プログラミング－言語・基礎・実践－ 13－10
（１９９３．８．２０）

並列単層配線を改良する新しい手法の提案

ケシェク ヘシャム、森眞一郎、中島浩、富田眞治
京都大学工学部

内容梗概

自動配線とは、ネットワーク上の複数個のピンをつなぐパスを見つける手法である。その際、新しいパスはすでに配線されているパスと交差してはならない。自動配線における基本的な問題点は、計算時間が長いこと、必要なメモリの量が多いことである。最近になって、並列計算機を用いて配線問題の高速化を図る研究がいくつか発表されている。我々は迷路法に基づく２つの並列アルゴリズムを考案した。本アルゴリズムの特徴は，分割された配線領域を跨ぐ短い配線が未配線のまま，ルーティングが進行するのを避けるため，境界線が移動するような領域分割を行う点にある．これにより、配線領域の大きさに比べてネットの配線長が短いような場合、特に高い並列性が得られる．本稿では，これらの手法とその実行結果を他のいくつかのアルゴリズムと比較して示す．

A new technique to improve parallel automated single layer wire routing.

Hesham KESHK, Shin-ichiro MORI,
Hiroshi NAKASHIMA,and Shinji TOMITA

Department of Information Science
Faculty of Engineering, Kyoto University
Yoshida-hon-machi, Sakyo-ku, Kyoto 606-01 Japan
E-mail:{keshk,moris,nakasima,tomita}@ kuis.kyoto-u.ac.jp

Abstract

Automated wire routing is to find a path between two or more network pins and this path must not intersect with previous drawn paths. The basic problems of automated wire routing are the long computation time and large memory size required. Recently, several researches tried to speed up the routing problem by using parallel computers. We develop two parallel algorithms based on maze running algorithm. Both of them have a new technique in dividing a single layer grid. These two algorithms give high speed specially if the net lengths are small with respect to grid dimensions. In the first algorithm, moving boundaries are used in dividing the grid. The second algorithm can be divided into two phases. In the first phase, we rotate the areas assigned to the processors to route all short nets. In the second phase we use the competing processors algorithm to route the remaining nets.

1. Introduction

Automated wire routing is one phase of CAD. The basic problems of automated wire routing are the long computation time and large memory size required. Although Lee's algorithm [1] which is known as Maze running was the first algorithm to solve the automated wire routing problem (1961), it is still used until now because it has a high degree of flexibility. There are many researches [2]-[7] which modified the original maze running to make it faster, but the improvement of the speed was not satisfactory. Some other researches[8][9] tried to improve the speed by using hardware implementation of maze running or line search.

Recently, several researches tried to speed up the routing problem by using parallel computers. Two of these trials are made by prof. Takahashi[10][11]. In the first one, they developed a parallel line search algorithm based on divide and conquer strategy and implemented it on a binary tree parallel computer called Coral 68. But the efficiency of the parallel computation was not satisfactory. In the second trial, they developed another parallel routing algorithm, in which a number of processors compete each other to route different nets independent of each other, and one master processor inspects and verifies the results.

Based on these two algorithms, we develop another two algorithms to overcome the disadvantages in their algorithms and to speed up the computation time. Our algorithms have higher degree of parallelism inside and it can be run faster. We implement our two algorithms and the other two algorithms mentioned in[10][11] on a message passing MIMD parallel computer called AP1000[13]. In this paper, our two algorithms are introduced. The results and a comparison are also included. Our algorithms can also be used to speed up the computational time for some other algorithms which use some techniques like rip-up and reroute to get high connection ratio.

2. Maze algorithm.

Automated wire routing is to find a path between two or more network pins and this path must not intersect with previously drawn paths. Maze[1] and line search algorithms are two famous algorithms for printed circuit boards. Maze running guarantees to find the shortest path between two points if one exists, but on the other hand, it takes large memory space and long time to be implemented. The time in the worst case is proportional to N^2, where N is the grid dimension. Line search is another algorithm which may run faster and need less memory than maze but it does not guarantee to find a path even if one exists. Line search also does not find the shortest path but it finds a path with minimum number of bends. In fact, line search runs faster than Maze only when number of obstructions is small. But if the grid is so congested, line search may not improve the speed.

Maze algorithm consists of three major phases: wave propagation (wavefront expansion), backtrace, and label clearance. Firstly, the grid will be divided to squares as shown in fig 1a, where each square can be occupied by only one net. Now, to find a path between two points (A,B), we start from any point as a source (point A in this example) and search for the target (point B). The black squares are obstacles which may be another networks which were already connected before. In the wave propagation phase we start by entering 1 in each empty square adjacent to point A, after that, enter 2 to each empty square adjacent to the squares which were already labeled by 1, and so on (fig 1b) until find point B or can not find any empty squares adjacent to the squares which are already labeled by m. In the back trace phase, if point B was already labeled by k, then from point B we look for a square which is labeled by k-1, and from this square look for another square which is labeled by k-2, and so on until find point A as shown in fig 1c. Note that we must not change direction unless it is necessary. In the label clearance phase, all the squares except those used for the drawn path are cleared.

In our algorithms, we make the wave propagation from the two points simultaneously to decrease the wave propagation area as shown in fig 1d. And this of course decreases the computation time.

3. Parallel routing algorithm with moving boundaries.

This algorithm could be considered as a modification of the hierarchical domain algorithm [10]. At first, the host computer broadcasts a list of all net terminal locations and the grid dimensions to all PEs. The grid area is divided by the total number of PEs. For example, if the grid dimensions is 256*256 and total number of PEs is 64 (consider it as 8*8 PEs), then the area assigned to each PE is 32*32 (as shown in fig 2 step 1). Every PE selects the nets which lie inside its assigned area from the list of all nets, sorts them in an ascending order according to their lengths, and routes them one after the other within its assigned area. There is no possibility of any confusion between different PEs. After finishing this step, the number of working PEs is decreased to 49 PEs(7*7) (as shown

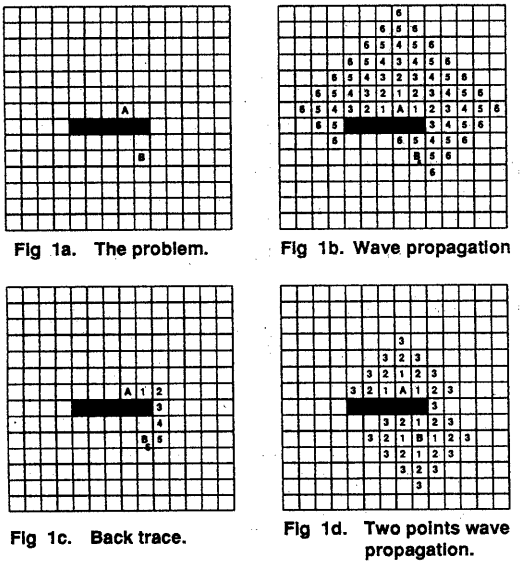

Figure 1: Maze running algorithm.

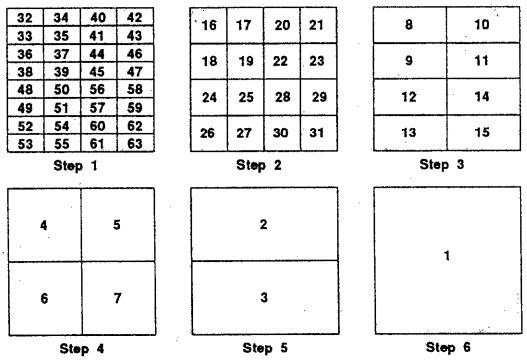

Figure 3: Hierarchically partitioned algorithm[10].

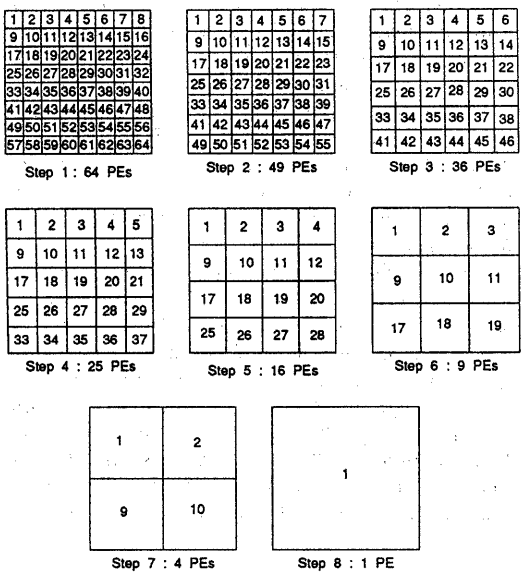

Figure 2: Grid partition by moving boundaries algorithm.

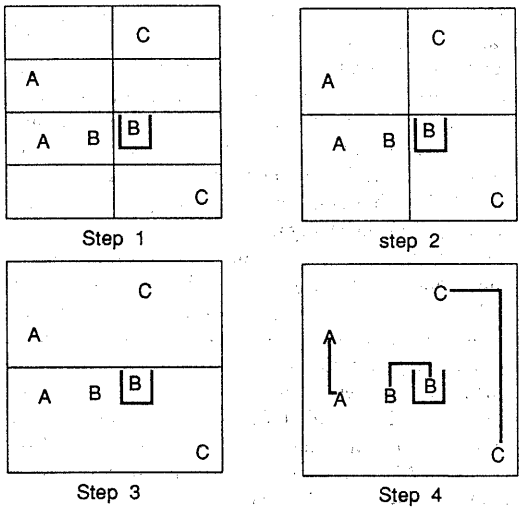

Figure 4: Routing nets in hierarchically partitioned algorithms.

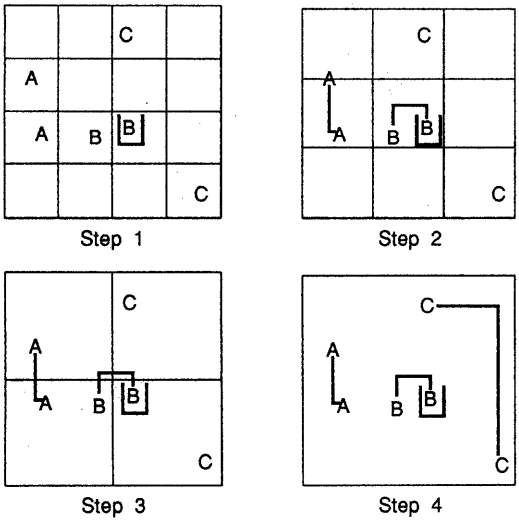

Figure 5: Routing nets in moving boundaries algorithm.

Table 1: Grid partition of hierarchically partitioned algorithm [10]

	PEs	area	x boundaries	y boundaries
step 1	32	64*32	64,128,192	32,64,96,128 160,192,224
step 2	16	64*64	64,128,192	64,128,192
step 3	8	128*64	128	64,128,192
step 4	4	128*128	128	128
step 5	2	256*128		128
step 6	1	256*256		

Table 2: Grid partition of moving boundaries algorithm

	PEs	area	x boundaries	y boundaries
step 1	64	32*32	32,64,96,128, 160,192,224	32,64,96,128 160,192,224
step 2	49	36*36	36,73,109 146,182,219	36,192,224 146,182,219
step 3	36	42*42	42,85,128 170,213	42,85,128 170,213
step 4	25	51*51	51,102,153, 204	51,102,153, 204
step 5	16	64*64	64,128,192	64,128,192
step 6	9	85*85	85,170	85,170
step 7	4	128*128	128	128
step 8	1	256*256		

in fig 2 step 2). The boundaries of every PE area are different from the first step. Every PE leaves part of its area and takes another part. So every PE obtains the new boundaries, sends the leaving part of the bit map, which contains the nets already connected, to its left and up neighborhood PEs, receives the taking part of the bit map from its right and down neighborhood PEs, and adds the taking part of the bit map to the remainder part of its old bit map to obtain the new bitmap which is 36*36 or 37*37 in this example. Then every PE from the working PEs selects again the unrouted nets which lie in its new area, sorts and routes them. Then the number of working PEs will be decreased to 36 PEs(6*6)(as shown in fig 2 step 3). PEs will again obtain the new boundaries, send and receive data, select, sort,and route the unrouted nets. And so on until the last step where one PE will be responsible about all the grid. Of course we can jump some steps, so we can make the steps as 8*8 → 7*7 → 5*5 → 3*3 → 2*2 → 1*1, or we can jump more steps or add some other steps.

The major difference between moving boundary algorithm and the hierarchical domain algorithm[10] is that the boundaries are moved in our algorithm while they are fixed in the other algorithm as shown in fig 3. For example, if we have 256 * 256 grid and 64 PEs, then the boundary for different steps will be as shown in table 1 and 2. We can see that in the hierarchical domain partition algorithm[10] the boundary 128 exists from the beginning until the last step. Then at the last step one PE will be responsible about routing the long nets (as in fig 4 type C) beside short nets which lie around this boundary (nets type A and B shown in fig 4). While in the moving boundary algorithm the short nets can be routed by more than one PE in some previous steps as shown in fig. 5 (both nets A and B are connected in step 2). In another words, the moving boundary algorithm has an advantage that no short nets will remain without routing until the end. So, the last steps, where few PEs are in work, will be aimed to route long nets only. While, in the hierarchical domain algorithm, short nets around the boundaries will remain without routing until the end.

This algorithm can get good results if the net lengths are short compared with the grid dimensions because most of the nets will be connected in the first steps where most of PEs are in work, while small number of long nets will stay without routing until the end. On the other hand, this algorithm will not give good results if the nets lengths are long compared with the grid dimensions. This is because only few PEs will be used to route a large number of long nets which require longer time for routing.

4. Parallel routing algorithm with rotating areas.

This algorithm can be divided into two major phases:

The first phase is aimed to route short nets. If the grid dimensions are L*L, and the number

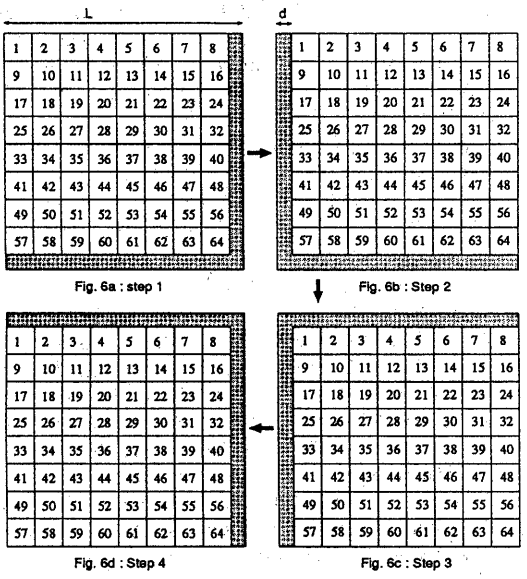

Figure 6: The first phase of algorithm 2.

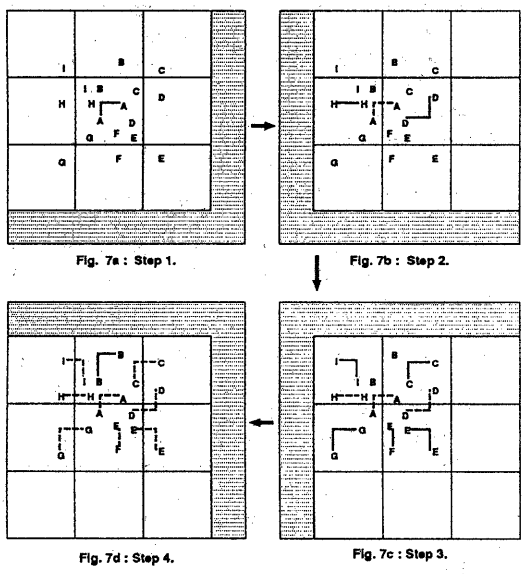

Figure 7: Example of routing at the first phase.

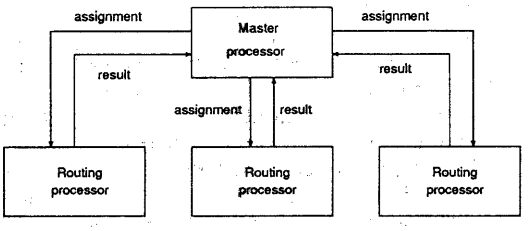

Figure 8: The second phase of algorithm 2.

of PEs is N^2, then this phase guarantees to deal with all nets which have lengths less than d, where d=$L/2(N+0.5)$. This phase may also deal with nets which have lengths greater than d. In this phase, The host computer broadcasts a list of all net terminals location and the grid dimension to all PEs as in the moving boundaries algorithm. The grid is divided also by the total number of PEs, but here, we do not divide all the grid. There is a part of the grid which is not assigned to any PE as shown by the shaded area in fig 6a. So, every PE is responsible for an area of $(L/(N+0.5))^2$ only (instead of $(L/N)^2$ in the moving boundary algorithm). Every PE selects the nets which lie in its assigned area from the list of all nets, sorts them in an ascending order according to their lengths, and routes them one after the other within its assigned area. In the beginning, all the nets in the net list are labeled by 0 which mean that no PE try to route this net yet. If both the net terminals are located in a PE area, then this PE tries to find a path for this net. The net will be labeled by 1 if the PE succeeds to find a path, and it will be labeled by 2 or 3 if the path can not be found at all or not found but may be found later because this net is near a boundary (like net B in fig 4). After finishing this step. The shaded area is changed, and the area of each PE is moved as shown in fig 6b. Every PE will send half of its bit map and the net list to the left neighborhood and receive another half bit map and net list from the right neighborhood. Every PE selects again nets (labeled by 0 or 3) which lie in its area, sorts and routes them. The shaded area will be changed again as shown in fig 6c and 6d. At the end of this phase, all the PEs send their bit map and the net list to the host computer.

By this complete rotation, we ensure that all short nets can be connected regardless of their locations. Fig 7 shows this property. In step 1, net A only is routed while all the other nets still without routing. In step 2, both nets D and H could be routed while nets C,E,G and I will be routed in step 3. Net B will be routed in step 4.

In the second phase of this algorithm, we use the parallel routing algorithm with competing processors[11]. In this algorithm, one processor or the host computer is used as a master and all the other PEs are used for routing(fig 8). The master PE broadcasts the bit map of the whole grid to all the PEs and assigns a net (which has label 0 or 3) from the net list to each PE which tries to find a path for it. The computed path is sent to the master PE which verifies if it intersects with any other previously accepted paths. If it intersects then the master PE will send it again to the sender PE with the latest accepted paths. But if it does not intersect then the master PE labels this net by 1, adds its path to the list of accepted paths, sends the latest accepted paths to the sender PE, and assigns another net to the sender PE.

By using two phases in this algorithm, we can connect all short nets in the first phase using few messages, while only long nets are connected in the second phase. This has two main advantages comparative with competing processors algorithm[11] which uses only the second phase to route both short and long nets

1. If we use the second phase to route short nets which require short routing time. Many messages will arrive to the master PE at the same time. Then the master PE will not be able to serve all the routing PEs on time especially if the number of PEs is large. The routing PEs have to wait long time for the master PE. But this will not happen for long nets which required longer time to be routed.

2. It decreases total number of messages from 2*(number of nets + number of reroute nets) to 2*(number of long nets + number of reroute nets) and of course the number of reroute nets will be decreased also.

The first phase can be used alone under the restriction that the maximum net length is small as compared with the grid dimensions. For example, if the maximum net length is shorter than 500 in a grid dimension 10000*10000 and the number of PEs is 64, then every PE will be responsible about area of 1176*1176 only. By this, we can decrease the required memory for each PE from 10000*10000 to 1176*1176. But, on the other hand, this may prevent routing some nets (label 3) which may be routed when using the second phase.

5. Results:

Several experiments have been done and many results have been taken to see the performance of the algorithms. Now, we will show some of these results with discussion. These results were taken on a grid dimensions 1000*1000 using random nets as input to our program. The random nets generator program output the x and y coordinates of the net terminals. The manhattan lengths of the nets obey the Rayleigh probability function.

We executed our program on a parallel computer called AP1000[13]. The AP1000 is a MIMD, distributed memory, message passing, torus network computer. It consists of 16 to 1024 processing elements (64 only in our system) and three independent communications networks.

We implement the two algorithms of prof. Takahashi together with our new two algorithms to compare their performances. The algorithms (when implemented on a grid dimensions 256*256) give connection ratios which are approximately same as the connection ratios obtained in [11]. From now, for simplicity, we will use the following notations:
HPA : the hierarchically partitioned algorithm[10] but with a small modification that we use all the PEs in the first step instead of half of them.
CPA : the competing processors algorithm[11].
MBA : the moving boundaries algorithm.
RAA : the rotating areas algorithm.

The average manhattan distance of the nets has a big effect on both HPA and MBA. That is because if the average distance is small then most of the nets will be routed in the first steps using most of the PEs, while if the average distance is large, then most of the nets will be routed in the last steps using less number of PEs which decreases the parallelism and increases the time. This average distance has less effect on both CPA and RAA as shown in figure 9. In this figure, we measure the routing time for different average manhattan distance for 4000 nets. As we can see MBA is faster than HPA for all lengths approximately and also MBA is faster than CPA for short average distance less than 70.

In order to know the effect of changing the number of nets on the different algorithms, we measure the time for different nets number with an average manhattan distance of 20. As we can see nets number has a big effect on CPA because it increases the number of messages. And if the average distance is small as in this case, then the routing PEs will route nets in a short time which will make overhead at the master PE because many messages will arrive at the same time. Increasing nets number does not increase the time so much in HPA, MBA and RAA because most of these nets will be routed at first step by all the PEs and the time for communication

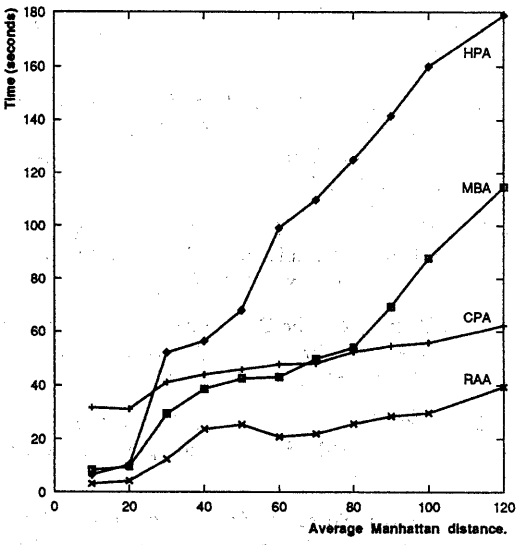

Figure 9: Time against average distance for 4000 nets.

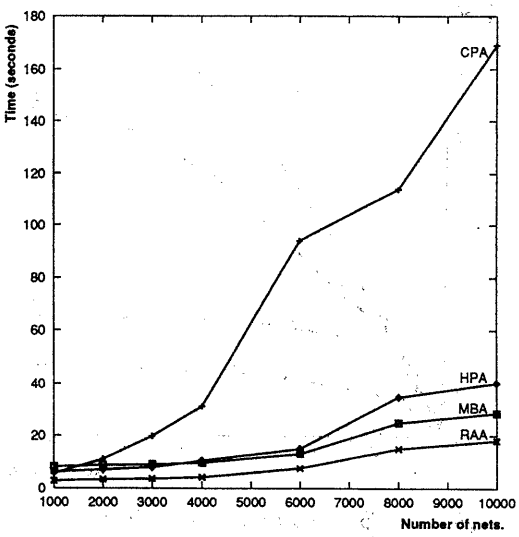

Figure 10: Time against nets number for average distance equal 20.

independent on nets number. Figure 10 shows time against nets number for average distance equal 20.

In figure 11 we change the average Manhattan distance but we keep the total nets lengths (average distance * nets number) constant equal to 200,000. For example, if the average distance is 10 then the nets number will be 200,000/10 =20,000 and so on. As we could see CPA takes a very long time for routing large nets number with small average distance while it will be better for longer nets distance with less nets number.

Figure 12 shows the speed up of the different algorithms when routing 8000 nets with average distance 20. The speed up is calculated by dividing the routing time using only one PE (sequential program) by the the routing time using different number of PEs.

6. Conclusion.

In this paper, two parallel routing algorithms for dividing a single layer grid has been introduced. Both of these algorithms are focused to decrease the routing time especially for short net lengths. In the first algorithm, the whole grid area is firstly divided to N partitions and each of these partitions is assigned to a processor to route the nets in parallel. After that the whole grid is re-divided to M partitions such that each new boundary is different from the previous boundaries of the N partitions. The second algorithm consists of 2 phases. In the

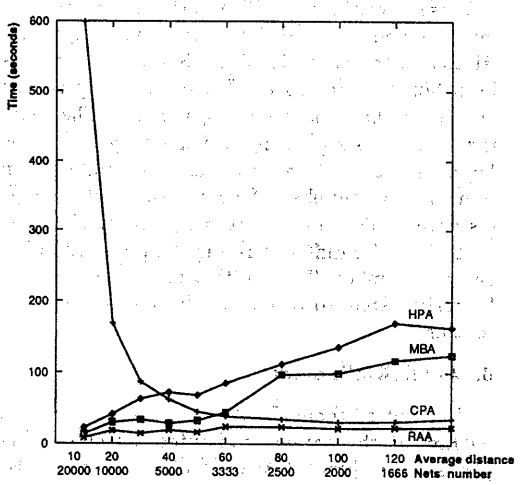

Figure 11: Time against nets number and average distance with fixed total length.

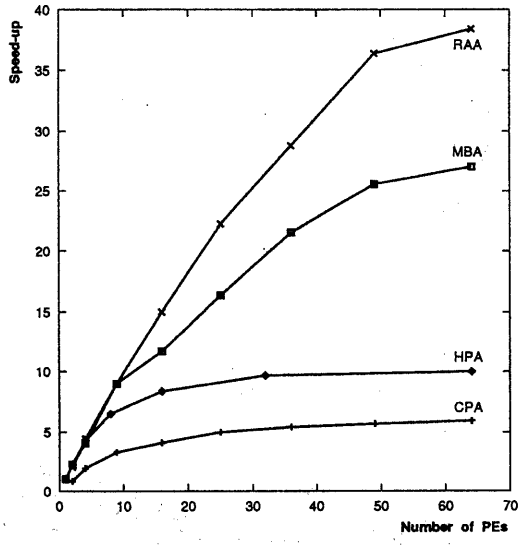

Figure 12: Speed up against number of used PEs for 8000 nets with average distance 20.

first phase, we exploit the parallelism in the area by assigning a partition to each processor then we move every partition to route all the nets shorter than a certain length. In the second phase, we exploit the parallelism in the number of nets using the competing processors algorithm.

These algorithms beside two other algorithms are implemented on a MIMD message passing parallel computer called AP1000. Several results has been taken which proved that our new algorithms are faster than the other algorithms. The moving boundaries algorithm, which is very easy to be implemented, gives a good result specially when routing large number of nets with short average distance. The rotating areas algorithm gives the best results for different average lengths and different nets number. The rotating area algorithm can be easily extended to be used in multi-layers wire routing.

6. Acknowledgment.

We want to thank Fujitsu Laboratories Ltd for offering us the parallel computer AP1000 to implement our parallel algorithms. We want also to thank all our laboratory members specially Mr. Matsumoto for their great help.

7. References

1. C.Y.Lee "An Algorithm for Path Connections and Its Applications", IRE Trans. on Electronic computers, vol. EC-10, pp. 346 - 365, 1961.

2. S.B.Akers,"A Modification of Lee's path Connection Algorithms", IEEE Trans. on Electronic Computers (short notes), vol. EC-16, pp. 97-98, 1967.

3. J.M.Geyer,"Connection Routing Algorithm for Printed Circuit Boards", IEEE Trans. on Circuit Theory, vol. CT-18, pp. 95-100, 1971.

4. L.C.Abel,"On the Ordering of Connections for Automatic Wire Routing", IEEE Trans. on Comput. vol. C-21, pp. 1227-1233, 1972.

5. F. Rubin,"The Lee Path Connection Algorithm", IEEE Trans. on comput. vol. C-23, pp. 907-914, 1974.

6. J.H.Hoel,"Some Variations of Lee's Algorithm",IEEE Trans. on comput. vol. C-25, pp. 19-24, 1976.

7. J. Soukup,"Fast Maze Router",Proc. 15th Design Automation Conf. pp. 100-102, 1978.

8. K.Suzuki,"A Hardware Maze Router with Application to Interactive Rip-Up and Reroute", IEEE Trans. on Computer A ided Design, vol. CAD-5, pp. 466-476, 1986.

9. K.Suzuki,"A Gridless Router: Software and Hardware Implementation", VLSI 87.

v

10. Y.Takahashi, "Parallel Maze Running and Line Search Algorithms for LSI CAD on Binary-tree Multiprocessor", Word Conference on Information/Communication, Seoul, pp. 128-136, June 1989.

11. Y. Takahasi,"Parallel Automated Wire Routing with a Number of Competing processors", International Conference on Supercomputing,ACM Vol.18,Number 3, pp. 310-317, 1990.

12. T.Yamauchi,"PROTON: A Parallel Detailed Router on an MIMD Parallel Machine" ICCAD-91 pp 340-343, 91.

13. H.Ishihata,"Third Generation Message Passing Computer AP1000" International Symposium on Supercomputer, pp.46-55, 1991.