The Parallel Decomposition and Implementation of an Integrated Circuit Global Router

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

Abstract

Better quality automatic layout of integrated circuits can be obtained by combining the placement and routing phases so that routing is used as the *cost function* for placement optimization. Conventional routers are too slow to make this feasible, and so this paper presents a parallel decomposition and implementation of an integrated circuit global router. The *LocusRoute* router is divided into three orthogonal "axes" of parallelism: routing several wires at once, routing segments of a wire in parallel, and dividing up the potential routes of a segment among different processors to be evaluated. The implementation of two of these approaches achieve significant speedup - wire-by-wire parallelism attains speedups from 6.9 to 13.6 using sixteen processors, and route-by-route achieves up to 4.6 using eight processors. When combined, these approaches can potentially provide speedups of as much as 55 times.

1 Introduction

The task of automatic layout of integrated circuits has traditionally consisted of two parts: automatic *placement* where the circuit modules are positioned and automatic *routing* in which the paths of the connecting wires are determined. The objective of both tasks is to result in a layout with as little area as possible. The best way to evaluate the "goodness" of a placement is to route it and determine its final area. Up to now this has not been feasible because routing itself is a difficult combinatorial optimization problem and common heuristics have been too slow to be used in this way.

The advent of usable commercial multiprocessors, with potentially enormous aggregate computation power may change this view if automatic routing can be decomposed into tasks that can be efficiently run in parallel. The aim of the *Locus Project* at Stanford University is to combine 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 parallel decomposition and implementation of the *LocusRoute* global router for integrated circuits. The goal of the router is to make the average routing time for one wire close to the time that it takes to recalculate more conventional cost functions. This means that the routing time must be on the order of one to five milliseconds per wire on a VAX 11/780-class machine [Sech85]. 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.

Prior work on parallel routing (see [Blan84] for a survey) has been done in isolation from the placement problem and has generally focused on the Lee routing algorithm [Lee61]. In most cases the algorithm has been fixed in hardware and as such lacks the flexibility that is always required in practical CAD software such as the global router described in [Yama85]. A far more versatile approach is to use general purpose parallel processors, which allow an application to be tuned in a manner similar to uniprocessors. 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.

The basic idea of the LocusRoute algorithm is to investigate a subset of the two-bend routes between pairs of pins to be routed. The uniprocessor LocusRoute program can route wires in average times from 45 ms to

935 ms on a DEC Micro Vax II depending on the size of the circuit. The routing speed is increased 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 6.9 to 13.6 using sixteen processors. The route-by-route approach achieves speedups of up to 4.6 using eight processors. These two axes of parallelism are orthogonal to each other.

This paper is organized as follows: Section 2 describes the standard cell layout methodology and defines the associated global routing problem. Section 3 describes the uniprocessor LocusRoute algorithm. Section 4 presents three approaches for speeding up the router using parallel processing, and gives performance results.

2 Standard Cell Layout

The standard cell-style layout is a common circuit design methodology in which all circuit modules are of equal height and are "butted" together to form rows as shown in Figure 1.

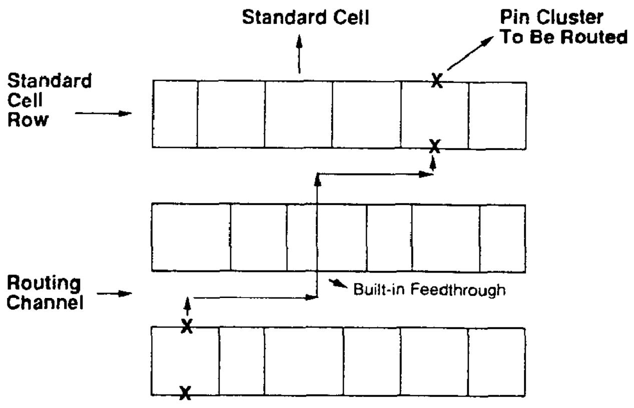

Figure 1 - *Standard Cell Layout*

Power and ground wires run horizontally through the cells and are connected by abutment. Cells have connection points on their top and bottom and typically one logical pin has two physical pins on each. This group of pins is called a *pin cluster*. Connections between adjacent rows are made by routing wires in the horizontal *routing channels* as shown in Figure 1. If a connection is required between two non-adjacent rows then either feedthrough cells are inserted in the intervening rows to make room for vertical connections or an uncommitted path in an existing cell (called a "built-in feedthrough") is used.

2.1 Problem Definition

Global routing for standard cells decides the following for each wire: First, for each pin cluster it decides which of the physical 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 decide which channel to use in the route from a pad into the core cells. The objective is to minimize the sum of the maximum widths of each routing channel (hereafter called the *total density*), and in so doing minimize the final area.

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 [Rose88b].

3 A Standard Cell Global Router

This section gives a brief description of the LocusRoute global router. A more complete discussion can be found in [Rose88b].

3.1 Routing Model

The LocusRoute algorithm uses the following 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 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 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. The routing problem for 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.

Under this model, 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 wire is routed through location (i , j)

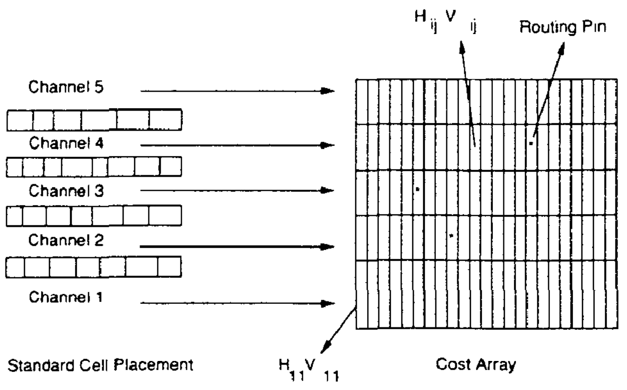

Figure 2 - *Routing Model*

its presence is recorded in the Cost Array (i.e. H_{ij} is incremented, as is V_{ij} if the direction is vertical) so that subsequent wires can take it into account. Thus the more wires going through a particular location in a channel, the less likely it is that area will be used.

3.2 The Global Routing Algorithm

There are five main steps in the LocusRoute global routing algorithm for standard cells. They are:

1. A multi-point wire is decomposed into two-point *segments*, by finding its minimum spanning tree using Kruskal's algorithm [Krus56].

2. The segments are further decomposed, if necessary, into *permutations*, which are the set of possible routes between each pin in a pin cluster. There are four possible routes, one between each of the two physical pins in each pin cluster. It has been experimentally determined that only when the clusters are greater than a certain horizontal distance apart (about 300 routing grids) is it necessary to evaluate all four permutations. Less than this distance, only the closest pin pair need be evaluated.

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. This step is described in further detail below, in Section 3.3.

4. Traceback. This is a cleanup step that provides enough information for later detailed routing.

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.

3.3 Route Evaluation

The LocusRoute algorithm searches for a low-cost path for a permutation by evaluating a number of different routes. The idea is to determine the cost of a subset of all two-bend routes between the two pins, and then choose the one with the lowest cost. Figure 3 illustrates three possible two-bend (or less) routes inside a representation of the Cost Array as a small example.

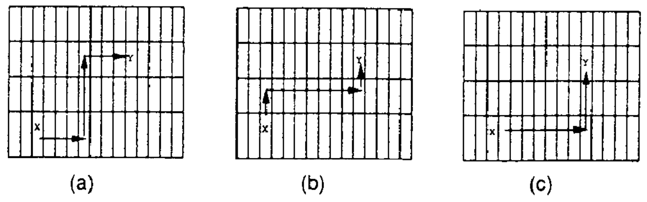

Figure 3 - *Sample Two-Bend Routes*

If the horizontal distance between the two pins is H routing grids, and the vertical difference in channels between the pins is C, 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 $\frac{TBP}{100} \times (C+H)$. When TBP is less than 100, then the routes are evaluated in a priority order [Rose88b]. Experimentally, it was determined that a TBP of 20% would result in a path as good as that found by an exhaustive maze router, as compared on the basis of total density for the entire circuit.

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.

The uniprocessor LocusRoute algorithm compares favorably with a widely used placement and global routing package [Sech85], and with a good quality industrial global router [Rose88b].

4 Parallel Decomposition & Implementation

In this section several ways of parallelizing the LocusRoute router are proposed and implemented. Figure 4 illustrates several such axes of parallelism:

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 3.2, can be evaluated in parallel.

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

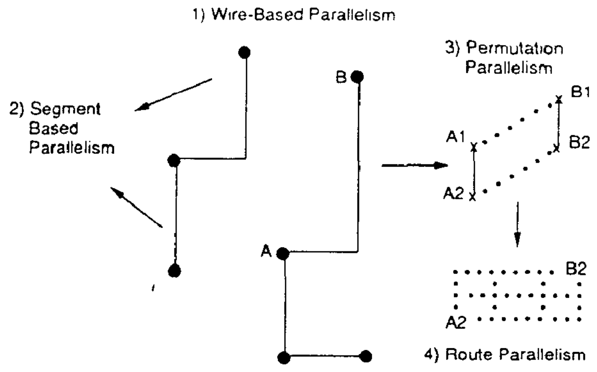
1) Wire-Based Parallelism
2) Segment Based Parallelism
3) Permutation Parallelism
B
A
A1
A2
B1
B2
B2
A2
4) Route Parallelism

Figure 4 - *Parallel Decomposition of LocusRoute*

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. On the other hand, other measurements 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.

To date, we have not considered pipelining as an axis of parallelism. A pipeline implementation would have the same stages as the basic algorithm described in Section 3.2. To some extent, pipelining uses the same axis of parallelism as wire-based parallelism since it also routes several wires at once. The best use of pipelining would be to execute the first two stages, segment and permutation decomposition, for all wires in parallel since these stages have no data dependencies on the routing of other wires. In the context of iterative improvement placement, however, the wire positions will not be known in advance as they are when considering the routing problem in isolation.

Each of the following sections discusses the details of the axes of parallelism that have been implemented. In the case where the quality of the answer of the parallel program is worse than the sequential program, a quantitative measure of the amount of degradation is given. This section is concluded by a discussion of the combination of two of the axes of parallelism. All decompositions assume a shared-memory multiprocessor.

4.1 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 3: 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 wires, 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. Other than a task queue, the cost array is the only shared piece of data. 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 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. Thus the total density will increase as the number of processors increases. The measured effect on total density is discussed below, in Section 4.1.1.

4.1.1 Wire-Based Parallel Results

Figure 5 is a plot of the speedup versus number of processors for a 3029-wire example running on an sixteen-processor shared-memory Encore MULTIMAX. The Encore uses National 32032 chip sets which, in our benchmarks, timed out slightly faster than a DEC Micro Vax II. 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 execution time measured *does not* include the time for input of the circuit, only the actual routing computation time. For this circuit the increase in total density due to the missing "knowledge" effect described in Section 4.1 from 1 to 16 processors is 6%, and the number of vertical hops increases 2%.

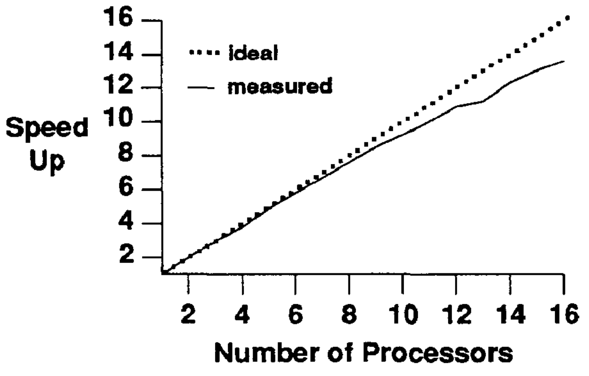

Figure 5 - *Wire-Based Speedup for 3029-Wire Circuit*

The program was run on several other circuits, which 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,Rose88a]. Table 2 gives the execution time and speedup using 1, 8 and 15 processors, for all the test circuits. The execution time is for four iterations over all the wires. The speedup ranges from 5.4 for a smaller circuit to 7.6 for the largest, using 8 processors. It ranges from 6.9 to 13.6 using 15 processors. The speedup is less for smaller circuits because they are done in such a short time, and the startup overhead becomes a factor.

Circuit Name	# Wires	1 Pr T (s)	8 Pr T (s)	15 Pr T (s)	8 Pr Spdup	15 Pr Spdup
BNRE	420	70.0	13.0	10.2	5.4	6.9
MDC	575	79.3	14.3	11.2	5.5	7.1
BNRD	774	139	21.0	16.0	6.6	8.7
Primary1	904	279	43.4	33.6	6.4	8.3
BNRC	937	198	30.5	21.4	6.5	9.3
BNRB	1364	565	84.6	63.4	6.7	8.9
BNRA	1634	725	112	77.8	6.5	9.3
Test06	1673	5684	797	465	7.1	12.2
Primary2	3029	3950	517	290	7.6	13.6

Table 2 - *Performance of Wire-Based Parallelism*

Table 3 gives the total density and vertical hop counts using 1, 8 and 15 processors. The increase in total density ranges between 1% to 7% for 15 processors. The increase in vertical hops is ranges from 1% to 9% but is generally less than 4%. In the placement context this level of degradation is tolerable. In the future, however, on machines with more processors, it will likely become more of a problem. We have considered three ways of reducing the effect of the missing knowledge due to simultaneous routing of wires. 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. 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. This severely degrades the new route of the wire itself, however, since it sees the old copy of itself while evaluating potential routes. Experimentally, the degradation was sufficient to nullify any gain from the approach. A third method not yet implemented is to route the wires in a different order for each iteration, (iteration is described in Section 3.3) so that the knowledge missing in one iteration is different from that in another.

Circuit Name	Total Density 1 Pr	Total Density 15 Pr	Total Density % More	Vertical Hops 1 Pr	Vertical Hops 15 Pr	Vertical Hops % More
BNRE	130	134	3%	449	490	9%
MDC	134	142	6%	241	243	1%
BNRD	176	181	3%	530	572	8%
Primary1	262	269	3%	940	966	3%
BNRC	191	193	1%	739	772	4%
BNRB	307	325	6%	1904	1974	4%
BNRA	298	320	7%	2106	2197	4%
Test06	318	338	6%	3221	3286	2%
Primary2	560	592	6%	3053	3126	2%

Table 3 - *Quality of Wire-Based Parallelism*

4.1.2 Gain Due to Removal of Locks

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 set of cells in the Cost Array. Locking the Cost Array during these operations ensures that two simultaneous operations on the same element does not prevent one of the operations from being lost. It does, however, cause a significant performance degradation. For example, for the Primary1 circuit the speedup decreased from 8.3 to 6.4 using 15 processors when Cost Array locking was used. For the Primary2 circuit the speedup for 15 processors was reduced to 12.1 from 13.0 due to locking.

The final routing quality, however, does not decrease when locking is omitted. The reason for this is that the probability of two processors accessing the *same* Cost Array element (of which there are on the order of 10000) 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. This was shown experimentally by performing ten runs with 15 processors on each of the above circuits, for both the locking and non-locking cases. Table 1 gives for the two circuits the average running time, and the average and standard deviation of the total density and number of vertical hops. From this table it can be seen that the quality in both cases is very nearly the same. Note that in a placement context in which many more wires will be ripped up and re-routed, the effect of these small errors would be cumulative and so an occasional correction step may be necessary if locks are not used.

Circuit & Lock Type	Avg T (s)	Density Avg.	Density SD	Vertical Hops Avg	Vertical Hops SD
Primary1 Locks	43.8	269	2.0	962	4.9
Primary1 NO Locks	33.7	272	3.0	964	3.4
Primary2 Locks	325	591	1.9	3126	7.5
Primary2 NO Locks	303	591	4.9	3122	4.0

Table 1 - *Speed & Quality Using and Not Using Locks*

4.2 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. On the surface this implies that a speedup of about two could be achieved using three processors. Unfortunately, this is not the case. Even though there are many wires that provide two or three-way parallel tasks, the size of those tasks are not necessarily equal. The amount of time taken by LocusRoute 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. Thus 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.

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 multiple segments. Though every processor won't be able to use a second processor all the time, some number of processors can be used in this way. This technique would become essential if many processors were used in wire-based parallelism, at the point where the number of processors was close to the number of wires. In that case the load balance would become a problem in wire-based parallelism because wires with many segments take much longer than wires with few segments. Hence segment-based parallelism could be used to speed up the routing of the larger wires.

4.3 Route-Based Parallelism

In route-based parallelism all of the two-bend routes to be evaluated are divided among separate 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 will be well-balanced because the routes are all of the same length, and the number of routes is evenly divided among the processors.

Figure 6 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.

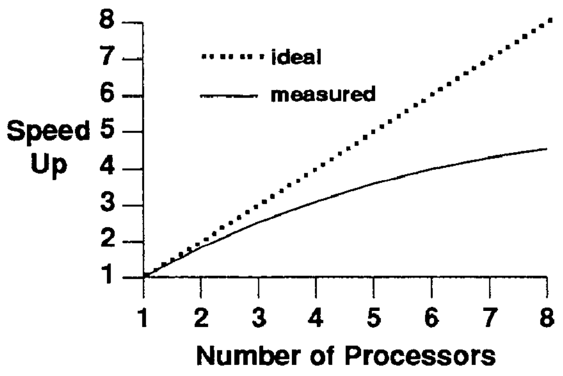

Figure 6 - *Route-Based Speedup for Circuit Test06*

Table 4 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 number of processors given for each circuit in the table are chosen by eye as to which number gives reasonable efficiency. It is clear that only the larger circuits benefit from more 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%. Other minor effects which degrade performance are the imbalance of processor task sizes due to integral numbers of routes and the fact that some segments have only a few potential routes.

Circuit Name	Best Route Speedup Speedup/#Proc
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	4.6/8
Primary2	3.3/5

Table 4 - *Performance of Route-Based Parallelism*

4.4 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 5 shows the best predicted speedup for the test circuits. Combined speedup ranges from 8.3 using 30 processors to 55 using 120 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 5 also contains the average routing time per wire on one processor, A_1, and what the the average routing time per wire 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 4.0ms to 17ms, and approaches our goal of one to five milliseconds per wire. It is interesting to note that even though the uniprocessor times are widely

Circuit	$\frac{S_w}{W}$	$\frac{S_r}{R}$	$\frac{S_w \times S_r}{W \times R}$	A_1	A_{RW}
BNRE	$\frac{6.9}{15}$	$\frac{1.2}{2}$	$\frac{8.3}{30}$	46ms	**5.6ms**
MDC	$\frac{7.2}{15}$	$\frac{1.3}{2}$	$\frac{9.4}{30}$	38ms	**4.0ms**
BNRD	$\frac{8.7}{15}$	$\frac{1.4}{2}$	$\frac{12.2}{30}$	50ms	**4.1ms**
Primary1	$\frac{8.3}{15}$	$\frac{1.8}{3}$	$\frac{14.9}{45}$	89ms	**6.0ms**
BNRC	$\frac{9.3}{15}$	$\frac{1.6}{3}$	$\frac{14.9}{45}$	59ms	**4.0ms**
BNRB	$\frac{8.9}{15}$	$\frac{2.1}{4}$	$\frac{18.7}{60}$	127ms	**6.8ms**
BNRA	$\frac{9.3}{15}$	$\frac{2.0}{4}$	$\frac{18.6}{60}$	134ms	**7.2ms**
Test06	$\frac{12.0}{15}$	$\frac{4.6}{8}$	$\frac{55}{120}$	935ms	**17ms**
Primary2	$\frac{13.6}{16}$	$\frac{3.3}{5}$	$\frac{45}{80}$	358ms	**8.0ms**

Table 5 - *Predicted Combined Performance*

varying, the best combined speedup results in average routing times that are all very close. This is because circuits with routes that take the longest have more parallelism.

Note that combining the two orthogonal axes of parallelism in the obvious way produces an obvious scheduling strategy: Each wire is assigned a constant number of processors to "help" in the route evaluation. While this static scheduling strategy has low overhead, it is clear that a dynamic approach that only assigns processors to wires when they really need it would be more processor-efficient. In this case wires that have many routes to be enumerated would use more processors, and those with less routes would use fewer processors.

4.5 Conclusions

The parallel implementation of an integrated circuit global routing algorithm has been presented. Two of the three axes of parallelism that were implemented achieved significant speedup - up to 13.6 using sixteen processors and 4.6 using eight processors. They should produce combined speedups of up to 55 times.

In the future, the combined approach will implemented with several scheduling strategies. Methods of reducing quality degradation in wire-based parallelism will be investigated. We are also looking at implementing a parallel version of the LocusRoute algorithm on a message passing architecture, such as an N-Cube [Haye86], and a massively parallel SIMD machine such as the Connection Machine [Thin87]. In addition the Locus placement environment is currently being developed, and will be combined with the LocusRoute global router. Our aim is to achieve smaller area by using routing as a measure of each placement.

5 References

Blan84

T. Blank, "A Survey of Hardware Accelerators Used In Computer-Aided Design," IEEE Design and Test, Vol. 1 No. 3, August 1984, pp. 21-39.

Haye86

J.P. Hayes, et. al, "A Microprocessor-based Hypercube Supercomputer," IEEE Micro, Vol. 6, No. 5, Oct. 1986, pp. 6-17.

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 Trans. on Electronic Computers, Vol EC-10, pp 346-365, 1961.

Nair87

R. Nair, "A Simple Yet Effective Technique for Global Wiring," IEEE Trans. on CAD, Vol CAD-6, No. 2, March 1987, pp. 165-172.

Prea87

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

Rose85

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

Rose88a

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. 7 No. 3, March 1988, pp. 387-396.

Rose88b

J.S. Rose, "LocusRoute: A Parallel Global Router for Standard Cells," to appear in the 1988 Design Automation Conference.

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.

Thin87

Thinking Machines Corp, "Connection Machine Model CM-2 Technical Summary," Technical Report # HA87-4, April 1987.

Yama85

M. Yamada, T. Hiwatashi, T, Mitsuhashi, K. Yoshida, "A Multi-Layer Router for Standard Cell LSIs," Proc. ISCAS 1985, 191-194.