A Parallel VLSI Circuit Layout Methodology

S. Bapat
J. P. Cohoon

Computer Science Report No. TR-92-29
September 15, 1992

A Parallel VLSI Circuit Layout Methodology

S. Bapat and J. P. Cohoon

Department of Computer Science
University of Virginia
Charlottesville, VA 22903
USA

Abstract

We propose a parallel computation layout technique that solves the layout problem directly rather than decomposing it into the traditional distinct steps of placement and routing. The method combines a superior geometric partitioning algorithm with extensive use of pre-computed minimum-length Steiner trees to produce layouts.

I. Introduction

To reduce the computational complexity of the physical layout problem, computer-aided design tool developers have traditionally decomposed the layout activity into separate steps of placement and routing[14]. This decomposition is clearly an artificial one as the placement and routing steps are intimately related — a poor placement can make it impossible to construct even a feasible routing solution. Similarly, a poor global routing can lead to a detailed routing solution with non-optimal connections. Recognizing this situation, several researchers have proposed to incorporate the global routing step of the interconnection activity into the placement activity [6, 17, 18].

Since interconnections occupy a significant portion of the layout surface, almost all placers make extensive use of routing estimators when evaluating the quality of proposed placements. However, in fear of computational intractability, the estimators are often quite crude. For example, its common for a placer to use one-half of the length of the perimeter of the net's bounding rectangle as an estimate of a net's routing length. As a result there is often great disparity between the estimate and the actual size of the circuit [13].

We have proposed a new layout approach, SHARP [1, 2], that neither decomposes layout problem into separate placement and routing steps nor uses traditional but inaccurate routing estimators. Instead, SHARP 'remarries' the placement and routing steps and examines the layout problem in its entirety and refines its solution over time. As shown in this paper, the refinement can be done very naturally in parallel. We call the parallel version, PARFECT. We note there are other parallel circuit layout efforts [3, 4, 16].

II. SHARP and PARFECT Basics

Unlike conventional partitioning-based placers (PBP) [7, 11, 18] that use the min-cut paradigm or a close variant [8, 10], we use a geometric partitioning algorithm that is designed explicitly for intra-package connections [1, 2]. The SHARP partitioning algorithm combines a robust hill-climbing method with built-in terminal propagation capabilities and heuristics that quickly generate and efficiently manipulate a rich collection of optimal Steiner tree routing topologies. During the layout process the algorithm simultaneously balances the preference for minimum-length interconnections with channel area utilization and interconnection congestion.

The geometric decomposition iteratively divides a given layout surface into an $m \times n$ partition, where $1 \leq m, n \leq 3$. The default partition is a 3×3 decomposition of nine subregions or *blocks* that roughly resembles a musical sharp as in Figure 1 (the other seven decompositions are used if the given layout surface is inappropriate). The decomposition of a block continues iteratively until an exact placement of the circuit elements and a coarse routing of the associated signals in the given block both become simple.

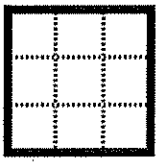


Figure 1. SHARP 3×3 Decomposition.

A more-detailed explanation and accompanying analysis of the partitioning scheme is presented elsewhere [1]. We note here that the decomposition geometry was selected as it is the smallest, nontrivial, symmetric decomposition that allows contiguous regions of the circuit surface that share similar routing features and problems (e.g. congestion) to be

processed as unit. For example, in determining the preferred interconnection given a net's block distribution, every minimum-length Steiner tree form can be considered since there are on average less than six such Steiner forms per decomposition.The trees given in Figure 2 are the six possible minimum-length Steiner tree forms corresponding to the given block distribution of circuit elements for the given net. A 4×4 decomposition has also recently been proposed independently [12]. However, this decomposition does not make extensive use of Steiner trees. This may be due to the significant blow-up in such tree forms for the 4×4 decomposition. For example, there are over one million different Steiner tree forms associated with such a scheme, while our decomposition scheme uses on the order of one thousand different tree forms.

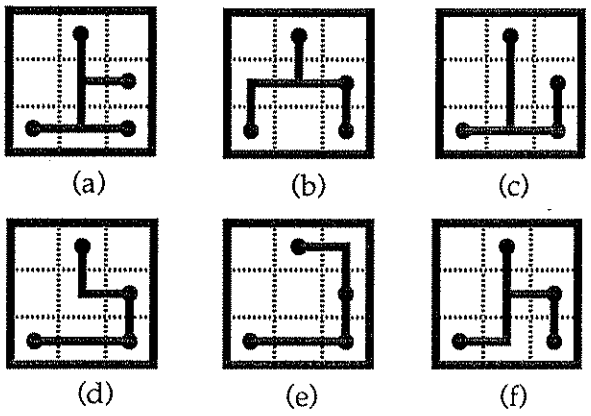


Figure 2. Steiner net route forms.

We divide the core layout area into three parts: *logic area* that is occupied by the circuit elements, *routing area* that is taken up by the wire to interconnect the circuit elements, and the remaining *unused area* that is neither utilized by the circuit elements nor the routing segments. Since we are concerned with structured design methodologies, layout area minimization reduces to minimizing the routing and unused areas. This implies that the important area minimization criteria are total wire length and the individual channel utilizations.

With our layout scheme, obtaining an accurate estimation of the total wire length is a natural by-product of the Steiner tree manipulations. We measure channel utilization in terms of *congestion imbalance*, where the *congestion* at a point p is the number of wires in the channel that intersect a line through p that is perpendicular to the channel's orientation. The height of a channel is a function of the point of maximum congestion in the channel. The unused area in a channel is a function of the variation from this maximum congestion value across the channel. The greater the variation in the congestion values, the greater the size of the unused area. The unused area is at a minimum when the congestion is constant at all points in a channel.

Although we are concerned with structured design styles, we do not require or assume — unlike most traditional partitioning-based placers — that the channel heights are uniform. This flexibility enables us to consider a broader solution domain. Rather than forcing all the channels to have equal heights, SHARP prefers a solution that minimizes the unused area in the individual channels.

The use of geometric partitioning and Steiner interconnections permits reduction in channel congestion imbalance and total wire length in two basic ways: re-arranging circuit elements among the SHARP partitioning regions, and selecting alternative Steiner trees for the nets. If the given structured design methodology prefers also to control the inter-channel balancing, the SHARP decomposition can balance congestion across groups of horizontal or vertical channels.

As implied in the above discussion, our geometric partitioning strategy [1] is the basis of our layout methodology. Briefly, the strategy works in the following manner. The partitioning algorithm first determines if an exact solution is readily constructed, otherwise the algorithm partitions the given circuit layout surface into the appropriate $m \times n$ decomposition. If necessary, the elements are assigned to specific blocks and the nets are assigned specific tree forms. The principal geometric partitioning phase then occurs with the algorithm alternately trying to improve the two wire usage components (routing area and unused area). This strategy contributes to SHARP's solution space hill-climbing ability. However, since a re-configuration strategy that operates individually on reducing the routing or unused areas can cause an increase in the other area, SHARP's objective function considers the two area measures as a weighted sum (we are currently investigating different schedules that dynamically specify the weights).

III. Parallel Layout

An overview of the parallel layout algorithm is given in Figure 3.

Initialization

The layout algorithm begins by computing the Steiner tree forms. Afterwards an initial partition, b, of the layout surface is determined. This partition becomes

the initial approximation (partial solution) of the circuit layout solution, S. As stated above, these activities require minimal time. The algorithm then begins an iterative phase where the solution S is repeatedly refined until an exact placement has been determined.

```
{— initialization —}
compute Steiner tree forms
compute SHARP partition b for circuit layout
   instance and use { b } as the initial partial
   solution set S
{— compute exact layout—}
while solution S is not exact do
   {— decompose partial solutions —}
   parfor each partial solution b in S do
        replace b in S with the sub-solutions that
             comprise b's SHARP partition
   end
   {— produce a tentative refinement —}
   parfor each partial solution b in S do
        create necessary virtual circuit elements
             V_b for b
        refine SHARP partition S_b of b with V_b
        update assignment of circuit elements
             and net routes in S using S_b
   end
   {— interconnect virtual element pairs —}
   parfor each virtual element pair v do
        interconnect pair v with an optimal
             connection
   end
   {— smooth out the refinement —}
   while solution quality is improving do
        for each template tiling T do
             parfor each tile t in T do
                  create necessary anchored
                       virtual circuit elements V_t
                       for t
                  refine SHARP partition S_t of t
                       with V_t
                  update assignment of circuit
                       elements and net routes in S
                       using S_t
             end
        end
   end
   {— rip-up and reroute —}
   parfor each partial solution b in S do
        reroute nets with circuit elements that
             have changed blocks
   end
end
```

Figure 3. Parallel circuit layout algorithm.

Decomposing partial solutions

The first step in the while-loop is a parallel one that decomposes the current partial solutions that comprise S. Each partial solution, b, is broken into partial sub-solutions that correspond to the SHARP partition elements for b. The set of such sub-solutions for all b comprises the new set S of partial solutions.

Once the new partial solutions are specified, the algorithm then begins a four step process that refines the new partial solutions an additional level. Once this occurs, the outer while-loop is then iterated to produce a new level of refinement. This process continues until an exact solution is determined. All four of the steps make extensive use of virtual circuit elements which are a generalization of Dunlop and Kernighan's terminal propagation technique [7].

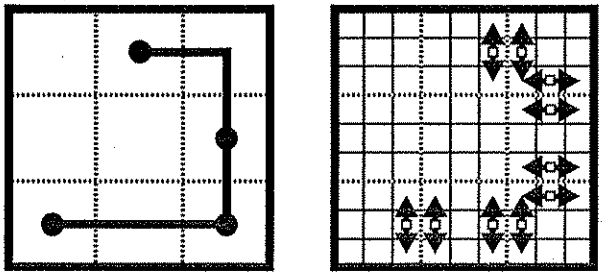
The first step in producing the new level of refinement is a parallel one that computes a tentative refinement of each partial solution. The second major step is performed in parallel and it connects any nets with sub-routes that are not electrically common. By the nature of the previous major step, all such fixes are trivial. The third major step has a parallel component and it performs a smoothing process that corrects any sub-standard circuit element assignments and net routes that may be introduced in the preceding two steps. The last major step examines the nets with circuit elements that have undergone significant migration from one partial solution to another. Each such net has its associated Steiner tree form reassessed, and if necessary, reassigned.

Producing the tentative refinement

In computing a refinement, it may be the case that the currently assigned route for a net, u, crosses multiple blocks. In particular, the route may involve blocks where the net does not have circuit elements (e.g., the six net routes of Figure 2). Such blocks when further refined, take into account the route of that net u in their routing area even though net u has no circuit elements in the blocks in question. Similarly, if a block is to be further refined that contains a circuit element for the net u, and the route for the net u must exit from one of these sub-blocks to the neighboring block then routing area is again set aside for that exiting connection.

To ensure that the necessary routing area is reserved, our approach uses *virtual circuit elements*. These pseudo-elements are constrained with respect to SHARP's geometric partitioning movements. There are two kinds of virtual circuit elements, *floating* and *anchored* (note only floating cells are introduced in the

current step). The floating virtual circuit elements are allowed one degree of freedom and thus can move within a specified row or column depending upon the associated routing segment. The anchored virtual circuit elements are constrained to a particular SHARP sub-block. In Figure 4.a, the route of Figure 2.e is reproduced. Figure 4.b further decomposes that layout surface by an additional level. The small white-filled squares within the double-sided arrows represent the floating virtual circuit elements associated with the route of Figure 4.a. The arrows indicate the range of permissible sub-block movement. Observe that the floating virtual circuit elements occur in pairs on either side of a block boundary.



(a) (b)

Figure 4. Floating virtual circuit elements.

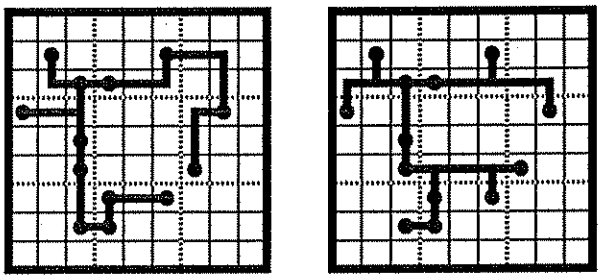
Once the floating virtual elements are inserted, the actual refinement of the partial solution proceeds by invoking the SHARP geometric partitioning algorithm on the actual and virtual circuit elements that lie in the given partial solutions's associated block. Upon computing the block refinement, an update of the positions of the affected circuit elements and net routes is performed.

Interconnecting virtual element pairs

As the partial solutions are refined, all of the floating virtual circuit elements are assigned to specific sub-blocks. To establish a connection between a pair of floating virtual circuit elements, the algorithm assigns a minimum-length connection to the pair in a manner that minimizes the local congestion measure.

Smoothing out the refinements

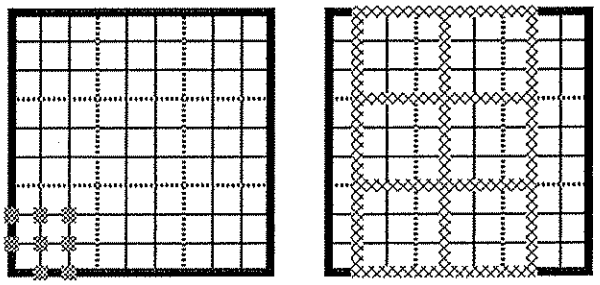
In spite of such activities as geometric partitioning, Steiner tree manipulation, congestion balancing, and virtual circuit elements, the independent sub-problem solving can introduce sub-optimal characteristics into the placement and associated routing. Although these sub-optimal features are infrequent, they generally occur near SHARP sub-region boundaries and are corrected heuristically in the *template smoothing* step. One such example is demonstrated in the net route given in Figure 5.a. Although the sub-routes in each of the nine major blocks are locally optimal, the overall connection does not have minimal length. The improved route in Figure 5.b is the result of the smoothing activity.



(a) (b)

Figure 5. A template smoothing result.

Template smoothing reduces routing irregularities by considering eight different subsections (*tilings*) of the original layout surface. The eight marked squares in Figure 6.a represent the lower left hand corners of the eight subsections. The wavy lines in Figure 6.b form the six *tiles* associated with one of the eight subsections. Each of the tiles covers a portion of the layout surface consistent with the current refinement. Four of the remaining seven tilings for the figure have four tiles per tiling and three of the remaining tilings have six tiles per tiling.



(a) (b)

Figure 6. Template tiling.

Every tile has the property that it covers a portion of at least two different blocks of the current refinement of *S*. This offset in block coverage ensures good inter-block connections. Although there are always eight different tilings per refinement, the number of tiles per tiling can increase by a factor of nine with each additional refinement.

All of the tiles of a given tiling can be processed in parallel as they represent distinct portions of the layout surface. As in the first parallel step of the refining activity, virtual circuit elements are introduced to

ensure that routing area is reserved for the inter-tile portions of the net routes (as all of the routes are sufficiently detailed, the virtual circuit elements are all anchored). Also in a similar fashion, the SHARP geometric partitioning algorithm is invoked on the actual and virtual circuit elements that lie in the particular tile t. The updating of S based on sub-solution S_t can require modifying information in each of the partial solutions that overlaps t.

To further improve solution quality, after the eight tilings are processed, the original partitioning boundaries for the current refinement are reconsidered, and if necessary the template smoothing process is iterated.

Ripping-up and rerouting

In addition to correcting some net routes during template smoothing, it will generally be the case that some circuit elements will *migrate*. If a circuit element's migration causes it to be reassigned to a block associated with a different partial solution, then the actual routes for the nets associated with that circuit element may not correspond to the previous level's Steiner forms for those nets. If this is the case for a particular net, then its refined routing is ripped-up, and the affected partial solutions are instead individually assigned minimum-length Steiner forms that best approximate this routing at the previous level.

At this point in the algorithm, the current level of refinement has been completed. The algorithm then repeats its major while-loop to compute a new level of refinement. The iteration continues until an exact placement has been constructed.

IV. Resource Utilization

The above circuit layout approach is designed for execution on a MIMD (multiple-instruction, multiple data) parallel machine. Since the algorithm essentially follows the divide-and-conquer paradigm, it can be readily implemented on either a message-passing or shared-memory architecture.

Our experiments have determined that when a sub-instance is reduced to eight circuit elements or less, then that sub-instance is sufficiently *simple* to be solved directly. Let s be the average number of circuit elements in a simple problem instance.

Since each SHARP block requires significant computation effort, PARFECT should be able to effectively use one processor per block. Hence, the desired number of processors for a circuit layout instance is basically a function of the number of SHARP blocks present at the maximum level of refinement which is approximated by n/s, where n is the number of circuit elements in the instance. Using Primary 1 with its 752 circuit elements and 904 nets as an example [15], we have found that our layout algorithm typically produces four levels of refinement before every SHARP block is sufficiently simple to be solved directly. In addition, there are usually on the order of 275 SHARP blocks at that maximum refinement level. Further analysis indicates that PARFECT should be able to effectively use 1500 - 2000 processors for a 10,000 circuit element instance.

The chart in Figure 7 indicates how much time the major steps of a sequential implementation require for a typical Primary 1 solution. A solution is generally

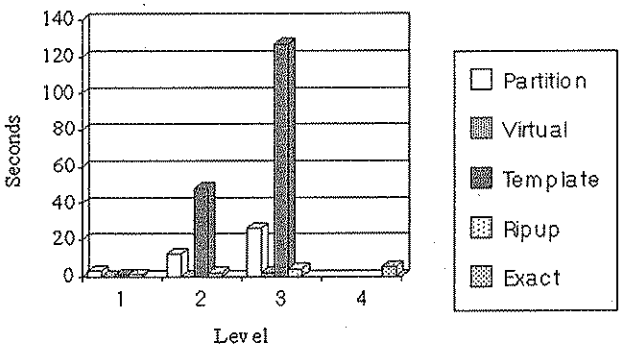


Figure 7. Running times.

computed in less than four minutes on a Sparc IPC which is approximately 20% of the time required by TIMBERWOLF [17]. By using the number of SHARP blocks present at a given level of refinement, we can extrapolate that the second level of refinement which currently requires on average slight more than a minute can be reduced to approximately 7 seconds. Similarly, a parallel implementation can reduce the third level of refinement from nearly three minutes to about three seconds. Overall, we expect a parallel implementation would require on the order of 14 seconds to solve the problem instance.

V. Solution Quality

Although our method is still actively evolving and thus the results are still being improved, the methodology consistently produces solutions with an expected core area size that is approximately the same as TIMBERWOLF's solution (within 2%). However, comparing expected wire usage by using the half-perimeter wire metric, we are still using 15-20% more wire. We expect to overcome this difference both by increasing the amount of circuit element movement in the partitioning phase and by fine-tuning the process-

ing of simple instances (Level 4 for the Primary 1 instance).

VI. Implementation Status

We have completed a preliminary sequential version in C and are currently converting this implementation to the MENTAT variant of C++ [9]. The MENTAT variation will be capable of both running in sequential or parallel mode. Independent of this implementation, we believe that the algorithm described here is part of a significant, new formulation for the circuit layout problem. We are also currently investigating a performance-driven version of the method that uses a Steiner tree variant [5] that accounts for propagation delay.

VII. Acknowledgments

The authors' work have been supported in part through National Science Foundation grants MIP-9107717 and CDA-8922545, Virginia CIT award 5-30971, and the SIGDA Design Automation fellowship. Their support is greatly appreciated.

VIII. References

[1] S. Bapat and J. P. Cohoon, Sharp-Looking Partitioning, *Second European Design Automation Conference (EDAC 91)*, Amsterdam, Netherlands, February 1991, 172-176.

[2] S. Bapat and J. P. Cohoon, Sharp: A Hierarchical VLSI Layout Methodology based on Geometric Partitioning and Steiner Routing, submitted to *1993 ACM-IEEE Design Automation Conference.*

[3] R. J. Brouwer and P. Banerjee, Phigure: A Parallel Hierarchical Global Router, *27th ACM/IEEE Design Automation Conference Proceedings,* Orlando, Florida, 1990, 650-653.

[4] A. Casotto, F. Romeo, and A. Sangiovanni-Vincentelli, A Parallel Simulated Annealing Algorithm for the Placement of Macro-Cells, *IEEE International Conference on Computer-Aided Design,* Santa Clara, California, 1986, 30-33.

[5] J. P. Cohoon and L. J. Randall, Critical Net Routing, *International Conference on Computer Design: VLSI in Computers,* Cambridge, Massachusetts, October 1991, 174-177.

[6] W. Dai and E. S. Kuh, Simultaneous Floor Planning and Global Routing for Hierarchical Building-Block Layout, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems cad-6,5* (September 1987), 828-837.

[7] A. E. Dunlop and B. W. Kernighan, A Procedure for Placement of Standard-Cell VLSI Circuits, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems cad-4,1* (January 1985), 92-98.

[8] C. M. Fiduccia and R. M. Mattheyses, A Linear-Time Heuristic for Improving Network Partitions, *19th ACM/IEEE Design Automation Conference Proceedings,* Las Vegas, Nevada, 1982, 175-181.

[9] A. Grimshaw, Easy-to-use Parallel Object-oriented Parallel Processing with Mentat, *IEEE Computer,* May 1993.

[10] B. W. Kernighan and S. Lin, An Efficient Heuristic Procedure for Partitioning Graphs, *Bell System Technical Journal* 49,2 (February 1970), 291-307.

[11] U. Lauther, A Min-Cut Placement Algorithm for General Cell Assemblies Based on a Graph Representation, *16th ACM/IEEE Design Automation Conference Proceedings,* San Diego, CA, 1979, 1-10.

[12] S. Mayrhofer and U. Lauther, Congestion-Driven Placement Using a New Multi-Partitioning Heuristic, *IEEE International Conference on Computer-Aided Design,* Santa Clara, CA, 1990, 332-335.

[13] M. Pedram and B. Preas, Interconnection Length Estimation for Optimized Standard Cell Layouts, *IEEE International Conference on Computer-Aided Design,* Santa Clara, California, 1989, 390-393.

[14] B. T. Preas and M. J. Lorenzetti, *Physical Design Automation of VLSI Systems,* Benjamin/Cummings, Menlo Park, California, 1988.

[15] B. T. Preas and K. Roberts, *IEEE/ACM SIGDA Physical Design Workshop: Placement and Floorplanning,* Hilton Head, South Carolina, April 1987.

[16] S. A. Kravitz and R. A. Rutenbar, Multiprocessor-based Placement by Simulated Annealing, *23rd ACM/IEEE Design Automation Conference Proceedings,* Las Vegas, Nevada, 1986, 567-573.

[17] C. Sechen and A. Sangiovanni-Vincentelli, The TimberWolf Placement and Routing Package, *IEEE Journal of Solid-State Physics* sc-20,2 (1985), 510-522.

[18] P. R. Suaris and G. Kedem, A Quadrisection-Based Combined Place and Route Scheme for Standard Cells, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems cad-8,3*(March 1989), 234-24.