LocusRoute: A Parallel Global Router for Standard Cells

Jonathan Rose
Computer Systems Laboratory
Center for Integrated Systems
Stanford University, Stanford CA 94305

Abstract

A fast and easily parallelizable global routing algorithm for standard cells and its parallel implementation is presented. LocusRoute is meant to be used as the cost function for a placement algorithm and so this context constrains the structure of the global routing algorithm and its parallel implementation. The router is based on enumerating a subset of all two-bend routes between two points, and results in 16% to 37% fewer total number of tracks than the TimberWolf global router for standard cells [Sech85]. It is comparable in quality to a maze router and an industrial router, but is factor of 10 times or more faster. Three approaches to parallelizing the router are implemented: wire-by-wire parallelism, segment-by-segment and route-by-route. Two of these approaches achieve significant speedup - route-by-route achieves up to 4.6 using eight processors, and wire-by-wire achieves from 5.8 to 7.6 on eight processors.

1 Introduction

The best way to evaluate a given placement of circuit modules is to route it and determine the final area. Since routing is a time-consuming task typical placement algorithms [Hana72,Breu77] use other metrics such as total wire length or crossing counts that are easier to calculate. The advent of usable commercial multiprocessors is leading us to consider using more compute-intensive cost functions *if* efficient parallel algorithms can be developed. The aim of the *Locus Project* is to integrate placement and routing into one optimization process, and to do this by using multiprocessing to increase the speed of the routing.

This paper presents the first step in the Locus Project: *LocusRoute*, a new global routing algorithm for standard cells, and its parallel implementation. Our goal is to make the average routing time for one net close to the time that it takes to recalculate more conventional cost functions such as that used in the TimberWolf [Sech85] Simulated Annealing algorithm. The intention is for the global router to be invoked to rip-up and re-route wires whose end points have changed when one or more cells are moved in an iterative improvement placement scheme. This means that routing time must be about one to five milliseconds per net on a VAX 11/780-class machine.

The routing performance of LocusRoute, as measured by total number of routing tracks, is better than that of TimberWolf [Sech85] and comparable to a maze router and an industrial router. It is fast because it investigates only a subset of two-bend routes between pairs of pins to be routed. The routing speed is increased further by parallelizing the algorithm in three ways: routing several wires at once, routing several two-point segments simultaneously, and evaluating possible two-bend routes in parallel. The wire-by-wire parallel approach achieves speedups ranging from 5.8 to 7.6 using 8 processors. The route-by-route approach achieves speedups of up to 4.6 using 8 processors. Since these two "axes" of parallelism are *orthogonal* to each other, their respective speedups will multiply.

This paper is organized as follows: Section 2 reviews related work. Section 3 defines the problem of global routing and gives our routing model. Section 4 describes the LocusRoute algorithm and compares it to other routers. Section 5 presents three approaches for speeding up the new router using parallel processing, and performance results.

2 Related Work

Previous work on parallel routing [Breu81, Blan81, Rute84, and many others] has generally focused on a fixed hardware mapping for the Lee routing algorithm [Lee61]. As such they lack the flexibility that is required in practical CAD software such as the global router described in [Kamb85]. 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. A more flexible approach is to use general purpose parallel processors, which can be adapted to many applications. Using the flexibility of a general purpose multiprocessor, several "axes" of parallelism can be exploited. If these axes are *orthogonal* to each other then when used together they can provide significant speedup. Two approaches to parallelizing an algorithm are said to be orthogonal if, when used together, the resulting speedup is the product of the speedup of the individual methods.

3 Problem Definition and Routing Model

Global routing for standard cells first decides for each group of electrically equivalent pins (pin clusters) which of those pins are actually to be connected. Second, 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. Lastly, it must determine the channel to use in routing from a pad into the core cells.

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. This does result in some inaccuracy in the track count. However, using this approximation (and the routing algorithm to be described) the 904-wire Primary1 circuit from the standard cell benchmark suite [Prea87] global routed to 249 tracks, using 995 vertical hops. The actual, post-process track count using 10 feedthrough cells and 985 built-ins was 253, only 1.6% more tracks. For the 3029-wire Primary2 circuit with 3424 vertical hops (287 feedthroughs, 3137 built-ins) the approximate track count was 546 and the post-process count was 590, an increase of 8%.

The usual objective of a global router is to minimize the sum of the channel densities of all the channels (hereafter called the *total density*). It is important to note that the total density can be traded off with the number of vertical hops, so to compare the total density of two global routings fairly they should both use the same number of vertical hops.

CH2540-3/88/0000/0189$01.00 © 1988 IEEE

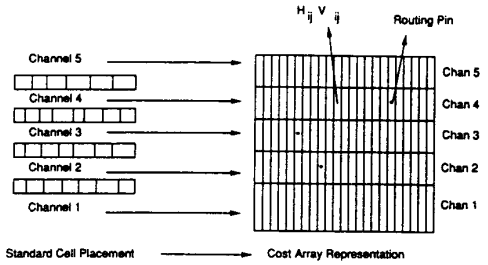

Figure 1 - *Routing Model*

3.1 Routing Model

All of the routing algorithms discussed here are based on the same routing model: Each possible routing position in a channel (also called *routing grid* of that channel) is represented as one element of an array as shown in Figure 1. 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 contains two values: H_{ij} and V_{ij}. H_{ij} contains the number of of wire routes that pass horizontally through the grid at channel i in position j. V_{ij} is the cost, assigned by parameter, of traversing a row in travelling from channel i to channel $i + 1$ at grid position j. A wire 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 objective is to find a minimum-cost path for each wire. The wire's cost is given by the sum of all of the H_{ij} and V_{ij} that it traverses. After a path is found for a wire that goes through location (i, j) its presence is recorded in the Cost Array (the appropriate H_{ij} and V_{ij} are incremented) so that subsequent wires can take it into account. The more wires going through a particular location in a channel, the less likely it is that area will be used. Note that in this model the total density is not directly minimized, but rather a combination of average density and wire length.

4 The LocusRoute Algorithm

In this section the uniprocessor LocusRoute algorithm is described, and a performance comparison with other routers is given. There are five steps in the LocusRoute global routing algorithm:

1. A multi-point wire is decomposed into two-point *segments*, using Kruskal's algorithm [Krus56]. This algorithm has running time $O(n^2)$ in the number of pin clusters. The effect of the sub-optimality of this decomposition is discussed in section 4.4 below.

2. The segments are further decomposed, if necessary, into *permutations*, which are the set of possible routes between each pin in a pin cluster.

3. A low-cost path in the Cost Array is found for each permutation by evaluating a subset of the two-bend routes between each pin pair. The permutation with the best cost is selected as the route for that segment.

4. Traceback. This step joins all the segments back together, 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. Wire *lay down*. The presence of the newly routed wire is put into the Cost Array by incrementing the array elements where the new wire resides. Once there, other wires can take it into account.

The details of the second and third steps are described in the following sections. The first and last two are simple enough that the above description suffices.

4.1 Decomposition Into Permutations

Each two-point segment consists of pairs of *pin clusters* that contain electrically equivalent pins. The LocusRoute algorithm considers routes between every pin in one cluster and every pin in the other cluster. Each such route is called a *permutation*. Figure 2 illustrates three of the four possible permutations between clusters A and B, which have two pins each. The four possible permutations are: (A_1, B_1), (A_1, B_2), (A_2, B_1), (A_2, B_2). If clusters A and B are separated by only a short horizontal distance, then the (A_1, B_2) permutation is most likely the least-cost path of the four. If the horizontal distance is large then it is possible that any one of the four permutations could have the low-cost path, and hence all should be investigated. This has been confirmed experimentally, and a constant horizontal separation (300 routing grids) has been determined beyond which total density will improve if all four permutations are evaluated.

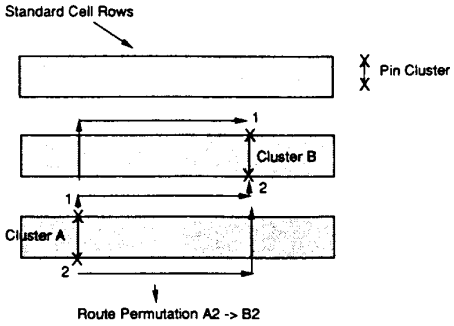

Figure 2 - *Permutation Decomposition of Segment*

4.2 Route Enumeration

The LocusRoute algorithm searches for a low-cost path for a permutation by *enumerating* a number of different routes. The idea is to evaluate the cost of a subset of all two-bend routes between the two pins, and then choose the one with the lowest cost. Generation of two-bend routes is discussed in [Ng86]. Figure 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 two-bend routes is $C+H$. A parameter, called the *two bend percent* (TBP) dictates the percentage of the total number possible two-bend routes to be evaluated. Thus the total number of routes evaluated is given by $\frac{TBP}{100} \times (C + H)$.

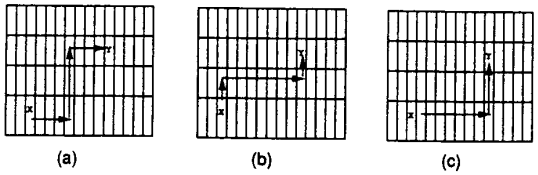

Figure 3 - *Sample Two-Bend Routes*

The priority order of the routes evaluated (when TBP is less than 100) is 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 because it is important that all of the potential channels for the route be 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 normalize to [0,1] then the routes are prioritized as $0, 1, \frac{1}{2}, \frac{1}{4}, \frac{3}{4}, \frac{1}{8}$, and so on.

The two-bend evaluation approach was calibrated against a least-cost path maze router between the two points. Note that both routers are not allowed to go beyond the bounding box of the two end points of the segment. This is different than comparing against a maze router for multipoint wires since that is a less constrained problem and the maze router will have more success, as discussed in Section 4.4. Experimentally, it was determined that a TBP of 20% would result in a path as good as that found by the maze router, as compared on the basis of total density for the entire circuit. On all of the test circuits used in the experiments discussed in the section 4.4, the LocusRoute router's total density 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 the standard cell routing problem.

4.3 Iteration

The LocusRoute algorithm makes use of a general iterative technique in the manner described in [Nair87]. Briefly, this means that after the first time all wires are routed, each is sequentially ripped up from the Cost Array and then re-routed. By routing each wire several times (typically four is sufficient), the wire order-dependency is reduced and the final answer is improved by five to ten percent. Also - of benefit to the end-purpose of integrated placement and routing - the nature of iteration is similar to the placement environment in which wires are ripped up and re-routed many times.

4.4 Uniprocessor Performance Results

This section compares the quality and execution time of LocusRoute with other routers.

Table 1 shows a comparison between the LocusRoute global router and the TimberWolf [Sech85] global router for several industrial circuits. These circuits are from several sources: The standard cell Benchmark suite (Primary1, Primary2, Test06 [Prea87]), 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 [Rose85, Rose88]. The TimberWolf version used was TimberWolf 4.1, obtained in July 1987. LocusRoute shows significantly better total density than does the TimberWolf global router, ranging from 16% to 37% fewer tracks. The principal reason is that the TimberWolf global router is constrained to use only the minimum number of vertical hops, whereas LocusRoute uses considerably more. This is a reasonable practice in current technology because many standard cells contain "free" built-in feedthroughs. The execution times of LocusRoute and TimberWolf are comparable for most of the examples, though TimberWolf is faster by a factor of 8 and 3 respectively for circuits Test06 and Primary2. This is due to the fact that the LocusRoute algorithm increases in running time proportional to the area covered by the wire, which is much larger in these two circuits.

Circuit Name	# Wires	Total Density		
		Locus	TWolf	% Few
BNRE	420	138	179	**22%**
MDC	575	150	179	**16%**
BNRD	774	188	225	**16%**
Primary1	904	262	316	**17%**
BNRC	937	202	247	**18%**
BNRB	1364	320	442	**27%**
BNRA	1634	315	432	**27%**
Test06	1673	335	537	**37%**
Primary2	3029	563	702	**20%**

Table 1 - *Comparison of LocusRoute and TimberWolf*

For comparison purposes a maze router [Lee61] was developed that exhaustively determines the optimal solution to the two-point routing problem as defined in Section 3. Note that it uses the same cost function as the LocusRoute router. It also determines a good approximation to the minimum-cost Steiner tree for multi-point wires using the approach described in [Aker72]. The maze router was carefully optimized for speed. Table 2 shows the comparison of total density and execution time for the maze router and the LocusRoute router, for all of the test circuits. The comparison is made on the basis of roughly equal numbers of vertical hops. Execution times are for four iterations over all wires on a DEC Micro Vax II.

Circuit Name	Total Density			Time (Micro Vax IIs)		
	Locus	Maze	Diff	Locus	Maze	Factor
BNRE	138	129	7%	88	2378	27x
MDC	150	141	6%	178	3173	18x
BNRD	188	182	3%	167	3306	20x
Primary1	262	255	3%	325	6534	20x
BNRC	202	189	7%	363	7250	20x
BNRB	320	308	4%	599	15116	25x
BNRA	315	294	7%	769	19652	26x
Test06	335	316	6%	5137	92272	18x
Primary2	563	549	3%	3758	48295	13x

Table 2 - *Comparison of LocusRoute and Maze Router*

For all circuits the LocusRoute total density (total number of routing tracks) is no greater than 7% more than that achieved by the maze router, and in some cases is as little as 3%. Most of this difference is due to the sub-optimality of dividing the wires up into two point nets. LocusRoute is markedly faster than the maze router - ranging from 13 to 27 times faster. This gain in speed is more than worth the increase in total density for the end-purpose of integrated placement and routing.

For two of our circuits, we can also compare the total routing density with the United Technologies global router used in the recent benchmark effort at the 1987 Physical Design Workshop [Prea87,Robe87]. The placements used above for circuits Primary1 and Primary2 were also routed by the UT router. Table 3 shows the comparison of total density for both circuits, with each router using roughly the same number of vertical hops. The total density of the UT router for circuit Primary1 is notably less than for the LocusRoute router. This is probably due to the fact that the UT router also performs neighbour exchanges and cell orientation changes on the placement in order to reduce the total number of tracks. The LocusRoute total density for circuit Primary2 is slightly less than that achieved by the UT router. We have no information on the execution time of the UT router, except that for circuits near the size of Primary2, it would take roughly 10000 Vax 11/780 seconds [Robe87].

Circuit Name	# Wires	Total Density	
		LocusRoute	Benchmark
Primary1	904	253	194
Primary2	3029	560	562

Table 3 - *Comparison of LocusRoute and Benchmark Router*

5 Parallelization

In this section several ways of parallelizing the LocusRoute router are proposed and implemented:

1. Wire-based Parallelism. Each processor is given an entire multi-point wire to route.

2. Segment-based Parallelism. Each two-point segment produced by the Kruskal decomposition can be routed in parallel.

3. Permutation-based Parallelism. Each of the four possible permutations, as discussed in Section 4.1, can be evaluated in parallel.

4. Route-based Parallelism. Each of the possible two-bend routes for every permutation can be evaluated in parallel.

Note that these are only *potential* axes of parallelism. It is possible to eliminate some of them as uneconomical by using statistical run-time measurements of the sequential router. For example, the number of two-point segments that actually need to have all four permutations evaluated is quite small with respect to the total. Thus, permutation-based parallelism is not going to provide significant speedup and isn't worth the time it requires to develop. Other measurements, however, show that the time spent evaluating the cost of two-bend routes ranges from 50 to 90 percent of the total routing time, so that some amount of speedup from route-based parallelism can be expected.

The following sections gives the details of three axes of parallelism, their performance and a quantitative measure of the of degradation in quality if there is some.

5.1 Wire-Based Parallelism

In Wire-Based parallelism, each multi-point wire is given to a separate processor, which runs the LocusRoute routing algorithm as described in Section 4. Thus, each processor executes the following "flow" for a different wire: Prior to decomposition, if the iteration technique is used, the wire must be "ripped up" out of the Cost Array. Next, each wire is decomposed into two-point nets, and possibly further into permutations. A subset of the potential two-bend routes is generated, and then evaluated by traversing the Cost Array. When a final route is chosen, the Cost Array is updated to reflect the new presence of that route.

The Cost Array is a shared data structure to which all processors have read and write access. This is an excellent axis of parallelism: if the sharing of the Cost Array does not cause performance degradation due to memory contention, the speedup should simply be the number of wires that are routed in parallel. The resulting parallel answer, however, will not necessarily be the same as the sequential answer. The problem is that the sequential router has complete knowledge of all wires that have already been routed, by virtue of their presence in the cost array. The parallel router has less information because it doesn't see the wires that are being routed simultaneously. The more wires routed in parallel, the less information each processor has to choose good routes that avoid congestion and hence the total density increases. The total density will increase as the number of processors increases. The measured effect on total density is discussed below, in Section 5.1.1.

An interesting issue is whether or not each processor should lock the Cost Array as it both rips up and re-routes wires in the Cost Array. The act of ripping up a route is essentially a decrement, and re-routing is an increment on a cell in the Cost Array. Locking the Cost Array during these operations (to ensure that two simultaneous operations on the same element does not prevent one of the operations from being lost) can cause a serious performance degradation. However, the final routing quality did not decrease when locking was omitted. The reason for this is that the probability of two processors accessing the same Cost Array element (of which there are many) at the same instant is very low. Even if very few increment or decrement operations are lost, the effect on final quality is negligible since only a few elements would be wrong by a small amount.

5.1.1 Wire-Based Parallel Results

Figure 4 is a plot of the speedup versus number of processors for the 904-wire (Primary1) example running on an eight-processor Encore MULTIMAX. The speedup for p processors, S_p is calculated as $\frac{T_1}{T_p}$, where T_1 is the execution time on one processor and T_p is the execution time using p processors. The Encore uses National 32032 chip sets which, in our benchmarks, timed out slightly faster than a DEC Micro Vax II.

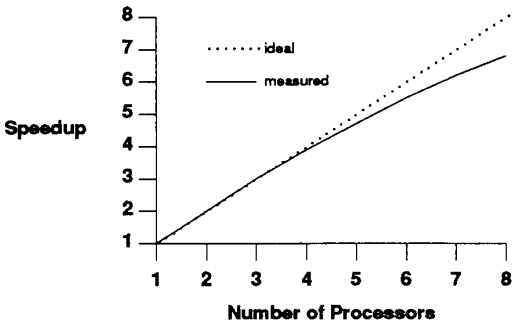

Figure 4 - *Wire-Based Speedup for Circuit Primary1*

Note that the execution time is only the actual routing computation time, excluding input time. The "knee" in the curve at five processors occurs because on an eight-processor Encore two processors share one cache. When five or more processors are used, pairs of processors interfere with each other more. For this circuit the increase in total density (between 1 and 8 processors) is negligible, and the number of vertical hops increases about 3%.

Table 4 gives the speedup using eight processors for the other test circuits. The speedup ranges from 5.8 for a smaller circuit to 7.6 for the largest. The speedup is less for smaller circuits because they are done in such a short time, and the startup overhead becomes a factor. The execution time is for four iterations over all the wires. It was discovered that very large global wires, such as TRUE or FALSE that have up to 150 pins, caused a severe degradation in speedup. This is because our system handles those nets just like any other, and the $O(n^2)$ nature of the Kruskal algorithm causes load balancing problems. Since most production systems treat TRUE and FALSE signal nets differently (usually tapping directly into the power lines with special cells) these were eliminated under the assumption that they could be handled quickly that way.

Table 5 gives the density and vertical hop counts for both 1 and 8 processors using wire-based parallelism. The degradation in total density ranges between .7% to 7.6%. The increase in vertical hops is generally 3% or less, with one exception. In the placement context this level of degradation is tolerable, though we have considered two ways of reducing the problem. The first is to try to ensure that the different processors only deal with wires that are in distinct physical areas, so that the wires routed simultaneously do not interact. This approach was not implemented because in the placement context (with incremental placement "moves") the wires are most likely to be in the same area and can't be separated.

Circuit Name	1-Proc Time (s)	8-Proc Time (s)	8-Proc Speedup
BNRE	78	13	5.8
MDC	88	15	5.9
BNRD	156	22	7.0
Primary1	321	47	6.8
BNRC	221	33	6.7
BNRB	697	92	7.6
BNRA	878	124	7.1
Test06	6261	869	7.2
Primary2	4334	574	7.6

Table 4 - *Wire-Based Parallelism Speedup*

The second way to reduce processor interference is not to rip up a route until the new route is determined. In this way there is a much shorter period of time in which the cost array does not contain the presence of the wire. Unfortunately, this severely degrades the new route of the wire itself since it sees the old copy of itself when new routes are being evaluated. Experimentally, the degradation was found to be bad enough to nullify any gain from the approach.

5.2 Segment-Based Parallelism

In segment-based parallelism, each two-point segment of a wire is given to a different processor to route. This is the stage following the Kruskal decomposition, but prior to the evaluation of different two-bend routes. 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. Even though there are many wires that provide two or three-way parallel tasks, however, the size of those tasks are not necessarily equal. The amount of time taken by the LocusRoute router to route 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.

Circuit Name	Density		Vertical Hops	
	1-Proc	8-Proc	1-Proc	8-Proc
BNRE	129	135	454	470
MDC	134	144	243	243
BNRD	181	185	528	562
Primary1	262	264	934	958
BNRC	193	199	739	749
BNRB	312	326	1897	1953
BNRA	300	311	2103	2154
Test06	325	336	3196	3253
Primary2	560	584	3022	3097

Table 5 - *Wire-Based Parallelism Quality*

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 on the order of the number of wires. In that case, the load balance would become 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 loads.

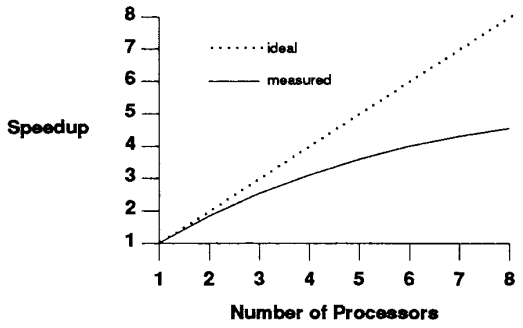

Figure 5 - *Route-Based Speedup for Test06*

5.3 Route-Based Parallelism

In route-based parallelism all of the two-bend routes to be evaluated are divided among processors. Each finds the lowest-cost path among the set of two-bend routes that it is assigned. When all processors finish, the route with the best overall cost is selected. In this case 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.

Figure 5 is a plot of the speedup versus number of processors for the circuit Test06, a large circuit. It achieves a speedup of 4.6 using 8 processors.

Table 6 gives the best speedup achieved for all of the test circuits, ranging from 1.2 using 2 processors to 4.6 using 8 processors. The principal reason for the limitation in 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 10-15%. Another reason is that some segments have only one potential route, limiting parallelism.

Circuit Name	Best Route-based Speedup (Speedup/#Processors)
BNRE	1.2/2
MDC	1.3/2
BNRD	1.4/2
Primary1	1.8/3
BNRC	1.6/3
BNRB	2.1/4
BNRA	2.0/4
Test06	3.6/5, 4.6/8
Primary2	3.3/5

Table 6 - *Performance of Route-Based Parallelism*

5.4 Combining Two Axes of Parallelism

The wire-based parallel and route-based parallel approaches are perfectly orthogonal; hence their speedups should "multiply". Assume, for a given circuit that a speedup of S_w is achieved using wire-based parallelism on W processors, and a speedup of S_r is achieved using route-based parallelism on R processors. Then, because the two approaches are orthogonal, the resulting speedup when they are used together should be $S_w \times S_r$ using $W \times R$ processors. This model neglects the effect of memory contention that may occur when the number of processors is increased dramatically. Table 7 shows the best predicted speedup for the test circuits. Combined speedup ranges from 7 using 16 processors to 33 using 64 processors. The smaller circuits are routed very quickly and so it is difficult to get speedups greater than 10 due to the startup overhead. The larger circuits benefit greatly from the combination of the approaches.

Table 7 also contains the average routing time per net on one processor, A_1, and what the the average routing time per net would be under the maximum speedup, A_{RW}. That is, $A_{RW} = \frac{A_1}{S_w \times S_r}$. The average routing times for all circuits, under the various speedups range from 5.0ms to 28ms, and approaches our goal of one to five milliseconds per net.

Circuit	$\frac{S_w}{W}$	$\frac{S_r}{R}$	$\frac{S_w \times S_r}{W \times R}$	A_1 (ms)	A_{RW} (ms)
BNRE	$\frac{5.8}{8}$	$\frac{1.2}{2}$	$\frac{7.0}{16}$	46	**6.6**
MDC	$\frac{5.9}{8}$	$\frac{1.3}{2}$	$\frac{7.7}{16}$	38	**5.0**
BNRD	$\frac{7.0}{8}$	$\frac{1.4}{2}$	$\frac{9.8}{16}$	50	**5.1**
Primary1	$\frac{6.8}{8}$	$\frac{1.8}{3}$	$\frac{12.2}{24}$	89	**7.2**
BNRC	$\frac{6.7}{8}$	$\frac{1.6}{3}$	$\frac{10.7}{24}$	59	**5.5**
BNRB	$\frac{7.6}{8}$	$\frac{2.1}{4}$	$\frac{16.0}{32}$	127	**8.0**
BNRA	$\frac{7.1}{8}$	$\frac{2.0}{4}$	$\frac{14.2}{32}$	134	**9.5**
Test06	$\frac{7.2}{8}$	$\frac{4.6}{8}$	$\frac{33}{64}$	935	**28**
Primary2	$\frac{7.6}{8}$	$\frac{3.3}{5}$	$\frac{25}{40}$	358	**14**

Table 7 - *Predicted Combined Speedup of Wire and Route Parallelism*

5.5 Conclusions

A new global routing algorithm for standard cells and its parallel implementation has been presented. The LocusRoute algorithm users significantly fewer tracks than the TimberWolf standard cell global router, and is comparable to a maze router and an industrial router. It is more than a factor of 10 faster than either of the two latter routers. Three axes of orthogonal parallelism were developed to speed up the LocusRoute router further. Two of the three axes that were implemented achieved significant speedup - up to 7.6 using eight processors and 4.6 using eight processors. They should produce combined speedups of up to 33 times.

In the future, the combined approach will be run on a multiprocessor with more processors. Using a sophisticated scheduling algorithm we hope to do better than simple multiplication of speedups. The Locus placement environment is currently being developed, and will be combined with the LocusRoute global router. Our aim is to achieve smaller final area by using the global routing as a better measure of each placement.

Acknowledgements

The author is grateful to Tom Blank who provided many good suggestions for 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 version 4.1 of TimberWolfSC.

6 References

[Aker72]
S. B. Akers, "Routing," Chapter 6 of **Design Automation of Digital Systems; Theory and Techniques,** M.A. Breuer, Ed., Englewood Cliffs, NJ, Prentice-Hall, 1972.

[Blan81]
T. Blank, M. Stefik, W. VanCleemput, "A Parallel Bit Map Processor Architecture FOR DA ALGORITHMS," Proc. 18th Design Automation Conference, June 1981, pp. 837-845.

[Breu77]
M.A. Breuer, "Min-Cut Placement," Journal of Design Automation and Fault-Tolerant Computing, Oct 1977, pp 343-362.

[Breu81]
M.A Breuer, K. Shamsa, "A Hardware Router," Journal of Digital Systems, Vol IV, Issue 4, 1981, pp. 393-408.

[Hana72]
M. Hanan, J.M. Kurtzberg, "Placement Techniques," Chapter 4 of **Design Automation of Digital Systems; Theory and Techniques,** M.A. Breuer, Ed., NJ, Prentice-Hall, 1972.

[Kamb85]
T. Kambe, T. Okada, T. Chiba, I. Nishioka, "A Global Routing Scheme for Polycell LSI," Proc. ISCAS 1985, pp. 187-190.

[Krus56]
J.B. Kruskal, "On The Shortest Spanning Subtree of a graph and the Traveling Salesman Problem," Proc. Amer. Math. Soc, 7, 1956, pp. 48-50.

[Lee61]
C.Y. Lee, "An Algorithm for Path Connections and Its Applications," IRE Transactions on Electronic Computers, Vol EC-10, pp 346-365, 1961.

[Nair87]
R. Nair,"A Simple Yet Effective Technique for Global Wiring," IEEE Transactions on Computer-Aided Design, Vol CAD-6, No. 2, March 1987, pp. 165-172.

[Ng86]
A P-C Ng, P. Raghavan, C.D. Thompson, "A Language for Describing Rectilinear Steiner Tree Configurations," Proc. 23rd Design Automation Conference, June 1986, pp. 659-662.

[Prea87]
B.T. Preas, "Benchmarks for Cell-Based Layout Systems," Proc. 24rd Design Automation Conference, June 1987, pp. 319-320.

[Robe87]
Ken Roberts used the United Technologies Standard Cell global router on the standard cell benchmark placements. Results were discussed at the 1987 DAC.

[Rose85]
J.S. Rose, W.M. Snelgrove, Z.G. Vranesic, "ALTOR: An Automatic Standard Cell Layout Program," Proc. Canadian Conference on VLSI, November 1985, pp. 168-173.

[Rose88]
J.S. Rose, W.M. Snelgrove, Z.G. Vranesic, "Parallel Standard Cell Placement Algorithms with Quality Equivalent to Simulated Annealing," IEEE Trans. on CAD, Vol. CAD-7, No. 3, March 1988, pp. 387-396.

[Rute84]
R.A. Rutenbar, T.N. Mudge, D.E. Atkins, "A Class of Cellular Architectures to Support Physical Design Automation," IEEE Trans. on CAD, Vol. CAD-3, No. 4, October 1984, pp. 264-278.

[Sech85]
C. Sechen, A. Sangiovanni-Vincentelli, "The Timberwolf Placement and Routing Package," IEEE JSSC, Vol. SC-20, No. 2, April 1985, pp 510-522. pp. 432-439.