Gridless Wire Routing using Cost Functions and Multiple Processors

Steven M. Rubin
Sun Microsystems Laboratories

May 23, 2008
Memo SML#: 2008-0232

Abstract

The task of routing an electrical circuit can consume large amounts of time and memory. Attempts to speed this task have used fixed grids to limit the routing choices, global preprocessing to decompose the task into smaller subtasks, and multiple processors to do all of this in parallel.

This paper describes a router that does not use a fixed grid, but instead allows the routes to run arbitrarily, limited only by blockages and design rules. Multiple parallel processors are used to speed up the routing of a single path as well as to find multiple routes at once. Finally, no global routing step is done because the parallel decomposer does much of this job when it assigns different routes to different processors.

The router is implemented in the Electric™ VLSI Design System [Rubin1983] and has demonstrated a speedup of over 4X on an 8-processor machine.

Routing

The task of routing wires was essentially solved nearly 50 years ago [Lee1961] and has only been improved upon slightly since then. Lee divided the space between a route's endpoints into a two-dimensional grid, where each point in the grid is a possible location of a wire. Blockages in the routing space were then marked in the grid to tell the routing algorithm where it can and cannot go. To find the route's path, a "wavefront" of markers was written in the grid, propagating outward from one of the endpoints. This wavefront expanded until it reached the other endpoint, and then path was determined by walking backwards to the original endpoint. The Lee algorithm finds the optimal path, but it consumes large amounts of memory.

There have been many attempts to improve on the Lee algorithm, but all of them perform essentially the same task. To reduce the time spent marking a wavefront, multiple steps can be made at once through the routing grid [Watanabe1987]. To reduce the amount of memory used, the route is often constrained to "lines" [Kozawa1972, Suzuki1987]. To reduce the complexity of the task, a global

routing step decomposes the task into many smaller tasks [Soukup1981]. Finally, many routers use two layers for routing: one for horizontal wires and one for vertical wires [Deutsch1976, Hsu1982]. This not only reduces the routing complexity, but also prevents a route from blocking subsequent routes.

The Sea-of-Gates Router

This paper describes a router called the "sea-of-gates" router. It has been implemented as part of the Electric VLSI Design System [Rubin1983] and is written in Java.

The sea-of-gates router uses a variant of the Lee search technique in which there is no fixed grid for the route. Instead, each advance of the wavefront moves as far as it can go given the limits set by blockages and design rules. Each such advance defines a segment of the final path. These segments, and all of the blockage information, are stored in a spatial data structure called an R-Tree [Guttman1984]. This structure allows rapid searching, and can represent any space regardless of grid alignment. Therefore, the router can search from any point without extra time or space costs.

No global routing step is performed: each route is handled in its entirety. The router starts by reducing the problem to a set of two-point routes. Any net that connects more than two points is broken down into multiple two-point routes. The router runs the power and ground nets first, and then runs the remaining routes, starting with the shortest.

The search method is based on Dijkstra search [Cormen1990]. This is a general search algorithm that finds a good path by using cost factors for each segment of the route. The search proceeds in 3-space, where segments can be run in any of 6 directions from a given point (4 directions within the layer plus two directions to adjoining layers). The search maintains a list of progress points, which initially consists of the starting point of the route. At each step, the lowest-cost point is taken off of the list and is replaced with up to 6 new points that define routing segments in the 6 directions. When a step moves by a large distance in X or Y, intermediate points are also created along the way with higher costs so that the search can "back up" if necessary. The list is never pruned: since its size is limited by the number of points in the search area, size has not been a problem.

Each of the new points has a cost, which is added to the previous cost at that point. Points in the search space are marked so that loops do not occur. After adding these 6 new points, one of them may be the "low cost" point of the list, or some earlier point may rank higher. The search continues until the goal is found. If the list of points empties, then no route can be found. The search is also aborted after a user-specified number of steps has been tried (initially 200,000).

It is interesting to note that different results are obtained depending on which of the two endpoints of the route is the start and which is the destination. Early versions of the router ran the search in both directions and then compared the results. However, it was found that one of the directions could sometimes get trapped and take as much as 10 times as long to find the destination. Traps occur when a blockage area is open on one side and the optimal path appears to go into the blocked area, causing much wasted time exploring the entire area before abandoning it. To solve this problem, the router starts both directions at the same time and takes turns advancing them. The first to reach its destination is not only the fastest, but is generally also the best.

Cost Functions

To more intelligently guide the search, there are a number of cost functions that are applied to each segment of the search. The most basic cost functions favor moving toward the destination endpoint. For example, a segment that moves away from the destination incurs an extra cost, as does a segment that changes layers in the wrong direction. To prevent zig-zag paths, every change of direction incurs a cost. The router does not force certain layers to be horizontal and others to be vertical, but it encourages such behavior by using cost functions. There is also a cost for changing layers, even when it moves toward the goal. This also prevents zig-zag routes, but in the vertical dimension.

Another cost function favors points that are on a grid. Although the router is gridless, it is not desirable to run wires at arbitrary locations because it may cause resolution errors. Therefore, a cost is incurred if a wire is off-grid. This forces most of the path to run on a grid and to go off the grid only to connect at the endpoints.

Users can request that certain layers be "favored," which then uses costs to encourage their use. Users can also request a particular layer not be used at all, but that is not implemented as a cost: the layer is simply removed from consideration.

One problem that occurs in routing is that wires which are placed earlier in the process tend to create blockages which make it more difficult to route wires later in the process. Some routers rip-up the early wires to get the most number of routes [Dees1982, Shin1987]. The sea-of-gates router employs a cost function to encourage each route to make optimal use of the open space. Each new segment is examined at both ends to see how much empty space exists beyond it. If there is a large amount of space on both sides, a cost is added to the segment. This is done to encourage segments to make wise use of space…if the segment abuts a used area, then it is sensibly filling the space. If the segment divides a large open area, then it is fragmenting the space and should be penalized.

This notion of using costs to guide routing has been used before [Linsker1984]. It allows the integration of many important factors including crosstalk and timing. The disadvantage of cost functions is that it reduces all information to a single number, and so it is hard to tune the costs relative to each other. The advantage, however, is that the overall algorithm is simpler.

Parallel Processing

The sea-of-gates router makes use of two different parallel processing techniques to achieve faster results. The most effective use of parallel processors is in handling the two directions of search. This technique uses two processors for each route. One processor searches for the path between endpoint A and B while the other processor searches for the path between endpoint B and A. Since these two paths are usually interleaved on a single processor, the use of two processors effectively halves the search time with very little overhead. The processor that finds its destination first aborts the other processor's effort.

The other parallel processing technique used by the sea-of-gates router is to run multiple routes in parallel. If the multiple routes do not intersect, then they can be planned independently of each other. If, however, they interfere with each other, then it may be necessary to reject some of them

[Takahashi1990]. The sea-of-gates router chooses routes that do not intersect so that rejection does not occur. To ensure that routes will not intersect, the router defines the "routing area" as the minimum bounding box of the two endpoints, plus an extra buffer amount. The routing process cannot go outside of this area, and all routes that are being run in parallel are required to have no intersection of their routing area with others. It was determined experimentally that most routes run inside a routing area that has a extra buffer whose size is 7% of the minimum cell dimension.

Results

Five different routing tasks were tested with the sea-of-gates router. Two of them involve pad routing (running from a core cell to a ring of pads) and three of them involve true "sea-of-gates" routing tasks (connecting a set of cells together). The more complex pad-routing task has a pathological situation in which a set of wires all cross each other.

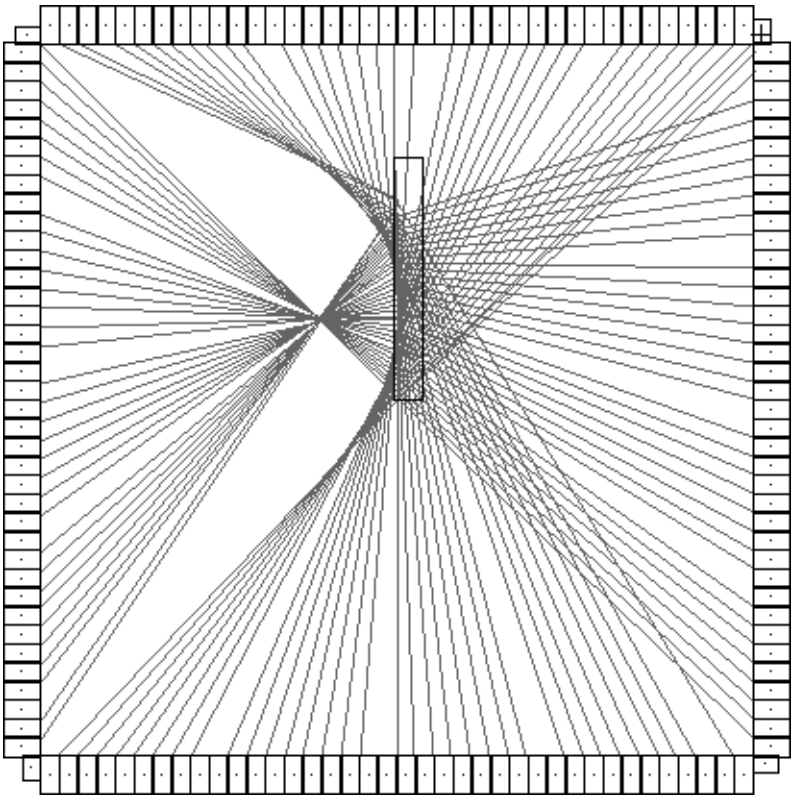

Figure 1: The large pad routing example

The sea-of-gates examples require arbitrary connections over a fully laid-out area of circuitry.

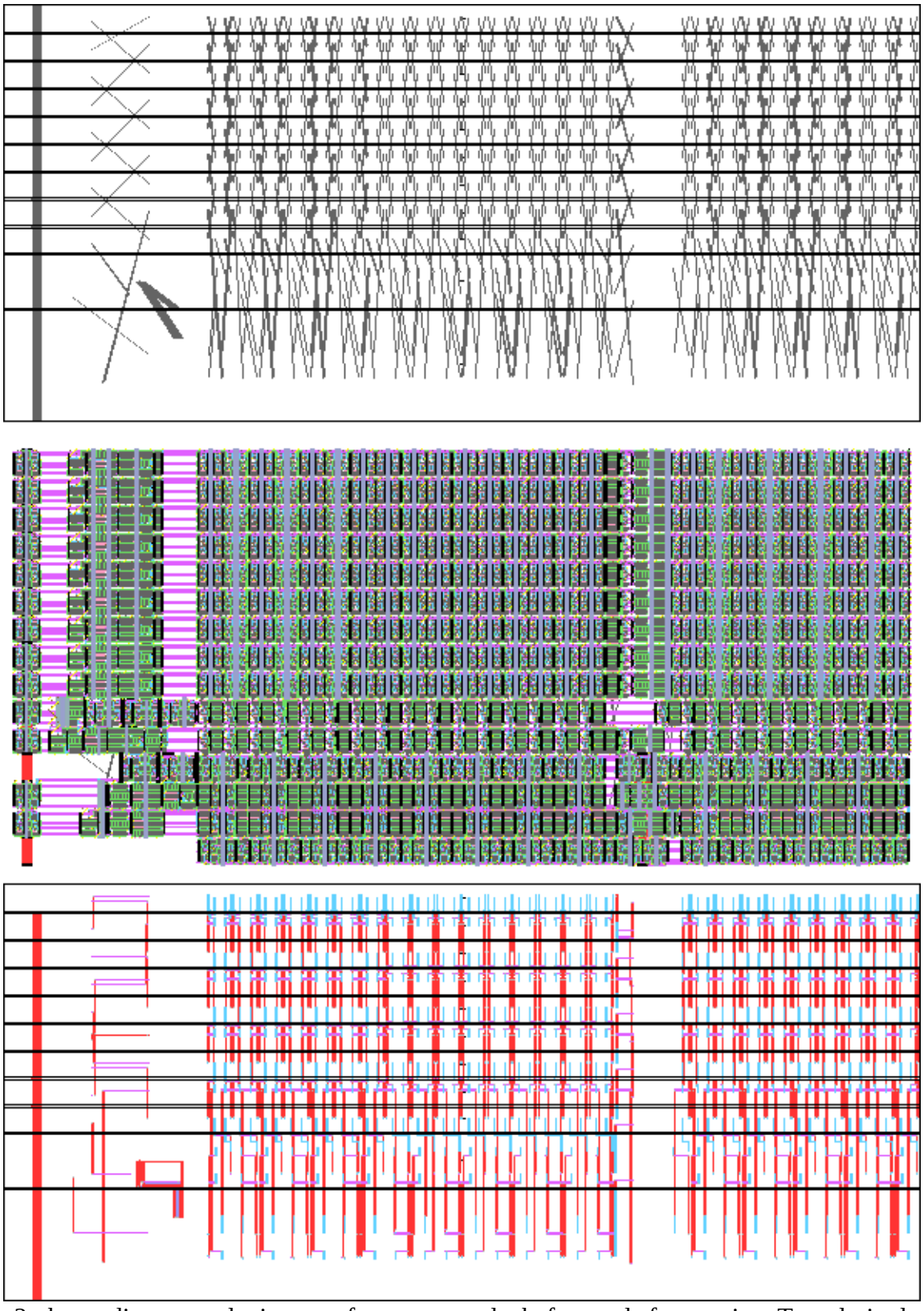

Figure 2: the medium-complexity sea-of-gates example, before and after routing. Top: desired routes; middle: circuitry; bottom: final routes.

The tasks were run on three different machines that ranged from 2 to 8 processors.

The following table shows the difference between using one processor and using two to run both directions in parallel:

		Machine A (2 proc)			Machine B (4 proc)			Machine C (8 proc)		
Example	**Segments**	**1 Proc**	**2 Proc**	**%**	**1 Proc**	**2 Proc**	**%**	**1 Proc**	**2 Proc**	**%**
Sea 1	45	5	4	20	2	1		7	4	43
Sea 2	682	659	290	56	1585	941		905	485	47
Sea 3	2439	814	688	16	335	304		776	516	34
Pads 1	29	46	18	61	65	29		117	49	58
Pads 2	135	100	73	27	355	183		102	66	35

This shows speedups that reduce routing time by as much as 58%. Considering that two processors should reduce the overall time by no more than 50%, this is curious. The reason is possibly related to the fact that machine C was shared with other users and had unknown loads. The results will be investigated further.

The following table shows the difference between using one processor and using as many as are available to run different routes in parallel:

		Machine A (2 proc)			Machine B (4 proc)			Machine C (8 proc)		
Example	**Segments**	**1 Seg**	**2 Segs**	**%**	**1 Seg**	**4 Segs**	**%**	**1 Seg**	**8 Segs**	**%**
Sea 1	45	5	5	0	2	2	0	7	9	-28
Sea 2	682	659	452	32	1585	612	61	905	221	76
Sea 3	2439	814	733	10	335	286	15	776	557	28
Pads 1	29	46	44	5	65	78	-20	117	124	-6
Pads 2	135	194	73	62	355	312	12	102	202	-98

These results are not as good as the ones that use multiple processors on each route. In fact, some of the pad routing tests actually slow down when run with multiple processors. The reason for this is that the pad tests are unable to find multiple routes to run in parallel, and so do not use all of the available processors. Because the multiple processors must wait for all of the routes to finish before advancing to the next set of routes, the slowest route delays all of the other processors.

The following table shows the difference between using one processor and using two to run both directions in parallel and to do this multiple times to run different routes in parallel:

		Machine B (4 proc)			Machine C (8 proc)		
Example	**Segments**	**1 Seg**	**2 Segs, 2 Proc**	**%**	**1 Seg**	**4 Segs, 2 Proc**	**%**
Sea 1	45	2	2	0	7	7	0
Sea 2	682	1585	596	62	905	202	88
Sea 3	2439	335	292	13	776	429	45
Pads 1	29	65	29	55	117	40	66
Pads 2	135	355	252	29	102	118	-16

Once again, the pad examples fail to show improvement because they cannot find enough multiple routes to run in parallel.

Conclusion

The new Sea-of-Gates router was developed very quickly. It is able to handle complex routing tasks, and still has room for improvement (by adding global routers, rip-up, and more). It demonstrates the power of Electric as a development environment, and provides Electric with a much-needed routing facility.

References

Cormen, T., Leiserson, C.E., and Rivest, R.L., "Introduction to Algorithms", MIT Press and McGraw Hill, New York, 1990.

Dees, William A. Jr. and Karger, Patrick G., "Automated Rip-Up and Reroute Techniques", Proc. 19th Design Automation Conference, 1982, 432-439.

Deutsch, David N., "A 'Dogleg' channel router", Proc. 13th Design Automation Conference, 1976, 425-433.

Guttman, Antonin, "R-Trees: A Dynamic Index Structure for Spatial Searching," ACM SIGMOD, 14:2, 47-57, June 1984.

Hsu, Chi-Ping, "A New Two-Dimensional Routing Algorithm", Proceedings 19th Design Automation Conference, 1982, 46-50.

Kozawa, T., Horino, H., Watanabe, K., Nagata, M. and Hukuda, H., "Block and Track Method for Automated Layout Generation of MOS-LSI Arrays," IEEE International Solid-State Circuits Conference, Digest of Technical Papers, Vol. XV, 1972, 62-3.

Lee, C. Y., "An Algorithm for Path Connections and Its Applications," *IRE Transactions on Electronic Computers*, EC-10, 346-365, September 1961.

Linsker, Ralph, “An iterative-improvement penalty-function-driven wire routing system”, IBM Journal of Research and Development, Vol 28 No 5, Sept 1984, 613-624.

Rubin, Steven M., “An Integrated Aid for Top-Down Electrical Design”, Proceedings, VLSI '83 (Anceau and Aas, eds.), North Holland, Amsterdam, 1983.

Shin, Hyunchul and Sangiovanni-Vincentelli, A., “A Detailed Router Based on Incremental Routing Modifications: Mighty”, IEEE Transactions on Computer-Aided Design, CAD-6, 6, Nov 1987, 942-955.

Soukup, Jiri, “Circuit Layout”, Proc IEEE, Vol 69 no 10, Oct 1981, 1281-1304.

Suzuki, K.; Ohtsuki, T.; and Sato, M., “A Gridless Router: Software and Hardware Implementations,” VLSI '87, 121-131, 1987.

Takahashi, Yoshizo and Sasaki, Shigetaka, “Parallel Automated Wire-Routing with a Number of Competing Processors,” Proceedings International Conference on Supercomputing, 310-317, 1990.

Watanabe, Takumi; Kitazawa, Hitoshi; and Sugiyama, Yoshi, “A Parallel Adaptable Routing Algorithm and its Implementation on a Two-Dimensional Array Processor,” IEEE Transactions on CAD, CAD-6, 2, 241-250, March 1987.