TR-0819

A Parallel Lookahead Line Search Router with Automatic Ripup-and-reroute

by
H. Date & K. Taki

December, 1992

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03)3456-3191 ~ 5
Telex ICOT J32964

Institute for New Generation Computer Technology

A Parallel Lookahead Line Search Router with Automatic Ripup-and-reroute

Hiroshi DATE

Institute for
New Generation Computer Technology
Tokyo 108, Japan

Kazuo TAKI

Dept. of Computer and Systems Engineering
Kobe University
Kobe 657, Japan

Abstract

In this paper, a new parallel routing method is proposed and evaluated. A key feature is object-oriented modeling. All line segments are modeled as independent objects communicating with each other through message exchanges. A distributed routing algorithm is designed based on a lookahead line search method that includes automatic ripup-and-reroute to retain the routing order. The program was implemented in a concurrent logic language, KL1, and was executed on PIM/m (Parallel Inference Machine), a distributed memory machine with 256 processors. The evaluation results for several industrial data are given.

1 Introduction

There have been many attempts to achieve high speed and good quality router systems with parallel processing. These attempts can be classified into two areas. One area focuses on the hardware engine which executes the specified routing algorithm efficiently [4, 6, 9]. The other area involves concurrent routing programs implemented on general purpose parallel machines [1, 7, 8, 11, 12]. The former approach can realize very high speeds, while the latter can provide great flexibility. We took the latter approach to realize both a high speed and flexible router system and targetted very large MIMD computers.

In general, large parallelism is needed to utilize a large MIMD computer. So we reported the preliminary measurement for a routing program based on a small granual concurrent objects model [3]. However, our router had two problems. One was that there was memory overflow for communication paths between processors due to increased numbers of objects for large-scale data. The other was the degradation in wiring rate due to net confliction among concurrently connected nets.

We improved our program to solve these problems as follows. For the first problem, we assigned a process, called a *distributor process* , to each processor. This process reduces the number of message paths between processors. We also adopted the ripup-and-reroute operation to retain the routing order for the second problem.

The routing method was implemented on the distributed memory machine, PIM/m (Parallel Inference Machine with 256 processors) [10], with a concurrent logic programming language KL1 [2]. We evaluated of the new router, from the viewpoints of (1) data size vs. speedup, and (2) the effectiveness of ripup-and-reroute.

The rest of the paper is organized as follows. Section 2 explains our previous work and the problems in our routing program. Section 3 describes our new routing program with an explanation of concurrent algorithms and implementation. Section 4 reports some measurements and evaluation results. Then, Section 5 concludes the paper.

2 Previous Work and Problems

In order to clarify the focus of our research, this section describes our previous work and the problems we faced.

Our router deals with the 2-layer routing problem. One layer is used for horizontal routing and the other for vertical routing.

In general, large parallelism is needed to utilize a large MIMD computer. So we designed a routing program based on a small granual concurrent objects model and reported the preliminary measurements[3]. However, two problems arose, as follows.

2.1 Memory overflow for communication paths

When we implement the concurrent program using KL1, two kinds of memory are necessary. The first is memory for representing processes. The other is memory for communication paths among processors.

In our routing program, the process structure shown in Figure 1 was implemented. Each process corresponds to the grid lines (*master line process*) and line segments (*line process*) on it. A master line process manages line processes on the same grid line and passes messages between the line processes and crossing line processes. So each master line process must communicate with all master line processes orthogonal to it. Therefore, the number of communication paths per master line process increases for large-scale data.

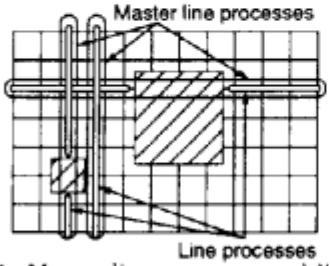

Figure 1: Master line processes and line processes

For efficient parallel execution on a distributed memory machine like PIM/m, processor mapping is important. In our router, a master line process and line processes on it are grouped. Then, each group is assigned to a processor randomly. This strategy mainly focuses on load balancing.

In this case, the communication between master line processes may be inter-processor communication. When the size of the routing grid becomes larger, many communication paths arise between the processors.

Previous experimental results have shown that the memory overflow for representing the communitaion paths between processors occurs for 750 × 750 grid size data on PIM/m. This limitation is too severe for use with practical amounts of data.

2.2 Degradation of wiring rate

When there is concurrent routing of multiple nets, different nets may conflict on the same line segment. In this situation, the first message to arrive (corresponding to net A) occupies the segment. The second message to arrive (net B) fails to complete a route and backtracks.

However, net A may backtrack afterwards and may release the line segment. In this case, net B does not visit the line segment anymore and the line segment may be left unused. This causes both lower quality routes (longer paths) and a lower wiring rate (more unconnected nets).

We studied the relation between wiring rates and parallelism in the experiments[3]. The routing quality was degraded for data with concentrated terminals.

3 Parallel Routing Program with Ripup-and-reroute

In order to solve the problems described above, we improved the process structure and added the ripup-and-reroute operation to the basic algorithm. In this section, these two points are discussed.

3.1 Process structure and processor mapping

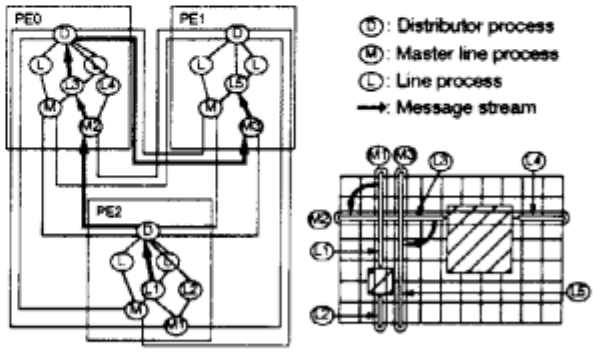

Figure 2: Improved process structure

Figure 2 shows the improved process structure and processor mapping. The key is the introduction of *distributor processes* to reduce the number of communication paths between processors. A distributor process is placed in each processor. The master line processes and the line processes are mapped as described in 2.1.

When the distributor processes communicate directly with each other, bottlenecks occur in each distributor process. We, therefore, adopted the process structure and processor mapping shown in Figure 2. Experiments on test data show that data forming up to a 5000 × 5000 grid can be handled.

3.2 Automatic ripup-and-reroute

In order to retain the sequential routing order, we added the automatic ripup-and-reroute operation to the basic algorithm. In our basic algorithm, two types of parallelism are realized. One is parallelism within a single lookahead operation and the other is concurrent routing of multiple nets.

When we use a sequential routing program, we must decide on the routing order of nets. We assume that the routing order of nets is given. Let the order of net_i be i. (i=1,···,N)

Each line process is in one of three states, *free*, *occupied* or *concrete*. The *free* status means that the line segment is not occupied by a net. The *occupied* status means that the line segment is occupied by some nets in permitting the multiple overlapping of routing paths. The *concrete* status means that the line seg-

ment is occupied by a net and is treated as a routing inhibited segment by other nets.

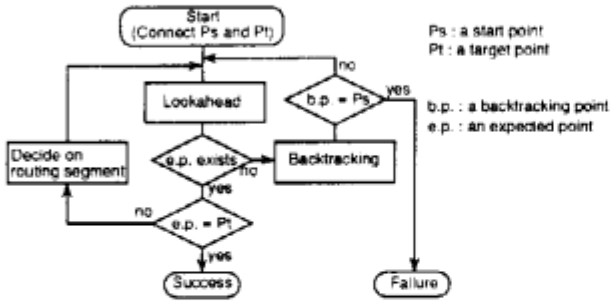

Figure 3: Flow of lookahead line search algorithm for one net

We used the lookahead line search method [5] shown in Figure 3 to search routing paths. Then we reconstructed the algorithm and included the automatic ripup-and-reroute operation to obtain a higher wiring rate, while keeping the concurrent objects model as a basic design framework.

The lookahead line search method is a line search algorithm coupled with a lookahead operation.

The routing procedure for one net is almost the same as that of the sequential lookahead line search algorithm, except that the lookahead operation is executed in parallel. In lookahead operations, the algorithm searches the line segment whose expected point is closest to a goal, where the expected point is defined as the closest location to a goal on a line segment. Its routing segment is decided from that search.

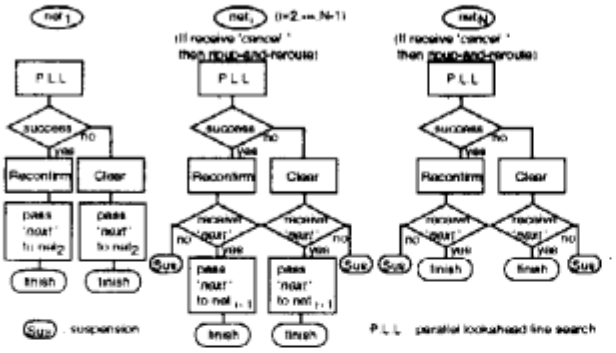

Figure 4: Parallel execution of expected points

Figure 4 shows the parallel lookahead line search with an automatic ripup-and-reroute algorithm. The routing of each net is executed concurrently. In reconfirm operation, the status of all line processes on the routing path changes to *concrete*. If a line process is occupied by other nets, the '*cancel*' message is passed to them. After completing the reconfirm operation, a message is passed to the next net to keep the routing order. The net receiving the '*cancel*' message starts the ripup-and-reroute.

Details of the flow are described in 3.3.

3.3 Examples

First, we assumed that the order of net_i is i (i=1,2,3). Here, we consider the problem to be as shown in Figure 5(a). For net_1, the routing path is decided from S1 to T1, then a '*reconfirm*' message is passed to start point S1 (Figure 5(b)). Next, the status of line processes is changed from *occupied* to *concrete* one line at a time. On its way to T1, a '*cancel*' message is passed to the net_2 which occupied the same line segment (Figure 5(c)). The net_2 that received the '*cancel*' message resets the status of the occupying line process to *free*, then again searches the routing paths between S2 and T2 (**ripup-and-reroute**).

When net_1 changes the status of all line processes on its routing path to *concrete*, a '*next*' message is passed to net_2 (Figure 5(d)). If the searching operation stops at a deadend, other routing paths are searched by backtracking operation (Figure 5(e)). net_2 completes its search of the routing paths from S2 to T2, and a '*reconfirm*' message is passed to start point S2 because of the arrival of the '*next*' message. Then, net_2 changes the status of all line processes on its routing path to *concrete* (Figure 5(f)).

An example follows, showing how the routing process is completed.

4 Measurements and Evaluation

We evaluate our router from the following two points of view : (1) Data size vs. Speedup, and (2) Effectiveness of ripup-and-reroute.

Table 1: Testing data

Data	Grid size	# of nets
D1	322×389	71
D2	262×106	136
D3	2746×3643	556
D4	2524×3424	1088

Four types of industrial LSI data were used. The features of these data are shown in Table 1. The terminals to be connected are concentrated locally in D1 and D3. Meanwhile, terminals that are distributed uniformly in D2 and D4.

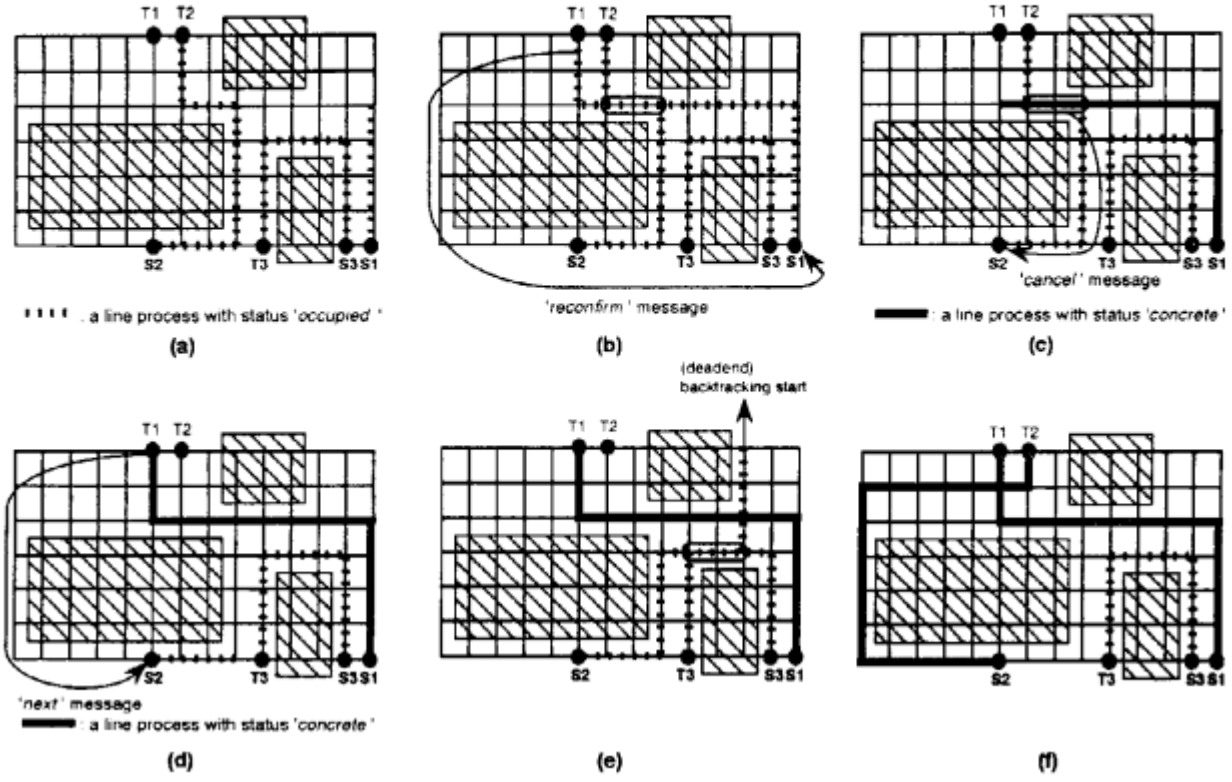

Figure 5: Example of parallel routing

4.1 Machine and language

The program was written in concurrent logic language KL1 and is executed on a MIMD machine PIM model m with distributed memory. The 256 processors are connected by a 2-dimensional mesh network with wormhole message routing.

4.2 Data size vs. speedup

Generally, when the data size increases, the number of processes increases too and more parallelism can be expected. We measured the relationship between the number of processors and speedup for various sizes of data.

Figure 6 shows the results of measurement. This graph shows that the speedup depends on the number of nets. When the number of nets is smaller than the number of processors (D1 and D2), the speedup curve peaked near the number of the nets. The results also show 49-fold speedup with 256 processors for D4 and do not show any sign of saturation yet. We need to investigate the speedup when the data size is increased further.

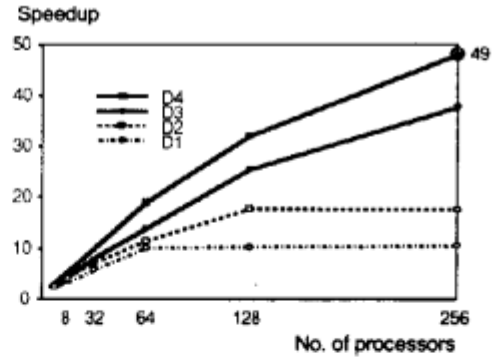

Figure 6: Data size vs. speedup

4.3 Effectiveness of ripup-and-reroute

Two cases of PIM/m measurements using 256PEs (with ripup-and-reroute and without ripup-and-reroute) are compared on PIM/m with one processor in Table 2.

Our routing system routes shorter nets first. At the initial stage of the concurrent routing process, almost no nets are in conflict, because the search space of each net is small. However, at later stages, the search

space becomes large. So we decreased the number of nets wired concurrently at later stages.

We made sure that our router found the same routing paths as found by the sequential router from D1 to D4 by comparing the routing results.

Table 2: Comparison of performance

Data		PIM/m 256 PEs with r&r	PIM/m 256 PEs without r&r	PIM/m 1PE (sequential)
D1	E	5.5	2.8	56.0
	W	100	72	100
D2	E	3.0	2.0	45.3
	W	100	100	100
D3	E	70.5	45.2	2712.0
	W	100	80	100
D4	E	30.0	17.3	1472.0
	W	100	98	100

E:execution time (Sec.),W:wiring rate(%)

5 Conclusion

We presented a new parallel routing method to retain the routing order of nets. This is suitable for very large parallel computers. The program was implemented on a distributed memory machine with 256 processors. Preliminary evaluation was done with industrial LSI data.

The experimental results showed that the greater the data size, the higher the efficiency attained by a maximum of 49-fold speedup with 256 processors against single processor execution. The speedup curve does not appear to be saturated for large amounts of data.

In experiments on the effectiveness of the ripup-and-reroute operation, a wiring rate of one hundred percent was attained for all testing data. We also confirmed that our router found the same routing paths as those of a sequential routing program based on the same algorithm. This means that the ripup-and-reroute operation works effectively.

As future works, we expect to realize both good execution times and good wiring rates by controlling the number of nets wired concurrently and changing the routing order automatically.

Acknowledgments

We would like to thank Dr. Kitazawa of NTT Co., Mr. Matsumoto and the members of PIC-WG, a working group in ICOT, for valuable discussions. We also acknowledge NTT Co. and Hitachi Ltd. for presenting us with LSI chip data.

References

[1] R. J. Brouwer and P. Banerjee, "PHIGURE : A Parallel Hierarchical Global Router," *Proc. 27th Design Automation Conf.*, pp. 650-653,1990.

[2] T. Chikayama, "Operating system PIMOS and Kernel Language KL1," *Proc. Int. Conf. on FGCS*, ICOT, Tokyo, pp. 73-88, 1992.

[3] H. Date, Y. Matsumoto, K. Kimura, K. Taki, H. Kato and M. Hoshi, "LSI-CAD Programs on Parallel Inference Machine," *Proc. Int. Conf. on FGCS*, ICOT, Tokyo, pp. 237-247, 1992.

[4] K. Kawamura, T. Shindo, H. Miwatari and Y. Ohki, "Touch and Cross Router," *Proc. IEEE ICCAD90*, pp. 56-59, 1990.

[5] H. Kitazawa, "A Line Search Algorithm with High Wireability For Custom VLSI Design," *Proc. ISCAS'85*, pp. 1035-1038, 1985.

[6] R. Nair, S. J. Hong, S. Liles and R. Villani, "Global Wiring on a Wire Routing Machine," *Proc. 19th Design Automation Conf.*, pp. 224-231, 1982.

[7] O. A. Olukotun and T. N. Mudge, "A Preliminary Investigation into Parallel Routing on a Hypercube Computer," *Proc. 24th Design Automation Conf.*, pp. 814-820, 1987.

[8] J. Rose, "Locusroute : A Parallel Global Router for Standard Cells," *Proc. 25th Design Automation Conf.*, pp. 189-195, 1988.

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

[10] K. Taki, "Parallel Inference Machine PIM," *Proc. Int. Conf. on FGCS*, ICOT, Tokyo, pp. 50-72, 1992.

[11] T. Watanabe, H. Kitazawa, Y. Sugiyama, "A Parallel Adaptable Routing Algorithm and its Implementation on a Two-Dimensional Array Processor," *IEEE Trans. on CAD*, Vol.CAD-6, No.2, pp. 241-250, 1987.

[12] Y. Won, S. Sahni and Y. El-Ziq, "A Hardware Accelerator for Maze Routing," *Proc. 24th Design Automation Conf.*, pp. 800-806, 1987.