A New Conflict Resolving Switchbox Router

Tae Won Cho, Sam S. Pyo† and *J. Robert Heath*

Department of Electrical Engineering
University of Kentucky
Lexington, KY 40506

Abstract

A parallel algorithm, called PARALLEX, which uses a conflict resolving method, has been developed for the switchbox routing problem in the parallel processing environment. PARALLEX can achieve a very high degree of parallelism by generating as many processes as nets. Each process is assigned to route a net, which bears the same identification number as the process. If any conflict is found for the current route of a net, then each process classifies the conflict segments with groups by their relations. Each process finds partial solutions, and merges them with the partial solutions from other processes to make a conflict-free switchbox. The speed-ups for 7-nets and 19-nets problem were 4.7 and 10 respectively.

1. Introduction

This article introduces a parallel algorithm, called PARALLEX, for switchbox routing in the shared-memory multiprocessor environment. A switchbox is a rectangular region with terminals on all four sides. Two or more layers of interconnections are available. There may be obstacles in the routing area. A net has two or more terminals. The problem is to route all of the nets inside the routing region without causing conflicts.

Many algorithms on channel routing problems have appeared recently [8,14,20,25]. Several switchbox routing were actively pursued since Burstein presented a hierarchical routing algorithm in 1983 [1]. Cohoon and Heck reviewed switchbox routing algorithms extensively [3]. Most of the researchers use sequential approaches [5,7,13,16,22]. Gerez and Herrmann [7] developed PACKER, a switchbox routing algorithm based on stepwise reshaping. PACKER uses an iterative method to rip-up and reroute with a new data structure. However, it may give different solutions, depending on the order of the scanning direction. Also, parallelism issues are not considered in the algorithm. Our algorithm uses a method similar to PACKER in segment (or fragment) notation and the initial routing of individual nets, but uses different data structures and a different tree search algorithm that easily enables parallelism. PARALLEX finds near-optimal solutions in a local conflict area using Lee's algorithm [15]. Local solutions for different conflict groups are compared and merged to make the best solution without any order dependencies.

Concerning parallelization efforts in routing, mostly small-grain parallelism was sought in the past. There have been many efforts to parallelize the Lee routing algorithm for array processors and special-purpose parallel machines [12,17,23,24]. Recently, several algorithms have been proposed that exploit large-grain parallelism [6], so that they can be implemented on general-purpose multiprocessor systems. Rose [21] presented a parallel global routing algorithm for standard cells, where a net is assigned to a processor for global routing. Zargham [25] proposed a parallel channel routing algorithm in which a channel is divided into multiple regions and each region is assigned to a processor for detailed routing. Rose's scheme offers a large degree of parallelism, since the upper bound is limited only by the number of nets to be routed. On the other hand, Zargham's scheme does not allow a large degree of parallelism, since a channel can be divided into only a limited number of regions. Unfortunately, Rose did not attempt to apply his method to detailed routing. Our algorithm is based on

† Samsung Advanced Institute of Technology, Computer & Communications Research Center, Korea

0-8186-2610-0/92 $3.00 © 1992 IEEE

the above observation and is an attempt to apply Rose's idea to detailed routing, more specifically, to the switchbox routing problem.

The basic idea of our algorithm is to generate as many parallel processes as nets. This enables us to achieve a high degree of parallelism. Each process is responsible for routing the assigned net. To maintain a high degree of parallelism during the routing process and to minimize the performance penalty, synchronization of processes that have routing conflicts is avoided as much as possible. Instead, parallel processes work independently of each other. This is called an asynchronous parallel algorithm [19]. It is made possible by using a special data structure describing the routing status.

The rest of the paper is organized as follows. In Section 2, related works are reviewed. In Section 3, the switchbox routing problem is defined and its performance criteria are discussed. Section 4 describes the PARALLEX algorithm. Concluding remarks are given in the last section.

2. Related work

Gerez and Herrmann [7] presented PACKER, a stepwise reshaping algorithm which utilizes an iterative rip-up and reroute method. PACKER uses a scan-line technique: all segments under the scan line are considered at the same time and a conflict-free subset of the segment is selected to remain; the rest must move to the next scan-line position. Scanning in different directions (from left to right, right to left, top to bottom, and bottom to top) continues until all conflicts have been resolved or a certain time-out condition is met. They did not consider parallelization of the algorithm.

Zargham [25] developed a parallel routing algorithm that is suitable for implementation in the shared-memory multiprocessor environment. Initially, all of the processors may be viewed as being in a "global pool." Then, a processor will leave the pool, producing a "pool of work." A work is considered to be the assignment of tracks to the nets which are passing through a column. The assignment is based on the global information about the channel (which is known in the beginning) and/or the tracks which already have been assigned to one of the neighboring columns. These works are independent of each other. That is, once a processor is assigned to a work, it does not need to communicate with other processors. His router achieved measured speed-ups of 2.1 using 3 processors, and 2.7 using 6 processors. Zargham's scheme does not allow a large degree of parallelism, since a channel can be divided into only a limited number of regions.

Rose's parallel algorithm [21] uses three orthogonal parallel decompositions: 1) wire-by-wire parallelism, 2) segment-based parallelism, 3) route-based parallelism. In *wire-by-wire* parallelism, each processor is given an entire multipoint wire to route. The master processor initializes the cost array, and sets up each individual wire as a task on one central task queue. All of the processors then remove wire tasks from the queue and perform the routing, read the shared cost array to evaluate the routes for each wire, and then update the cost array with the best route that is found. In *segment-based* parallelism, each two-point segment of a wire (produced by the minimum spanning tree decomposition) is given to a different processor to perform the routing. In *route-based* parallelism, all of the two-bend routes to be evaluated are divided among the processors. Each finds the lowest-cost path among the set of routes it is assigned. When all of the processors finish, the route with the best overall cost is selected.

His approach achieved measured speed-ups from 5 to 14 using 15 processors, and 6 speed-ups using 10 processors. His algorithm does not guarantee that the same solution will be found every time it is executed. The solution depends on the order of execution of the individual processors. Also, Rose did not attempt to apply his method to detailed routing.

3. Problem statement

In this paper, Luk's terminology [16] is used as otherwise mentioned. Terminals are located on the four boundaries of the routing region. All of the terminals that bear the same net ID constitute a net that should eventually be connected together. The problem is finding a solution for connecting all of the terminals that belong to the same net within the given routing region without causing any conflict. A conflict exists when a grid point is occupied by more than one net. It is assumed that two layers are available, one for vertical wires (or tracks) and the other for horizontal wires. Connection is made by a wire which is allowed to run either horizontally or vertically. A connection between a horizontal wire and a vertical wire is made through a *via* (or *contact*). A grid point, either horizontal or vertical, can be occupied at most by a single wire segment.

For routing performance, the *speed-up* is the primary concern for PARALLEX, since most parallel algorithms are usually measured in terms of speed-ups obtainable.

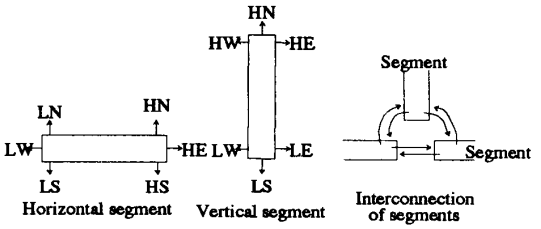

Figure 1. Horizontal/vertical segment and interconnection of segments.

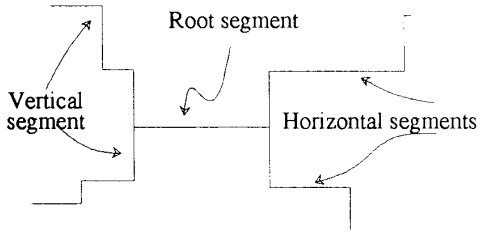

Figure 2. A net segment tree with root segment.

4. The PARALLEX algorithm

4.1. Representation of a route

The representation scheme of a routed net is shown by two types of segments: *horizontal-wire-segment* and *vertical-wire-segment*. Each segment has eight possible directions in which other segments at both ends can be connected. HN, HE, HS and HW are for one end (called high end) of the segment, and LN, LE, LS and LW are for the other end (called low end) of the segment. A segment is represented by five tuples: net ID, Is_horizontal_or_vertical, row (column) index for horizontal (vertical) segment, column (row) indices of the high end, and the low end for horizontal (vertical) segment (or row numbers for vertical segment).

Connections between segments are made by a pair of pointers as shown in Figure 1. Whenever two segments are connected, there implicitly is a via or a contact cut. A segment represents an interconnection between two vias.

4.2. Data structures

For managing the horizontal/vertical segments and detecting conflicts easily, segments are stored in three data structures.

1) *Net Segment Tree* (NST): An NST stores information of the vertical and/or horizontal segments of a net in a tree structure as shown in Figure 2. Those segments are found by the Rectilinear Steiner Tree [8,11,18]. An NST shows physical connections of a net. Furthermore, two adjacent segments in an NST are doubly-linked, and an NST can be reached from any segment in the tree. Thus, this net segment tree structure is suitable for manipulating the changes obtained from the path-finding procedure by using `MOVE`, `FIND`, `DELETE` and `MERGE` operations.

2) *Horizontal Segment Tree* (HST): An HST stores information on the horizontal segments of all of the nets. For each row, horizontal segments located in the row are stored in an HST and sorted by the non-decreasing order of the left end. If the left ends of two or more segments are equal, they are sorted by the non-decreasing order of the right end.

3) *Vertical Segment Tree* (VST): Similarly, a VST stores information on the vertical segments of all of the nets.

A set of NSTs for all nets is global data accessible and maintained by all the processes, while an HST and a VST are private data for each process. A set of NSTs of all nets forms a switchbox. Also an HST and a VST form a switchbox. Thus a set of NSTs of all nets can be converted to an HST and a VST, and *vice versa*. Generally, NSTs will be used to hold the nets' information, while, the HST/VST will be used to determine the relationships among the segments of nets. So, the existence of any conflicts is detected by traversing the HST and the VST. If we trace by each net, it means that we are tracing each NST. If we trace by each row or column, then we are tracing an HST or a VST.

4.3. Conflict group

There are three kinds of conflicts among segments in a row or a column, called a *grid-conflict*, a *segment-conflict* and a *via-conflict*. A *grid-conflict* exists if two or more horizontal (or vertical) segments, belonging to different nets, meet in a grid unit of a row (or a column) as in Figure 3 (a). Two segments of different net, A and B, are located in the same row and meet on a grid unit. A *segment-conflict* exists between two grid-conflicts if a segment is involved in both grid-conflicts. In Figure 3 (b), there are two grid-conflicts; between segment A and B and between segment A and C, where they share a segment A in common. A *via-conflict* exists if there are two segment-conflicts, or a grid-conflicts, which share a via as shown in Figure 3 (c). One or more conflicts related to each other by a *grid* or *segment* or *via-conflict* are considered a *conflict group*, which implies that the solution

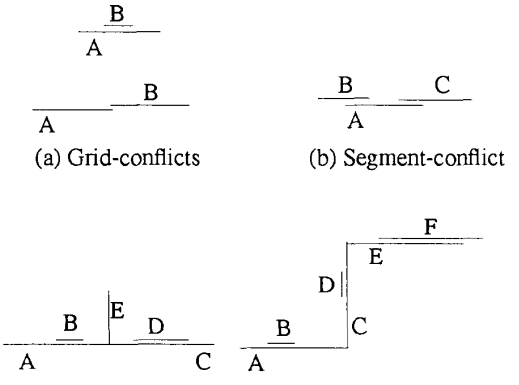

Figure 3. Types of conflicts.

of a conflict influences the solution of another conflict in a group. The conflict group is constructed with a set of segments involved in the conflicts when an HST and a VST are built from NSTs. If we look at Figure 4 (a), we observe two conflict groups, one group by conflicts 1, 2 and 3, the other by conflict 4 itself.

4.4. Fixed segments and re-routable segments

For the convenience of explanation, we classify segments into three types: the *main-conflict segment*, the *neighbor segment* and the *fixed segment*.

A *main-conflict segment* is the segment directly involved in the conflict, as shown in Figure 4 (b), where net 1 and 4 are extracted from Figure 4 (a). *Neighbor segments* are segments connected at both ends of the main-conflict segment, which may have maximum of 6 *neighbor segments*. A *main-conflict segment* and *neighbor segments* will be re-routed in the path-finding procedure. All of the segments, except main-conflict segment and neighbor segments are *fixed segments* including segments in nets not related to any conflicts at all. Also, segments in other conflict groups are considered to be *fixed segments* for the current conflict group. The *fixed segment* is regarded as an obstacle in the path-finding procedure.

4.5. Search lists

A search list is an order of segments, which will be used for searching the paths for the conflict group. If there are n segments in a conflict group, n search lists can be constructed by selecting the first segment of each search list from the n segments in a conflict group. The main difference among the search lists is

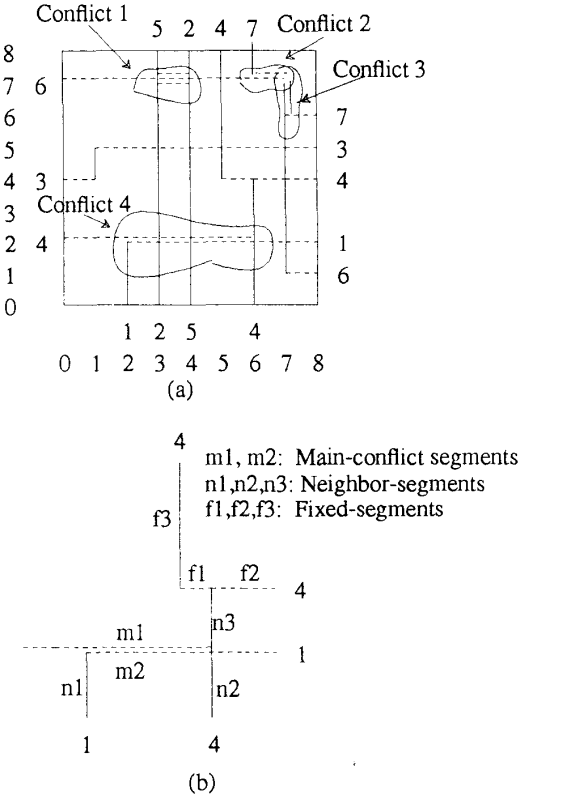

Figure 4. Initial routing and types of segments.
(a) Initial routing phase of the pedagogical problem
(b) Types of segments in a net segment tree

the first segment. The order of the remaining segments, with the exception of the first segment in a search list, can be chosen arbitrarily. These n search lists are sufficient to find the conflict-free routes for a conflict group. This is possible with the aid of the maze routing procedure described later in the next section.

Each search list will be assigned to the process which bears the same ID of the first segment in the search list. The order in each search list is used to find the local solutions in a conflict area by keeping the first segment path (optimal by itself) unchanged, while changing the paths of the other segments, one-by-one, according to the order in the search list.

4.6. Algorithm

The following is a brief description of the PARALLEX algorithm. The overall framework of the algorithm is a *distributed branch-and-bound* algorithm. The cost of the current feasible solution is used as a

lower bound. The cost of a path of a net is defined as the weight function value, to be defined later. The cost of the current solution is the sum of the paths of all of the nets. The following equation defines the weight function used in PARALLEX:

$W_k = v \cdot p + H + V$

where W_k : weight of net k,
v : cost for via,
p : number of vias,
H : sum of the horizontal segments of the current path of net k,
V : sum of the vertical segments of the current path of net k.

The cost of the current solution is defined by the following equation: $C_{current} = \sum W_k$.

A. **Initialize**: Each process extends inward the length of a grid unit from each terminal of the net. This is based on the rules for interconnection, which prohibit routing on the perimeter of the switchbox.

B. **Find an RST**: Each process finds a Rectilinear Steiner Tree (RST) for the net assigned to it. The RST is found by using a method based on Ho and Wong [10]. An RST is one of the many possible shortest paths for a net. The RST that has the smallest weight function value is selected initially as the initial path of the net.

Figure 4 (a) illustrates an initial routing phase of the pedagogical problem obtained in step B. In the figure, solid lines represent vertical wire segments and broken lines represent horizontal wire segments.

C. **Construct NST[i]** : An NST[i], where i is the net ID, is stored into the shared memory based on the RST which has built in step B. Once all of the NSTs are stored into the shared memory, they are copied to the private memory of each process, so that they can be accessed and manipulated individually without further synchronization.

D. **Find conflict group**: Each process maps all NSTs in the shared memory to the HST and the VST in the private memory. If there are *grid-* and/or *segment-*conflicts, record them onto an array of the conflict group in the private data area. After completion of mapping to the HST/VST, if there exist any *via* conflicts among the conflict groups, those groups will be merged.

E. **Find conflict-free paths**: From each conflict group, CG_k, construct search lists. Only the lists whose first segment's net ID is identical to the process ID are chosen for each process from every conflict group. Each segment in a list is given a priority number

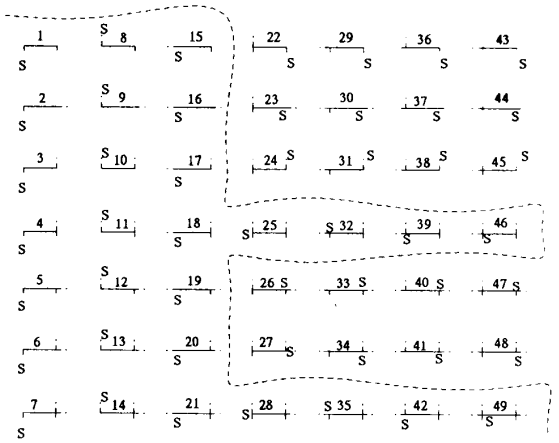

Figure 5. Patterns of a horizontal main-conflict segment and starting point, S, for maze routing.

according to the search order. The higher the order, lower the priority number, i.e., the first segment in an order is given priority number 1, the second segment is given priority number 2, and so on. Also, the neighbor segments are given equal priority to their main-conflict segment. Fixed segments are given the highest priority, 0, and not to be re-routed. In the path-finding procedure, if the priority number of the current segment is lower than the existing segment in the HST or the VST, the existing segment is regarded as eliminated and to be connected later in its own turn in the path-finding procedure.

E.1 Maze routing: In a maze routing phase, the path of the first segment in a search list is unchanged, while other segments in a search list are re-routed in turn. To start maze routing, a source and a target must be selected.

There are 49 patterns in a horizontal *main-conflict segment* to be distinguished from in selecting the source and the target as shown in Figure 5. Another 49 patterns exist for the vertical *main-conflict segment* if we rotate the patterns 90 degrees in Figure 5. The main-conflict segment is denoted with a solid line, while the neighbor segments are denoted with a dotted line as shown in Figure 5. The source, S, of each segment is selected from an end point of the neighbor segment. The target is located at the opposite side of the starting point, as one or two segments. The wave front from the starting point is expanded to the target.

E.2 Weight of via and slack-cost in a routing phase: Maze routing consumes a lot of memory space and takes exponential time. It guarantees an optimal

solution. However, an optimal solution in an order of search list may not be the best solution, considering the whole net and switchbox. Even the optimal solution of a procedure in the order of a search list may block any solution at all. If all of the solutions in a maze routing phase are found, it guarantees solutions in the next path-finding procedure, if any. However, all of the solutions do not lead to a feasible solution by nature; the more that vias exist, the less solutions are likely. The increase in slack-cost in a solution leads to a less feasible solution capability for the next procedure. Thus, there are three constants which control the number of solutions in a maze routing procedure: the slack-cost in a search list, the slack-cost in a net, and the weight of via. By selecting these three parameters properly, we can avoid using unnecessary memory space and save operating time.

- Slack-cost in a search list (SCAS): In each path-finding procedure, more wave front expansions are allowed to be searched for within the range of the slack-cost beyond the optimal length. If this slack-cost is large enough, all of the feasible solutions are found.

- Slack-cost in a net (SCAN): Slack-cost in a net, which is an extra cost allowed to permit a search beyond the optimal cost, confines the number of partial solutions for each net to eliminate any inefficient solutions which are far from optimal.

- Weight of via (WV): If the weight of via is increased, the cost in a net is increased. This results in less solutions within a slack-cost in a net. So, the increase of via weight leads to a cost increase, which in turn leads to the decrease of the number of feasible solutions within the range of slack-cost.

Each process which has conflicts tries to find conflict-free solutions for the segments involved in the search list. The solutions found are recorded in the partial solution partition of the shared memory, while the location of the partial solutions are posted to the index partition for the access of other processes.

F. **Merge the paths found**:

Each process searches the indices of the other partial solutions from the index partition of the shared memory, and makes the task queue to the private memory. By the order of this queue, each process merges the partial solutions found from other processes. If there are any conflicts on the way to the merging procedure, it uses another solution in accordance with its queue. If there is a solution, the current cost is recorded to the cost partition, so that other processes can recognize the existence of the solution and its cost. If it is asked to find only one solution, then all of the processes stop their tasks. If it is asked to find the multiple solution, the cost partition is renewed by the new lower cost until all of the processes finish their tasks. At the end of the procedure, the one best solution which is optimal will be reported.

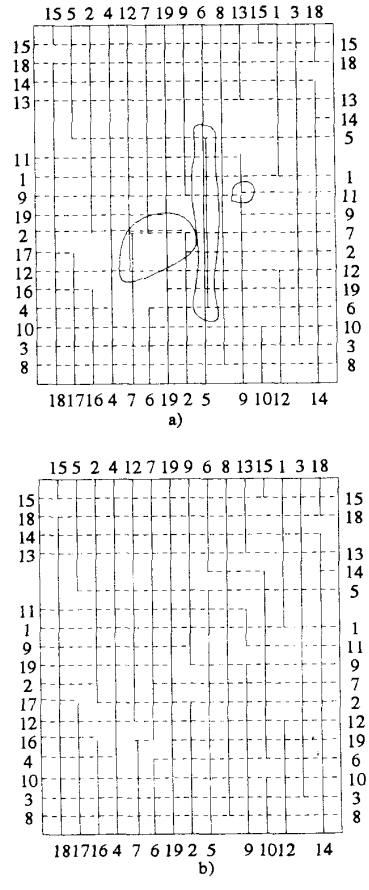

Figure 6. Conflicts shown on HST/VST and solution of example 2.

5. Experiment and results

In this paper, three experiments are illustrated. Example 1 is a pedagogical problem as shown in Figure 4 (a). Example 2 is a variant of the Modified Dense Switchbox [3]; one terminal at LEFT(13) is moved to LEFT(12), as shown in Figure 6. Example 3 is Burstein's Difficult problem as shown in Figure 7. Experiments of the algorithm show the speed-ups of 4.71 to 9.95 as shown in Table 1. For example 3 (Burstein's Difficult problem [1]), with 24 nets, speed-up is only 3.43. The main reason for this lower speed-

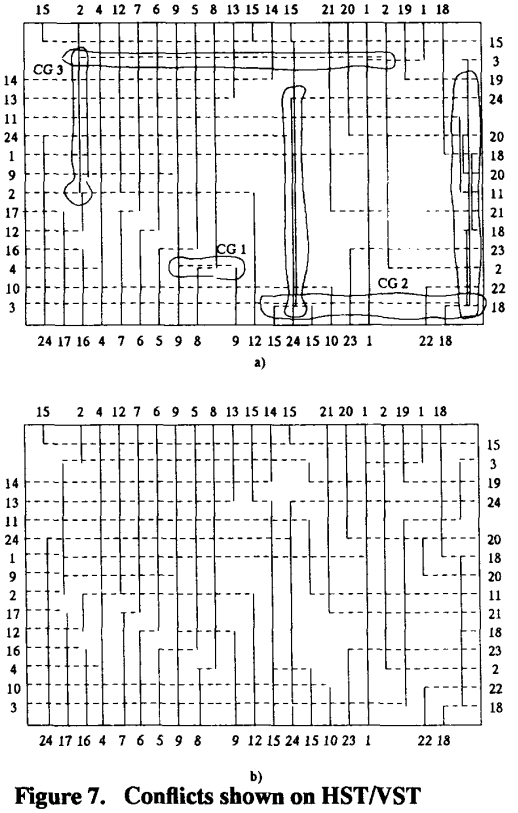

Figure 7. Conflicts shown on HST/VST and solution of example 3.

up is the involvement of the many segments in a conflict group. From Figure 7 (a) for example 3, we see that one conflict group is related with 12 segments and four indirect conflicts, while in example 2 as shown at Figure 6, conflicts are evenly distributed throughout the nets. One of the many search lists for the conflict group of example 3 is

3→15→15→18→24→15→24→18→3→18→11→20

where the number indicates the net ID of the conflict segment. Net IDs 15 and 18, are involved in the conflict three times, and net IDs 3 and 24 are involved twice. This long list of the search list takes much more time to get the partial solutions. Other processes must wait until they recognize the arrival of the partial solutions from the process which has finished tasks last. Thus, five processes, 3, 15, 24, 20 and 18 have consumed most of the time as shown in Table 2. For most of the experiment, one iteration from procedure D to F was able to solve the problem. Details of the study are illustrated in the author's thesis[2].

Table 1. Speed-ups of switchboxes.

Switchbox	Number of nets	Sequential(sec)	Parallel (sec)	Speed-ups
Example1	7	4.90	1.04	4.71
Example2	19	20.70	2.08	9.95
Example3	24	86.41	25.21	3.43

Table 2. Process time of Example3 executed parallelly.

Process ID	Time (sec)	Process ID	Time (sec)
23	0.68	9	2.54
7	1.05	2	4.08
22	1.10	8	6.19
16	1.16	11	7.28
14	1.21	4	8.89
17	1.26	1	9.32
19	1.29	12	10.22
5	1.32	3	13.29
21	1.33	15	15.73
13	1.37	24	23.73
6	1.50	20	24.93
10	1.78	18	25.21

6. Concluding remarks

A new large-grain parallel routing algorithm, PARALLEX, is presented for switchbox routing. PARALLEX has been developed to run in the shared-memory multiprocessor environment. An early version of PARALLEX has been successfully run on the Sequent Symmetry system with 26 processors. It found solutions for many switchbox routing problems, sequentially or in parallel. For this paper, the results of inquiring for a single solution are presented. The results of inquiring for an optimal solution will be included in the author's thesis[2]. By changing the value of the slack-cost and the weight of via, we can obtain an optimal solution based on the wire length or based on the number of vias. We are currently tuning the system to improve performance.

There are some modifications to the algorithm which have not yet been implemented, but are planned for further experimentation. One of the possible modifications is in the merging step. Instead of waiting for the partial solutions from each search list to be posted to the index partition, the index partition can be updated every time a process finds a new feasible solution in the search list. Another modification is construction of a conflict group with segment-conflicts

instead of via-conflicts, which may reduce the load for each process in a partial solution procedure. This modification to decompose via-conflicts in a partial solution procedure will utilize processes more efficiently and reduce the computational complexity in the merging procedure. A more interesting modification is to let processes asynchronously iterate steps D, E, and F without synchronizing at the end of step F. This is a form of the asynchronous iterative algorithm [19].

In addition, PARALLEX can be modified to effectively handle other routing problems (e.g., channels or routing regions embedded with obstacles, the knock-knee model or multilayer.)

References

[1] Michael Burstein and Richard Pelavin, "Hierarchical Wire Routing," IEEE Trans. on Computer-Aided Design, CAD-2, pp. 223-234, October 1983.

[2] Tae Won Cho, "Parallel Approach in Switchbox Routing," PhD thesis, University of Kentucky, 1991.

[3] J. P. Cohoon and A P. L. Heck, "BEAVER: A Computational-Geometry-Based Tool for Switchbox Routing," IEEE Trans. on Computer-Aided Design, vol. 7, 1988.

[4] David N. Deutsch, "Solutions to a Switchbox Routing Problem," IEEE Trans. on Comp-Aided Design, CAD-4, NO. 2, pp. 163, 1985.

[5] R. J. Enbody and H. C. Du, "General Purpose Router," Proc. 24th ACM/IEEE Design Automation Conference, pp. 637-640, 1987.

[6] R. A. Finkel, "Large-grain Parallelism - Three Case Studies," The Characteristics of Parallel Algorithms, The MIT Press, pp. 21-63, 1985.

[7] Sabih H. Gerez and Otto E. Herrmann, "Switchbox Routing by Stepwise Reshaping," IEEE Trans. on Computer-Aided Design, vol 8 pp. 1350-1361, Dec. 1989.

[8] M. Hanan, "On Steiner's Problem with Rectilinear Distance," SIAM Jour. Appl. Math., vol. 14, pp. 255-265, March 1966.

[9] A. Hashimoto and J. Stevens, "Wire routing by optimizing channel assignment within large apertures," Proc. 8th Design Automation Workshop, pp. 155-169, 1971.

[10] D. Hightower, "A Solution to the Line Routing Problem on Continuous Plane," Proc. 6th Design Automation Workshop, pp. 1-24, 1969.

[11] Jan-ming Ho, Gopalakrishnan Vijayan and C. K. Wong, "A New Approach to the Rectilinear Steiner Tree Problem," Proc. 26th ACM/IEEE Design Automation Conference, pp. 161-166, 1989.

[12] A. Iosupovicz and A. Vahidsafa, "Parallel Routing on a Hardware Array Router," Proc. IEEE Int'l Conf. on Computer-Aided Design, pp. 136-138, 1985.

[13] R. Joobbani, "WEAVER: An Application of Knowledge-Based Expert Systems to Detailed Routing of VLSI Circuits," Carnegie Mellon University Research Report, CMUCAD-85-56, June 1985.

[14] E. S. Kuh, "Efficient algorithms for channel routing," IEEE Trans. on Computer-Aided Design, CAD-1, pp. 25-35, Jan. 1982.

[15] C. Lee, "An Algorithm for Path Connections and its Applications," IRE Trans. Electron. Computer, EC-10, pp. 346-365, Sept. 1961.

[16] W. K. Luk, "A greedy switch-box router," Integration, the VLSI journal, vol. 3, pp. 129-149, 1985.

[17] T. Ohtsuki, M. Tachibana and K, Suzuki, "A Hardware Maze Router with Rip-up and Reroute Support," Proc. IEEE Int'l Conf. on Computer-Aided Design, pp. 220-222, 1985.

[18] R. C. Prim, "Shortest Connection Networks and Some Generalizations," Bell STJ, 36, pp. 1389-1401, November 1957.

[19] S. S. Pyo, "Asynchronous Algorithms for Distributed Processing," PhD thesis, Carnegie Mellon University, 1985.

[20] Ronald L. Rivest and Charles M. Fiduccia, "A Greedy Channel Router," Design Automation Conference, pp. 418-424, 1982.

[21] Jonathan Rose, "Parallel Global Routing for Standard Cells," IEEE Trans. Computer-Aided Design, CAD-9, No. 10, pp. 1085-1095, Oct. 1990.

[22] H. Shin and A. Sangiovanni-Vincentelli, "A Detailed Router Based on Incremental Routing Modifications: Mighty," IEEE Trans. on Computer-Aided Design, CAD-6, No. 6, November 1987.

[23] K. Suzuki, Y. Matsunaga, M. Tachibana, and T. Ohtsuki "A Hardware Maze Router with Application to Interactive Rip-up and Reroute Support," IEEE Trans. Computer-Aided Design, CAD-5, No. 4, pp. 466-476, 1986.

[24] Youngju Won and Sartaj Sahni, "Maze Routing on a Hypercube Multiprocessor Computer," Proc. of the International Conference on Parallel Processing, pp. 630-637, 1987.

[25] Mehdi R. Zargham, "Parallel Channel Routing," Design Automation Conference, vol. 25, pp. 128-133, 1988.