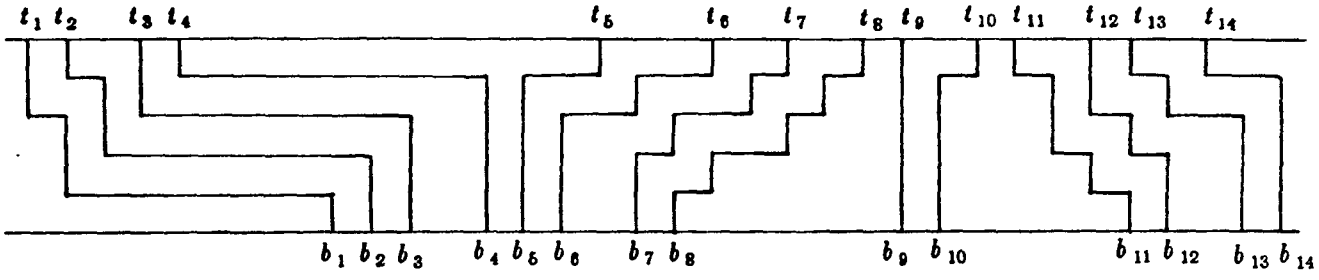

Figure 2: Minimum separation with offset 7

$$\alpha_{id} \leq k \leq \beta_{id} \tag{3}$$

where α_{id} and β_{id} are given by

$$\alpha_{id} = d - 1 - (b_i - t_{i-d-1})$$

$$\beta_{id} = (t_{i+d-1} - b_i) - (d - 1)$$

Such a k exists if, and only if,

$$max_i \alpha_{id} \leq min_i \beta_{id} \tag{4}$$

Lemma 3 *An offset of k which induces a separation of d exists if, and only if, condition4 is satisfied for each i.*

Proof: It is clear that 4 is equivalent to (1) and (2). A set of nets can be river routed within a channel of width d if, and only if, for each net i, the distances between b_i and $t_{i-(d-1)}$ and between b_i and t_{i+d-1} are large enough to accomodate the $d-1$ nets between t_i and $t_{i-(d-1)}$ and between t_i and t_{i+d-1}, i.e., if and only if, the number of vertical tracks is at least $d-1$.

A straightforward binary search over the interval (D_l, D_r) using the above lemma will yield an optimal offset in $O(\log\ n)$ iterations. On a sequantial machine each iteration will require $O(n)$ time, while on a PRAM each will require $O(\log\ n)$ time. The obvious implementation on the mesh of each iteration will require $O(\sqrt{n})$ time. Therefore we have the following theorem.

Theorem 2 *Finding an offset that generates the minimum separation can be done on a PRAM with $O(n)$ processors in time $O(\log^2\ n)$. On the mesh, this algorithm will take $O(\sqrt{n}\log\ n)$ time.*

5 Routing In a Simple Polygon

The routing problem of nets within a simple rectilinear polygon introduced in([P]) is a generalization of the standard river routing problem. In this case we are supposed to connect a set of terminals $a_1, a_2, \ldots, a_n$ on the boundary of a simple rectilinear polygon to another set of terminals $b_1, b_2, \ldots, b_n$ on the boundary of the same polygon such that all the wires lie within the polygon and no two wires intersect. *Routability testing* is to determine whether or not a one layer routing is possible and *detailed routing* is to specify the actual wiring of the n nets, if they are routable. In the serial

case, a simple greedy strategy ([P]) produces an optimal solution for detailed routing. However, such a scheme is inherently sequential and hence an alternative method should be developed for the parallel case.

Our strategy for the detailed routing problem will consist on identifying a set of critical nets and then performing the wiring of each such net with the nets "covered" by it separately. It turns out that these special nets will also play a crucial role in the routability testing problem.We will restrict ourselves to the rectangle case. However all the algorithms can be generalized to any rectilinear polygon.

5.1 Detailed Routing

We will begin with few definitions. Given the set of nets $\{N_i =< a_i, b_i > \mid 1 \leq i \leq n\}$ whose terminals lie on the boundary of a rectangle R, we let the lower left corner of R be the origin of an (x, y) coordinate system. The four corners of R have coordinates $(0,0)$,$(l,0)$,(l,h) and $(0,h)$, where l and h are respectively the length and height of the rectangle. If we cut R at $(0,0)$ and straighten the boundary clockwise into a line, the corresponding linear coordinate of a point P on the boundary will be denoted by $d(P)$. It is trivial to compute $d(P)$ from the two-dimensional coordinates of P.

Let $N_i =< a_i, b_i >$ be an arbitrary net. The terminals a_i and b_i divide the boundary of R into two parts. The part of length $\leq h + l$ will be called the *internal boundary* of N_i. The other part will be called the *external boundary*. A net N_i is *covered* by another net N_j if the terminals of N_j are in the external boundary of N_i and the terminals of N_i are in the internal boundary of N_j. A *representative net* is a net that is not covered by any other net. Figure 3 shows an example of a detailed routing problem such that N_1, N_6 and N_{14} are the representative nets. We can partition the nets into *groups* such that each group consists of a representative net and all the nets covered by it. The groups in Figure 3 are $\{N_1, N_2, N_3, N_4, N_5\}$,$\{N_6, N_7, N_8, N_9, N_{10}, N_{11}, N_{12}, N_{13}\}$, and $\{N_{14}, N_{15}\}$.

Given a net N_i, trace the boundary of R counterclockwise starting from any point on the external boundary of the net, the terminal that we meet

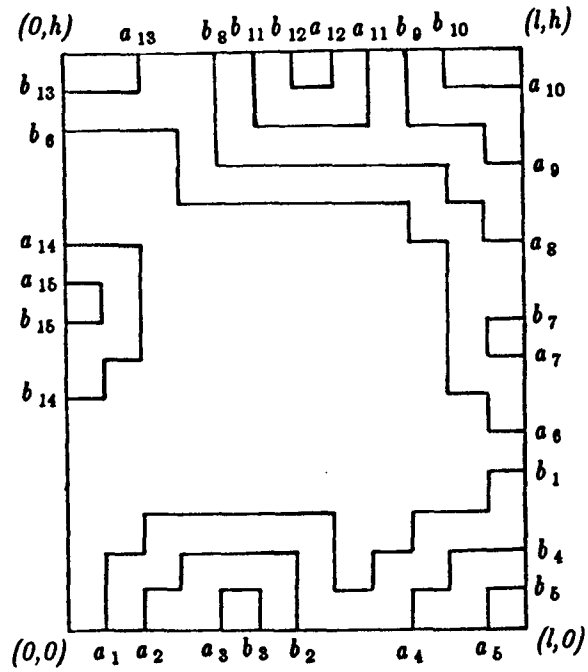

Figure 3: Basic river routing around a rectangle boundary

first is the *left* terminal, while the other is the *right* terminal. The net adjacent to the left terminal will be called the *left* net of N_i, while the net adjacent to the right terminal will be called the *right* net of N_i. For example, in Figure 3, a_4 and b_4 are respectively the left and right terminals of net N_4. N_2 is the left net and N_1 is the right net of N_4.

Lemma 4 *Suppose a given instance of the above problem is routable. Then the routing can be performed by routing each group of nets separately.*

Proof: Take any net N that does not cover any other net. Route it as close to the boundary as possible. Remove N and consider the contour formed by the new wiring and the old boundary. Repeat the procedure until all the nets are wired. It is clear this strategy will result in a legal wiring if the nets are routable. Hence each group of nets can be routed separately.

The general strategy for specifying the routing will be the following: (i) identify the proper groups , (ii) find the representative nets, and (iii) specify the routing of each group. We will start by tackling (i) and (ii). Essentially we want to create a chain of the nets involved in each group such that a representative net is a sink. The terminals of each net $N_i =< a_i, b_i >$ will be labeled as follows. Let $N_k =< x, y >$ be the net in the external part of N_i with x closest to a_i. If a_i lies in the internal part of N_k, then set $label(a_i) = 0$ (N_i is not a representative net). Else, a_i and b_i lie in the external part of N_k. If the distance between y and b_i in the external part of N_k is $\leq h + l$, then set $label(a_i) = i$ (N_i is a representative net). Otherwise, set $label(a_i) = k$ (N_i is a representative net if and only if N_k is). In a similar fashion we can determine $label(b_i)$. For example, in Figure 3, $label(a_2) = 0$, $label(b_2) = 4$, $label(a_1) = 14$ and $label(b_1) = 1$. Now chains whose nodes are the nets are created by setting the link of an arbitrary net $N_i =< a_i, b_i >$ as follows.

- If $label(a_i) = i$ or $label(b_i) = i$, then set $link(N_i) = N_i$.
- If $label(a_i) = 0$ or $label(b_i) = 0$, then set $link(N_i) = 0$.
- Else set $link(N_i) = N_{label(a_i)}$, assuming a_i is the left terminal.

The lists created above are chains with possibly one of them being a circular list. We can identify the circular list (if any) fast by a path doubling technique and break it arbitrarily. From now on, we assume that we have a set of chains. Referring to Figure 3 again, we have $link(N_1) = N_1$, $link(N_6) = N_6$, $link(N_8) = N_7$, $link(N_{14}) = N_{14}$, and $link(N_i) = 0$ for all the remaining nets. The following lemma holds.

Lemma 5 *A net N_i is a representative net if and only if the sink reachable is not* 0.

Proof: Suppose N_i is not a representative net. Then there exists another net N_k that covers it. Choose N_k to be closest to one of the terminals of N_i. If two terminals of N_i and N_k are adjacent, then it is easy to check that $link(N_i) = 0$. Otherwise there are several nets between N_i and N_k. N_i will be linked to some of these nets such that the last one on the chain is adjacent to N_k, i.e., the sink will be labelled 0.

Lemma 6 *Let $N_{r1}, N_{r_2}, \ldots, N_{r_k}$ be all the representative nets and let $R(N_{r_i})$ be the number of nets in the internal boundary of N_{r_i}. Then $\sum_{i=1}^{k}(R(N_{r_i}) + 1) = n$. Moreover, there exists a wiring strategy such that N_{r_i} has at most $2(R(N_{r_i}) + 1)$ bend points.*

The proof is simple and will be omitted.

Corollary: The total number of bend points of all the representative nets is $O(n)$, where n is the number of nets.

We are now ready to state the algorithm to identify the representative nets and all the nets within each group.

Algorithm Groups and Representatives

Input: A set of nets N_i specified by their terminals.

Output: The goups and their representatives.

1. Compute the d-coordinate value of each terminal and sort these values.

2. Determine $label(x)$ for each terminal x and set up the corresponding chains breaking the circular list if it exists.

3. For each node, determine the sink reachable from that node. Determine all the representative nets.

4. Create new chains as follows. Link each *nonrepresentative* net point to its left net. Make each *representative* net into a sink. Identify the corresponding groups and compute $R(N_{r_i})$ for each representative net N_{r_i}.

Lemma 7 *Let n be the number of nets. Then the above algorithm can be implemented to run in time $O(\log\ n)$ with $O(n)$ processors on the PRAM. On a two-dimesional mesh , it runs in time $O(\sqrt{n})$.*

Proof: The fast efficient algorithms for sorting, path doubling and prefix computation can be used to obtain the run times stated in the lemmma.

We now turn to the problem of routing each group separately. Our goal here is to identify the bend points of each representative net. Note that in general the total number of bend points of all the nets could be $\Omega(n^2)$, where n is the number of nets. However the total number of bend points of the representative nets is always $O(n)$.

Let $N =< x, y >$ be a net in a group whose representative is N_r. Let k be the number of nets between N and N_r, including both N and N_r. The *bounding perimeter of rank k* is the rectilinear boundary of the region determined by N such that the wiring of N_r cannot lie inside it, i.e., this is the boundary of the region within the rectangle of all the points of distance $\leq k$ of the rectangle boundary determined by N. Consider again the case of Figure 3. Let $B_{k,i}$ be the bounding perimeter of rank k induced by net N_i. Figure 4 shows the contours $B_{3,3}$, $B_{3,5}$, $B_{2,2}$, $B_{2,4}$ and $B_{1,1}$. We claim that the following lemma is true.

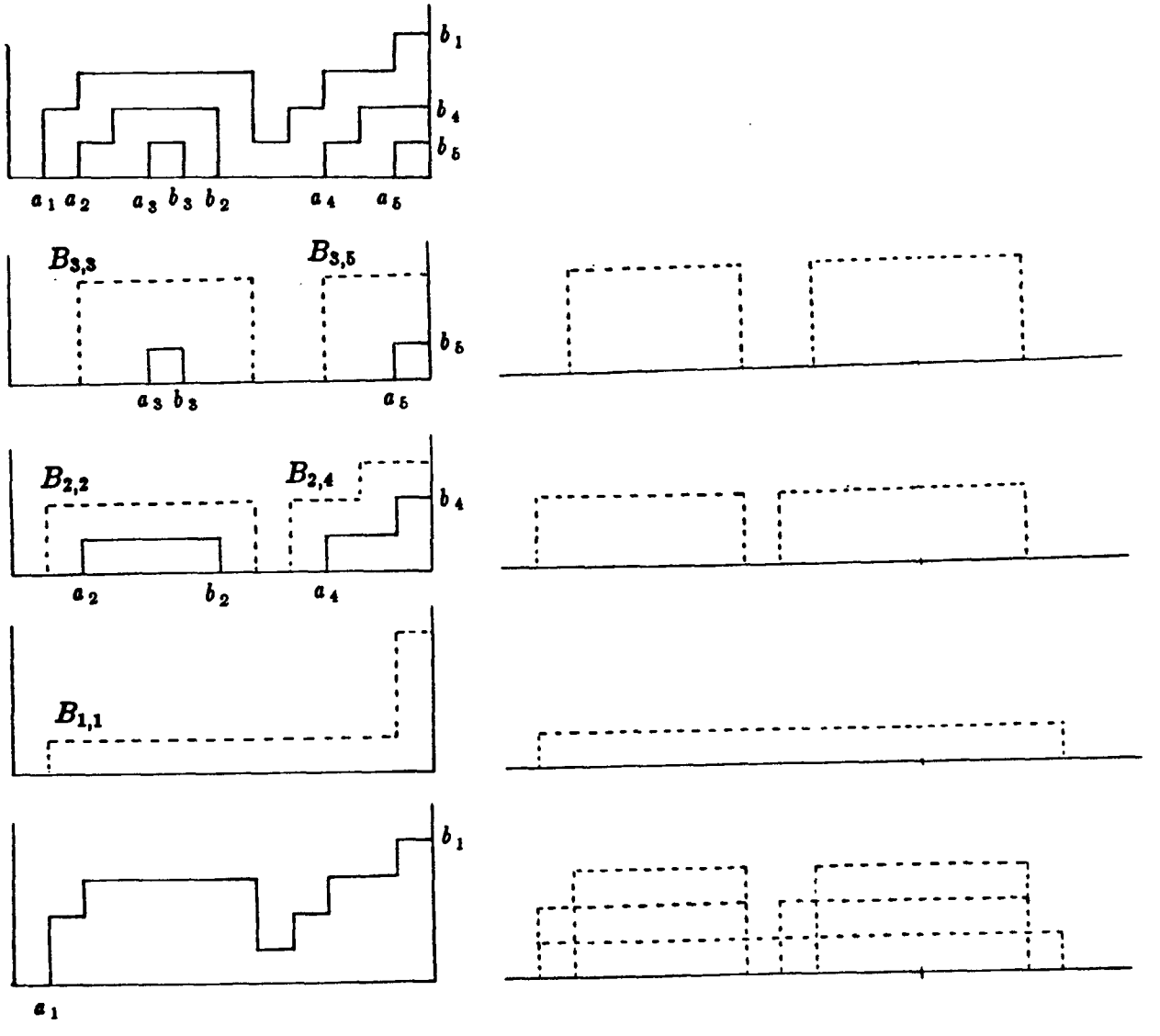

Figure 4: The union of all bounding perimeters

Lemma 8 *The union of all the bounding perimeters of all the nets within a group determines the contour of the group and hence determines the wiring of the representative net.*

Proof: The proof is by induction on the number of nets within a given group. The basis $k = 1, 2$ can be checked easily.

Suppose that the lemma holds for groups with $k > 2$ nets. Let N be the $(k+1)$st net added to a given group. Define the rank of a net to be the number of nets between the representative net N_r and the net. If the rank of N is ≤ 2 it is easy to check by inspection that the lemma holds. Suppose the rank of N is greater than 2. Then pick any net of rank 2 and remove it. Apply the induction hypothesis and put back the removed net. The details can be verified easily.

We now discuss the problem of determining the countour of a group of nets from the corresponding bounding perimeters. Flatten the rectangle into a line as before by making a cut at the left terminal of the representative net. Suppose a terminal p gets mapped into $\bar{p}$. A bounding perimeter connecting p and q of rank k will get mapped into a simple rectangle with endpoints $\bar{p}$ and $\bar{q}$ and height k. Denote the mapped bounding perimeters by $\bar{R}_1, \bar{R}_2, \ldots, \bar{R}_l$. These rectangles determine a contour R given by the coordinates of its bend points. Each such extreme point (p, h), where p is the linear coordinate and h is the distance from the line, is mapped into (p', h), where p' is on the boundary of the rectangle corresponding to the point of linear coordinate p on the line. Few of these points around the corners may not be mapped into extreme points of the contour within the rectangle, but rather onto the boundary. These can be determined quickly and then eliminated. We are now ready to state the algorithm.

Algorithm Contour

Input: A group of nets with their representative.

Output: The bendpoints of the corresponding contour.

1. Assign a weight of 1 to the left terminal and -1 to the right terminal of each net. Order the terminals counterclockwise starting from the left

terminal of the representative net. For each terminal, compute the sum of the weights of all terminals preceeding it, including itself. Call that sum *the rank* of the terminal. The *rank* of a net is the rank of its left terminal. The rank of each net is equal to the number of nets between itself and the representative net.

2. Determine all the bounding perimeters.

3. Cut the boundary of the rectangle at the left terminal of the representative net and strech it clockwise into a line. Map the bounding perimeters into this line . Each corresponding perimeter can be identified by $(\bar{p}, \bar{q}, k)$.

4. Sort the triplets $(\bar{p}, \bar{q}, k)$ according to k. For each k, determine the union of line segments at distance k.

5. From each line segment generated at step 4, determine the corresponding bendpoints. The overall contour can be specified by the bend points.

6. Map the bend points of the contour on the line back into the rectangle. Eliminate those points within the rectangle which are not bend points.

Lemma 9 *If the number of nets in the group is n, then algorithm contour can be implemented in time $O(\log n)$ with $O(n)$ processors. On the mesh, the algorithm runs in time $O(\sqrt{n})$.*

The proof is easy and will be omitted.

The main result of this section is stated in the following theorem.

Theorem 3 *Detailed routing of n nets within a simple rectilinear polygon can be done in time $O(\log n)$ with $O(n)$ processors. On a two-dimensional mesh, the algorithm runs in time $O(\sqrt{n})$.*

5.2 Routability Testing

In this subsection we address the problem of testing whether or not it is possible to route all the nets within the given rectangle without actually determining the routing. The above algorithm assumes that the nets are routable. Notice that the detailed routing algorithm does not generate enough information to do the routability testing since it only produces the bend points of the representative nets. We will present a fast parallel algorithm to handle the testing problem . Several subproblems which had to be solved are interesting in their own and require new parallel techniques for their solutions.

Given a set of nets $\{N_i =< a_i, b_i >, 1 \leq i \leq n \}$, where the terminals lie on the boundary of the rectangle R, these nets may be unroutable for one of the following reasons:

- The graph determined by the nets when restricted to lie within the rectangle is *nonplanar.*
- The wiring of all the nets requires more area.

We start by handling the first issue (easy case). The algorithm tests for any given net $N =< x, y >$, whether any net with a terminal in the internal boundary of N has the other terminal in the external boundary of N.

Algorithm Interconnect Planarity

Input: A set of nets $\{N_i =< a_i, b_i >, 1 \leq i \leq n\}$, where the terminals lie on the boundary of a rectangle.

Output: Determine whether the interconnect pattern is planar.

1. Compute the d-coordinates of all the terminals. Sort the terminals according to these coordinates.

2. For each net $N_i =< a_i, b_i >$ assign the following weights to its terminals: if $d(a_i) > d(b_i)$, then $w(b_i) = 1$ and $w(a_i) = -1$; else, $w(b_i) = -1$ and $w(a_i) = 1$.

3. Calculate the rank (i.e., sum of weights of all predecessors) of each terminal.

4. Output no if there exists a net $N =< x, y >$ (say $d(x) > d(y)$) such that $rank(y) \neq rank(x) + 1$. Otherwise output yes.

Lemma 10 *The above algorithm correctly checks whether the interconnection pattern of the given nets is planar. This algorithm runs in time* $O(\log\ n)$ *time on the PRAM model and in time* $O(\sqrt{n})$ *on a two-dimensional mesh.*

Before we tackle the question of whether the rectangle is large enough to accomodate all the nets, several definitions are needed. A *single side* net is a net whose terminals lie on the same side of the rectangle. If the terminals lie on adjacent sides then the net is called *corner* net. It is a *cross* net if the terminals lie on opposite sides. Partition the single side nets corresponding to each specific side into *single side blocks* such that each net except one (*cover net*) is covered by one or more nets in the block. Moreover each such block is maximal. A *corner block* is a maximal set of corner nets corresponding to the same corner such that each net except one (*cover net*) is covered by one or more nets within the block. Moreover no other net outside a block is covered by the cover net. For example, in Figure 3, N_2 is a single side net, N_1 is a corner net and N_6 is a cross net. The single side blocks are $\{N_2, N_3\}$, $\{N_7\}$, $\{N_{11}, N_{12}\}$ and $\{N_{14}, N_{15}\}$, whose corresponding cover nets are N_2,N_7,N_{11} and N_{14}. $\{N_4, N_5\}$, $\{N_9, N_{10}\}$ and $\{N_{13}\}$ are the corner blocks with N_4,N_9,N_{13} as the corresponding cover nets.

We start by deciding whether the above blocks are routable.

Algorithm: Wiring of Blocks

Input: A set of nets whose terminals lie on the boundary of a rectangle and whose interconnection pattern is planar.

Output: whether or not the single side and corner blocks can be wired within the given rectangle.

1. Determine the single side and the corner blocks. For each block, specify the cover net.

2. Determine the wiring of all single side cover nets and all the corner cover nets by using algorithm Contour.

3. Check whether there is any intersection between the wires of the cover nets.

Lemma 11 *The above algorithm correctly determines whether or not the single side blocks and the corner blocks can be wired within the rectangle. The CREW PRAM complexity of the algorithm is* $O(\log n)$ *time with* $O(n)$ *processors, while its mesh complexity is* $O(\sqrt{n})$.

Proof: Since the blocks can be wired independently of each other, it is clear that a valid wiring exists if there is no intersection between the cover nets.

As for the running time, the first two steps are easy to implement. Step 3 can be implemented as follows. Suppose we want to check whether any cover net which has one or two terminals on the same side (say bottom side) intersects any other cover net. Sort the convex corners of the two vertical and upper sides and the vertical line segments of the bottom side by their x-coordinates. For each convex corner a, let $p(a)$ be the line segment whose x-coordinate is the closest to a from below and let $q(a)$ be the line segment closest to a from above. Based on this we can determine whether a is inside some wiring with at least one terminal on the bottom side. A similar strategy will work for all the other cases.

Once the block cover nets are wired, it should be checked whether there is enough space to route the remaining nets. The techniques developed below could be used to determine whether any wiring of two nets is legal (i.e. the corresponding wires donot intersect). Our approach consists of determining *the wiring capacity* and the *wiring density* between blocks. The wiring capacity between two blocks is the number of nets that can be wired between these two blocks, while the wiring density is the number of

wires that have to be wired between these two blocks. Determining the density is relatively easy and will be outlined in the algorithm below. The capacity between blocks on two orthogonal sides of the rectangle boundary is computed as follows. Given a block B consider all the convex corners of B. Generate 45 degree "rays" from each such corner and determine the line segment where it intersects another block contour or the original rectangle boundary. Based on this information, one can determine the width of the narrowest passage between B and any other block. The details are provided in the algorithm below.

Algorithm Intersection

Input: Contours of single side and corner blocks on two orthogonal sides of rectangle boundary.

Output: Intersection points of rays emanating from convex corners.

1. Consider the case of the the lower right corner. The other cases can be dealt with in a similar fashion. Sort all the line segments determined by the block contours and the right side of the rectangle R. Determine the projection of each line segment on the diagonal, say line segment i is projected into line segment $p(i)$ on the diagonal.

2. Sort the projections according to their order on the diagonal and compute $p'(i) = p(i) - \bigcup_{j=1}^{i-1} p(i)$.

3. For each ray y coming out of a corner of contour on the horizontal side of the original rectangle, find its intersection with the diagonal. If the intersection point lies in $p'(j)$, then ray y intersects segment j. Determine the intersection point of ray y and line segment j.

4. If a ray y intersects the original rectangle boundary, then rotate to find the intersection with the next line segment belonging to some block contour (see Figure 5 and ray y_B). Now determine the point of intersection.

For example, one can check that in Figure 5 $p'(CD) = C'D'$ and $p'(EF) = D'F'$. Hence rays y_A and y_B intersect CD and EF respectively. If we rotate y_B, we can find the intersection with the next line segment GF.

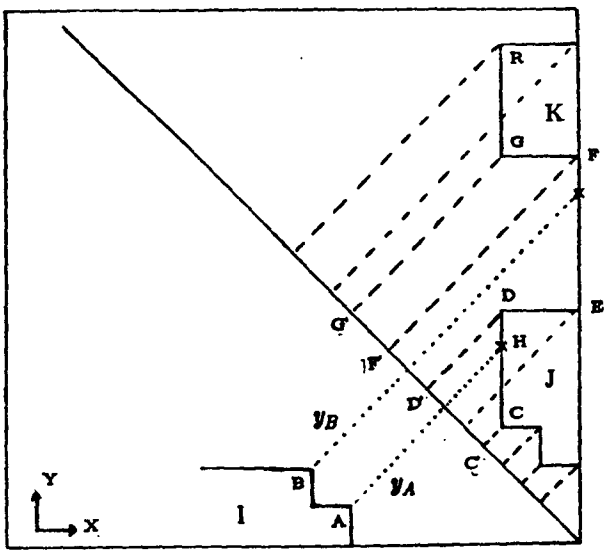

Figure 5: Intersection between rays and block contours

Lemma 12 *The above algorithm finds the intersection points of rays emanating from convex corners with the line segments of contours on two orthogonal sides of the bounding rectangle in time $O(\log n)$ time with $O(n)$ processors. Its mesh implementation takes time $O(\sqrt{n})$.*

Proof: Since the nets are on two orthogonal sides, it is clear that the line segment intersected by a ray can be determined by the projections on the diagonal. The running time of the algorithm can be easily verified.

Algorithm Density and Capacity

Input: The wiring of cover nets of the blocks and the remaining corner or cross nets.

Output: Determine whether or not there is enough space between the contours of the block cover nets on orthogonal sides to wire the remaining nets

1. Assume that the d-coordinates of all terminals are available. Cut the rectangle R at the left terminal A of a cover net such that $d(A)$ is mini-

mum and straighten it into a line. Let d' be the new coordinate system. Determine $d'(P)$ for all terminals P.

2. Assign weights to the terminals of each corner or cross net as follows: +1 to the terminal with smaller d'-coordinate, -1 otherwise. Order these terminals according to their d'-values and compute the *rank* of each terminal. Each block is assigned the *rank* of terminal adjacent to the left terminal of its cover net. The density between two adjacent blocks is equal to the difference in their ranks.

3. Use algorithm Intersection to compute the intersection point of each ray with a single side contour, corner block contour or the original boundary of the rectangle. The capacity between blocks can then be calculated easily.

Lemma 13 *Testing the routability of n nets between two orthogonal sides of a rectangle can be done in* $O(\log\ n)$ *time with* $O(n)$ *processors on the CREW PRAM model and in time* $O(\sqrt{n})$ *on the two-dimensional mesh.*

Proof: The correctness proof involves two steps. The first is to show that the density can be computed correctly at step 2 (easy) and the second step is to show that the capacity determined by each ray is enough to determine the routability of nets between orthogonal sides. We will concentrate on the latter step.

A typical case is shown in Figure 5 . If a ray emitted from the convex corner A intersects at point H of block J, then the capacity between A and J is given by $c_{AJ} = max\{|x_A - x_H|, |y_A - y_H|\}$. If $c_{AJ} \geq d_{AJ}$ (density between A and block J), then there is enough space between A and H to route the cross nets. Now consider any convex corner R of a block (say K) above H in the right side of the rectangle. Then $c_{AR} = c_{AH} + |y_H - y_R|$. But $d_{AR} \leq d_{AH} + d_{HR}$ and hence $d_{AR} \leq c_{AR}$ since $d_{HR} \leq c_{HR} \leq |y_H - y_R|$. It follows that if it is routable between I and J, then it is routable between I and K, for any K above J. The other cases can be treated similarly.

The running times can be verified easily.

We now address the routability problem between two opposite sides of the bounding rectangle. It seems that the generation of horizontal, vertical

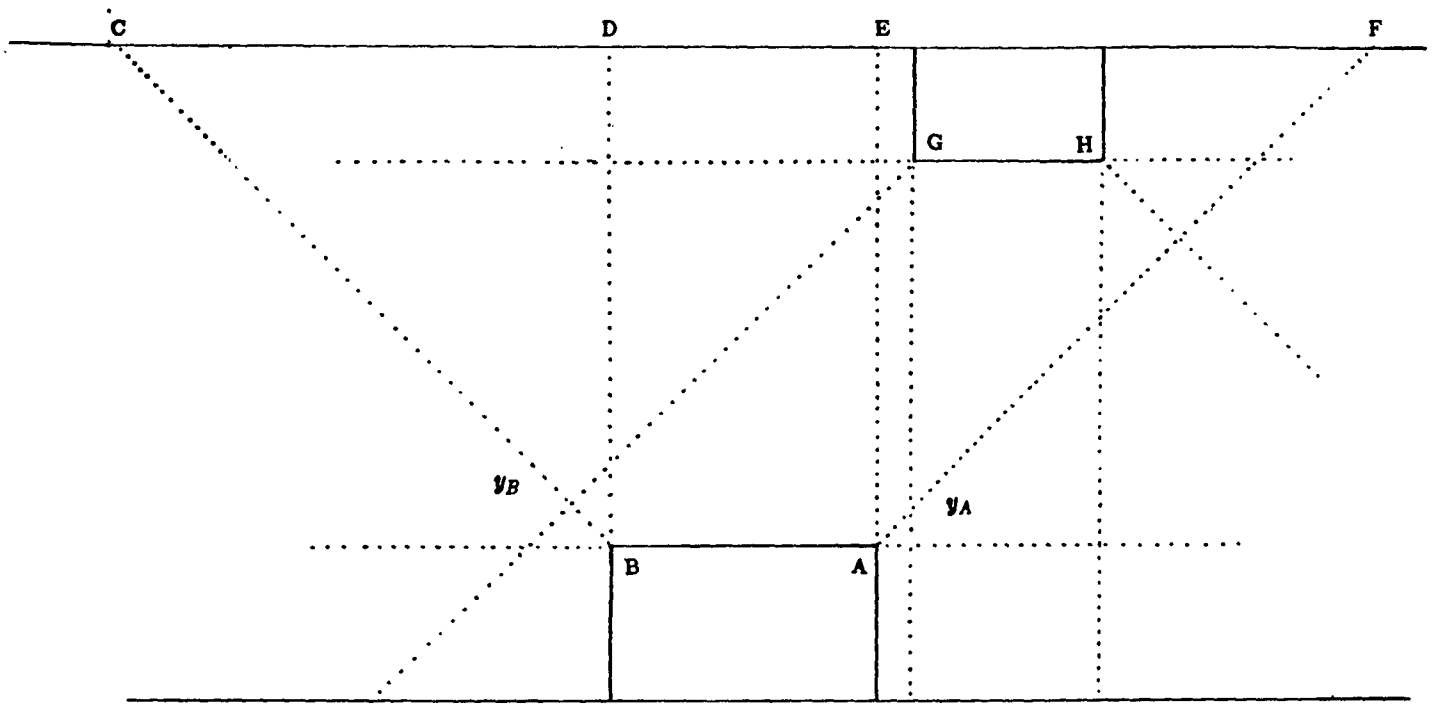

Figure 6: Routability testing between two opposite blocks

and 45 degree rays from each convex corner is not enough to determine the routability between two opposite sides. Consider the case shown in Figure 6. Let k be the number of nets with one terminal to the right of ray y_A and with the other terminal to the left of ray y_B. These k nets may be blocked by blocks between C and D and blocks between E and F. However determining the capacity induced by 3 rays from each corner will not necessarily reveal this. We will use a divide-and-conquer strategy to handle this case.

For any two blocks I and J on opposite sides, let R and S be two adjacent nets between I and J. We can test the routability between I and J as follows. Using the detailed routing algorithm, find the wiring of R (not necessarily final) as close to I as possible, and the wiring of S as close to J as possible. Check to find out whether there is any intersection between

the two wires. Apply the algorithm recursively to the wires to the left of R , and the wires to the right of S (see Figure 7). In order to make the procedure efficient, the numbers of nets to the left of R and of nets to the right of S are almost equal. The details are given in the algorithm below.

Algorithm Routability Testing

Input: Contours of single side and corner blocks, and the remaining nets.

Output: Whether or not the nets are routable.

1. Use algorithm Density and Capacity to test whether there is enough space between any two block cover nets on orthogonal sides to wire corner nets.

2. Partition corner nets into groups such that each group is a maximal set of corner nets with the property that exactly one of its members covers all the remaining members. These groups could be identified by a technique similar to the one used in algorithm Groups and Representatives.

3. Determine the wiring of each of the cover nets introduced in step 2. Check whether there is any two such wires intersect. In the affirmative, the nets are not routable.

4. If there are no cross nets, then *exit* (routability is feasible). Else, assume without loss of generality that all cross nets are between the top and bottom sides. Select two adjacent cross nets N_i and N_j that split the nets almost evenly. Let N_i be to the left of N_j.

5. Find the temporary wiring of N_i as close to the left as possible and the temporary wiring of N_j as close to the right as possible. Check whether any intersection will result.

6. Repeat steps 4-6 for the cross nets to the left of N_i and for the cross nets to the right of N_j separately.

Theorem 4 *Testing the routability of n nets within a simple rectilinear polygon could be done in $O(\log^2 n)$ time with $O(n)$ processors on the CREW PRAM model and in time $O(\sqrt{n})$ on the two-dimensional mesh.*

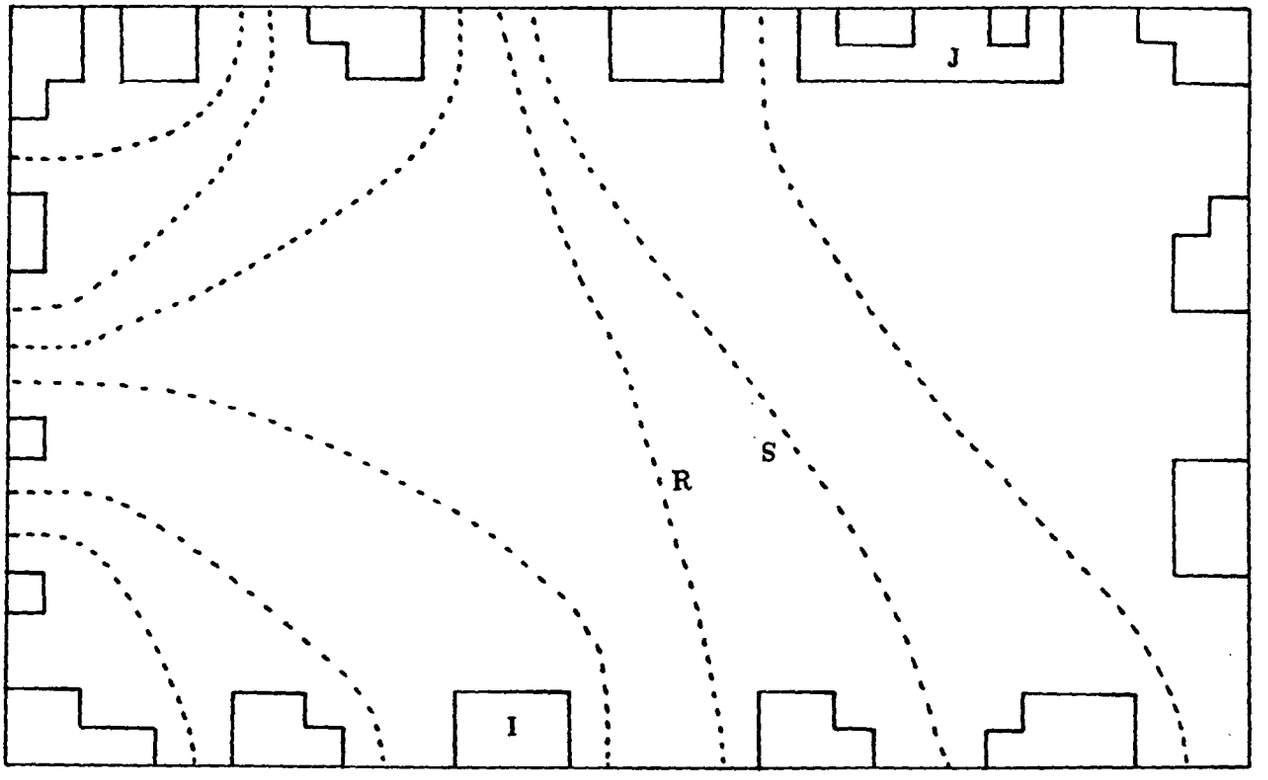

Figure 7: Routability between two blocks in opposite sides

Proof: It is enough to show that the wiring of the cross nets is tested properly.

The running time of the PRAM model can be easily checked. The running time of the mesh algorithm satisfies the following recurrence:

$$T(n) = T(\frac{n}{2}) + O(\sqrt{n})$$

$$T(1) = O(1)$$

and therefore the theorem follows.

6 References

[AH] M. Atallah and S. Hambrusch, " Solving tree problems on a mesh-connected processor array," Proceedings of the 26th Symp. FOCS, 1985, pp. 222-231.

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

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

[KL] M. Kramer and J. van Leeuwen, "Wire routing is NP-complete," technical report, University of Utrecht, the Netherlands, February 1982.

[L] A. LaPaugh, "Algorithms for integrated circuit layout: an analytic approach," Ph.D. dissertation, MIT, Cambridge, MA, November 1980.

[LP] C. Leieserson and R. Pinter, "Optimal placement for river routing," SICOMP 12(3), August 1983, pp. 447-462.

[P] R. Pinter, "River routing: methodology and analysis," Proceedings of the third CALTECH Conference on Very Large Scale Integration, March 1983, pp. 141-163.

[SB] S. Sahni and A. Bhatt, "Complexity of the Design Automation Problem," Proceedings of the 17th Design Automation Conference, June 1980, pp. 402-411.

[SD] A. Seigel and D. Dolev, "The separation for general single layer wiring barriers," Proceedings of the CMU Conference on VLSI Systems and Computations, October 1981, pp. 143-152.

[S] T. Szymanski, "Dogleg Channel routing is NP-complete," manuscript, Bell Laboratories, Murray Hill, NJ, September 1981.

[T] M. Tompa, "An optimal solution to a wire routing problem," Proceedings of the 12th Annual Symposuim on Theory of Computing, April 1980, pp. 161-176.