Parallel Global Routing for Standard Cells

JONATHAN ROSE, MEMBER, IEEE

Abstract—**Combined placement and routing has the potential to achieve better quality automatic layout because the placement optimization can use the actual wire paths to make better decisions. This approach has been considered infeasible because of the computational requirements of the routing task. In this paper, we investigate the potential speedup of a standard cell global router using a general-purpose multiprocessor. We present *LocusRoute*, a global routing algorithm for standard cells, and its parallel implementation. The uniprocessor speed and quality of LocusRoute is comparable to modern global routers—LocusRoute compares favorably with the TimberWolf 5.0 global router [16], and a maze router that searches the same space more completely. Two successful methods of parallel decomposition of the router are presented. The first, in which multiple wires are routed in parallel, uses the notion of *chaotic* parallelism to achieve significant performance gains by relaxing data dependencies, at the cost of a minor loss in quality. Using iteration and careful assignment of wires to processors, this degradation is reduced. The approach achieves measured speedups from 5 to 14 using 15 processors. The second parallel decomposition technique is the evaluation of different routes for each wire on separate processors. It achieves speedups of up to 6 using 10 processors. We demonstrate that when these two approaches are combined, the aggregate speedup is the product of the individual approaches' speedup and, using an improved scheduling approach, it can be even greater. With a simple model based on these results, we predict speedups of more than 75 using 150 processors.**

I. Introduction

FULLY combined automatic placement and routing, in which every placement is judged on the basis of the routed interconnections, can provide the best possible measure of the *goodness* of a placement. Evidence from recent research has shown that information provided by the routing can be helpful: Burstein and Hong [7] describe a gate array layout system in which routing is interleaved with the placement process. Dai and Kuh [11] perform simultaneous floorplanning and global routing by hierarchically decomposing the problem to reduce the computational complexity of the routing problem. Suaris and Kedem [33] also integrate placement and routing by successively refining a hierarchical routing at the same time the placement hierarchy is refined in a quadrisection-based approach.

Each of these approaches reduce the computational complexity of the routing either by i) routing only after some amount of placement optimization [7] or by ii) routing at the current level of hierarchy [11], [33]. To achieve fully combined placement and routing, global routing must be invoked whenever changes to the placement are considered and each affected wire must be completely rerouted. This approach may achieve further gains in placement quality but requires so much computation that up to now it has been considered infeasible. In this paper we address this problem by investigating the performance gains possible for global routing using a multiprocessor. We present LocusRoute, a global routing algorithm for standard cells and its parallel implementation.

Currently, we envision two approaches to combined placement and routing. We summarize the approaches here to further motivate the parallel router presented in this paper.

1.1. Direct Area Estimation

A fast standard cell global router could be invoked to route and reroute wires as the positions of the endpoints of the wires are changed during the placement process. This allows the direct measurement of the area of the placement. Other work [32], [23] suggests that it is difficult to optimize area-based cost functions because the so-called *energy landscape* or *convexity* of the cost function is unfavorable, particularly if the placement optimization uses single cell moves or two-cell exchanges. We regard this problem, however, as an interesting open question in automatic placement.

1.2. Multiway Partition-Based Placement Optimization

This is an N-way recursive partitioning algorithm, as opposed to 2-way in min-cut placement [5] and 4-way in quadrisection [33]. In this kind of approach a coarse grid of N partitions is recursively imposed on an initial placement. The placement is perturbed by proposing and possibly accepting moves that change the partition assignment of one or more cells. The cost function is determined from the number of wires crossing each *segment* of a partition line, and the partition balance requirements. To calculate the segment crossing counts, the paths of all the wires must be determined. This is exactly the global routing problem, and thus the motivation for a fast parallel router is to make the segment crossing count recalculation as fast as possible.

While the context of LocusRoute is as a tool to gain higher quality placement, we show that it is a reasonable global router for standard cells in its own right. The routing performance of LocusRoute, as measured by the total

Manuscript received September 21, 1988; revised October 8, 1989. This work was supported by DARPA under Contract N00014-87-K-0828, a Stanford Center for Integrated Systems Seed Research Grant, and an NSERC Post-Doctoral Fellowship. This paper was recommended by Associate Editor R. H. J. M. Otten.

The author was with the Computer Systems Laboratory, Stanford University, Stanford, CA. He is now with the Department of Electrical Engineering, University of Toronto, Toronto, Ont., Canada M5S 1A4.

IEEE Log Number 9036683.

number of routing tracks, is comparable to that of the TimberWolf 5.0 global router [16] and a maze router.

We present three approaches to the parallel decomposition of LocusRoute: routing several wires at once, routing several two-pin segments of the wire simultaneously, and evaluating possible routes in parallel. In the first approach, we use the notion of *chaotic* parallelism in which data dependencies are relaxed in order to gain performance. This causes a loss of quality in the results, but because the underlying algorithm is iterative, we can recover some of the lost quality. The quality is improved further by assigning wires to processors in such a way as to reduce the data dependency violations. Also, we argue throughout this paper that, in a combined placement and routing scheme, it is advantageous to trade a small amount of quality for a large gain in speed, allowing more placements to be evaluated. This approach achieves measured speedups ranging from 5 to 14 using 15 processors depending on the size of the circuit.

A second parallel decomposition, the route-by-route approach, achieves speedups of up to 6 using 10 processors. Since the route-by-route approach is *orthogonal* to the wire-by-wire scheme, when the two are used together the total parallelism should be the product of the parallelism of the two individual methods. This is demonstrated on 15 processors, and used to predict speedups of more than 75 using 150 processors for standard benchmark circuits, using a static scheduling policy. We show that when a dynamic scheduling approach is used, greater speedup and efficiency results.

1.3. Related Work

Previous work on parallel routing [6], [4], [1], [18], [30], [12], [35] has focused on a fixed hardware mapping for the Lee maze routing algorithm [15]. As such they lack the flexibility that is required in practical CAD software. Another drawback of special hardware for the Lee algorithm is that a uniprocessor implementation can be made very efficient using special software data structures that cannot be put easily into fixed hardware.

Until recently, there have been few publications on global routing for standard cells. Supowit [34] presents a simple heuristic for solving a limited subcase of the global routing problem. Sechen [31], [32] describes the original TimberWolf global router which decomposes multipin wires into two-pin wires using a minimum spanning tree algorithm and then employs an iterative track-switching algorithm to reduce channel densities. It suffered from the inability to cross any standard cell row more than once. Two global routers are described in [13] and [36] but give little detail of the process. An early version of the work presented in this paper was given in [28]. More recently, [16] and [10] have introduced new global routing algorithms that consider the problem much more carefully. In [16], a Steiner tree approximation for a wire is followed by a coarse global routing in which paths are chosen to minimize a penalty function that includes the channel density. Feedthroughs are assigned, and particular attention is paid to aligning the feedthrough paths. Finally, the spanning tree is refined using more detailed knowledge of the previous steps, and a final optimization of the switchable segments is performed. In [10], a minimal spanning forest is determined, which has the unique feature that it considers all wires at once. The algorithm of [16] is quantitatively compared to the LocusRoute algorithm in this paper. A survey of global routing that touches on standard cells appears in [17].

1.4. Organization

This paper is organized as follows: Section II defines the global routing problem for standard cells and describes the sequential LocusRoute algorithm. Section III gives performance comparisons with the TimberWolf 5.0 global router [16] and a maze router. Section IV presents three approaches for speeding up the new router using parallel processing, and performance results. Section V presents experiments with combining two of the aproaches, using two scheduling policies, which are then used to model and predict speedups for larger numbers of processors.

II. The LocusRoute Algorithm

This section defines the standard cell global routing problem, and describes the new LocusRoute approach to solving it.

2.1. Problem Definition

Global routing for standard cells decides the following for each wire in the circuit.

1) For each group of electrically equivalent pins (pin clusters in the terminology of [31]) it determines which of those pins are actually to be connected.
2) If there is no path between channels when one is required, it must decide either which built-in feedthrough to use or where to insert a feedthrough cell.
3) It decides which part or parts of a channel a wire should occupy.
4) It must determine the channel to use in routing from a pad into the core cells.

The objective of a global router is to minimize the sum of the channel densities of all the channels (hereafter called the *total number of tracks*).

In this discussion of global routing there will be no differentiation between feedthrough cells and built-in feedthroughs—they are referred to jointly as *vertical hops*. The decision to insert a feedthrough cell or use a built-in feedthrough is deferred to a post-processing step. Whenever comparisons are made to other global routers, the post-processing step is invoked to obtain accurate track counts.

2.2. The LocusRoute Algorithm

In the LocusRoute algorithm, the following five steps are executed for *each* wire. The details of each step are contained in the subsequent sections.

1) *Segment decomposition:* Each multipoint wire is divided into a set of two-point *segments* (where a "point" is a pin cluster), using a minimum spanning tree algorithm. By considering only two-point wires we introduce a suboptimality, but some of the negative effect is mitigated by a local optimization performed in the reconstruction: step 4) below. A direct comparison with a multipoint maze route is given in Section III-3.2.
2) *Permutation decomposition:* Since each pin cluster can consist of two physical pins (one on the top of a cell and one on the bottom), there are four possible ways to connect two pin clusters. Each of these possible routes is called a *permutation*. This step decomposes the segment into permutations and uses a simple heuristic (described below) to see if all the permutations should be evaluated.
3) *Route generation and evaluation:* A low-cost path is found for each permutation by evaluating a subset of the two-bend routes between physical each pin pair. The *cost* of a wire is defined below, in Section II-2.2.3. The permutation with the best cost is selected as the route for that segment.
4) *Reconstruct:* This step joins all the segments back together, performs a local optimization, and assigns unique numbers to distinct segments of the same wire in each channel. This is so that a channel router can distinguish between two segments and will not inadvertently join them together.
5) *Record:* The presence of the newly routed wire is recorded so that later wires can take it into account.

LocusRoute uses the iterative technique described in [19]. Briefly, this means that after the first time all wires are routed, each is sequentially ripped up and then rerouted. By routing each wire several times (typically four iterations gives the best answer, but little improvement is seen after two iterations), the final answer is improved by 5–10% because later wires can take earlier wires into account. Iteration also reduces the effect of the wire order dependency. Wires are routed in the order they are input.

The details of each of the above steps are described in the following sections.

2.2.1. Segment Decomposition: In this step each wire with more than two logical pins is decomposed into pairs of logical pins. This is done by determining a minimum spanning tree by using Prim's algorithm [24] (formerly we used Kruskal's algorithm [14], but [16] observed that Prim's algorithm is more efficient for complete graphs). A complete graph is constructed, with each logical pin assigned to one node of the graph. The position of each logical pin is set to be the average of all its constituent physical pins. The weights on each edge of the graph are a function of the horizontal and vertical distance between the two nodes of the edge. If the two nodes are separated by H horizontal routing grids (where a routing grid is the routing pitch used in the standard cell system), and C routing channels, then the weight on edge W is given by

$$W = vC + H$$

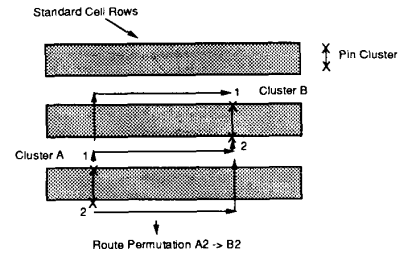

Fig. 1. Permutation decomposition of segment.

where v is a weighting factor for crossing a row. We have found that setting v to 20 gives a good number of tracks with a reasonable number of vertical hops. Prim's algorithm has a running time of $O(n^2)$ in the number of logical pins.

2.2.2. Decomposition into Permutations: Each two-point segment consists of pairs of *pin* clusters that contain electrically equivalent pins. The algorithm considers routes between every pin in one cluster and every pin in the other cluster. Each pair of pin clusters is called a *permutation*. Fig. 1 illustrates three of the four possible permutations between clusters A and B, which have two physical pins each. The four possible permutations are: (A_1, B_1), (A_1, B_2), (A_2, B_1), (A_2, B_2).

It is not always necessary to evaluate every permutation. For example, if clusters A and B occupy the same channels, then it is only necessary to investigate the connections between pins in the same channel—that is, permutations (A_1, B_1) and (A_2, B_2). (Note that only routes inside the bounding box of the wire are evaluated.) In addition, if two pin clusters are separated by a short horizontal distance, then it is likely that the best route joins the two closest physical pins—permutation (A_1, B_2) in Fig. 1. However, if the horizontal distance is larger, then it may be worthwhile (to avoid some heavily congested area) to be able to connect to a cell in a different row, and hence, make it worthwhile to consider all four permutations. These observations have been confirmed experimentally—if pin clusters are separated by less than 300 routing grids (where one grid is the size of the routing pitch) then it is only necessary to evaluate routes between the closest physical pins. Otherwise, all four permutations should be evaluated. This modification reduces the computation time significantly, but leaves the number of tracks the same.

2.2.3. Route Evaluation: The route evaluation step introduces two crucial notions of the LocusRoute algorithm: the cost model, which dictates the cost assigned to a path chosen for a wire, and the method of choosing routes based on paths that have two or less bends.

a) Cost model: Each routing position in a channel (also called *routing grid* of that channel) is represented as one element of an array as shown in Fig. 2. The array, called the *cost array*, has a vertical dimension of the number of rows plus one, and a horizontal dimension of the width of the placement in routing grids. Each element of the cost array, H_{ij}, contains the number of wire routes that

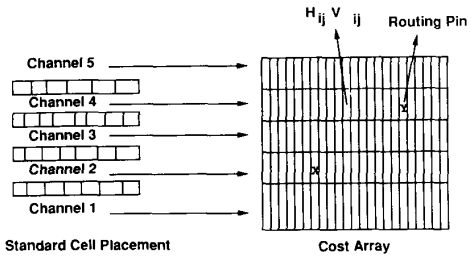

Fig. 2. Cost model.

pass horizontally through channel i in routing grid j. These values change as wires are routed. A wire to be routed is represented as a list of (i, j) pairs of locations in the cost array, corresponding to the locations of pins to be joined. The path of a wire, P, that joins the pins is also a set of (i, j) locations in the cost array. The cost of a path is given by

$$\text{Cost}(P) = \sum_{P} H_{ij} + v \times C$$

where C is the number of cell rows that are crossed in the path, and v is the assigned cost of a row crossing, the same parameter used in the minimum spanning tree decomposition discussed in Section II-2.2.1 above.

This model implies that more than one vertical hop can exist in one grid location, and that the assignment of a vertical hop does not disturb the placement. These assumptions are strictly incorrect. However, when the standard cells contain sufficient built-in feedthrough cells, the placement is disturbed little because few feedthrough cells are inserted. In the case where there are no built-ins, the fact that all the rows will have feedthrough cells keeps the relative positions of the pins similar and tends to minimize the effect of the error.

This type of grid-based cost model is typically used for unconstrained (nonchannel-based) routing problems [29]. Patel [21] applied it to gate arrays. To our knowledge, this is the first use of such an approach in standard cells.

b) Two-bend route generation and evaluation: After each permutation is identified, a low-cost path for a permutation is determined by evaluating the cost of a number of different routes and choosing the best. The approach is to evaluate a subset of all two-bend routes between the two physical pins, and then choose the one with the lowest cost. Generation of two-bend routes is discussed in [20]. Fig. 3 illustrates three possible two-bend (or less) routes inside a representation of the cost array as a small example.

If the horizontal distance between the two pins is H routing grids, and the vertical difference is C channels then the total number of possible two-bend routes is $C + H$. In the LocusRoute algorithm the percentage of all the possible two-bend routes to be evaluated is a parameter. If fewer than 100% of all the routes are to be evaluated, the set of all possible routes is prioritized as follows: first all principally horizontal routes (those with bends only at the left and right extremes) are evaluated. Then the principally vertical routes (those with bends at the upper and lower extremes) are evaluated. Horizontal routes are evaluated first to ensure that all of the potential channels for the route are examined at least once. Within the horizontal and vertical groups, routes are searched in bisection order; i.e., if the limits of the group span are normalized to [0, 1] then the routes are prioritized as 0, 1, 1/2, 1/4, 3/4, 1/8, and so on. This ensures that the possible space of routes is evenly spanned.

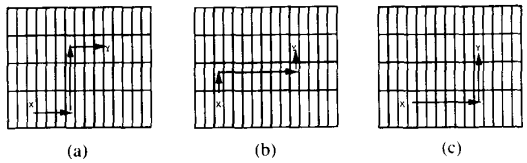

Fig. 3. Sample two-bend routes.

To calibrate the percentage of two-bend routes to be evaluated, the two-bend router was compared against a least-cost path maze router. Both routers were not allowed to go beyond the bounding box of the two end points of the permutation. Experimentally, it was determined that if only 20% of the two-bend routes were evaluated, then this would result in a path as good as that found by the maze router, as compared on the basis of total track count for the entire circuit. On all of the test circuits used in the experiments discussed in the Section III, the LocusRoute router's track count was within 2% of that obtained by the two-point maze router, with one exception of 3.3%. Most of the differences were below 1%. This is surprising in that the maze router looks for not only two-bend routes but for three or more bend routes. It implies that two-bend routes provide a sufficiently rich route set for avoiding the congestion that occurs in standard cell routing.

2.2.4. Reconstruction: In this step the two-point segments are rejoined into multisegment wires. One of the major suboptimalities with considering only two logical pins at a time is that the router may assign two or more tracks to the same network in the same area of a channel—one each for connecting a pin to two other pins. During reconstruction, LocusRoute detects duplicate connections to a single pin and collapses the two (or more) tracks into one.

2.2.5. Wire Recording: The last step in the algorithm is to record the presence of the wire's route in the cost array, so that the cost of using any part of that path will increase for wires routed subsequently. This is done by incrementing the appropriate H_{ij} cells of the cost array. In the succeeding iterations, the wire is "ripped up" by decrementing those same cells of the cost array, before the route evaluation step.

III. Performance Comparisons

This section compares the quality and execution time of LocusRoute with two other routers. Here we wish to demonstrate that LocusRoute is comparable to the best routers in both the number of tracks used and computation time.

TABLE I
QUALITY COMPARISON OF LOCUSROUTE AND TIMBERWOLF

Circuit Name	# Wires	Track Count LocusRoute	TimberWolf 5.0	% Difference
BNRE	420	149	177	+19%
MDC	575	158	170	+8%
BNRD	774	200	218	+9%
Primary1	904	273	260	-5%
BNRC	937	213	241	+13%
BNRB	1364	369	428	+18%
BNRA	1634	373	417	+12%
Test06	1673	381	507	+33%
Primary2	3029	596	558	-6%

TABLE II
EXECUTION TIME OF LOCUSROUTE AND TIMBERWOLF 5.0

Circuit Name	# Wires	Time (DECstation 3100) (s) LocusRoute	TimberWolf 5.0
BNRE	420	3.4	8
MDC	575	5	11
BNRD	774	7	13
Primary1	904	13	25
BNRC	937	10	18
BNRB	1364	25	29
BNRA	1634	31	36
Test06	1673	229	73
Primary2	3029	167	107

3.1. Comparison with TimberWolf

Table I shows a comparison between the LocusRoute global router and the TimberWolf 5.0 [16] global router for nine industrial circuits. These circuits are from several sources: the standard cell benchmark suite (Primary1, Primary2, Test06 [22]), Bell-Northern Research Ltd. (BNRA → BNRE), and the University of Toronto Microelectronic Development Centre (MDC). The placement for all of the circuits was done by the ALTOR standard cell placement program [25], which is a min-cut based approach [5]. Table I gives the number of wires in each circuit, the track count achieved by LocusRoute and TimberWolf, and the percentage difference in tracks between LocusRoute and the TimberWolf 5.0 global router.

To make a fair comparison, the LocusRoute output is passed through ALTOR, so that the post-process step of converting all vertical hops to either built-in feedthroughs or actual feedthroughs is performed and the track counts are exact. For two of the circuits, Primary1 and Primary2 from the standard cell benchmark suite, the TimberWolf router achieves about 6% fewer tracks in LocusRoute. For the other circuits, LocusRoute achieves a range from 8 to 33% fewer tracks than the TimberWolf global router. This is probably due to the difference between Primary1 and Primary2, which have many built-in feedthroughs in the standard cells, while the other circuits have none. In all cases the number of actual feedthrough cells inserted by LocusRoute and TimberWolf were similar, with TimberWolf using slightly more.

Table II gives the execution time for the LocusRoute and TimberWolf runs on a DECstation 3100, which is a 12 MIP machine. The LocusRoute runs use two iterations over all the wires. The execution time for LocusRoute includes the time for input and output of the global routing. For the TimberWolf runs, it is the entire time from input of the placement file, costing, and global routing (no placement is done). Table II indicates that the runtimes of LocusRoute and TimberWolf are very similar.

3.2. Comparison with Maze Router

For comparison purposes a maze router [15] was developed, using the same cost model as LocusRoute, which determines exhaustively the optimal solution to the two-point routing problem. It also determines a good approximation to the minimum-cost Steiner tree for multipoint wires using the approach described in [2]. The maze router was carefully optimized for speed. Table III shows the comparison of track count and execution time for the maze router and LocusRoute. Each router used the same number of vertical hops. The post-process step of converting vertical hops to feedthroughs or built-ins is not done here because both algorithms use vertical hops and thus can be fairly compared without the conversion. Execution times are for two iterations over all wires on a DECstation 3100.

TABLE III
COMPARISON OF LOCUSROUTE AND MAZE ROUTER

Circuit Name	Track Count Locus	Maze	Diff	Time (DECstation 3100 s) Locus	Maze	Factor
BNRE	133	128	+4%	3.4	87	26x
MDC	144	140	+3%	5	118	24x
BNRD	181	178	+2%	7	121	17x
Primary1	266	259	+3%	13	236	18x
BNRC	193	190	+2%	10	272	27x
BNRB	311	301	+3%	25	560	22x
BNRA	304	295	+3%	31	731	24x
Test06	324	308	+5%	229	3381	15x
Primary2	563	559	+1%	167	1817	11x

For all circuits the LocusRoute track count is no greater than 5% more than that achieved by the maze router, and in some cases is as little as 1% more. Most of this difference is due to the suboptimality of dividing the wires up into two point wires. The speed of LocusRoute is from 11 to 27 times faster than the maze router. Since the purpose of the router is to evaluate a placement in a placement algorithm, we will always be willing to trade this slight loss in quality for such a large gain in speed, allowing many more placements to be evaluated in the same amount of time.

IV. PARALLEL DECOMPOSITION AND IMPLEMENTATION

The previous section makes the case that the uniprocessor LocusRoute algorithm is competitive with modern global routers in both quality and time. Our purpose is to determine how fast this basic algorithm can be made to run on a parallel processor.

We are willing to consider a tradeoff between execution time and quality of the final answer, because, if a parallel decomposition allows a significantly faster routing with only a small loss of quality, then an automatic placement process would be able to evaluate many more placements.

One such approach will be discussed in Section IV-4.1. This kind of tradeoff is possible to consider because we have a particular end purpose in mind—in general this is not possible in parallel processing where the exact same (correct) answer must be guaranteed.

Prior work on parallel routing [6], [4], [1], [18], [30], [12], [35] has focused on fixed hardware implementations of the maze routing algorithm [15]. A more flexible approach is to use general purpose parallel processors, which can be adapted to many applications. Also, using the flexibility of a general purpose multiprocessor, different parallel decompositions can be exploited on the same machine. If these decompositions are *orthogonal*, and they are used in tandem, then they can achieve significant speedup. Two approaches to parallelizing an algorithm are said to be othogonal if, when used together, the resulting speedup is the product of the speedup of the individual methods. In this section, three orthogonal parallel decompositions of the LocusRoute router are proposed, implemented, and measured for performance.

1) *Wire-by-wire parallelism:* Each processor is given an entire multipoint wire to route.
2) *Segment-based parallelism:* Each two-point segment produced by the minimum spanning tree decomposition is routed in parallel.
3) *Route-based parallelism:* Potential routes for each *permutation* of a segment (described in Section IV-4.3) are evaluated in parallel.

The following sections give the details of the three parallel approaches, their performance, and a quantitative measure of the degradation in quality if there is some. All decompositions assume a shared-memory multiprocessor.

4.1. Wire-by-Wire Parallelism

In the wire-by-wire parallel approach, each processor routes a different wire at the same time, using the algorithm described in Section II. The cost array is stored in shared memory and all processors access and change this structure simultaneously. The general flow of this parallel decomposition is shown in Fig. 4. The master processor initializes the cost array, and sets up each individual wire as a task on one central task queue. All processors then remove wire tasks from the queue and execute the LocusRoute algorithm—reading the shared cost array to evaluate the routes for each wire and then updating the cost array with the best route that is found.

This wire-by-wire approach does not exactly reproduce the serial LocusRoute algorithm. In the sequential algorithm there is a *data dependency* which dictates that wires be routed sequentially: after the wire is routed, its presence is recorded in the cost array so that subsequent wires may take the new path into account. In the wire-by-wire parallel approach, we *relax* this data dependency by allowing several wires to be routed simultaneously. This kind of approach is called *chaotic* parallelism—where data dependencies are relaxed under the assumption that the data used will only be marginally different from the correct data, and thus will still lead the system toward a good solution.

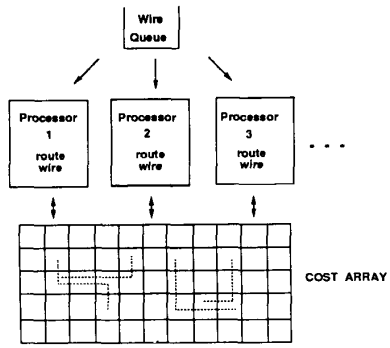

Fig. 4. Wire-by-wire parallel decomposition.

In a chaotic approach, iteration is typically used so that the system converges to the same answer as the sequential program. For example, in the parallel simulated annealing work for placement of standard cells [26], [27], [8], [9], [3] the iteration of simulated annealing overcomes the error due to the chaotic parallel approach. While LocusRoute does iterate over the solution there is no guarantee that this kind of iteration will converge to the same answer as the sequential algorithm. Hence, there is a degradation in the quality of the final answer. We are willing, as discussed earlier, to trade some amount of quality for increased speed, for the end purpose of combined placement and routing. Quantitative measurements of the amount of degradation are given below, in Section IV-4.1.3. The following section describes a technique for reducing the degradation.

4.1.1. Geographic Wire Assignment (GWA): It is possible to reduce the quality degradation by taking steps to ensure that fewer data dependencies are violated. In this approach, called *geographic wire assignment* (GWA), each processor is assigned wires that reside in a specific area of the chip. In the ideal case, a processor will change areas of the cost array that only it reads, and so no data dependencies are violated. To do this, a task scheduler must assign wires to processors that do not overlap in space at any time during the evaluation phase of computation. Unfortunately, this assignment requires too much computation, and can drastically reduce the amount of parallelism, since there may not be many totally independent wires.

A simpler version of this approach has been implemented—the chip is divided into distinct geographic areas, one for each processor. Each area has a task queue associated with it, and the processor takes tasks out of that queue. A wire is assigned to a queue if the position of its leftmost pin falls within the assigned area of that task queue. This does not generate totally independent tasks, because wires from different task queues/processors can still overlap in space and time, but it does reduce the amount of overlap.

This approach causes an imbalance of workload among the processors, because each queue may not represent the same amount of work. To prevent loss of performance when a queue becomes empty, the processor associated with that queue searches for an nonempty queue. It chooses among the available queues to balance the number of processors working on all the queues. The geographic wire assignment approach was implemented and resulted in significant improvement in the final quality of the routing, without loss of performance, as shown in Section IV-4.1.3.

4.1.2. Elimination of Synchronization Overhead: In the wire-by-wire approach, several processors may be changing the cost array at the same time—incrementing the cost array when the best route is found, or decrementing it during rip-up for the second and further iterations. In strict parallel programming, the cost array must be *locked* to provide exclusive access to the array during the update operations. Without locking, if two processors increment or decrement the same cost array element at the same time, then one of those operations will be lost. This synchronization, however, reduces the number of operations that can be performed at the same time, degrading performance.

For typical circuits, however, there are thousands of cost array elements. The chances of two processors operating on the same element at the same time is very small. This chance is further reduced in the geographic wire assignment approach. If such operations do collide, all that happens is that the cost array element would be wrong by an amount of one. In the LocusRoute algorithm, this kind of error is unlikely to change the results because the routes are evaluated based on many cost array elements. As such, we eliminated the locking of the cost array—this results in increased performance and no loss in quality, as shown in the following section.

4.1.3. Wire-by-Wire Performance Measurements: Fig. 5 is a plot of the speedup versus number of processors for the 3029-wire (Primary2) example running on an sixteen-processor Encore MULTIMAX, using the geographic wire-assignment approach, and not locking the cost array. The speedup for p processors, S_p, is calculated as T_1/T_p, where T_1 is the execution time on one processor and T_p is the execution time using p processors. The Encore uses the National 32032 microprocessor, in our benchmarks, which is about the same speed as a DEC VAX 11/780.

It is clear from the figure that the wire-based approach achieves excellent speedup—14.6 times faster using 16 processors. Note that the execution time is only the actual routing computation time, and excludes the input time. We exclude input time because the router is intended to be used inside a placement program, and it is the actual routing, not the input, that needs to be fast for the placement optimization process.

Fig. 6 is a plot of the number of tracks achieved by the router versus the number of processors, for two approaches—using geographic wire assignment and not

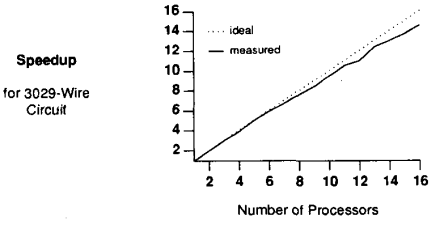

Fig. 5. Wire-based speedup for circuit Primary2.

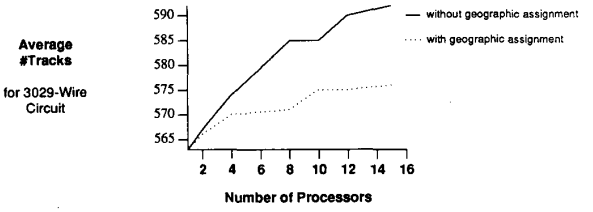

Fig. 6. Tracks versus number of processors for Primary2.

using geographic wire assignment. The number of tracks are averaged over 10 runs for each of the number of processors. The geographic assignment reduces the quality degradation by a significant amount—cutting the increase in tracks by about half, without loss of performance. In fact, the performance improves slightly in the geographic approach because it exhibits more data locality which improves the performance of the processor caches.

Table IV gives the speedup using fifteen processors for the other test circuits. The speedup ranges from 5 for a smaller circuit to 13.8 for the largest. The speedup is less for smaller circuits because they are done in such a short time, so that the startup overhead and workload balance become factors. The execution time is for two iterations over all the wires.

Table V gives the track and vertical hop counts for 1 and 15 processors using wire-by-wire parallelism and geographic wire assignment, averaged over 10 runs. The degradation in track count ranges between 2–10%, mostly less than 5%. The increase in vertical hops is 9% or less. The larger circuits exhibit less degradation because the geographic wire assignment has a larger area to spread the processors over, and they are less likely to interact.

a) Gain due to removal of locks: Table VI gives measurements of speed and quality comparing the performance of the wire-by-wire approach with and without locking of the cost array as discussed in Section IV-4.1.2. The table gives the execution time, number of tracks, and vertical hops averaged over 10 runs for the circuits Primary1 and Primary2, using 15 processors. It also gives the standard deviations (SD) for the track and vertical hop counts. For the Primary1 circuit the speedup decreased from 10.4 to 7.3 using 15 processors when cost array locking was used. For the Primary2 circuit the speedup for 15 processors was reduced from 13.5 to 12.4 due to locking. The final routing quality, however, does not de-

TABLE IV
WIRE-BASED PARALLELISM SPEEDUP

Circuit Name	1-Processor Time (s)	15-Processor Time (s)	15-Processor Speedup
BNRE	29	4.4	6.6
MDC	40	8.1	4.9
BNRD	58	5.4	10.8
Primary1	125	11.6	10.7
BNRC	92.4	10.8	8.7
BNRB	246	21.5	11.4
BNRA	298	25.4	11.7
Test06	2484	190	13.1
Primary2	1772	128	13.8

TABLE V
WIRE-BASED PARALLELISM QUALITY

Circuit Name	Track Count			Vertical Hops		
	1-Proc	15-Proc	%Increase	1-Proc	15-Proc	%Increase
BNRE	133	138	4%	460	490	7%
MDC	134	148	10%	253	262	4%
BNRD	180	187	3%	537	577	7%
Primary1	265	270	2%	942	992	5%
BNRC	193	199	2%	762	828	9%
BNRB	310	318	3%	1927	2015	5%
BNRA	304	311	2%	2129	2191	3%
Test06	323	338	5%	3238	3319	3%
Primary2	563	576	2%	3057	3132	3%

TABLE VI
SPEED AND QUALITY USING AND NOT USING LOCKS FOR 15 PROCESSORS

Circuit & Lock Type	Avg T (s)	Track Count Avg.	Track Count SD	Vertical Hops Avg	Vertical Hops SD
Primary1 Locks	17.1	270	2.8	983	13
Primary1 NO Locks	12.0	272	3.3	992	20
Primary2 Locks	143	591	1.9	3159	15
Primary2 NO Locks	131	592	4.0	3168	15

crease when locking is omitted, due to the low probability of collisions as discussed in Section IV-4.1.2.

4.2. Segment-Based Parallelism

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. Measurements of the sequential router showed that about 60% of the routing time was spent on wires with more than one segment. This means that a speedup of about two might be expected using three processors. However, even though there are many wires that provide two or three-way parallel tasks, the *size* of those tasks are unequal. The amount of time taken by the LocusRoute to route between two points is proportional to the Manhattan distance between the two points. If, in a three-point wire, two of the points are close together and the third is far away, it will then take much longer to route one segment than the other. The processor assigned to the short segment will be idle while the longer one is being routed. This unequal load prevents a reasonable speedup. On the test circuits a speedup of about 1.1 using two processors was measured.

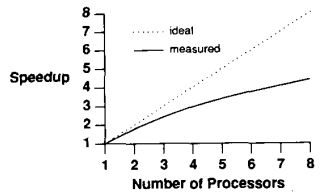

Fig. 7. Route-based speedup for Primary2.

It is fairly clear, however, that an extra processor could be assigned to a *number* of processors that are routing different wires. It is likely that at any given time, one of them will be able to use the extra processor to route an extra segment. This technique would become essential in wire-based parallelism if the number of processors were increased much beyond sixteen. In that case, the load balance becomes a problem because wires with many segments take much longer than wires with few segments. Hence, segment-based parallelism could be used as a method to balance those workloads.

4.3. Route-Based Parallelism

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 processors finish, the route with the best overall cost is selected. Here the processor loads are well balanced because the routes are all of the same length, and the number of routes is evenly divided among the processors.

Fig. 7 is a plot of the speedup versus number of processors for the circuit Primary2, for the route-based approach. It achieves a speedup of 4.4 using 8 processors.

Table IV gives the best speedup achieved for all of the test circuits, ranging from 1.2 using 2 processors to 6 using 10 processors. The number of processors given in the table is at the "knee" of the speedup curve—the point where increasing the number of processors does not appreciably improve performance. The principal reason for the limitation is speedup is the sequential portion of the routing: the wire decomposition and the post-route processing that places the presence of the route into the cost array. On the small circuits that have lesser speedup, the sequential portion is about 50% of the total routing time, while on the larger circuits, which have better speedup, the sequential portion ranges from 10 to 15%.

V. COMBINING TWO ORTHOGONAL PARALLEL DECOMPOSITIONS

The wire and route parallel approaches described above are orthogonal, and so when they are combined we can expect a multiplication of their respective speedups. In this section experiments are performed to demonstrate this effect on the Encore MULTIMAX, including a discussion and experiments on scheduling of the two kinds of tasks. Using a simple model, the speedup for a larger number of processors is predicted.

TABLE VII
PERFORMANCE OF ROUTE-BASED PARALLELISM

Circuit Name	Best Route-based Speedup (Speedup/#Processors)
BNRE	1.2/2
MDC	1.2/2
BNRD	1.3/2
Primary1	1.8/3
BNRC	1.5/3
BNRB	2.2/4
BNRA	2.1/4
Test06	6.0/10
Primary2	4.1/7

Note that there is a significant difference between the two kinds of parallelism: the wire-by-wire approach causes a degradation in the quality of the results that is an increasing function of the number of processors. The route-by-route parallel decomposition reproduces the sequential algorithm exactly. As such it is important to control the number of processors using the wire-by-wire approach, so that there is control on the quality of the answer. For this reason, the number of wires allowed to be routed at the same time is an input parameter.

5.1. *Implementation on the MULTIMAX*

Because there are different kinds of tasks to be executed, the major challenge of combining the wire and route approaches is the *scheduling* of those tasks. We present two approaches: a static schedule which permanently assigns processors to a wire, and a dynamic one which allows processors to be applied wherever they are needed.

5.1.1. Static Schedule: This scheduling strategy is implied by the notion of orthogonality: for each wire that is being routed by one processor in the wire-based approach, we now design a constant number of processors to aid in the parallel execution of the route-based tasks. This scheme is depicted in Fig. 8. The extra processors are used only during the two-bend route generation and evaluation. The approach is implemented by assigning a separate task queue to each wire stream, from which the route processors remove route-based tasks.

Several experiments were performed to show that the combined speedup of the wire and route-based approaches was indeed the multiplication of the individually measured speedups. Table VIII gives the result of those experiments for the 3029-wire Primary2 circuit. For each experiment it gives the number of wires being routed in parallel (M), the number of processors assigned to each wire to do the routing tasks (N), the total number of processors ($M \times N$), the speedup predicted by multiplying the wire-based speedup using M processors and the route-based speedup using N processors (from Tables IV and VII), and the *measured* combined speedup. From this table it is clear that the speedups very nearly multiply. The small difference is due to increased contention for shared memory and the central bus, the fact that two processors contend for one cache in the Encore MULTIMAX, and the increased cache coherence traffic that occurs with more processors.

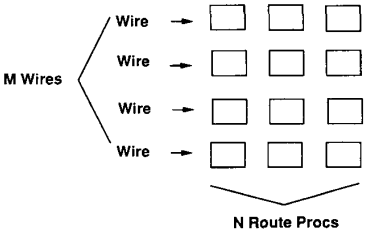

Fig. 8. Static scheduling policy.

TABLE VIII
STATIC SCHEDULE EXPERIMENTS FOR CIRCUIT PRIMARY2

# Wires in Parallel (M)	#Route Procs Per Wire (N)	Total (M × N)	Speedup Predicted	Speedup Measured
3	4	12	8.9	8.5
4	3	12	9.5	9.3
6	2	12	10.6	10.1
3	5	15	10.2	9.1
5	3	15	12.2	10.9
7	2	14	12.0	11.8

5.1.2. Dynamic Schedule: A drawback of the static scheduling policy is that it cannot assign processors where they will be of best use. If one wire has very few routes while another has many, the processors assigned to the first are not used by the second. In addition, there is a portion of the wire routing procedure that only uses one processor, so the other processors will be idle. A dynamic scheduling approach allows any idle processor to be used by any wire that has a need for it. This was implemented by assigning all "route-by-route" tasks from any wire to a single task queue, from which any processor performing route tasks can retrieve tasks. Here M wire processors add tasks to the queue, and R route processors remove and execute tasks, where M and R are parameters specified by the user—a total of $M + R$ processors are used. The *granularity* of the routing tasks in the dynamic scheme, the number of two-bend routes assigned to one processor to evaluate per task, was tuned to achieve the best speedup. The best performance was achieved when the number of tasks was several times the number of available processors, indicating that the load balance effect was more significant than the overhead of starting up a task.

Table IX gives an experimental comparison of the static and dynamic scheduling approaches, also for circuit Primary2. In each experiment, the total number of processors is the same—the only difference is the scheduling discipline. The table gives the number of wires being routed simultaneously (M), the number of additional processors used to help in routing tasks (R), the total number of processors ($M + R$), the speedup for the static and

TABLE IX
STATIC VERSUS DYNAMIC PERFORMANCE FOR CIRCUIT PRIMARY2

# Wires in Parallel (M)	#Route Procs (R)	Total (M + R)	Measured Speedup Static	Dynamic	% Increase
3	9	12	8.5	9.6	13%
4	8	12	9.3	10.4	11%
6	6	12	10.1	10.5	5%
3	12	15	9.1	10.7	17%
5	10	15	10.9	13.0	19%
7	7	14	11.8	12.8	9%

dynamic scheduling approach, and the percentage increase in speedup of the dynamic approach over the static.

The improvement in performance of dynamic scheduling over static scheduling ranges from 5 to 19% for the six examples. The difference is greater when more route processors are involved, because there is more opportunity to take advantage of the inefficiency of static scheduling.

5.2. *Performance Prediction on Larger Number of Processors*

Since we have experimentally demonstrated that the static schedule performance of the combined approach does indeed nearly multiply the speedups attained by the individual methods, it is possible to predict the performance of the static schedule on many more processors. Assume, for a given circuit that a speedup of S_M is achieved using wire-based parallelism on M processors, and a speedup of S_N is achieved using route-based parallelism on N processors. Then, because the two approaches are orthogonal, the resulting speedup when they are used together should be $S_W \times S_N$ using $M \times N$ processors. This model neglects the effect of memory contention that may occur when the number of processors is increased dramatically. Table X shows the predicted speedup for the test circuits, using the data from Tables IV and VII. Combined speedup ranges from 6 using 30 processors to 79 using 150 processors. The smaller circuits are routed very quickly and so it is difficult to get speedups greater than 6 due to the startup overhead, and load balance effects. The larger circuits benefit greatly from the combination of the approaches. In addition, we can expect even better performance using the dynamic scheduling approach.

VI. CONCLUSIONS

A new global routing algorithm for standard cells and its parallel implementation has been presented. The LocusRoute algorithm is a simple and effective way of searching for good paths for routes in the standard cell layout style, yet it is comparable in speed and quality to the TimberWolf 5.0 standard cell global router. It achieves similar quality to a much slower maze router that searches the same space of possible routes. Two of the three parallel decompositions that were implemented achieve significant speedup—up to 13.8 using fifteen processors and 6 using 10 processors. They should produce combined speedups of more than 75 times.

TABLE X
PREDICTED COMBINED SPEEDUP OF WIRE AND ROUTE PARALLELISM

Circuit	$\frac{S_M}{M}$	$\frac{S_N}{N}$	$\frac{S_M \times S_N}{M \times N}$
BNRE	$\frac{6.6}{15}$	$\frac{1.2}{2}$	$\frac{8}{30}$
MDC	$\frac{4.9}{15}$	$\frac{1.2}{2}$	$\frac{6}{30}$
BNRD	$\frac{10.8}{15}$	$\frac{1.3}{2}$	$\frac{14}{30}$
Primary1	$\frac{10.7}{15}$	$\frac{1.8}{3}$	$\frac{19}{45}$
BNRC	$\frac{8.6}{15}$	$\frac{1.5}{3}$	$\frac{13}{45}$
BNRB	$\frac{11.4}{15}$	$\frac{2.2}{4}$	$\frac{25}{60}$
BNRA	$\frac{11.7}{15}$	$\frac{2.1}{4}$	$\frac{25}{60}$
Test06	$\frac{13.1}{15}$	$\frac{6}{10}$	$\frac{79}{150}$
Primary2	$\frac{14.6}{16}$	$\frac{4.1}{7}$	$\frac{60}{112}$

We introduce a number of techniques to achieve these results.

i) A chaotic parallel approach in which data dependencies are relaxed to allow a tradeoff between execution time and quality.
ii) Careful assignment of tasks to processors to reduce the quality degradation of the chaotic approach.
iii) Elimination of costly locking with no loss of quality, due to the low statistical chance of interactions.
iv) Using two *orthogonal* parallel decompositions, which when combined give a speedup which is the product of the individual approaches' speedups.
v) Dynamic (as opposed to static) scheduling of parallel tasks to improve the parallel performance.

In the future, we hope to integrate the global router into a placement algorithm and see how the added information can be used to improve the placement. Several issues that have been raised by the parallel router will also be pursued: theoretical study of the effect of relaxing data dependencies in parallel combinatorial optimization; more flexible scheduling; parallelizing more of the algorithm such as the minimum spanning tree decomposition; and implementation of a similar approach on a massively parallel machine.

ACKNOWLEDGMENT

The author is grateful to John Hennessy for the encouragement and support of this work, and to Tom Blank for many good suggestions of an earlier version of this paper. Thanks also to Grant Martin of Bell-Northern Research for the use of company circuits and to the people involved in the standard cell benchmark effort for supplying those test circuits. Carl Sechen provided the version of 5.0 of TimberWolfSC.

REFERENCES

[1] H. G. Adshead, "Employing a distributed array processor in a dedicated gate array layout system," in *Proc. ICCC*, Sept. 1982, pp. 411–414.

[2] S. B. Akers, *Design Automation of Digital Systems; Theory and Techniques*, M. A. Breuer, Ed. Englewood Cliffs, NJ: Prentice-Hall, 1972, chap. 6.
[3] P. Banerjee and M. Jones, "A parallel simulated annealing algorithm for standard cell placement on a hypercube computer," in *Proc. ICCAD '86*, Nov. 1986, pp. 34–37.
[4] T. Blank, M. Stefik, and W. VanCleemput, "A parallel bit map processor architecture for DA algorithms," in *Proc. 18th Design Automation Conf.*, June 1981, pp. 837–845.
[5] M. A. Breuer, "Min-cut placement," *J. Design Automat. Fault-Tolerant Comput.*, pp. 343–362, Oct. 1977.
[6] M. A. Breuer and K. Shamsa, "A hardware router," *J. Digital Syst.*, vol. IV, no. 4, pp. 393–408, 1981.
[7] M. Burstein and S. J. Hong, "Hierarchical VLSI layout: Simultaneous placement and wiring of gate arrays," in *Proc. VLSI '83*, 1983, pp. 45–60.
[8] A. Casotto, F. Romeo, and A. Sangiovanni-Vincentelli, "A parallel simulated annealing algorithm for the placement of macro-cells," in *Proc. ICCAD 86*, Nov. 1986, pp. 30–33.
[9] ——, "A Parallel Simulated Annealing Algorithm for the Placement of Macro-Cells," *IEEE Trans. Computer-Aided Design*, vol. 6, pp. 838–847, Sept. 1987.
[10] J. Cong and B. Preas, "A new algorithm for standard cell global routing," in *Proc. ICCAD 88*, Nov. 1988, pp. 176–179.
[11] W-M. Dai and E. S. Kuh, "Simultaneous floorplanning and global routing for hierarchical building-block layout," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 828–837, Sept. 1987.
[12] A. Iosupovici, "A class of array architectures for hardware grid routers," *IEEE Trans. Computer-Aided Design*, vol. CAD-5, pp. 245–255, Apr. 1986.
[13] T. Kambe, T. Okada, T. Chiba, and I. Nishioka, "A global routing scheme for polycell LSI," in *Proc. ISCAS*, 1985, pp. 187–190.
[14] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," in *Proc. Amer. Math. Soc.*, vol. 7, 1956, pp. 48–50.
[15] C. Y. Lee, "An algorithm for path connections and its applications," *IRE Trans. Electron. Comput.*, vol. EC-10, pp. 346–365, 1961.
[16] K-W. Lee and C. Sechen, "A new global router for row-based layout," in *Proc. ICCAD 88*, Nov. 1988, pp. 180–183.
[17] M. J. Lorenzetti and D. S. Baeder, "Routing," in *Physical Design Automation of VLSI Systems*, B. Preas and M. Lorenzetti, Ed. Menlo Park, CA: Benjamin/Cummings, 1988, chap. 5.
[18] R. Nair, S. J. Hong, S. Liles, and R. Villani, "Global wiring on a wire routing machine," in *Proc. 19th Design Automation Conf.*, pp. 224–231, June 1982.
[19] R. Nair, "A simple yet effective technique for global wiring," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 165–172, Mar. 1987.
[20] A. P.-C. Ng, P. Raghavan, and C. D. Thompson, "A language for describing rectilinear Steiner tree configurations," in *Proc. 23rd Design Automation Conf.*, pp. 659–662, June 1986.
[21] A. M. Patel, N. L. Soong, and R. K. Korn, "Hierarchical VLSI Routing—An approximate routing procedure," *IEEE Trans. Computer-Aided Design*, vol. CAD-4, pp. 121–126, Apr. 1985.
[22] B. T. Preas, "Benchmarks for cell-based layout systems," in *Proc. 24rd Design Automation Conf.*, June 1987, pp. 319–320.
[23] ——, private communication.
[24] R. Prim, "Shortest connecting networks and some generalizations," *Bell. Syst. Tech. J.*, vol. 36, pp. 1389–1401, 1957.
[25] J. S. Rose, W. M. Snelgrove, and Z. G. Vranesic, "ALTOR: An automatic standard cell layout program," in *Proc. Canadian Conf. on VLSI*, Nov. 1985, pp. 168–173.
[26] J. S. Rose, D. R. Blythe, W. M. Snelgrove, and Z. G. Vranesic, "Fast, high quality VLSI placement on an MIMD multiprocessor," in *Proc. ICCAD 86*, Nov. 1986, pp. 42–45.
[27] J. S. Rose, W. M. Snelgrove, and Z. G. Vranesic, "Parallel standard cell placement algorithms with quality equivalent to simulated annealing," *IEEE Trans. Computer-Aided Design*, vol. 7, pp. 387–396, Mar. 1988.
[28] J. S. Rose, "LocusRoute: A parallel global router for standard cells," in *Proc. 25th Design Automation Conf.*, June 1988, pp. 189–195.
[29] F. Rubin, "The Lee path connection algorithm," *IEEE Trans. Comput.*, vol. C-23, pp. 907–914, Sept. 1974.
[30] R. A. Rutenbar, T. N. Mudge, and D. E. Atkins, "A class of cellular architectures to support physical design automation," *IEEE Trans. Computer-Aided Design*, vol. CAD-3, pp. 264–278, Oct. 1984.
[31] C. Sechen and A. Sangiovanni-Vincentelli, "The Timberwolf placement and routing package," *IEEE J. Solid-State Circuits*, vol. SC-20, pp. 510–522, Apr. 1985.
[32] C. Sechen, D. Braun, and A. Sangiovanni-Vincentelli, "TimberWolf: A complete standard cell layout package," *IEEE J. Solid-State Circuits*, vol. 23, pp. 410–420, Apr. 1988.
[33] P. R. Suaris and G. Kedem, "A quadrisection based combined place and route scheme for standard cells," *IEEE Trans. Computer-Aided Design*, vol. 8, pp. 234–244, Mar. 1989.
[34] K. Supowit, "Reducing channel density in standard cell layout," in *Proc. 20th Design Automation Conf.*, June 1983, pp. 263–269.
[35] Y. Won, S. Sahni, and Y. El-Ziq, "A hardware accelerator for maze routing," in *Proc. 24th Design Automation Conf.*, June 1987, pp. 800–806.
[36] M. Yamada, T. Hiwatashi, T. Mitsuhashi, and K. Yoshida, "A multilayer router for standard cell LSIs," in *Proc. ISCAS*, 1985, pp. 191–194.

*

Jonathan Rose (S'79-M'86), for a photograph and biography please see page 259 of the March 1990 issue of this TRANSACTIONS.