A PARALLEL ROW-BASED ALGORITHM FOR STANDARD CELL PLACEMENT WITH INTEGRATED ERROR CONTROL †

Jeff S. Sargent and Prith Banerjee

Computer Systems Group
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Urbana, IL 61801

ABSTRACT

A new row-based parallel algorithm for standard-cell placement targeted for execution on a hypercube multiprocessor is presented. Key features of this implementation include a dynamic simulated-annealing schedule, row-partitioning of the VLSI chip image, and two novel approaches to control error in parallel cell-placement algorithms: (1) Heuristic Cell-Coloring; (2) Adaptive Sequence Length Control.

1. INTRODUCTION

Simulated annealing is a general-purpose optimization method that has been successfully applied to solve a large variety of combinatorial optimization problems including many in VLSI design. Annealing is computationally very expensive, hence efforts to improve execution time has proceeded along two fronts: (1) accelerating the annealing schedule, and (2) parallelizing the annealing algorithm for execution on multiprocessors.

Parallel implementations of annealing as applied to the cell placement application either attempt multiple cell moves in parallel [1-7], or distribute the evaluation phases of a single move across several processors [8]. Jones and Banerjee [1, 2] developed a parallel algorithm based on the TimberWolf cell placement program [9] for a distributed-memory hypercube, that evaluated multiple moves in parallel, then synchronized (i.e. updated cell-position information at all processors) after each parallel move set. This global synchronization overhead is an unavoidable constraint on the performance of algorithms written for distributed-memory machines If the synchronization is performed less frequently, the error in cost calculation due to interacting moves would increase because of out-of-date cell position information.

We have developed a new parallel cell-placement algorithm for a hypercube multiprocessor utilizing an adaptive annealing schedule with integrated error control. Two methods of error control are developed: (1) **Heuristic Cell Coloring** which identifies sets of non-interacting cells that can be moved repeatedly, and in parallel, with no buildup of error in the placement cost; and (2) **Adaptive Sequence Length Control** which allows multiple parallel cell moves to take place between global cell-position "updates" with a feedback mechanism controlled by an error bound. The dynamic annealing schedule is a parallel variation of Huang's schedule [10].

† This research was supported in part by NASA under Grant NASA NAG 1-613, and in part under National Science Foundation Equipment Grant NSF CCR 87-05240.

2. ROW-BASED PLACEMENT ALGORITHM

Overview of Parallel Algorithm

The basic theme of our parallel algorithm is to divide up the area of the VLSI chip image into equal-sized sub-areas, and allocate each sub-area to a separate processor. Cells are assigned an initial position but are free to migrate across the layout. The cell's associated data structure passes from processor to processor as it crosses sub-area boundaries during placement evolution. Processors pair up to evaluate four different move-types (inter-processor and intra-processor cell displacements and exchanges), drawing potential cells from the sub-areas assigned to processors in a pair. On a $P=2^n$ processor hypercube, a total of $P/2$ moves are evaluated in parallel at each "parallel move".

Row partitioning

In an earlier study, Jones and Banerjee had reported a parallel cell-placement algorithm for a hypercube multiprocessor [2], which partitioned the chip area into square blocks, or a grid. Our new algorithm instead incorporates row partitioning, where each processor of a 2^n processor hypercube gets an entire row or set of rows of the standard cell layout. The mapping between row positions and processors corresponds to the binary reflected Gray code for linear arrays [11]. Let $G_n = (g_0, g_1, ..., g_{2^n-1})$ be the sequence of n-bit binary numbers of a binary reflected Gray code. Then row i gets mapped to processor g_i. For example, for a standard cell circuit with 8 rows of cells, each processor in a 8-processor hypercube will be assigned an entire row as shown in Fig. 1.

The primary benefit of row partitioning is that the entire cost of any move-type can be computed solely on the basis of local information, along with its partner node. With grid partitioning it was necessary to maintain information on the placement of cells at a node's east and west neighbors to assess the two penalty components of the cost function: the non-ideal row length penalty and the overlapping cell overlap penalty [2]. The overhead of maintaining this "neighborhood" placement information can be very high since this information must be exchanged after every parallel move to maintain state cost accuracy. By nature, row partitioning has all such neighborhood placement information, eliminating this costly exchange. Most importantly, with exact knowledge of the two penalty cost components at each processor, the *only error that accumulates due to parallel moves is error in total wire-length.*

Inter-processor Communication Pattern

Processors pair up to evaluate a single move-type - either a displacement or an exchange. The row-wise mapping using the binary reflected Gray code has the property that g_i and $g_{i\pm 2j}$ differ in exactly two bits for all $j>0$. Hence any two processors with node numbers g_i and $g_{i\pm 2j}$ will be separated by at most two link-hops for $0 \le i \pm 2j \le 2^n - 1$. An example of this mapping, and the pairwise communication that is possible with such a mapping, is shown on the right side of Fig. 1. Arrows originating in row 3 allocated to processor g_3=2 indicate pairwise move resolution with other rows $3\pm 2j$ belonging to processors $g_{3\pm 2j}$, for $0<j<2$. This provides geographic uniformity for hypercubes of any size.

Annealing Schedule

In any simulated annealing algorithm, four important criteria are the choice of the initial temperature, the equilibrium detection condition at a particular temperature, the rate of decrease of the temperature, and the frozen or termination condition. All previously reported implementations of parallel simulated annealing use a fixed temperature sequence derived empirically [2-4, 6-8]. Huang et al have proposed an adaptive cooling schedule for serial simulated annealing algorithms based on the characteristics of the cost distribution and the annealing curve itself [10]. The overhead of adaptive scheduling is not large for a uniprocessor algorithm such as TimberWolf, but can be very substantial in an algorithm such as ours that proposes cell-moves in parallel. The system cost would have to be sampled after each accepted move to build an accurate cost distribution, requiring a costly global synchronization and updating after every parallel move. To experimentally validate Huang's schedule for parallel annealing we have implemented a version that only allows one parallel move-set between updates.

3. INTEGRATED ERROR CONTROL

In algorithms such as ours that move multiple cells independently and simultaneously among different pairs of processors, errors may be produced in the interconnection length portion of the cost function when more than one processor pair simultaneously alters cell positions of a shared net. Such an error has been classified as *temporary error* by researchers, since there is an implied procedure of resynchronization and updating of distributed cell positions after each parallel move. Large amounts of temporary error do not seem to impair convergence to a "good" solution. Of more interest is the error accumulated after several parallel moves with no intervening cell-position update and is classified as *cumulative error*. Clearly the amount of misinformation will increase with each accepted move. However if this expensive synchronizing update could be reduced in frequency, overall execution time would be significantly improved, especially at low temperature. We have developed two new approaches for controlling error: **Heuristic Cell-Coloring** to eliminate temporary error; and **Adaptive Sequence-Length** Control to constrain cumulative error. A related approach to control error in parallel annealing has been presented in [12].

Heuristic Cell-Coloring

We have implemented an efficient circuit pre-processing algorithm called **Heuristic Cell-Coloring**, that completely eliminates temporary error in our parallel placement algorithm by identifying sets of noninteracting cells. Noninteracting cells can be moved repeatedly, and in parallel, without any accumulation of cell position misinformation in the distributed database.

Finding sets of unconnected nodes in an arbitrary graph is analogous to graph coloring. Clearly the circuit description of a standard-cell circuit can be directly posed as a graph where cells correspond to nodes and nets correspond to edges. Now the graph is colored so that no two connected vertices are the same color. All vertices (cells) of the same color are non-interacting, and can be moved repeatedly between updates without any error accumulation. Though optimal graph-coloring for arbitrary graphs is NP-complete, fast heuristic graph-coloring methods are available for graphs that are not "pathological cases" [13]. Table 1 shows some example colorings of standard cell circuits using two algorithms.

Adaptive Sequence-Length Control

We now describe a method used to control cumulative error. Our parallel algorithm dynamically extends and contracts the parallel move sequence length constrained by a bound on allowable error derived shortly. We wish to find an upper bound on the maximum permissible average error at a particular temperature. By adjusting the sequence length dynamically after each sequence the average error can be limited to a specific range. We base our approach on the characteristics of the move-acceptance curve. The probability of accepting a move is given by

$$P = e^{-\Delta C/T} \cdot P\{\Delta C > 0\} + P\{\Delta C < 0\}$$

In the presence of error the composite acceptance rate changes slightly, however the probability of generating good or bad moves is invariant with respect to error :

$$P_E = e^{-(\Delta C \pm E)/T} \cdot P\{\Delta C > 0\} + P\{\Delta C < 0\}$$

To bound the acceptance rate with error P_E to within 5% of normal, i.e. $\frac{P - P_E}{P} \le 0.05$, we find the following bound on the magnitude of error:

$$E_- \le -T / \ln(1+0.05) \approx T/21$$

If the average error computed after a sequence is higher than (T/21), the sequence length is reduced; if the average error is very low (T/42) then sequence length is increased slowly. Table 2 shows average sequence lengths for various circuits on different sized hypercubes.

4. IMPLEMENTATION AND RESULTS

The performance of this row-based placement algorithm is much faster than a previous grid-based parallel algorithm for the hypercube [2]. We have implemented the row-based algorithm on the Intel iPSC-2/D4/MX hypercube multiprocessor. Three variants of the algorithm were developed with different levels of error control. The simplest algorithm, FIXED-ROW, fixed the sequence length at one, and did not use heuristic cell-coloring. This version basically verified that Huang's annealing schedule could be adapted to a parallel annealing algorithm. The second algorithm, COLOR-ROW, is identical to FIXED-ROW, with the addition of heuristic cell-coloring. Lastly the third algorithm, ADAPTIVE-ROW, incorporates dynamic sequence-length adjustment and heuristic cell-coloring. The placement and runtime results are shown in Table 3 for various test circuits and hypercube sizes. The entries in the table showing "na" were runs that could not be completed due to memory limitations or excessive runtimes. The FIXED-ROW algorithm had good convergence characteristics but had poor runtime. The COLOR-ROW algorithm plainly did not converge for three of the circuits probably because of the restricted move-set available to the annealing algorithm under Heuristic Cell-Coloring. The ADAPTIVE-ROW algorithm which combined cell coloring and sequence length control performed the best in terms of placement quality and runtime. The resulting parallel algorithm using an adaptive sequence length control and cell coloring is outlined below.

```
Node_Program
{
  Determine initial temperature (use Huang's schedule).
  While placement not "frozen" (use Huang's schedule) do {
    For i=1 to sequence length S do {
      Select cell of a particular color from current node.
      If current color exhausted, then switch colors.
      Evaluate cell move with next Neighbor
        processor for same color.
      If move successful, record event in MoveQueue.
    }
    Broadcast local MoveQueue to all processors.
    Assemble global MoveQueue, update cell positions.
    Broadcast sequence of local delta-costs.
    Assemble global delta costs, update cost distribution.
    Adjust S based on perceived average error
    If at thermal equilibrium for this temperature,
     then reduce temperature (use Huang's schedule).
  }}
```

5. CONCLUSIONS

We have presented a new row-based parallel cell-placement algorithm based on the simulated annealing technique, designed to run on a hypercube multiprocessor. We have implemented a dynamic annealing schedule with an integrated error-control mechanism. We have addressed the problem of error in parallel cell-moves with two new techniques: **Heuristic Cell-Coloring** and **Adaptive Sequence-Length Control**. The algorithms have been implemented on the Intel iPSC/2 Hypercube. It may be noted from the results that the parallel annealing algorithms gave results comparable to uniprocessor results. However, the runtimes are quite large in both our sequential and parallel implementations because we have not used any sophisticated programming techniques such as the ones used in Timberwolf [14] since the purpose of this research was to evaluate the impact of parallelization in annealing and not to produce the most efficient implementation.

REFERENCES

[1] P. Banerjee and M. Jones, ''A Parallel Simulated Annealing for Standard Cell Placement on a Hypercube Computer,'' *Proc. Int. Conf. Computer-Aided Design*, pp. 34-37, Nov. 1986.

[2] M. Jones and P. Banerjee, ''Performance of a Parallel Algorithm for Standard Cell Placement on the Intel Hypercube,'' in *Proc. 24th Design Automation Conf.*, Miami Beach, FL, pp. 807-813, Jun. 1987.

[3] J. S. Rose, W. M. Snelgrove, and Z. G. Vranesic, ''Parallel Standard Cell Placement Algorithms with Quality Equivalent to Simulated Annealing,'' *IEEE Trans. Computer-Aided Design*, pp. 387-396, Mar. 1988.

[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*, pp. 838-847, Sep. 1987.

[5] F. Darema and G. F. Pfister, ''Multipurpose Parallelism for VLSI CAD on the RP3,'' *IEEE Design and Test of Computers*, vol. 4, pp. 19-27, Oct. 1987.

[6] A. Casotto and A. Sangiovanni-Vincentelli, ''Placement of Standard Cells Using Simulated Annealing on the Connection Machine,'' *Proc. Int. Conf. Computer-Aided Design*, pp. 350-353, Nov. 1987.

[7] C. P. Wong and R. D. Fiebrich, ''Simulated Annealing-Based Circuit Placement on the Connection Machine System,'' *Proc. Int. Conf. Computer Design*, pp. 78-82, Oct. 1987.

[8] S. A. Kravitz and R. A. Rutenbar, ''Placement by Simulated Annealing on a Multiprocessor,'' *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 534-549, Jun. 1987.

[9] C. Sechen and A. Sangiovanni-Vincentelli, ''TimberWolf3.2: A New Standard Cell Placement and Global Routing Package,'' *Proc. 23rd Design Automation Conf.*, pp. 432-439, Jun. 1986.

[10] M. D. Huang, F. Romeo, and A. Sangiovanni-Vincentelli, ''An Efficient General Cooling Schedule for Simulated Annealing,'' *Proc. Int. Conf. Computer-Aided Design*, pp. 381-384, Nov. 1986.

[11] T. F. Chan and Y. Saad, ''Multigrid Algorithms on the Hypercube Multiprocessor,'' *IEEE Trans. Computers*, vol. C-35, pp. 969-977, Nov. 11, 1986.

[12] R. Jayaraman and F. Darema, ''Error Tolerance in Parallel Simulated Annealing Techniques,'' in *Proc. Int. Conf. Computer Design*, New York, NY, pp. 545-548, Oct. 1988.

[13] Jonathan S. Turner, ''Almost All k-Colorable Graphs Are Easy to Color,'' *Journal of Algorithms*, vol. 9, pp. 63-82, March, 1988.

[14] C. Sechen and K. W. Lee, ''An Improved Simulated Annealing Algorithm for Row-based Placement,'' *Proc. Int. Conf. Computer-Aided Design*, pp. 478-481, Nov. 1987.

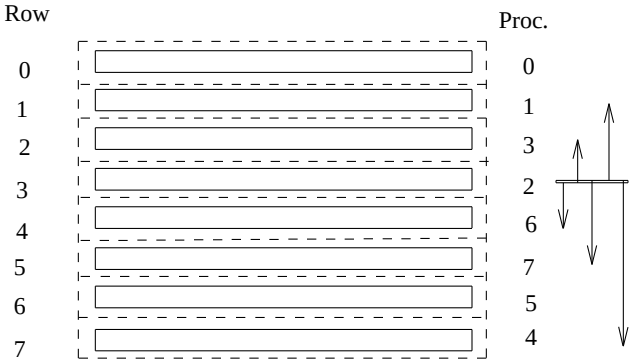

Fig. 1. Row partitioning of an 8-row circuit to 8 processor hypercube.

Table 1. K-Colorability of Sample Standard-Cell Circuits.

Number of Cells	Connectivity			Colors Needed	
	Min	Max	Avg	Random	Greedy
32	3	10	6	6	6
64	3	16	6	11	11
183	1	23	8	15	15
286	1	123	40	24	30
800	1	123	22	24	24

Circuit	No. of Proc.	Sequence Length
32	1	72
	2	60
	4	44
64	1	157
	2	115
	4	57
	8	33
183	1	70
	2	78
	4	60
	8	75
	16	40
286	1	329
	2	304
	4	275
	8	210
	16	105
469	1	na
	2	na
	4	287
	8	na
	16	145
800	1	na
	2	na
	4	316
	8	na
	16	174

Table 2. Average ADAPTIVE Move-Sequence Length

Table 3. Comparison of placement results and runtimes of algorithms FIXED-ROW, COLOR-ROW, ADAPTIVE-ROW on various test circuits and hypercube sizes.

Circuit Size	Number of Processors	FIXED-ROW		COLOR-ROW		ADAPTIVE-ROW	
		Wire-length	Time (minutes)	Wire-length	Time (minutes)	Wire-length	Time (minutes)
32	1	4666	8	6803	7	4930	7
	2	4761	9	6485	6	4855	6
	4	5276	7	6663	5	4995	3
64	1	14604	32	25098	38	14448	28
	2	14830	30	24385	31	14553	17
	4	14633	24	24497	22	14626	10
	8	14497	22	22320	20	14541	7
183	1	80588	406	102191	542	81422	434
	2	80747	451	109021	465	75324	433
	4	86932	447	114025	247	77968	125
	8	82751	123	119648	128	89537	118
	16	82594	118	125545	97	83095	58
286	1	126478	2940	na	na	144939	1350
	2	132140	2415	na	na	135517	1114
	4	131887	1135	na	na	172017	348
	8	153923	720	na	na	149587	390
	16	143310	482	na	na	142849	170
469	1	na	na	na	na	na	na
	2	na	na	na	na	na	na
	4	na	na	na	na	245808	4149
	8	na	na	na	na	na	na
	16	na	na	na	na	249162	1018
800	1	na	na	na	na	na	na
	2	na	na	na	na	na	na
	4	na	na	na	na	544721	6225
	8	na	na	na	na	na	na
	16	na	na	na	na	553875	1260