A PARALLEL HIERARCHICAL ALGORITHM FOR MODULE PLACEMENT BASED ON SPARSE LINEAR EQUATIONS

*Zhaoyun Xing and Prithviraj Banerjee**

Computational Science and Engineering
469 Computer and Systems Research Lab
1308 W. Main St.
University of Illinois
Urbana, IL 61801
{banerjee, xing}@crhc.uiuc.edu

ABSTRACT

We present a fast and effective module placement algorithm which is based on the PROUD algorithm. The PROUD algorithm uses a hierarchical decomposition technique and the solution of sparse linear systems of equations based on a resistive network analogy. It has been shown that the PROUD algorithm is suitable for solving the placement problem for very large circuits, and obtains placement qualities that are comparable to the best placement algorithms based on simulated annealing, but is several orders of magnitude faster. In this paper, we first report on an improved hierarchical placement algorithm which is based on perturbing the matrices in the matrix equation solution stage of the PROUD algorithm. The new modified PROUD algorithm performs much faster that the original PROUD algorithm. We subsequently propose parallel versions of the original and modified algorithms that combine both fine grain and coarse grain parallelism to obtain another order of magnitude improvement in the runtime without loss of the quality of the layout. Results are reported using MPI on various multiprocessor systems for a variety of large layout benchmark circuits.

1. INTRODUCTION

The module placement problem involves placing a set of modules (standard cells, gate arrays, sea-of-gates) on a VLSI chip, given a netlist that specifies the connectivity between the modules. The objective is to minimize the total wire-length between the modules. In this paper, we deal with the placement problem of point modules (where the modules are of equal size, and the connections to the nets are assumed to be at the center of the module) and assume that all nets are two-terminal nets. Multi-pin nets are preprocessed and replaced with two-pin nets. Various approaches are available for solving the placement problem[13].

In modern integrated circuit design, with the trend of rapidly increasing gate counts, the use of parallel processing is becoming an attractive solution to reduce the inordinate amount of time spent in VLSI circuit design. In this paper we propose a parallel algorithm for module placement that is based on a hierarchical algorithm called PROUD [15], which is the basis of a commercial tool from ARCSYS.

PROUD uses the quadratic placement formulation [7], however, instead of determining the eigenvalues and eigenvectors of a large matrix, the concept of resistive network optimization is used [5]. The PROUD algorithm is especially suitable for extremely large designs having tens of thousands of gates. and produces placement qualities comparable to those obtained by simulated annealing (e.g. Timberwolf [14]) but in runtimes that are orders of magnitude faster.

In this paper, we first propose a modification to the serial module placement algorithm based on the concept of *perturbation* of the connection matrix. We show experimentally how this modified algorithm improves the runtime of the serial algorithm by an order of magnitude without loss of quality of the layout. We subsequently parallelize the original and modified module placement algorithms and show that we can further reduce the runtimes by another order of magnitude without loss of quality. We report on the implementation of the parallel hierarchical algorithms using Message Passing Interface (MPI) on various multiprocessor systems and report the results of the algorithm on a set of large layout benchmarks.

2. RELATED WORK

We will briefly review some parallel algorithms for placement proposed by other researchers [1]. Parallel algorithms for placement have been based on *iterative pairwise exchange* [6, 17], and *force-directed* approaches [8]. While these algorithms are fast, and a large degree of parallelism can be exploited, the solution quality obtained from the algorithms are inferior to simulated annealing.

Several parallel algorithms based on *simulated annealing* have been reported. The advantage of these algorithms is that it gives the best quality of the solution among all sequential algorithms for the placement problem. Parallel algorithms have been proposed based on move decomposition [11] and parallel move evaluation [2, 4, 11].

Other researchers have proposed parallel algorithms for placement based on *simulated evolution* [10], and *genetic algorithms* [12]. The algorithms require runtimes comparable to simulated annealing, but their quality of solutions is inferior; however, these algorithms produce good speedups.

The parallel algorithms for placement based on *hierarchical decomposition* use a divide and conquer method of solution by recursively decomposing the placement solution on the entire chip into subproblems of the placement of cells on smaller chip regions [3].

In this paper, we propose a parallel hierarchical algorithm for module placement that is suitable for very large chips and that is based on an extremely efficient serial algorithm, namely PROUD [15, 16]. The advantages of this approach are that it is possible to exploit a large degree of parallelism, and that is based on an algorithm whose results have been known to be comparable to those based on simulated annealing at much lower runtimes.

*This research was supported in part by the Semiconductor Research Corporation under contract SRC 94-DP-109 and the Advanced Research Projects Agency under contract DAA-H04-94-G0273 administered by the Army Research Office.

3. OVERVIEW OF THE PROUD ALGORITHM

In this section we will review the PROUD algorithm [15, 16] on which our parallel algorithm is based. PROUD is a hierarchical algorithm. It has two major components: global placement by solving a system of equations, and partitioning based on the result of placement. Global placement determines the relative positions of the modules in the current block for the partitioning routine. By sorting the modules by the positions, the partitioning stage bisects the current block into two sub-blocks.

Global placement is based on two assumptions, namely, that all modules are point modules and that all nets are two-pin nets. Thus multi-pin nets are expanded into two-pin net cliques. The weight of each edge of an r-terminal net is set to $2/r$. Let matrix $C = (c_{ij})$, where $c_{ii} = 0$ and c_{ij} be the number of nets connecting module i and module j. Let (x_i, y_j) represent the location of module i, and $\vec{x}$ and $\vec{y}$ represent the n-dimensional vectors which specify the coordinates of the n modules. The objective is to minimize the sum of the squared wire lengths $L(\vec{x}, \vec{y}) = \frac{1}{2}\sum_{i,j=1}^{n} c_{ij}[(x_i - x_j)^2 + (y_i - y_j)^2]$. Let D be diagonal matrix with entries $d_{ii} = \sum_{j=1}^{n} c_{ii}$ and $B = D - C$, then

$$L(\vec{x}, \vec{y}) = \vec{x}^T B\vec{x} + \vec{y}^T B\vec{y}$$

Based on a resistive network analogy, the minimization of the term $\vec{x}^T B\vec{x}$ can be achieved by solving the following system of equations.

$$B_{11}\vec{x}_1 + B_{12}\vec{x}_2 = 0 \tag{1}$$

where $\vec{x}_1$ is the vector of x coordinates of all movable modules in current block and $\vec{x}_2$ is the vector of x coordinates of all the pads and all other modules not inside current block. We assume that all I/O pad locations are fixed. B_{11} consists of columns of B with the same indices as $\vec{x}_1$. Since B is symmetric, B_{11} is also symmetric. B_{12} consists of columns of B with the same indices as $\vec{x}_2$. The reader is referred to [15] for details.

At each level of the hierarchy, there is a coordinate used by global placement. Suppose the current coordinate is x. Global placement obtains the relative x coordinate positions of all the modules inside the current block by solving Equation (1). According to the relative positions, over the x coordinate, the chip area is partitioned into two equal-sized sub-blocks with roughly equal number of modules in each sub-block. Then the new blocks will placed by global placement. The loop of global placement and partitioning is executed a few times to make the placement more accurate. This number is called the block Gauss-Seidel number. In the next level of the hierarchy, the coordinate is switched to y. This process continues until each block contains at most one module.

4. A MODIFIED SEQUENTIAL PROUD ALGORITHM WITH PERTURBATION THEORY

It can be noticed from the description of the PROUD algorithm of the previous section that the solution of Equation (1) dominates the computation of PROUD. If we let $\vec{b} = -B_{12}\vec{x}_2$, then Equation (1) becomes

$$B_{11}\vec{x}_1 = \vec{b} \tag{2}$$

By definition of B_{11}, it is always symmetric. When solving a system of equations like (2) by an iterative method such as Gauss-Jacobi or Gauss-Seidel method, the convergence rate is determined by the condition number of the coefficient matrix B_{11}[9]. When B_{11} tends to be singular, its condition number increases, and thus the convergence rate of Equation 2 decreases. Since the magnitude of diagonal elements of B is the sum of its off diagonal elements, it can be shown that at top level of PROUD, the matrix B_{11} could be singular if there are no pads, or not many pads specified for the chip and hence the convergence of the solver of Equation (1) could be very slow.

Definition 1 Let $B_{11} = D - C$, where $D = (d_{ii})$ is sub-matrix of B_{11} with diagonal entries and $C = (c_{ij}), i \neq j$ is sub-matrix of B_{11} with off-diagonal entries. We call $B'_{11} = (1+\epsilon)D - C$ the diagonally perturbed matrix of B_{11}. If we use B'_{11} for B_{11} in (1) of PROUD, the algorithm is called modified PROUD.

The nice property of modified PROUD is that the matrix B_{11} in Equation (1) is always nonsingular, and the condition number is always smaller than that in PROUD. Since the solution of Equation (1) is used only for the partition, when ϵ is small the solution of Equation (1) will not change much. Therefore the partition should not change much either. This can be proven both in theory and by our experiment. Details are described in [?].

4.1. Comparison of PROUD and Modified PROUD

In this section, we will show how the perturbation can speed up the performance of the serial PROUD algorithm, and how much quality difference it makes. In theory, based on matrix B, ϵ can be computed such that after perturbation the relative position sequence does not change. Initially we start with a big ϵ value, then according to the quality of the result, we gradually reduce the ϵ value. The effect of the perturbation is significant when the circuit has a fairly large size.

We chose a benchmark circuit *Industry-1* from the MCNC layout synthesis benchmarks as our test circuit. When comparing the original PROUD and the modified PROUD algorithms, we compared the overall performances and the qualities of the layouts measured in terms of total wire length.

As a comparison, on a Sun4/690MP we ran TimberWolf 6.0 based on simulated annealing and the modified PROUD with $\epsilon = 0.1$ for the *Industry-1*, assuming that all circuits have equal size modules and all nets have two terminals. We first ran the modified PROUD and then input the placement of the modified PROUD to TimberWolf 6.0 to evaluate the cost of the placement. The result is given in Table 1.

TimberWolf 6.0 has a running time of 49364.3 seconds and a final placement cost of 19947479. Compared with TimberWolf 6.0, PROUD has a speedup of 35 and a quality degradation of 6.7%. But when $\epsilon = 0.05$ the modified PROUD has a speedup of 180 and a quality degradation of only 10.0 %. As stated in the analysis section, after perturbation, the regularity of the coefficient matrix is totally changed. For the circuit *Industry-1*, compared with PROUD, the speedup of the modified PROUD is quite significant as shown in Table 1. For example when $\epsilon = 0.05$ the overall speedup is 5.08. But the cost increase is only about 3.1%. Though by intuition, the modified PROUD is an approximation of PROUD, it should not do better than PROUD, this is not always true. For example, when $\epsilon = 0.001$, the modified PROUD outperforms PROUD by 1.0% and has 1.29 over all speedup.

5. PARALLEL PROUD ALGORITHMS

We will now report on a parallel version of both PROUD and modified PROUD algorithm for module placement so that we can further improve the runtimes by another order of magnitude without loss of quality. We report on the implementation of the parallel hierarchical algorithm on various multiprocessor system such as the Intel Paragon, the SGI Challenge, and a cluster of

Table 1. Variation of the running time and cost of Industry-1 with various perturbation coefficient ϵ

ϵ	running time (second)	speedup over TimberWolf	speedup over PROUD	cost	change % from TimberWolf	change % from PROUD
0.0	1392.47	35.46	1.00	21285672	6.7	0.0
0.001	1076.19	45.88	1.29	21072600	5.6	-1.0
0.01	559.08	88.31	2.49	21984874	10.2	3.3
0.05	274.00	180.1	5.08	21957892	10.0	3.1

Table 2. Runtimes (seconds) and speedups of parallel PROUD on the Intel Paragon

	1 proc	2 procs	4 procs	8 procs	16 procs
Industry1	489	493(0.99)	247(1.98)	139(3.51)	92(5.31)
Industry2	36034	19516(1.84)	10432(3.45)	7406(4.87)	4752(7.58)
Industry3	4152	2610(1.59)	1742(2.38)	1031(4.03)	746(5.59)

workstations, and report the results of the algorithm on a set of large layout benchmarks.

Figure 1 illustrates our design of parallel PROUD. Suppose we have 16 modules to be placed and there are 4 processors. At the first level of the hierarchy, all 4 processor participate in the global placement. A master process will control which variables a slave processor needs to iterate when concurrently working on one Gauss-Seidel iteration. After placement the master processor bisects the portion of the chip it owns into two sub-chips with roughly equal number of modules on each. At the same time the working group is partitioned into 2. Processor p_0 and p_1 will jointly place 8 modules and p_2 and p_3 will place the remaining 8 modules. Both 2 processor groups work exactly the same way as the first hierarchy. After one more level each working group will have only 1 processor. At this time this processor will execute the sequential modified PROUD algorithm. An outline of the parallel placement algorithm is given in Procedure 1.

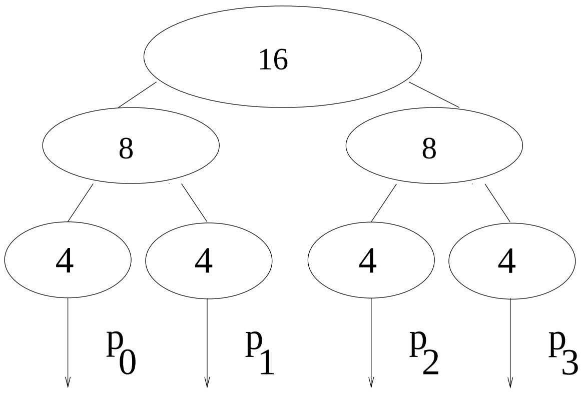

Figure 1. Illustration of our parallel algorithm on a chip with 16 cells using 4 processors

Procedure 1 (Parallel PROUD Algorithm)

Input: net list, cell library, I/O pad location, number of BGS number, number of hierarchies, and perturbation ϵ

Output: placement result

Algorithm:

1. Assign all modules to one internal block, and assign all I/O pads to one I/O block.
2. For each level in the hierarchy

   Repeat number of BGS iterations (repeat only once for the first hierarchical level)

Table 3. Characteristics of the benchmark circuits

circuit	cell number	pad number	net number
Industry1	2271	814	2593
Industry2	12142	495	13419
Industry3	15059	374	21940

   For each internal block

   (a) Construct B_{11} and B_{12}, and determine $\vec{x}_1$ and $\vec{x}_2$.
   (b) Determine slaves
   (c) Until change in $\vec{x}$ is small enough
      i. Send work to each slave
      ii. Do its own part of BGS iteration
      iii. Send slaves its work
      iv. Receive slave's work
   (d) Temporarily cut the block into two sub-blocks according to the cut method for this hierarchy. Assign the modules into separate sub-blocks by sorting.

3. Fix the cut at center for each internal block.
4. If there are slaves then send block to one of the slave. That slave becomes a new master. It receives half of the slaves of its old master.
5. Form the new set of internal blocks.

5.1. Experimental Evaluation

We implemented the parallel algorithm outlined in Procedure 1 using MPI, a standard specification for message-passing libraries, on various multiprocessor systems such as the Intel Paragon, the SGI Challenge, and a cluster of Sparc5 workstations. We report results on a set of benchmarks from the MCNC layout benchmark set. Table 3 shows the characteristics of the benchmarks.

5.2. PROUD vs. Parallel PROUD

We now report the runtimes (in seconds), the speedups of our parallel algorithm with respect to the PROUD algorithm. This corresponds to the case $\epsilon = 0$ in the modified PROUD. Table 2 shows that speedups of parallel PROUD versus PROUD on the Intel Paragon. Table 4 shows the speedups on Sparc5 workstation clusters.

5.3. Parallel Modified PROUD vs. Modified PROUD

We now report on the runtimes and speedups of the parallel versions of the modified PROUD algorithm in Table 5 and Table 6 for the Intel Paragon and the SGI Challenge respectively. In all

Table 4. Runtimes (seconds) and speedups of parallel PROUD on Sparc5 workstation cluster

	1 proc	2 procs	4 procs
Industry1	858	579(1.48)	551(1.58)
Industry3	6504	5365(1.21)	4016(1.62)

our experiments, the perturbation parameter $\epsilon = .1$. Our experiments show that the parallel version of the modified PROUD algorithm provides fairly good speedups in most cases except when we run the circuit Industry2 on the Intel Paragon. At that time since the circuit has a large number of two terminal nets, the communication cost becomes dominant.

Table 5. Runtimes (seconds) and speedups of parallel modified PROUD on Intel Paragon

	1 proc	4 procs	8 procs	16 procs
Industry1	169	90(1.88)	58(2.91)	40(4.23)
Industry2	1081	991(1.09)	1216(0.89)	1812(0.6)
Industry3	854	368(2.32)	242(3.53)	168(5.1)

Table 6. Runtimes (seconds) and speedups of parallel modified PROUD on SGI Challenge

	1 proc	2 procs	4 procs
Industry1	590	482(1.22)	329(1.55)
Industry2	3944	2579(1.53)	1568(2.52)
Industry3	2864	1835(1.56)	1315(2.18)

6. CONCLUSION

In this paper, two techniques have been proposed to speedup PROUD, one of the the best deterministic module placement algorithms which can produce results of comparable quality to simulated annealing much faster:

1. Using matrix diagonal perturbation, the regularity of coefficient matrix of some system of equations is changed. This speeds up the process of solving the system of equations by the Gauss-Seidel method. Since the solution of this system of equations is the dominant part of PROUD computation, this technique speeds up the sequential PROUD significantly. Because of the unique property of PROUD algorithm, this approximation only slightly degenerate the placement result. In some cases, it can even improves the placement result.

2. Combining fine-grain parallelization at the top level and coarse-grain parallelization of the lower level after there is enough work for all processors, we designed a parallel algorithm for MIMD parallel architectures. This algorithm speeds up the perturbed version of the PROUD algorithm by another order of magnitude. Since the communication overhead is not avoidable because of the data dependence structure of Gauss-Seidel method, it is not possible to get linear speedup. But from our experiments, we do get good speedups when the circuits are large.

REFERENCES

[1] P. Banerjee, *Parallel Algorithms for VLSI Computer-Aided Design,* PTR Prentice Hall, Englewood Cliffs, New Jersey 07632, 1994.

[2] P. Banerjee, M. H. Jones, and J. S. Sargent. Parallel Simulated Annealing Algorithms for Standard Cell Placement on Hypercube Multiprocessors. *IEEE Trans. Parallel Distributed Systems*, 1(1):91–106, Jan. 1990.

[3] R. J. Brouwer and P. Banerjee. PARAGRAPH: A Parallel Algorithm for Simultaneous Placement and Routing Using Hierarchy. *Proc. European Design Automation Conf. (EDAC-92)*, Mar. 1992.

[4] A. Casotto, F. Romeo, and A. Sangiovanni-Vincentelli. A Parallel Simulated Annealing Algorithm for the Placement of Macro-Cells. *IEEE Trans. Computer-aided Design Integrated Circuits Systems*, pages 838–847, Sept. 1987.

[5] C. Cheng and E. Kuh, Module placement based on resistive network optimization, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* CAD-3, (3), pp.218-225, 1984.

[6] D. J. Chyan and M. A. Breuer. A Placement Algorithm for Array Processors. *Proc. 20th Design Automation Conf.*, pages 182–188, June 1983.

[7] K. Hall, An r-dimensional Quadratic Placement Algorithm. *Management Sci.*, vol. 17, no.3, pages 219-229, Nov. 1970.

[8] E. I. Horvath. A Parallel Force Directed Standard Cell Placement Algorithm. Technical report, Dept. Computer Science, Florida Atlantic University, Boca Raton, FL, 1992.

[9] D. Kincaid and W. Cheney, *Numerical Analysis*, Brook/Cole Publishing Company, Pacific Grove, California 93950, 1991.

[10] R. M. Kling and P. Banerjee. Concurrent ESP: A Placement Algorithm for Execution on Distributed Processors. *Proc. Int. Conf. on Computer-aided Design,* pages 354-357, Nov. 1987.

[11] S. A. Kravitz and R. A. Rutenbar. Placement by Simulated Annealing on a Multiprocessor. *IEEE Trans. Computer-aided Design Integrated Circuits Systems*, CAD-6(4):534–549, June 1987.

[12] S. Mohan and P. Mazumder. Wolverines: Standard Cell Placement on a Network of Workstations. *IEEE Trans. Computer-aided Design Integrated Circuits Systems,* 12(9):1312-1326, Sept. 1993.

[13] B. Preas and M. Lorenzetti. *Physical Design Automation of VLSI Systems.* Benjamin-Cummings Publishing Co., Menlo Park, CA, 1988.

[14] C. Sechen and A. Sangiovanni-Vincentelli, The TimberWolf placement and routing package. *IEEE Journal for solid State Circuits,* vol. SC-20, pp. 510-522. Apr. 1985.

[15] pR. Tsay and E Kuh, Module Placement for Large Chips Based on Sparse Linear Equations. *International Journal on Circuit Theory and Applications,* vol. 16, pp. 411–423, 1988.

[16] R. Tsay and E Kuh, A Fast Sea-Of-Gates Placement Algorithm, *Proc. of 25th ACM/IEEE Design Automation Conference*, June 1988, pages 318-322.

[17] K. Ueda, T. Komatsubara, and T. Hosaka. A Parallel Module Placement Approach for Logic Module Placement. *IEEE Trans. Computer-aided Design Integrated Circuits Systems of Integrated Circuits and Systems*, CAD-2(1):39–47, Jan. 1983.

[18] Z. Xing and P. Banerjee, A Parallel Hierarchical Algorithm for Module Placement Based on Sparse Linear Equations, *UIUC-CRHC Technical Report*, 1996.