Rectangular Spatial Decomposition Methods for Parallel Simulated Annealing

Daniel R. Greening [*]
Frederica Darema [†]

Abstract

Research on VLSI placement has extended the standard sequential simulated annealing technique to two multiprocessing variants. In one technique, processors perform moves on disjoint partitions of locally-stored circuit grids. In the other, processors perform simultaneous moves on a shared grid. Our research explores new techniques in the first category—called spatial decomposition algorithms.

We describe the impact of cell mobility and cost-function errors in parallel simulated annealing. We show that changing the partition shape can affect these measures, and the quality of the final result. We also show a trade-off: execution speed vs. increased cell mobility and decreased cost-function errors.

We present four rectangular decomposition methods. Using two circuit examples, we compare their convergence properties to that of a "standard" random spatial decomposition technique. Runs were performed in a simulated RP3 environment.

One method we developed, "sharp random rectangles," converged better than the other techniques we studied. On one example, sharp random rectangles on 8 processors converged better than standard sequential algorithms. This promising technique allows us to increase the stream-length, and thereby reduce execution time.

The authors continue their research to better quantify cell mobility and cost-function errors, and to run the rectangular algorithms on other types of multiprocessors to obtain speed-up information.

[*]UCLA Computer Science Department
[†]IBM T.J. Watson Research Center

1 Introduction

Simulated annealing is a class of statistical hill-climbing algorithms for solving difficult combinatorial optimization problems, such as VLSI placement or the traveling salesman problem [7]. Simulated annealing gives near-optimal results for these problems, but requires a substantial investment of computational resources. Research papers discuss several techniques to reduce total execution time, including highly-tuned annealing schedules [11] [13] [16] and parallel processing [2] [4].

Most parallel simulated annealing methods use a method related to chaotic relaxation [3], in that they operate on partially erroneous information. Simulated annealing makes hill-climbing moves based on a Monte Carlo approach, therefore it can tolerate some error in cost-function calculations. Thus, annealing algorithms can evaluate the cost of a move using slightly incomplete or out-of-date information, but still converge to a reasonable solution.

Error tolerance provides a great advantage in multiprocessing: when several processors proceed independently on different parts of the problem, they need not synchronously update state information in other processors. A processor can save several updates, then send them in a block to the other processors. The processor will send less control information and compress multiple moves for a single cell into one move. Thus block transmissions reduce total communication traffic.

With a few limitations, updates can occur out-of-order, greatly reducing the number of synchronization operations.

Quantifying the error tolerance of simulated annealing is an open theoretical issue. Relying on empirical evidence, [6] claims that simulated annealing can tolerate cost function errors of 10% with limited effect on convergence. However, those results may strongly depend on the partitioning used in [6]—random cell allocation with unlimited Manhattan swap-distance. Grover states that the largest tolerable error is about half the current annealing temperature [5], but clearly this is dependent on the cost-function used. Rose claims error reduction is extremely important at high-temperatures [14], but less important at lower temperatures.

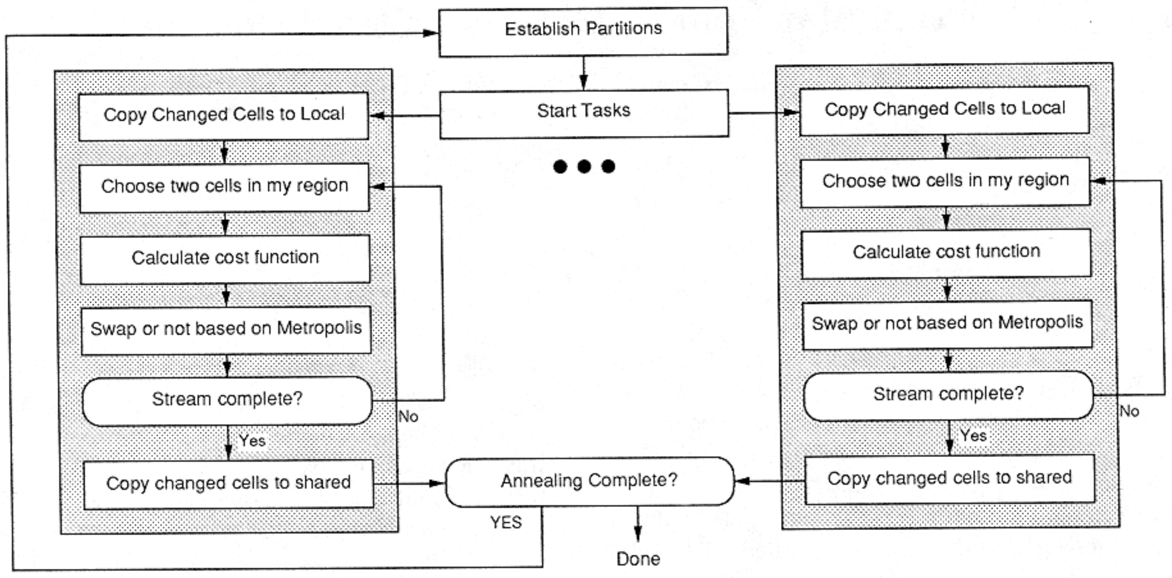

Establish Partitions
Start Tasks
Copy Changed Cells to Local
Choose two cells in my region
Calculate cost function
Swap or not based on Metropolis
Stream complete?
No
Yes
Copy changed cells to shared
Annealing Complete?
YES
Done

Figure 1. Parallel Execution Model

1.1 Shared Cell-Map Algorithms

Two methods of parallel simulated annealing dominate the literature: The "shared map" algorithms use a shared memory cell map [1] [4] [10]. Each process randomly selects two cells for an attempted move, synchronously locks the cells, calculates the cost-function assuming the cells were swapped, and based on the Metropolis algorithm [12] either swaps the cells or leaves them in place. At the end of the move, the algorithm unlocks the cells. Cost-function errors occur when two processors simultaneously manipulate cells on the same net.

Some algorithms [4] also lock attached nets before swapping—assuring accurate cost-function calculations at a greatly increased synchronization cost. Darema [4] shows that the increased accuracy of net-locking did not make a significant difference in the convergence quality on a simulated multiprocessor system.

1.2 Spatial Decomposition

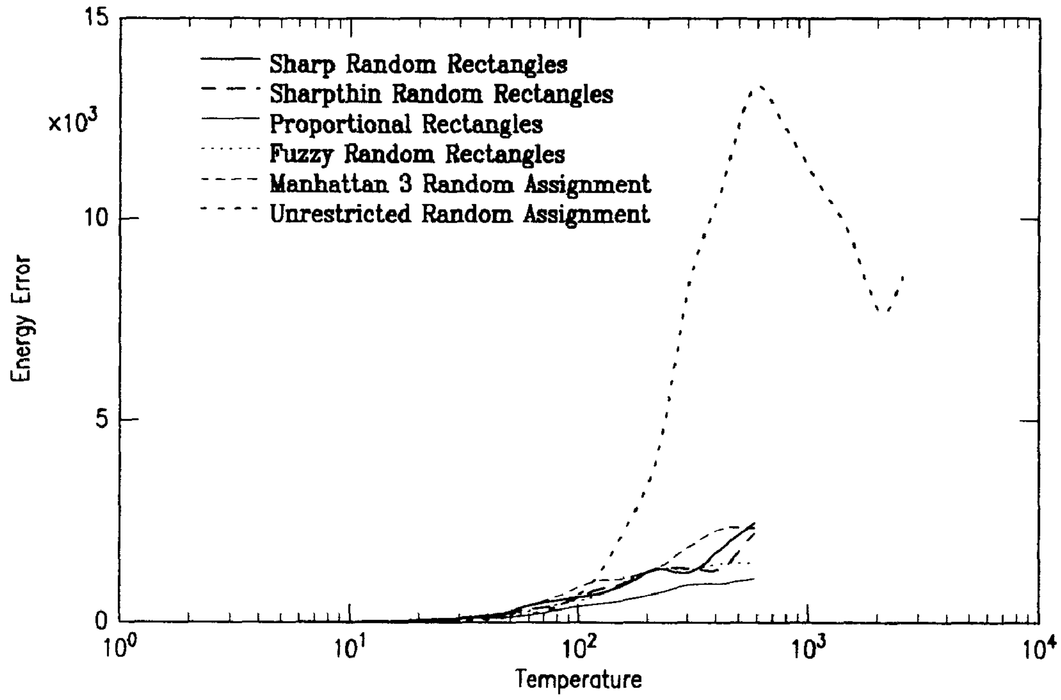
The "spatial decomposition" algorithms divide the a cell map into mutually disjoint partitions, either randomly [2] [6] or using a fixed pattern [8] [9], and assign each partition to a different processor. A processor randomly selects two cells within its partition, calculates the cost-function assuming the cells were swapped, and uses the Metropolis algorithm to decide whether to swap the cells. No locking is required in these algorithms, because the processors operate on disjoint sets of cells. A processor may complete several swaps, in a "swap stream," before sending the updated information to a common database.

Each processor acts independently on its partition, assuming that cells in other partitions are stationary. Cost-function errors result because cells in other partitions are *not* stationary—they are being modified by different processors. After a number of tries, the processes update a global map, the controlling task repartitions the grid, and the process repeats. The reader can see that using a larger swap stream (allowing more swaps between updates) will increase calculation errors. Note that any successful algorithm must change partition boundaries between streams, otherwise limitations on cell-mobility will preclude a good result.

1.3 Our Work

Our research explores four new "random rectangles" spatial decomposition schemes. Our approach divides a virtual circuit grid into rectangular regions—the algorithms differ in the restrictions placed on where the boundaries can fall. One method, "sharp random rectangles" converges better than the others we studied. For certain examples the resulting layout quality of sharp random rectangles was higher than standard sequential approaches!

We show that our results can be reasonably interpreted in terms of the increased cell-mobility and de-



P9: 4 *Streams/64 Tries: Error at Different Allocations*

Figure 2. Cost Function Errors vs. Decomposition Method

creased cost-function errors that occur with the rectangular partitioning approaches.

2 The Algorithm

Our spatial decomposition algorithm first assigns the cells in a VLSI circuit to random locations on a virtual grid.

Figure 1 shows the parallel execution model. It begins with a single processor decomposing the grid into mutually disjoint partitions—in our case these partitions are rectangular regions from the virtual grid. For comparison purposes, we can also generate random regions, where each cell is assigned to a random processor. These partitioning algorithms will be described in detail later.

All processes then proceed independently to complete a "stream" of trial moves on their respective regions. Each process copies the entire cell map to its local memory—we optimize this step by copying only *changed* cells to the local memory.

A process randomly chooses two cells within its region. If the two cells have Manhattan distance greater than 3, the process repeatedly retries, randomly selecting another two cells until it finds two at Manhattan distance less than 4.

It then calculates the cost function based on its local copy of the cell map. While cells within the partition accurately reflect their present position, cell positions outside the partition will become outdated when other processes move their cells around.

Using the estimated change in the cost-function for the circuit layout in the Metropolis algorithm, the process may swap the two selected cells. If the cells are swapped, the local cell map is changed, but copies of the cell map in other processors are *not* updated.

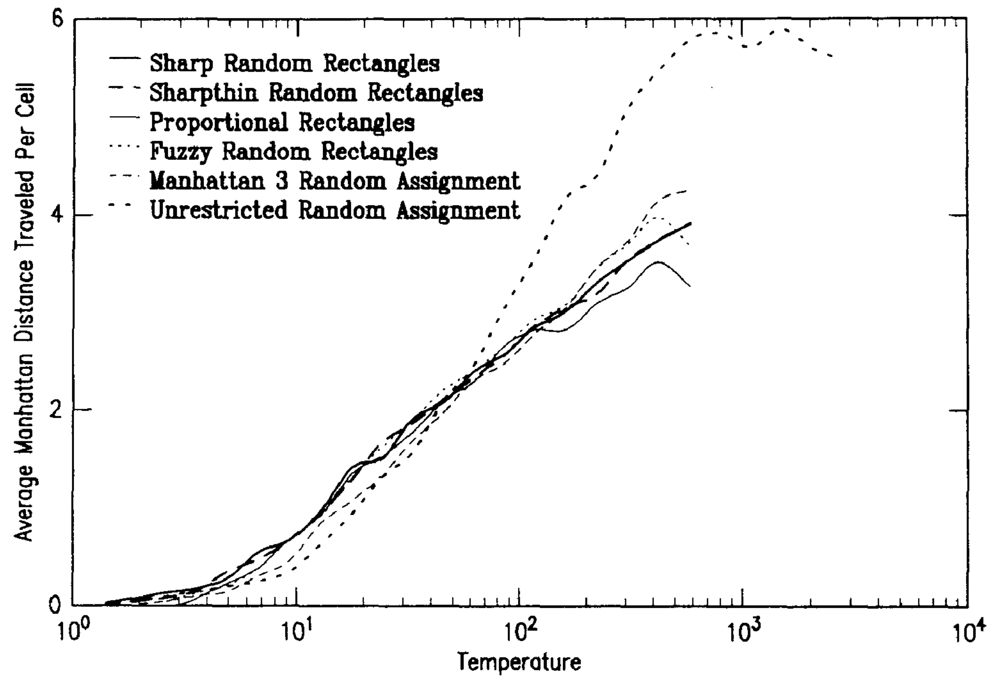
Finally, if the number of tries equals the "stream length," the process relinquishes control back to the supervisor process. Otherwise, it again randomly selects two cells, as above, and the sequence repeats.

When all processes have completed their streams, the supervisory process creates an accurate shared copy of the cell map, reflecting changes made in individual processors. It determines whether the annealing schedule is complete. If not, it changes the annealing constraints according to the schedule, re-establishes the partitions, and restarts the parallel processes.

2.1 Speed vs. Accuracy and Mobility

When a process moves chips within its region, the global states viewed by the other processes become inconsistent. This introduces a computation error which, uncontrolled, would destroy the quality of the resulting layout.

To minimize these errors, the stream length must be kept reasonably small. However, as the stream length decreases, the interprocess communication increases, reducing the benefits of multiprocessing. Any spatial decomposition algorithm must balance these conflicting



P9: 4 *Streams/64 Tries: Mobility at Different Allocations*

Figure 3. Cell Mobility vs. Decomposition Method

goals: higher accuracy improves the result, but requires more time.

Partition *shape* can influence errors. Figure 2 shows cost-function errors for different spatial decomposition methods, while annealing our *P9* example with 4 processors and a stream length of 64 tries.

Suppose cells are assigned to partitions at random, so that a cell's neighbors are not likely to be within the same partition. In a circuit where cells tend to have strong local connectivity (most circuits are like this), calculations errors are likely to be high. Swaps within these random partitions would cause dramatic changes in network length, and those networks would probably be connected to cells in other partitions. The large cost-function errors for *Random3* and unrestricted random assignment shown in Figure 2 typify this phenomenon.

Cell mobility also competes with execution time. Suppose, for example, that partition boundaries never change. Then all swaps will occur within a region, and the cells in the region will never leave. To allow cells to travel any location on the virtual grid, we must periodically redraw the boundaries. This repartitioning incurs execution time costs, because individual processes must copy global state information on the cells in their region before each stream. As we decrease stream length, we redraw partition boundaries more often—cell mobility increases, but the execution speed decreases. Again, we see a fundamental conflict: higher cell mobility improves the result, but requires more time.

Partition shape can also affect cell mobility. Figure 3 shows how average Manhattan distance traveled per cell varies with different spatial decomposition methods—again we show the *P9* example with 4 processors and 64 tries per stream.

Suppose that partitions in a spatial decomposition scheme always take on a wide, flat shape—as wide as the entire virtual grid. The non-random spatial decomposition scheme described in [9] follows this approach. Cells exhibit high mobility in the horizontal direction, but low mobility in the vertical direction. A cell can travel from grid left to grid right in one stream, but it may take several streams to travel from top to bottom.

The random rectangles approach uses partition shape to reduce errors and improve mobility over standard spatial decomposition methods. Recently obtained results indicate that we are on the right track.

2.2 Four Rectangular Approaches

To evaluate this class of parallel simulated annealing algorithms, we tried four different variations of random rectangular partitioning, as shown in Figure 4.

2.2.1 Proportional Rectangles

This is the simplest of the four techniques. Here we divide the layout into equally sized rectangles, roughly proportional to the overall layout size. We then randomly select the origin for this grid. Note that the shapes of the rectangles in this group never change.

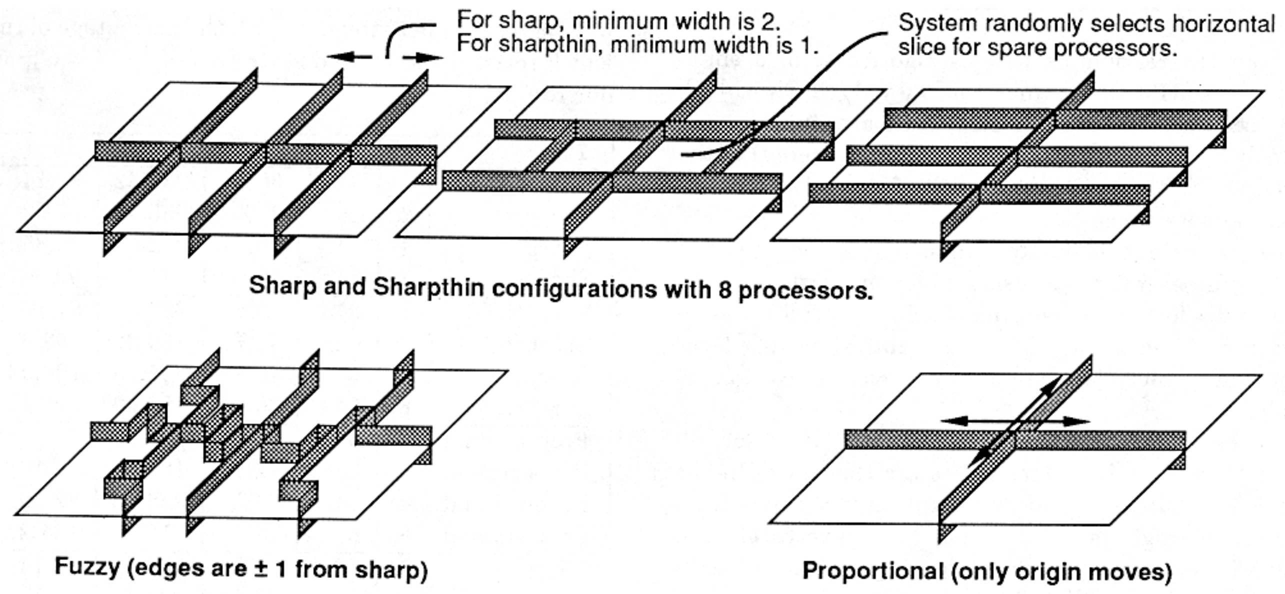

For sharp, minimum width is 2.
For sharpthin, minimum width is 1.
System randomly selects horizontal slice for spare processors.
Sharp and Sharpthin configurations with 8 processors.
Fuzzy (edges are ± 1 from sharp)
Proportional (only origin moves)

Figure 4. Rectangular Decomposition Schemes

2.2.2 Sharp Random Rectangles

This algorithm divides grid g into rectangles with a minimum width and height of 2. First, we choose the number of rectangles in the X direction. Let w_g be the grid width, and h_g be the grid height. Let p be the number of processors. We randomly select the width of each rectangle r from the set $w_r \in \{\lceil 2p/h_g \rceil, \ldots, \min(p, \lfloor w_g/2 \rfloor)\}$. Grid g holds $w_g^r = \lfloor w_g/w_r \rfloor$ rectangles in the x direction. Rectangle height h_r is then $h_r = \lfloor p/w_g^r \rfloor$. Grid g holds $h_g^r = \lfloor h_g/h_r \rfloor$ rectangles in the y direction.

There are two problems. First, it is likely that $w_r \cdot w_g^r \neq w_g$ or $h_r \cdot h_g^r \neq h_g$. To resolve this we widen a randomly chosen column of rectangles, r_{x*} to $w_{r_{x*}} = w_g - w_r(w_g^r - 1)$. We heighten a randomly chosen row, r_{*y} to $h_{r_{*y}} = h_g - h_r(h_g^r - 1)$.

Second, $p \leq h_g^r \cdot w_g^r$, so with no modifications we may not use all processors. To increase the number of utilized processors to p, we increase the rows in column r_{x*} to $h_g^{r_{x*}} = p - w_g^r(h_g^r - 1)$. $h_r \cdot h_g^{r_{x*}}$ may not equal h_g. In that case, we heighten one row in this single column r_{x*}, as discussed above.

2.2.3 Sharpthin Random Rectangles

This spatial decomposition algorithm applies the sharp random rectangles allocation method, except that we randomly select $w_r \in \{\lceil p/h_g \rceil, \ldots, \min(p, w_g)\}$. In sharp random rectangles $w_r \geq 2$ and $h_r \geq 2$. Sharpthin allows $w_r \geq 1$ and $h_r \geq 1$.

It was thought that rectangles with width or height 1 would significantly increase the calculation error and reduce the quality of the resulting layout. Empirical results appear to confirm these initial thoughts.

2.2.4 Fuzzy Random Rectangles

Fuzzy random rectangles also applies the sharp random rectangles method discussed above, except that cells lying on rectangle borders are assigned randomly to any rectangle adjacent to the cell. This creates rectangles with "fuzzy" borders.

2.2.5 Random3

This is a variation of Jayaraman's technique [6]. Before a swap stream, each chip is randomly assigned to a processor. The chips are divided roughly equally among the processors and no chip is allocated to more than one processor each time. When the swap stream completes, the chips are again redistributed to the processors.

The difference between our implementation and Jayaraman's is that we disallow exchanges of pairs of chips whose Manhattan distance exceeds 3—serial algorithms [7] typically use this restriction to accelerate the annealing process.

We tried Jayaraman's original technique, hoping to provide some data for comparison, but when the number of processors exceeded 4, annealing runs using unlimited Manhattan distance would not converge in a reasonable time.

3 Empirical Results

We ran the random rectangles algorithms in a simulated IBM RP3 environment, using VM/EPEX C [15]. We used the classic VLSI placement cost-function [7], namely $E = L + C^2$, where E is the cost (energy) of the system, L is the total wire length of the networks, and C is the wire congestion.

To evaluate a simulated annealing algorithm, one must run several trials using different random seeds. The mean final cost over many trials provides a good measure of the quality of a particular annealing technique. For each data point in our results, we ran 50 trials.

At the boundary condition of a single processor, all spatial decomposition techniques are the same. In the tables which follow, we simply duplicated the data from one set of single processor trials in all five categories. Had we made separate runs, the single processor data points would differ slightly due to the random nature of the runs.

3.1 The P9 Circuit

Two circuit systems were tried. The first, *P9*, is a uniform 9×9 grid with immediate neighbor cells connected by two-cell nets. There are 81 cells, and 144 nets. It has a known ground state, under our cost-function, of $E = 144$, $L = 144$ and $C = 0$. The ground state appears in Figure 5.

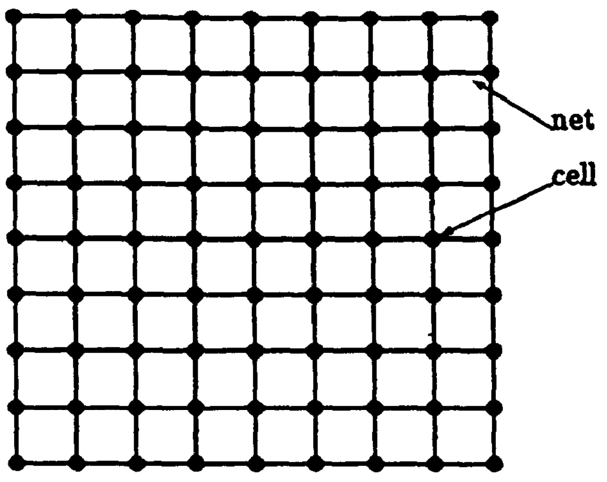


Figure 5. P9 Ground State

We evaluated each of the four rectangular spatial decomposition techniques with 1, 2, 4, and 8 processors. A fifth set of runs using *Random3* provides a baseline from which we can measure improvement. Table 1 shows the *type* of partitioning, the number of processors (P), the number of tries per stream (T/S), the percentage of runs which reached the ground state cost of 144 (*% G*), and our results.

Type	*P*	*T/S*	*% G*	*Ave E*	*Std E*
Sharp	1	64	38%	233.32	97.14
Sharp	2	64	27%	255.32	97.62
Sharp	4	64	26%	252.64	90.85
Sharp	8	64	100%	144.0	0.00
Sharpthin	1	64	38%	233.32	97.14
Sharpthin	2	64	38%	236.82	102.61
Sharpthin	4	64	40%	245.68	106.93
Sharpthin	8	64	20%	287.92	97.35
Proportional	1	64	38%	233.32	97.14
Proportional	2	64	72%	180.68	60.19
Proportional	4	64	24%	258.66	92.50
Proportional	8	64	18%	316.60	134.22
Fuzzy	1	64	38%	233.32	97.14
Fuzzy	2	64	36%	244.72	97.70
Fuzzy	4	64	38%	254.92	106.96
Fuzzy	8	64	15%	262.28	91.05
Random3	1	64	38%	233.32	97.14
Random3	2	64	38%	231.36	86.69
Random3	4	64	15%	298.01	98.42
Random3	8	64	0%	363.44	74.96

Table 1: P9: Convergence Statistics

Note that as the number of processors increases, most runs show an increasing average cost-function value. The number of runs which result in the ground state of $E = 144$ typically decreases. *Random3* shows the worst degradation as the number of processors increases. Since *Random3* creates the highest calculation errors, we expected that result.

Finally, one example, *Sharp*, shows the best result with 8 processors—all runs reached the ground state. We would assume this to be anomalistic, however with an entirely different circuit, described below, *Sharp* performed better than the other techniques.

3.2 The ZA Circuit

The other problem we tried, called *ZA*, is a real printed circuit layout problem. All cells in this problem are uniformly square. Most of the 359 cells in *ZA* are unconnected. There are 50 networks, each with an average of 4.04 attached cells.

Overall, there are three blocks of interconnected cells. Two blocks include a few simple 2-cell networks—cell swaps in those blocks have minor effects on the cost-function. The third block includes many multiple-cell networks, typically 9 to 11 cells per net. Each cell has numerous connections to other cells. One swap in this group typically causes dramatic changes in the cost function.

There is no known ground state to *ZA*. We obtained data for the four rectangular spatial decomposition methods, and for the *Random3* method. Each of these partitioning methods were tried with 1, 2, 4, 8, 16, and 32 processors. As with *P9*, data for the single processor case is duplicated in all categories. Table 2 shows our results.

Type	*P*	*T/S*	*Ave E*	*Std E*
Sharp	1	64	139.08	1.83
Sharp	2	64	138.74	1.99
Sharp	4	64	139.58	2.22
Sharp	8	64	140.40	2.51
Sharp	16	64	141.02	2.71
Sharp	32	64	142.02	2.53
Sharpthin	1	64	139.08	1.83
Sharpthin	2	64	138.69	1.93
Sharpthin	4	64	139.26	2.49
Sharpthin	8	64	140.16	2.59
Sharpthin	16	64	141.56	2.26
Sharpthin	32	64	142.50	3.05
Proportional	1	64	139.08	1.83
Proportional	2	64	139.52	2.31
Proportional	4	64	139.46	2.31
Proportional	8	64	140.00	2.44
Proportional	16	64	140.78	2.67
Proportional	32	64	142.54	2.99
Fuzzy	1	64	139.08	1.83
Fuzzy	2	64	139.30	2.04
Fuzzy	4	64	139.28	2.06
Fuzzy	8	64	139.70	2.24
Fuzzy	16	64	140.34	2.63
Fuzzy	32	64	144.48	3.82
Random3	1	64	139.08	1.83
Random3	2	64	139.46	2.39
Random3	4	64	139.34	2.15
Random3	8	64	142.54	2.50
Random3	16	64	150.12	3.22
Random3	32	64	166.30	5.93

Table 2: ZA: Convergence Statistics

As with the *P9* example, in Table 2 we see the best results in the *Sharp* partitioning method. *Fuzzy* looks very promising until we reach 32 processors. *Random3*, as in the *P9* case, produces terrible results when the number of processors becomes high.

4 Conclusion

We showed that increased mobility and decreased cost-function errors are important goals in spatial decomposition methods for simulated annealing. We qualitatively described a trade-off where increased parallelism can decrease cell mobility or increase cost-function errors, resulting in a less desirable annealing result. We showed that partition shape can affect both cell mobility and cost-function errors.

We presented four new rectangular spatial decomposition techniques for parallel simulated annealing. Our rectangular techniques use partition shape to help increase cell mobility and decrease cost-function errors. This allows us to increase the stream length, providing greater parallelism and decreasing execution time on multiprocessors.

One rectangular technique, *Sharp Random Rectangles*, appears to perform better than the others. The authors are actively pursuing research in this area, and expect to develop more quantitative measures for cell mobility and cost-function errors. We will be running the algorithms on mesh-connected transputers, conventional LAN-connected workstations, and the shared-memory RP3 multiprocessor to obtain accurate speed-up information. We are also investigating the theoretical implications of error-tolerant parallel simulated annealing on convergence and execution time.

References

[1] Emile H.L. Aarts, Frans M.J. de Bont, Erik H.A. Habers, and Peter J.M. van Laarhoven. Parallel implementations of the statistical cooling algorithm. *Integration, the VLSI Journal*, 4:209–238, 1986.

[2] A. Casotto, F. Romeo, and A. Sangiovanni-Vincentelli. A parallel simulated annealing algorithm for the placement of macro-cells. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD-86)*, page 60, IEEE Computer Society Press, 1986.

[3] D. Chazan and W. Miranker. Chaotic relaxation. *Linear Algebra and Its Applications*, 2:199–222, 1969.

[4] Frederica Darema, Scott Kirkpatrick, and Alan V. Norton. Parallel algorithms for chip placement by simulated annealing. *IBM Journal of Research and Development*, 31(3):391–402, May 1987.

[5] L.K. Grover. A new simulated annealing algorithm for standard cell placement. In *Proceedings of the International Conference on Computer-Aided Design*, pages 378–370, IEEE Computer Society Press, November 1986.

[6] Rajeev Jayaraman and Frederica Darema. Error tolerance of parallel simulated annealing techniques. In *Proceedings of the International Confer-*

ence on Computer-Aided Design, IEEE Computer Society Press, 1988.

[7] S. Kirkpatrick, Jr. C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[8] Ralph Kling. Novel approaches to cell placement. September 1988. Presentation of work at IBM T.J. Watson Research Center.

[9] Ralph-Michael Kling and Prithviraj Banerjee. Concurrent esp: a placement algorithm for execution on distributed processors. In *Proceedings of the 1987 International Conference on Computer Design*, pages 354–357, October 1987.

[10] Saul A. Kravitz and Rob A. Rutenbar. Placement by simulated annealing on a multiprocessor. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, CAD-6(4):534–549, July 1987.

[11] Jimmy Lam, Jean-Marc Delosme, and Carl Sechen. A new simulated annealing schedule for row-based placement. In *MCNC International Workshop on Placement and Routing*, 1988.

[12] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, and A.H. Teller. Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1091, 1953.

[13] Debasis Mitra, Fabio Romeo, and Alberto Sangiovanni-Vincentelli. Convergence and finite-time behavior of simulated annealing. In *Proceedings of 24th Conference on Decision and Control*, pages 761–767, December 1985.

[14] J.S. Rose, W.M. Snelgrove, and Z.G. Vranesic. *Parallel Standard Cell Placement Algorithms with Quality Equivalent to Simulated Annealing.* Technical Report, Stanford University, Stanford, CA, 1987.

[15] Chang Whei-Ling and Alan Norton. *VM/EPEX C Preprocessor User's Manual Version 1.0.* Technical Report RC 12246, IBM T.J.Watson Research Center, Yorktown Heights, NY, October 1986.

[16] Steve R. White. Concepts of scale in simulated annealing. In *Proceedings of the International Conference on Computer Design*, page 646, IEEE Computer Society Press, 1984.