Structural Design Using Cellular Automata

Douglas J. Slotta
Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061-0106

Brian Tatting
ADOPTECH, Inc.
Virginia Tech Corporate Research Center
Blacksburg, VA 24060

Layne T. Watson
Departments of Computer Science and Mathematics
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061-0106

Zafer Gürdal
Departments of Aerospace and Ocean Engineering, and Engineering Science and Mechanics
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061-0203

Key words: cellular automata, massively parallel computation, structural analysis.

Abstract

Traditional parallel methods for structural design do not scale well. This paper discusses the application of massively scalable cellular automata (CA) techniques to structural design. There are two sets of CA rules, one used to propagate stresses and strains, and one to perform design analysis. These rules can be applied serially, periodically, or concurrently, and Jacobi or Gauss-Seidel style updating can be done. These options are compared with respect to convergence, speed, and stability.

1. Introduction

The traditional method of doing structural analysis and design uses finite element based numerical analysis programs. While this approach works well for many problems, it does not parallelize efficiently on massively parallel processors (MPPs), thus limiting the size and complexity of the designs that can be analyzed and optimized. A new approach is needed that works well on MPPs. This method need not be faster than those currently used for serial machines on problems that do not exhaust the machines' resources, rather it needs to allow each processor of a MPP enough useful work such that large problems beyond the resources of serial or moderately parallel machines can be solved in acceptable times.

Cellular automata (CA) were used at least as early as 1946 by Weiner and Rosenblunth (1946) to describe the operation of heart muscle, even though their use was not computationally feasible at the time. CA tiles a problem domain into *cells* of equal size. Each cell has the same set of simple rules that dictate how it behaves and interacts with its neighboring cells. The principle is that an overall global behavior can be computed by a group of cells that only know local conditions (Wolfram, 1994). If each cell only needs to know local conditions, then this minimizes the communication requirements and therefore the problem scales well on a MPP. A CA is the archetypical algorithm for the SIMD parallel architecture (Toffoli and Margolus, 1991).

A cellular automaton is a discrete dynamical system (Wolfram, 1994). It is discrete in the sense that space and time are discrete. Each cell is a fixed point in a regular lattice. The state of each cell is updated at discrete time steps, based upon conditions in previous time steps. All of the cells are updated every time step, thus the state of the entire lattice is updated every time step.

In general, CA are used to simulate the dynamic behavior of physical systems, and have been used successfully to represent a variety of phenomena such as diffusion of gaseous systems, solidification and crystal growth in solids, and hydrodynamic flow and turbulence (Toffoli and Margolus, 1991). CA has also recently been used in conjunction with genetic algorithms to derive the rules required at each cell to perform structural analysis (Hajela and Kim, 2000). CA rules have recently been devised for the simultaneous analysis and design of simple two-dimensional structures (Gürdal and Tatting 2000; Tatting and Gürdal, 2000); that work is the basis for this paper. In the case of structural design, the intention is to describe a static equilibrium of a structure under a system of forces acting on it. In this sense, time is not being simulated, rather each step of the automaton is used to propagate (local) stresses and strains through a structure to allow it to reach equilibrium state while simultaneously determining the shape and/or dimensions of the cells associated with this equilibrium state. This is continued until the entire process converges (ideally) to a global state where there is no significant change in the structure for every subsequent iteration, corresponding to a static equilibrium state. Note that analysis and design are done simultaneously and locally by each cell.

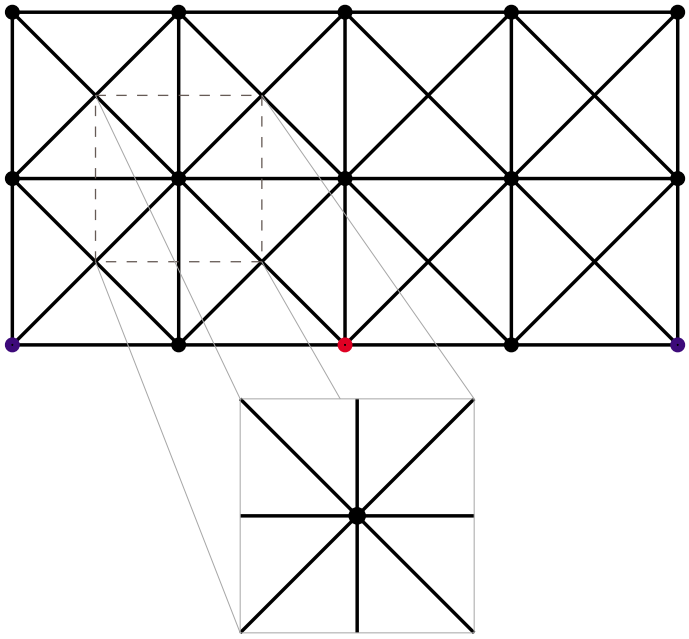

FIGURE 1. *Example CA domain with a single cell denoted by the dashed line.*

This paper will begin by describing the CA used to do structural design on ground trusses in Section 2 and give an example. Section 3 will discuss the merits of two different iteration methods. Section 4 will explain how the CA is parallelized, and finally Section 5 will discuss some preliminary results.

2. Method Description

The basic elements of the structural design CA consist of the division of the problem into cells and the three types of rules that can operate on those cells. Each of the rules operates on the cells using the information in a Moore neighborhood, which consists of the surrounding nine cells. The first set of rules are used to do analysis only, determining the stresses and strains in each cell. The second set of rules does the design work, changing the areas of the connecting beams to withstand the stresses. The final set of rules performs simultaneous analysis and design.

2.1 Domain Definition

Each cell of this CA is an eight-beam truss where each beam starts at the center of the cell and connects to its opposite member in an adjacent cell as illustrated in Figure 1.

This type of structure is known as a *ground truss*. Those cells which fall on the border of the rectangular domain are not partial cells requiring special rules, but are complete cells with the area of the beams that fall outside the computational domain set to zero. In addition, they are connected to an invisible set of surrounding cells that are turned *off* and that also have all of their beam areas set to zero. Cells that are turned off are not part of the computation, being used only to make the rules for the border and non-border cells consistent, since the stress analysis rules require the displacements of all eight surrounding cells.

The actual border of the computational domain of the CA need not be rectangular. Any shape can be defined for the truss by turning off any cells that are not within the computational

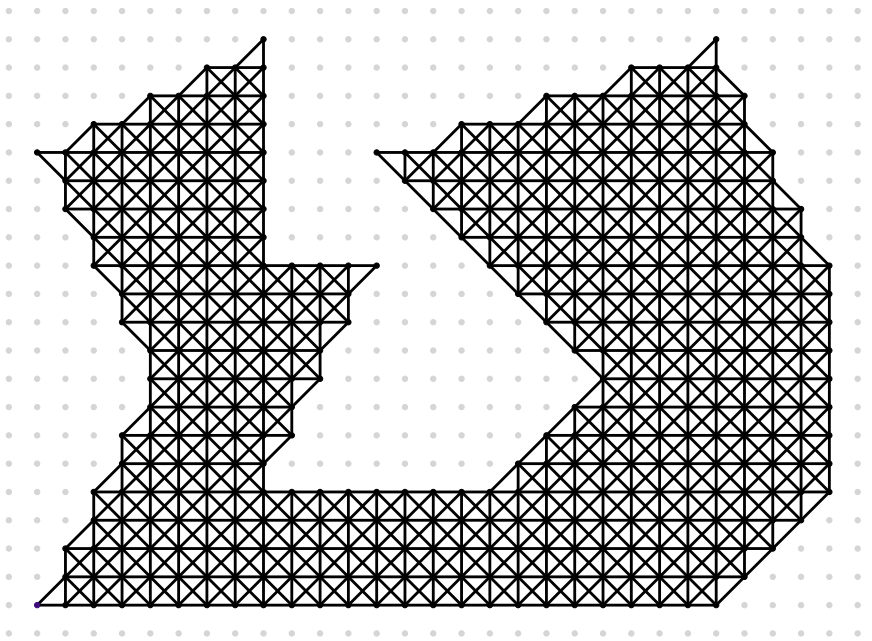

FIGURE 2. *Example CA domain with a non-rectangular computational domain.*

domain, as illustrated in Figure 1. A simple method to define a shape for the truss is to define an enclosing polygon, and then turn off every cell that does not fall within the polygon. The "edge crossings" algorithm (O'Rourke, 1994) to determine those points within the polygon can be used; it is simple and parallelizes well. A more sophisticated method could be used to allow for holes, circular regions, or other, nonstraight boundaries.

As seen in Figure 2, only those polygons that are composed of lines with slopes 0, 1, -1, or ∞ will be represented exactly. This is the same aliasing problem that bitmaps face. A better resolution can be obtained by decreasing the cell size in the domain, thereby increasing the number of cells that form the shape. This is the same as smoothing the outline of a bitmap by increasing the number of pixels that form the bitmap. The amount by which the original cells have been subdivided to increase the resolution is known as the cell density factor (CDF).

2.2 CA Rules

There are two sets of rules used to compute an optimal solution to a given structural problem. *Optimal solution* in this sense means a set of truss beams with the minimum size required to withstand the applied forces.

2.2.1 Displacements

The first set of rules is (normally) executed at every iteration to determine the strains in each cell. The cell attempts to reach equilibrium with the surrounding cells by displacing itself to minimize the potential energy.

Within a cell, each truss member (indexed relative to the cell by $k = 1, \ldots, 8$) has an elastic modulus E, length L_k, a cross-sectional area A_k, and an orientation angle θ_k from the cell center. Denote the displacement of the kth truss member's near end from the original cell center by (u, v), and the displacement of the far end from the neighboring cell's center by (u_k, v_k). These

neighboring displacements (u_k, v_k) are taken as fixed when the CA calculates the displacements (u, v) for each cell. The extension Δ_k, strain ε_k, and force F_k within each member are calculated from these properties and displacements by

$$\Delta_k = (u_k - u)\cos\theta_k + (v_k - v)\sin\theta_k, \tag{1}$$

$$\varepsilon_k = \Delta_k / L_k, \tag{2}$$

$$F_k = E A_k \varepsilon_k. \tag{3}$$

Taking into account the applied external force (F_x, F_y), the total (internal strain plus external) potential energy V for a cell is given by

$$V = \sum_{k=1}^{8} \frac{E A_k L_k \varepsilon_k^2}{2} - F_x u - F_y v. \tag{4}$$

Setting the partial derivatives of the potential energy with respect to the cell displacements to zero gives the equilibrium equations

$$\frac{\partial V}{\partial u} = 0, \qquad \frac{\partial V}{\partial v} = 0. \tag{5}$$

In general this is a system of two equations with two unknowns. If there is an (externally) applied displacement along a single axis, then (5) reduces to a single equation with one unknown, and if there are (externally) applied displacements along both axes, then there is nothing to solve. The forces acting upon the cell may be computed for reference, but this is not needed for the overall computation.

2.2.2 Beam Sizing

Designing the structure requires resizing the beams in the cells. If displacements have already been calculated, as in Section 2.2.1 for example, then some scheme for changing the cross sectional areas A_k is required. In terms of allowable stress σ_{allow}, which is chosen by the user as the maximum stress that any given beam should endure, one scheme for computing a new cross sectional area A_k^{new}, based upon the previous cross sectional area A_k^{old}, is

$$A_k^{\text{new}} = \frac{E|\varepsilon_k|}{\sigma_{\text{allow}}} A_k^{\text{old}}. \tag{6}$$

If the displacement calculation and sizing are done sequentially, the sizing period (how often sizing is done) depends on many factors: the number of cells in the domain, the locations and relative placements of the applied forces and displacements, the iteration method (Jacobi vs. Gauss-Seidel), and for Gauss-Seidel implemented in parallel, the number of processors used.

The last two items will be discussed in Sections 3 and 5.

3. Iteration Methods

Each cell depends on the displacements of the surrounding cells to calculate its own displacement, thereby propagating the stresses and strains. This is repeated until the structure no longer changes appreciatively, at which point it is said to have *converged*. The convergence criterion for the displacement of a cell is defined by the condition that the change in displacement is a small fraction or percentage (usually 10^{-6}) of the maximum displacement within the structure. The sizing

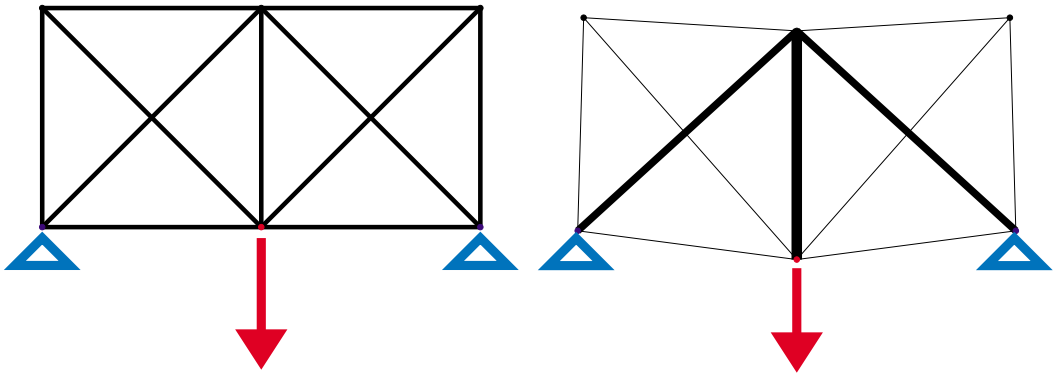

FIGURE 3. *Simple bridge truss, before and after running CA.*

convergence criterion is analogous. For an entire structure to be considered converged with respect to displacement or sizing, every cell within that structure must meet the convergence criterion for that update rule.

One method of implementing a CA is to keep two copies of the array of cells, one to represent time t, and the other to represent time $t+1$. The values for the cells at $t+1$ are calculated from the cells at t. At the end of this iteration, the labels of the arrays are swapped, and the process is repeated. This is a Jacobi iteration, where all of the new values are calculated from the old values.

For any process that converges, using a Jacobi iteration method can be inefficient (Bertsekas and Tsitsiklis, 1989). By using a Gauss-Seidel iteration method, where new values are calculated using updated values, the process should converge using fewer iterations. This means that only one copy of the array is kept. When a new displacement is calculated for one cell, then the next adjacent cell will use that updated value when calculating its own displacement. Note that this does not apply to the sizing rules since, as defined, their application is independent of the information in the surrounding cells.

3.1 Example

Consider the problem of a simple bridge truss. The first image in Figure 3 shows a CA with six cells. The bottom two corner cells have an applied displacement of (0,0) so they are fixed in place. The bottom middle cell has an applied force of $100kN$ downward. The width of the bridge is 50 meters and the height is 25 meters. The bars are composed of medium steel ($E = 200GPa$ and $\sigma_{\text{allow}} = 250MPa$). Each beam has an initial area of $0.0175m^2$.

Running the CA on the bridge problem using the Gauss-Seidel iteration method for displacements and applying the sizing rules every sixth iteration until it converges at iteration 253, the result shown in the second image of Figure 3 is obtained. Since the bridge is $50m$ across and the steel beams are no more than a few cm thick, the areas in this view are exaggerated by a factor of 3000 to show the differences in the beam sizes.

The bridge in Figure 3 is only composed of eleven trusses, and the solution could easily have been computed by hand. But if each beam is required to be less than $25m$ long, the complexity of the problem rises. Figure 4 shows a problem of the exact same dimensions, where each cell is 40 times smaller than previously. Each horizontal and vertical beam is $0.625m$ long, rather than $25m$.

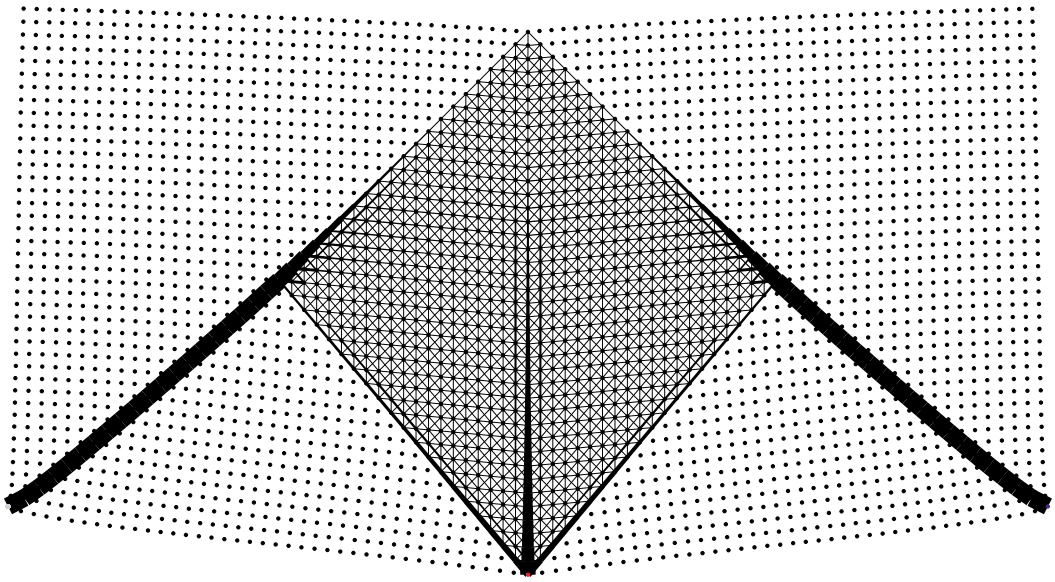

FIGURE 4. *Bridge truss with same dimensions, smaller cells, converged.*

3.2 Convergence Analysis

To analyze the efficacy of the iteration method used, it is useful to transform the CA into an equivalent system of linear equations. Recall from Section 2.2.1 that each cell is computing its position (u, v) based upon the position of the surrounding cells. If each cell were assigned unique variables for its position, such that cell 1 has u_1 and v_1, cell 2 has u_2 and v_2, and so forth, then the equations for each cell can be expressed in terms of the variables for the surrounding cells. For a CA structure composed of 6 cells, this will form a linear system of 12 equations and 12 unknowns.

This standard system of linear equations,

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{7}$$

can be solved by the Jacobi and Gauss-Seidel fixed-point iteration methods or block versions thereof, which are the exact mathematical formulations of the local cell calculations. For the Jacobi, $\mathbf{A}$ is split into its strictly 2×2 block lower triangular ($\mathbf{L}$), 2×2 block diagonal ($\mathbf{D}$), and strictly 2×2 block upper triangular ($\mathbf{U}$) parts,

$$\mathbf{A} = \mathbf{L} + \mathbf{D} + \mathbf{U}. \tag{8}$$

The system is then rewritten as a fixed point iteration where the next iterate $\mathbf{x}^{(n+1)}$ is computed from the previous iterate $\mathbf{x}^{(n)}$ via

$$\mathbf{x}^{(n+1)} = \mathbf{B}\mathbf{x}^{(n)} + \mathbf{C}, \tag{9}$$

where

$$\mathbf{B} = -\mathbf{D}^{-1}(\mathbf{L} + \mathbf{U}), \quad \mathbf{C} = \mathbf{D}^{-1}\mathbf{b}. \tag{10}$$

Note that $\mathbf{A}\mathbf{x} = \mathbf{b}$ if and only if $\mathbf{x} = \mathbf{B}\mathbf{x} + \mathbf{C}$, assuming $\mathbf{D}^{-1}$ exists. For Gauss-Seidel the iteration $\mathbf{x}^{(n+1)} = \mathbf{B}\mathbf{x}^{(n)} + \mathbf{C}$ has

$$\mathbf{B} = -(\mathbf{D} + \mathbf{L})^{-1}\mathbf{U}, \qquad \mathbf{C} = (\mathbf{D} + \mathbf{L})^{-1}\mathbf{b}, \tag{11}$$

CDF	n	Jacobi	Gauss-Seidel
1	8	0.794104	0.611503
2	26	0.949748	0.8982
3	52	0.979368	0.959006
4	86	0.989496	0.978893
5	128	0.993801	0.987605
6	128	0.995959	0.991925
7	236	0.997181	0.994365
8	302	0.997933	0.995865
9	376	0.998426	0.996853
10	458	0.998765	0.997532

TABLE 1. *Spectral radius of the bridge truss for various CDFs.*

assuming $(\mathbf{D}+\mathbf{L})^{-1}$ exists.

The fixed point iteration (9) converges for any starting point $\mathbf{x}^{(0)}$ if and only if all of the eigenvalues of $\mathbf{B}$ are less than one in absolute value (Issacson and Keller, 1966). The maximum absolute value of the eigenvalues for a matrix is called the *spectral radius*. The spectral radius for the bridge structure at various CDFs is shown in Table 1.

This table shows that the Jacobi or Gauss-Seidel CA iteration for analysis does converge, but extremely slowly. For larger CDFs, the improvement of Gauss-Seidel over Jacobi is marginal. Even with massive parallelism, any competitive advantage of CA (over solving the linear system with standard iterative numerical methods) must come by combining analysis with sizing.

Aitken's δ^2 method (Issacson and Keller, 1966) was also explored. This method uses the (scalar) values of three successive iterations to extrapolate a value (hopefully) closer to the fixed-point value. This method requires that (for scalar values $\mathbf{x}_n$)

$$\frac{\Delta\mathbf{x}_{n+1}}{\Delta\mathbf{x}_n} \approx \frac{\Delta\mathbf{x}_n}{\Delta\mathbf{x}_{n-1}} \approx \frac{\Delta\mathbf{x}_{n-1}}{\Delta\mathbf{x}_{n-2}} \approx A, \tag{12}$$

where $|A| < 1$. However, this requirement was not met by the components $\mathbf{x}_i^{(n)}$ of the vector iteration (9), and therefore Aitken's δ^2 acceleration method was not applicable.

4. Parallel Implementation

The code for this CA was implemented in Fortran 90 using the Message Passing Interface (MPI) library as its parallel communication mechanism. It has been tested on both an Origin 2000 with 64 processors and a Beowulf cluster with 32 processors.

A parallel decomposition was performed by dividing the computational domain into vertical strips and assigning each strip of contiguous cells to a single processor. Each strip has an additional column of border cells on either side that are turned off. These border cells represent the connected cells located on the adjacent processors. At every iteration, a processor computes the updated values for its cells, and then exchanges its left and right columns with its neighbors. These updated values are stored in the border cells and used for the next iteration. Therefore the natural communication topology is a ring topology, which easily maps into most other communication topologies.

Stripped partitioning works well for a rectangular shaped domain because it provides a good balance of computation to communication. It does have some limitations, for example, given a

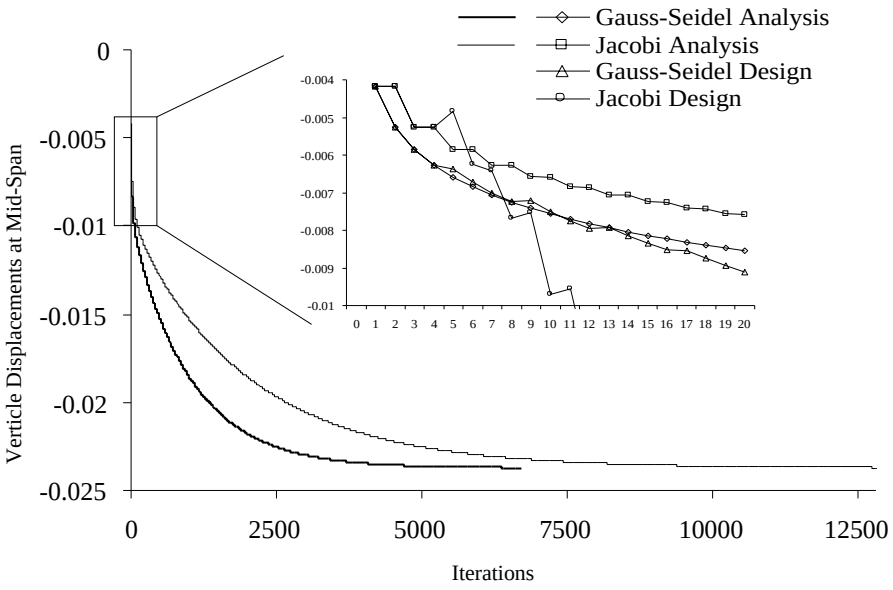

FIGURE 5. *Comparison between Jacobi and Gauss-Seidel iteration methods on the bridge structure with a CDF of 16.*

domain size of N rows $\times$ M cols, there can be at most M processors; if $M < N$ then the lattice can be trivially rotated. For those problems with an irregular shape, there will not be the same balance of computation to communication at every processor. For these cases, a more efficient partitioning method (e.g., graph-based partitioning) should be used.

When implementing the Gauss-Seidel iteration method in parallel, no attempt is made to keep the order in which the cells are updated the same as in the non-parallel implementation. Instead, each processor iterates over its collection of cells as if it were the sole processor operating on the domain, where the domain consists of its assigned cells, plus a group of surrounding "dead" cells that are not computed. The "dead" cells are used to contain the updated values from the adjacent processors that are sent at the end of every iteration.

Since the Gauss-Seidel iteration is contained solely within each processor, the rate of convergence differs depending upon the number of processors used. This has an effect upon the stability of the calculation as will be seen in Section 5.1. On the other hand, the programming task is much easier since, except for the initial setup and the communication at the end of every iteration, the program is exactly the same as the Gauss-Seidel iteration for a single processor.

5. Results

5.1 Jacobi vs. Gauss-Seidel

When comparing the performance of Jacobi and Gauss-Seidel iteration methods it is useful to look at the number of iterations it takes the displacements (without sizing) to converge using a single processor. Figure 5 compares the number of iterations to the vertical displacement of the mid-span of the bridge truss with a CDF (cell density factor) of 16. The mid-span will have the largest displacement for any correct solution to the problem because it has the only externally applied force. Figure 5 shows that it takes 12,808 iterations to converge with the Jacobi method and only 6,723 iterations using the Gauss-Seidel.

The speed of convergence for a structural analysis CA is affected by more than just the iteration method if sizing rules are used as well. Figure 5 also has an expanded view of the first 20 iterations that includes sizing rules applied every four iterations. Note that when using the Jacobi method with sizing, the maximum displacement diverges away quickly from the maximum displacement using analysis only. In fact, for this sizing period, the CA is non-convergent. For the Gauss-Seidel method with sizing applied every n iterations, the CA is usually more stable and converges quicker.

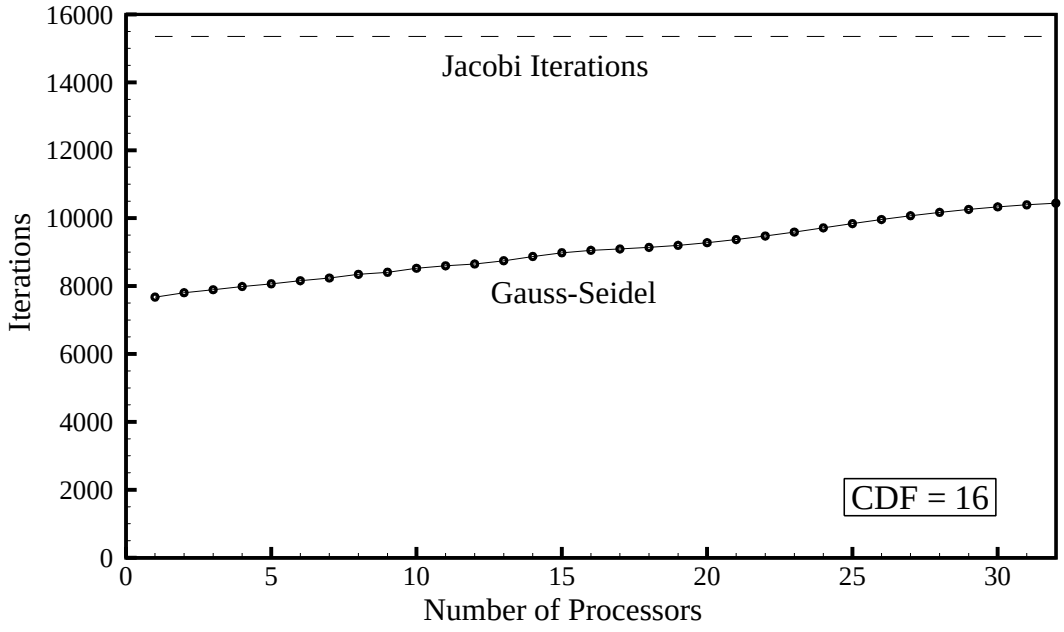

FIGURE 6. *Comparison between Jacobi and Gauss-Seidel iteration methods for analysis only on the bridge structure with a cell density factor of 16.*

As mentioned previously, when using a Gauss-Seidel iteration, the rate of convergence differs depending upon the number of processors used. As shown in Figure 6 for analysis with no sizing, the number of iterations needed to converge increases as the number of processors increases. As the number of processors increases, the smaller the number of cells each processor contains, and therefore the less area each stress and strain can propagate each iteration. Intuitively, this continues until each processor has exactly one cell and (parallel) Gauss-Seidel iteration is exactly the same as Jacobi.

5.2 Sizing Period Using Gauss-Seidel

As seen in the previous section, the choice of how often to apply sizing rules is very important to the speed and the stability of the CA. Since the design equations allow the areas of the bars to adjust fully to the surrounding stresses and strains, it is possible that if a sizing is performed before all the stresses and strains have propagated, bars that are important to the final structure could be allowed to disappear too soon. In this case, the CA will not converge within a reasonable time, or even oscillate and never converge.

Figure 7 shows the bridge structure CA convergence rates in a density plot as a function of the sizing period and the cell density factor. The gray tones represent the number of iterations the CA with sizing needed to converge normalized with respect to the number of iterations needed for

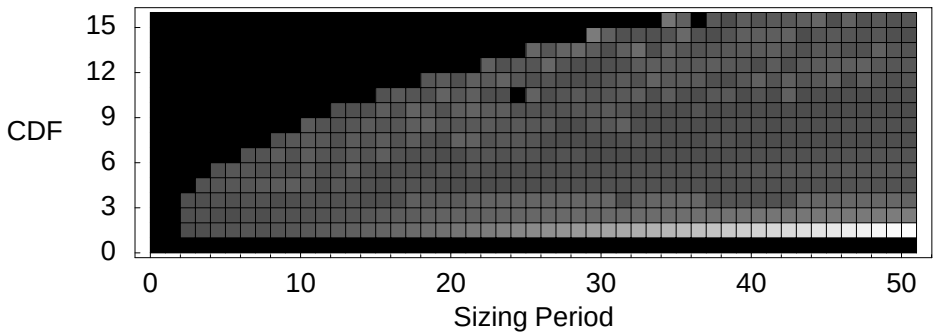

FIGURE 7. *Convergence data using undamped sizing with Gauss-Seidel iteration.*

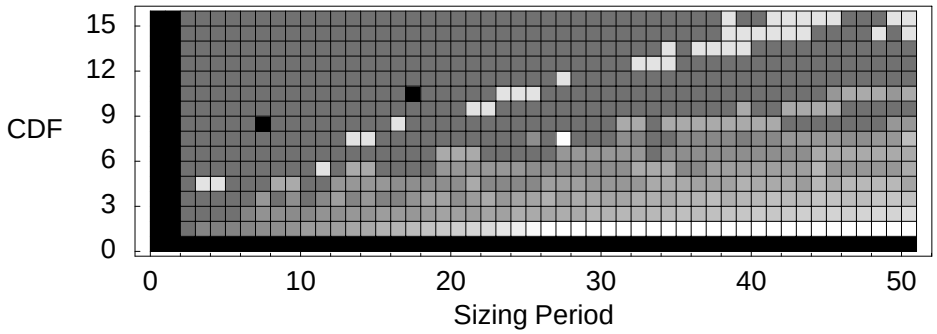

FIGURE 8. *Convergence data using damped sizing with Gauss-Seidel iteration.*

convergence with analysis only. The range goes from one (white) to three (dark grey), with black squares indicating those combinations that did not converge. Therefore the lighter the gray, the faster the CA converged.

Since the CA tends to converge faster with smaller sizing periods it is better to choose a period as small as possible, but if the period is too small then convergence might never be achieved. To allow smaller sizing periods to be chosen, damping (in terms of the rate in which the cross-sectional areas can change) was applied to the design equations. Damping the sizing function limits the area to be within a certain percentage of the current value of the bar area. Figure 8 shows the results for the same problem as in Figure 7 with 10% damping applied.

In this case, far more combinations of CDFs and sizing periods converged than previously, especially when using smaller sizing periods. However, more iterations were needed for each computation to converge than for sizing without damping. Thus there is a tradeoff between speed of convergence and robustness of the code.

5.3 Parallel Speedup

Execution time for the Fortran 90 code was measured for the Gauss-Seidel version of the code since it seemed to be the most stable version. Timing runs were done using a fixed number of iterations that are below the number of iterations required for full convergence. This was to eliminate the vagaries of convergence using Gauss-Seidel on different numbers of processors. Parallel overhead was measured by taking a time stamp at the beginning and end of every communication

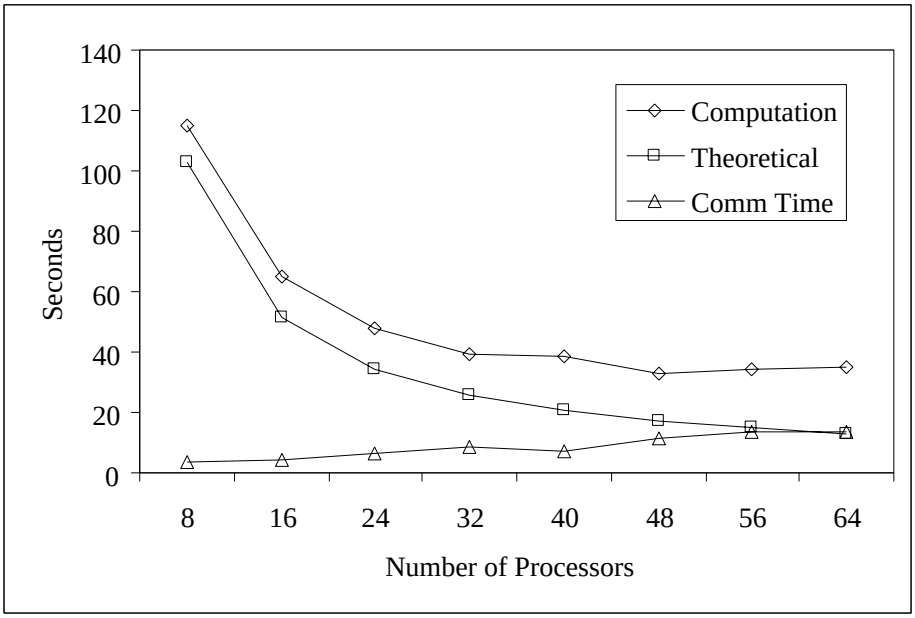

FIGURE 9. *Timings on an Origin 2000 machine for the bridge structure with a CDF of 40 for 20,000 iterations.*

between processors at a single processor. The cumulative time difference is the parallel overhead needed to pack the information, send it to adjacent processors, wait for the updates from the other processors, and then finally unpack the information.

The communication is using blocking MPI calls to aid in measurement, therefore the computation time is the overall time minus the parallel overhead. The design easily allows for a more efficient implementation overlapping the computation and communication by using non-blocking MPI calls.

Figure 9 shows the timing results for an Origin 2000 using shared memory for the communication channels. The data shown in Figure 9 is the average of 5 runs, and the standard deviations ranged between 0.11 and 1.19. In this case, communication overhead does not begin to dominate until about 64 processors are used.

Figure 10 shows the timing results for a Linux Beowulf system using 100Mbit Ethernet between processor nodes. The data shown is the average of 5 runs, and the standard deviations ranged between 0.29 and 0.46. Here the communication overhead dominates from the beginning. The theoretical curve is the ideal speedup, serial time divided by the number of processors.

6. Conclusions

Cellular automata techniques can be applied to structural design and allow efficient use of MPPs. This potentially allows problems of far greater complexity to be solved in a reasonable time. The technique is also easy to implement and is versatile in design of truss topologies. Slow convergence and divergence of the CA is mathematically explained by the spectral radius of the iteration matrix, for analysis only, but adding sizing makes the fixed point iteration $\mathbf{x}^{(n+1)} = F(\mathbf{x}^{(n)})$ nonlinear. A topic for future work is the mathematical analysis of the full nonlinear iteration. Future work could also include extending the method to three dimensions and to creating more types of structures other than ground trusses.

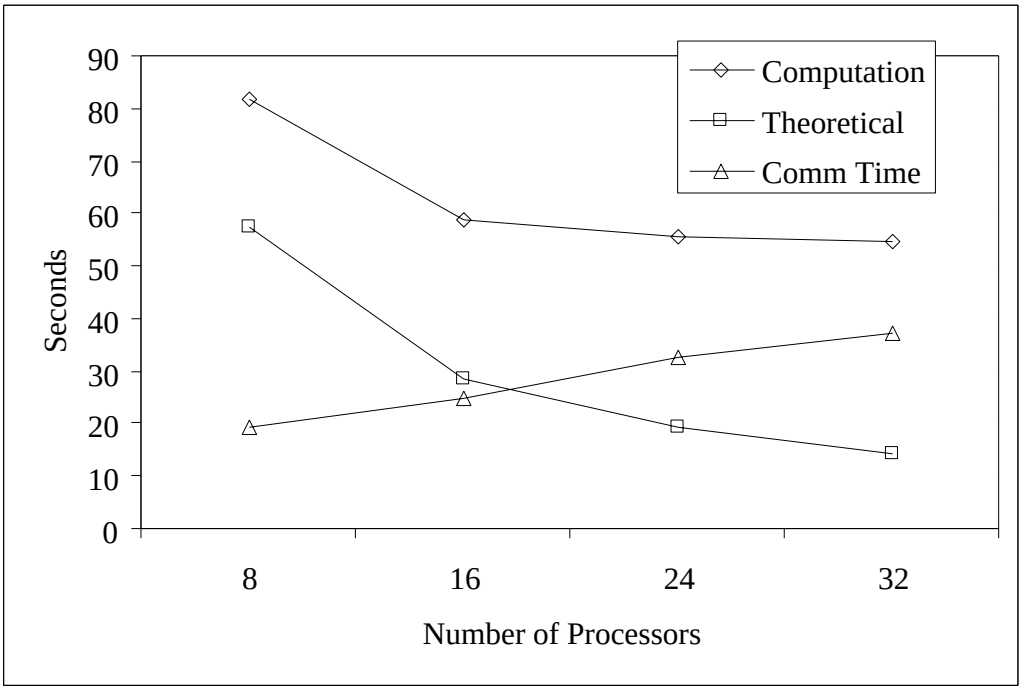

FIGURE 10. *Timings on a Beowulf cluster for the bridge structure with a CDF of 40 for 20,000 iterations.*

References

Bertsekas, D. and Tsitsiklis, J. (1989), *Parallel and Distributed Computation, Numerical Methods*, Prentice Hall, Englewood Cliffs, NJ.

Ca, J. and Thierauf, G. (1997), "Evolution strategies in engineering computation", *Engineering Optimization*, Vol. 29, pp. 177–199.

Gaylord, R. and Nishidate, K. (1996), *Modeling Nature: Cellular Automata Simulations with Mathematica*, Springer-Verlag, New York.

Gürdal, Z. and Tatting, B. (2000), "Cellular automata for design of truss structures with linear and nonlinear response", *Proc. 41st AIAA/ASME/ASCE/AHS Structures, Structural Dynamics, and Materials Conf.*, AIAA Paper 2000-1580, Atlanta, GA.

Hajela, P. and Kim, B. (1999), "GA based learning in cellular automata models for structural analysis", *3rd World Congress on Structural and Multidisciplinary Optimization*, Niagara Falls, NY.

Hajela, P. and Kim, B. (2000), "On the use of energy minimization for CA based analysis in elasticity", manuscript, in preparation.

Issacson, E. and Keller, H. (1966), *Analysis of Numerical Methods*, Dover, New York.

O'Rourke, J. (1994), *Computational Geometry in C*, Cambridge University Press, Cambridge, MA.

Quinn, M. (1993), *Parallel Computing, Theory and Practice*, McGraw Hill, New York.

Quinn, M. (1993), *Designing Efficient Algorithms for Parallel Computers*, McGraw Hill, New York.

Tatting, B. and Gürdal, Z. (2000), "Cellular automata for design of two-dimensional continuum structures", *Proc. 8th AIAA/NASA/ISSMO Symp. on Multidisciplinary Analysis and Optimization*, AIAA Paper 2000-4832, Long Beach, CA.

Toffoli, T. and Margolus, N. (1991), *Cellular Automata Machines*, MIT Press, Cambridge, MA.

Weiner, N. and Rosenblunth, A. (1946), "The mathematical formulation of the problem of conduction of impulses in a network of connected excitable elements, specifically in cardiac muscle", *Arch. Inst. Cardiol. Mexico*, Vol. 16, pp. 205–265.

Wolfram, S. (1994), *Cellular Automata and Complexity: Collected Papers*, Addison-Wesley, Reading, MA.