A Parallel Implementation of the Cellular Potts Model for Simulation of Cell-Based Morphogenesis

Nan Chen[1], James A. Glazier[2], and Mark S. Alber[1]

[1] Department of Mathematics and Center for the Study of Biocomplexity, University of Notre Dame, Notre Dame, IN 46556, USA
{nchen1, malber}@nd.edu
[2] Department of Physics and Biocomplexity Institute, 727 East Third Street, Swain Hall West 159, Indiana University, Bloomington, IN 47405, USA
glazier@indiana.edu

Abstract. Glazier and Graner's Cellular Potts Model (*CPM*) has found use in a wide variety of biological simulations. However, most current CPM implementations use a sequential modified Metropolis algorithm which restricts the size of simulations. In this paper we present a parallel CPM algorithm for simulations of morphogenesis, which includes cell-cell adhesion, haptotaxis and cell division. The algorithm uses appropriate data structures and checkerboard subgrids for parallelization. Communication and updating algorithms synchronize properties of cells simulated on different computer nodes. We benchmark our algorithm by simulating cell sorting and chondrogenic condensation.

Keywords: Computational biology, morphogenesis, parallel algorithms, Cellular Potts Model, multiscale models, pattern formation.

1 Introduction

Simulations of complex biological phenomena like development, wound healing and tumor growth, collectively known as *morphogenesis*, must handle a wide variety of biological agents, mechanisms and interactions at multiple length scales.

Glazier and Graner's Cellular Potts Model (*CPM*) [1] has become a common technique for morphogenesis simulations because it easily adapts to describe cell differentiation, growth, death, shape changes and migration and the secretion and absorption of extracellular materials. CPM simulations treat many biological and non-biological phenomena, including sorting due to cell-cell adhesion, chicken limb bud growth, Dictyostelium discoideum morphogenesis, liquid drainage in fluid foams and foam rheology [2-6].

The CPM approach to modeling makes several choices about how to describe cells and their behaviors and interactions. First, it describes cells as spatially extended but internally structureless objects with complex shapes. Second, it describes most cell behaviors and interactions in terms of effective energies and elastic constraints. These first two choices are the core of the CPM approach. Third, it assumes perfect

S. El Yacoubi, B. Chopard, and S. Bandini (Eds.): ACRI 2006, LNCS 4173, pp. 58 – 67, 2006.
© Springer-Verlag Berlin Heidelberg 2006

damping and quasi-thermal fluctuations, which together cause the configuration and properties of the cells to evolve continuously to minimize the effective energy, with realistic kinetics. Fourth, it discretizes the cells and associated fields onto a lattice. Finally, the classic implementation of the CPM employs a modified Metropolis Monte-Carlo algorithm which chooses update sites randomly and accepts them with a Metropolis-Boltzmann probability.

Since these choices are relatively independent from each other, we can modify some of them to optimize our computation without discarding our basic modeling philosophy. For example, because the acceptance probabilities for updates can be small (10^{-4} – 10^{-6}) the classic lattice-based Metropolis algorithm may run slower than continuum off-lattice implementations. Since the typical discretization scale is 2-5 microns per lattice site, CPM simulations of large tissue volumes require large amounts of computer memory. Current practical single-processor sequential simulations can handle about 10^5 cells. However, a full model of the morphogenesis of a complete organ or an entire embryo would require the simulation of 10^6 – 10^8 cells, or between 10 – 1000 processor nodes.

Clearly, we need a parallel algorithm which implements the CPM and runs on the Beowulf or High Performance Computing Clusters (*HPCC*) [7] available in most universities. Wright *et al.* [11] implemented a parallel version of the original Potts model of grain growth. In this model the effective energy consists only of local grain boundary interactions, so a change of a single pixel changes only the energies of its neighbors.

Gusatto *et al.*'s recent random-walker (*RW*) implementation of the CPM [15] ran approximately six times faster than the standard algorithm on a single processor. In addition, their algorithm parallelizes fairly easily, though a two processor implementation ran only about 15% faster than a one processor version. The standard CPM Metropolis algorithm always rejects spin flips inside a cell, which wastes much calculation time. The RW approach attempts flips only at cell boundaries, reducing the rejection rate and increasing speed. However, the parallel scheme for this algorithm requires shared memory with all processors sharing the same lattice sites, limiting the total lattice size to the memory size of a single computer. Adapting the RW algorithm to accommodate large scale simulations on distributed memory clusters will still require development of an appropriate spatial decomposition algorithm.

The main difficulty in all forms of CPM parallelization is that the effective energy is non-local. The effective energy terms for cell-cell adhesion, haptotaxis and chemotaxis are local, but the constraint energy terms, *e.g.* for cell volume and surface area, have an interaction range of the diameter of a cell. Changing one lattice site changes the volume of two cells and hence the energy associated with all pixels in both cells. For example, if a cell's pixels are divided between the subdomains located on two nodes and the nodes attempt updates affecting the cell simultaneously, without communication, one node has stale information about the state of the cell. If we use a simple block parallelization, where each processor calculates a predefined rectangular subdomain of the full lattice, non-locality greatly increases the frequency of interprocessor communication for synchronization and, because of communication latency, the time each processor spends waiting rather than calculating. To solve this problem, we use an improved data structure to describe cells and decompose the

subdomain assigned to each node into smaller subgrids chosen so that corresponding subgrids on different nodes do not interact, a method known as a *Checkerboard Algorithm*. These algorithms are based on those Barkema and his collaborators developed for the Ising model, see, *e.g.* [9]. These methods allow successful parallel implementation of the CPM using MPI [9, 10].

On the other hand, an intrinsic inconvenience of the classical CPM ameliorates one difficulty which Ising model parallelization faces. In MPI parallelization, the larger the number of computations per pixel update, the smaller the ratio of message passing to computation, which results in less latency delay and greater efficiency. In the Ising model, the computational burden per pixel update is small (at most a few floating point operations), which increases the ratio of message passing to computation in a naive partition. However, in the CPM, the ratio of failed update attempts to accepted updates is very large (10^4 or more in some simulations). Only accepted updates change the lattice configuration and potentially stale information in neighboring nodes. The large effective number of computations per update reduces the burden of message passing. However, because we can construct pathological situations which have a high acceptance rate, we need to be careful to check that such situations do not occur in practice.

2 The Glazier-Graner Cellular Potts Model

Glazier and Graner's CPM generalizes the Ising model from statistical mechanics, and shares its core idea of modeling dynamics based on energy minimization under imposed fluctuations. The CPM uses a lattice to describe cells. We associate an integer index to each lattice site (pixel) to identify the space a cell occupies at any instant. The value of the index at a pixel (i, j, k) is l if the site lies in cell l. Domains (*i.e.* collection of pixels with the same index) represent cells. Thus, we treat a cell as a set of discrete subcomponents that can rearrange to produce cell motion and shape changes. As long as we can describe a process in terms of a real or effective potential energy, we can include it in the CPM framework by adding it to the effective energy. The CPM models chemotaxis and haptotaxis by adding a chemical potential energy, cell growth by changing target volumes of cells and cell division by a specific reassignment of pixels. If a proposed change in lattice configuration (*i.e.* a change in the index number associated with a pixel) changes the effective energy by ΔE, we accept the change with probability:

$$P(\Delta E) = 1, \Delta E \leq 0; \qquad P(\Delta E) = e^{-\Delta E / T}, \Delta E > 0 \tag{1}$$

where T is the *effective temperature* of the simulation in units of energy.

A typical CPM effective energy might contain terms for adhesion, a cell volume constraint and chemotaxis:

$$E = E_{Adhesion} + E_{Volume} + E_{Chemical} \tag{2}$$

We discuss each of these terms below.

Cell–cell adhesion energy: In Equation 2, $E_{Adhesion}$ phenomenologically describes the net adhesion/repulsion between two cell membranes. It is the product of the binding

energy per unit area, $J_{\tau(\sigma)\tau'(\sigma')}$ and the area of interaction of the two cells. $J_{\tau(\sigma)\tau'(\sigma')}$ depends on the specific properties of the interface between the interacting cells:

$$E_{Adhesion} = \sum_{(i,j,k)(i',j',k')} \{J_{\tau(\sigma)\tau'(\sigma')}[1-\delta(\sigma(i,j,k),\sigma'(i',j',k'))]\} \tag{3}$$

where the *Kronecker delta*, $\delta(\sigma,\sigma')=0$ if $\sigma \neq \sigma'$ and $\delta(\sigma,\sigma')=1$ if $\sigma=\sigma'$, ensures that only the surface sites between different cells contribute to the adhesion energy. Adhesive interactions act over a prescribed range around each pixel, usually up to fourth-nearest-neighbors.

Cell size and shape fluctuations: A cell of type τ has a prescribed *target volume* $v(\sigma,\tau)$ and *volume elasticity* λ, *target surface area* $s(\sigma,\tau)$, and *membrane elasticity* λ'. Cell volume and surface area change due to growth and division of cells. E_{Volume} exacts an energy penalty for deviations of the actual volume from the target volume and of the actual surface area from the target surface area:

$$E_{volume} = \sum_{\text{all-cells}} \lambda_\sigma (v(\sigma,\tau)-v_{\text{target}}(\sigma,\tau))^2 + \sum_{\text{all-cells}} \lambda'_\sigma (s(\sigma,\tau)-s_{\text{target}}(\sigma,\tau))^2 \tag{4}$$

Chemotaxis and haptotaxis: Cells can move up or down gradients of both diffusible chemical signals (*i.e.* chemotaxis) and insoluble extracellular matrix (*ECM*) molecules (*i.e.* haptotaxis). The energy terms for both chemotaxis and haptotaxis are local, though chemotaxis requires a standard parallel diffusion equation solver for the diffusing field:

$$E_{chemical} = \mu(\sigma)C(\vec{x}) \tag{5}$$

where $C(\vec{x})$ is the local concentration of a particular species of signaling molecule in extracellular space and $\mu(\sigma)$ is the effective chemical potential.

3 Data Structures and Algorithms

System Design Principles

Our parallel CPM algorithm tries to observe the following design principles: to implement the CPM model without systematic errors, to homogeneously and automatically distribute calculations and memory usage among all processor nodes, and to use Object-Oriented programming and MPI to improve portability.

Spatial Decomposition Algorithm

Our parallel algorithm homogeneously divides the lattice among all processor nodes, one subdomain per node. During a CPM simulation some cells cross boundaries between nodes. If nodes attempted to update pixels in these cells simultaneously, cell properties like volume and surface area would stale and energy evaluations would be

incorrect. We use a multi-subgrid checkerboard method to solve this problem: In each node we subdivide the subdomain into four subgrids indexed from 1-4. During the simulation, at all times we restrict calculations in each node to the same index subgrid. Since these subgrids are much larger than a cell diameter, we guarantee that no calculation in one node affects the calculations occurring simultaneously in any other node. In principle, we should switch subgrids after each pixel update to recover the classical algorithm. Since acceptance rates are low, on average, we should be able to make many update attempts before switching between subgrids. However, because acceptance is stochastic, we would need to switch subgrids at different times in different nodes, which is inconvenient. In practice we can update many times per subgrid (which means accepting that we will sometimes use stale positional information from the adjacent subgrids), because the subgrids are large, the acceptance rate small and the effects of stale positional information just outside the boundaries fairly weak. We use a random switching sequence (the switching sequence each time is different and random, for example, 1234, 2341, 4123, 3124 …) to switch between subgrids frequently enough to make the effect of stale positional information negligible compared to the stochastic fluctuations intrinsic to Monte Carlo methods. Fig. 1 illustrates the algorithm.

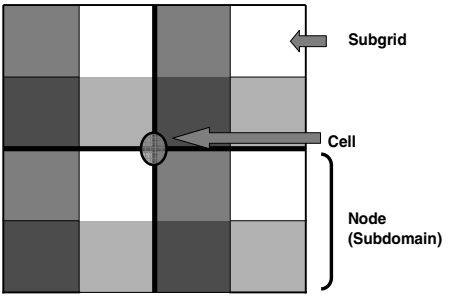

Fig. 1. Spatial decomposition: Each computer node hosts a subdomain which has four subgrids. At any time, each node performs calculations on only one subgrid. At all times, all nodes work on the subgrids with the same index number (indicated by the shading in the figure).

Data Structures

Two basic data structures of the parallel CPM algorithm are the cell and the pixel. During simulations, cells move between subdomains controlled by different nodes. Cells can also appear due to division and disappear due to cell death. In the classical single-processor algorithm, each cell has its own global cell index number. This data structure works efficiently on a single processor. In a parallel algorithm, this data structure for cells requires a Cell Index Number Manager to handle cell division, disappearance and handoff between nodes. For example, when a cell divides in a particular node, the node sends a request to the Manager to obtain a new cell index number and the Manager needs to notify all other nodes about the new cell. Instead, we assign each cell two numbers, a node ID and an index ID. The Node ID is the index number of the node in which the cell was generated and the index ID, like the

old index number, is the index number of the cell generation sequence. Since cell IDs are now unique, each node can generate new cells without communicating with other nodes. Since cells may move between nodes, we dynamically allocate the memory for cell data structures on creation or appearance and release it when a cell moves out of the node or disappears. To optimize the usage of memory and speed data access, the index in each pixel is a pointer to the cell data structure.

Communication and Updating

In the spatial decomposition algorithm, when the program switches between different subgrids, the communication algorithm transfers two types of information: lattice configurations and cell volumes. In 2D, each subgrid needs to communicate with 8 neighboring subgrids (in 3D, 26 neighboring subgrids) and the communication algorithm sequentially sends and receives corresponding data according to the spatial organization of the subgrids. Sending and receiving could take place within a node, in which case the algorithm is just a memory copy. Fig. 2 illustrates the communication algorithm. After the communication, the program needs to dynamically update cell structures and buffers. The program also needs to check whether any cells cross between subgrids and implement the corresponding creation or destruction operations.

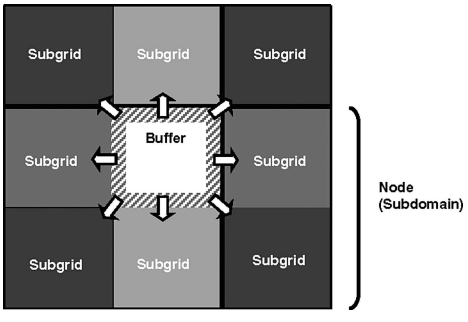

Fig. 2. Communication algorithm: After each change of subgrid, each node needs to transfer data to neighboring nodes. Lattice sites and associated variables (volume, surface area,…) located within the buffer area are transferred so neighboring subgrids contain correct cell configurations and characteristics.

4 Benchmark Results

The following benchmarks used the *Biocomplexity* cluster at the University of Notre Dame. The cluster consists of 64 dual nodes, each of which contains two AMD 64 bit Opteron 248 CPUs (clock frequency 2.2 GHz) and 4GB of RAM.

Cell Sorting

Steinberg's Differential Adhesion Hypothesis (*DAH*), states that cells adhere to each other with different strengths depending on their types [12]. Cell sorting results from

random motions of the cells that allow them to minimize their adhesion energy, analogous to surface-tension-driven phase separation of immiscible liquids. If cells of the same type adhere more strongly, they gradually cluster together, with less adhesive cells surrounding the more adhesive ones. Based on the physics of the DAH, we model cell-sorting due to variations in cell-specific adhesivity at the cell level. Fig. 3 shows two simulation results for different adhesivities. All other parameters and the initial configurations of two simulations are the same. In simulation (a), cell type 1 has higher adhesion energy with itself (is less cohesive) than cell type 2 is with itself. The heterotypic (type 1-type 2) adhesivity is intermediate. During the simulation cells of type 2 cluster together and are surrounded by cells of type 1. In simulation (b), the adhesivity of cell type 1 with itself is the same as the adhesivity of cell type 2 with itself and greater than the heterotypic adhesivity. This energy hierarchy results in partial sorting.

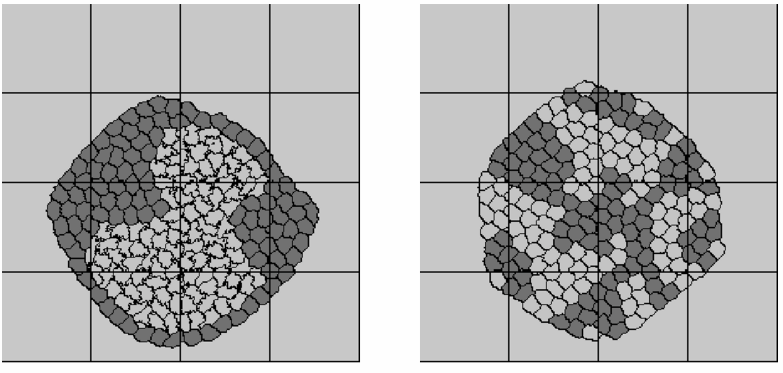

Fig. 3. Cell sorting simulation: Cell type 1 (Dark). Cell Type 2 (Light). The two simulations use the same initial cell configuration and target volumes (150), the only differences between (a) and (b) are the different adhesion constants. (a) Adhesion constants: $J_{1\text{-}1}$=14, $J_{2\text{-}2}$=2, $J_{1\text{-}2}$=11, $J_{1,2\text{-}ECM}$=16. (b) Adhesion constants: Adhesion energy $J_{1\text{-}1}$=14, $J_{2\text{-}2}$=14, $J_{1\text{-}2}$=16, $J_{1,2\text{-}ECM}$=16. The lines indicate the boundaries of the subdomains assigned to each node in a 16 node simulation.

In this simulation the lattice size is (288x288) and we distributed it in homogeneous subdomains of size 72x72 on a 16 node cluster. Each subgrid has 36x36 pixels.

Simulation of Chondrogenic Condensation

Fig. 4 shows the simulation result for a simulation of chondrogenic condensation (cartilage formation) in a chicken limb bud simulation run on 16 nodes with a total lattice size of 1200x1200 sites. In this simulation, we used an externally-supplied chemical pre-pattern (Activator concentration calculated from a pair of coupled reaction-diffusion equations) to control cell differentiation and condensation.

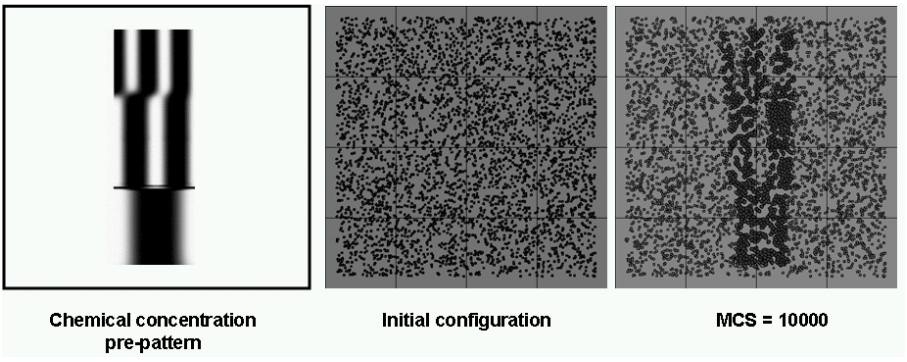

Fig. 4. Simulation of chondrogenic condensation during limb-bud formation. The lines indicate the boundaries of the subdomains assigned to each node for a 16 node simulation.

Efficiency of the Parallel Algorithm

We used the cell sorting (lattice size 288x288) and chondrogenesis simulations (lattice size1200x1200) to analyze the efficiency of our parallel algorithm. We ran both simulations on 4, 9 and 16 nodes with switching between subgrids after each Monte Carlo Step (defined as as many lattice update attempts as the number of lattice sites in the subgrid). This switching rate is relatively slow and results in significant effects from stale parameters. Table 1 summarizes the simulation running times. We define the relative efficiency, f:

$$f = \frac{T_4 / 4}{T_n / n} \tag{6}$$

where T_n is the running time of the simulation on n nodes. Since the smallest cluster on which our program runs uses 4 nodes, we use the running time on 4 nodes as a reference value. Fig. 5 plots the relative efficiency *vs.* the number of nodes. The cell sorting simulation is less efficient than the limb bud simulation because the small (288x288) lattice increases the ratio of communication time to computation time. The larger the subdomain size, the more efficient the calculation.

Table 1. Calculation time for different tests

Tests	Number of Nodes		
	4	9	16
Cell Sorting Simulation. Lattice size 288x288. 10,0000 MCS	3351 Sec.	2352 Sec.	1807 Sec.
Chondrogenesis Simulation. Lattice size 1200x1200. 10,000 MCS	4188 Sec.	2050 Sec.	1305 Sec.

The Gillespie stochastic simulation algorithm acceleration strategy based on “tau-leaping” is a powerful tool for large-scale stochastic biochemical simulations [13][14]. Instead of processing each reaction event, it moves forward in time by “leaps” that include many reaction events. Though it currently applies only to spatially

homogeneous models, its extension to parallel simulation of inhomogeneous models would be valuable and could greatly increase the size of feasible CPM simulations.

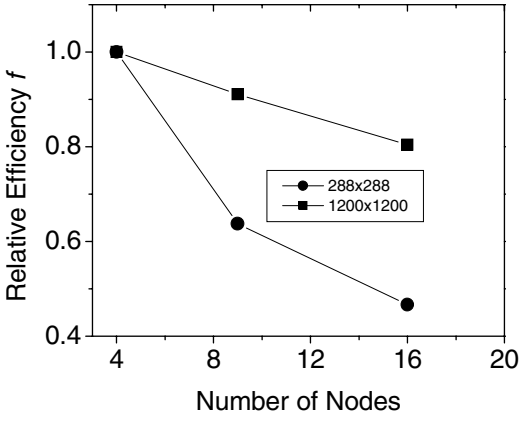

Fig. 5. Relative Efficiency as defined in equation 6 *vs*. the number of nodes used in the calculation. Bullets and solid squares correspond to cell sorting and chondrogenesis simulations respectively.

5 Discussion and Future Work

One issue with our algorithm is whether its results deviate from the classical algorithm significantly. Our switching algorithm works on subgrids one at a time. If the configuration is far from equilibrium, energies and configurations change rapidly and the dynamics of cells at subgrid boundaries could differ from those in the classical algorithm. For instance, if a cell's target volume is much larger than its current volume, the cell should grow rapidly and isotropically, while in our algorithm, a cell at a subgrid boundary might grow anisotropically. A higher switching frequency reduces this problem but also reduces the computational efficiency. In such case, smoothly changing the target value to the equilibrium one would solve this problem.

The parallel algorithm uses the standard CPM site selection algorithm which wastes time by selecting non boundary spins that cannot be updated. We plan to combine our parallel algorithm with the Random Walker algorithm [15] which selects only boundary spins to further improve our simulation efficiency.

6 Conclusions

Sequential versions of the CPM model are extensively used to simulate cell morphogenesis. However, large-scale morphogenesis simulations require a parallel implementation. In this paper, we have proposed a parallel CPM algorithm using appropriate data structures and checkerboard updating. The algorithm reproduces examples of cell sorting and limb bud formation and shows good scalability, which an improved site-selection algorithm like the RW algorithm should be able to improve further.

Acknowledgments. This work was partially supported by NSF Grant No. IBN-0083653 and NIH Grant No. 1R0-GM076692-01: Interagency Opportunities in Multiscale Modeling in Biomedical, Biological and Behavioral Systems NSF 04.6071. Simulations were performed on the Notre Dame Biocomplexity Cluster supported in part by NSF MRI Grant No. DBI-0420980.

References

1. Graner, F. and Glazier, J. A.: Simulation of biological cell sorting using a two dimensional extended Potts model. *Phys. Rev. Lett.* **69**, (1992) 2013–2016.
2. Chaturvedi, R., Huang C, Izaguirre, J. A., Newman, S. A., Glazier, J. A., Alber, M. S.: On Multiscale Approaches to Three-Dimensional Modeling of Morphogenesis, *J. R. Soc. Interface* **2**, (2005) 237-253.
3. Mombach, J., and Glazier, J. A.: Single cell motion in aggregates of embryonic cells. *Phys. Rev. Lett.* **76**, (1996) 3032–3035.
4. Alber, M.S., Kiskowski, M.A., Glazier, J.A., and Jiang, Y., On Cellular Automaton Approaches to Modeling Biological Cells, in J. Rosenthal and D.S. Gilliam (Eds.), Mathematical Systems Theory in Biology, Communication, and Finance, IMA Volume 134, Springer-Verlag, New York, 1-39, 2003.
5. Jiang, Y. and Glazier, J. A.: Foam Drainage: Extended Large-Q Potts Model Simulation. *Phil. Mag. Lett.* **74**, (1996) 119–128.
6. Jiang, Y., Swart, P., Saxena, A., Asipauskas, and Glazier, J. A.: Hysteresis and Avalanches in Two Dimensional Foam Rheology Simulations. *Phys. Rev. E.* **59**, (1999) 5819-5832.
7. See http://www.beowulf.org and links therein for a full description of the Beowulf project, access to the Beowulf mailing list, and more.
8. Barkema, G. T. and MacFarland, T.: Parallel simulation of the Ising model. *Phys. Rev. E* 50, (1994) 1623–1628.
9. Gropp, W., Lusk, E., and Skjellum, A.: *Using MPI: Portable Parallel Programming with the Message Passing Interface, 2nd edition*. MIT Press, Cambridge, MA (1999).
10. Gropp, W., Lusk, E., and Thakur, R.: *Using MPI-2: Advanced Features of the Message-Passing Interface*. MIT Press, Cambridge, MA (1999).
11. Wright, S. A., Plimpton, S. J., Swiler, T. P., Fye, R. M., Young, M. F. and Holm, E. A.: *Potts-model Grain Growth Simulations: Parallel Algorithms and Applications*, SAND Report 97-1925, August (1997).
12. Davis, G. S., Phillips, H. M., and Steinberg, M. S. Germ-layer surface tensions and "tissue affinities" in *Rana pipiens* gastrulae: quantitative measurements. *Dev. Biol.* **192**, (1997) 630-644.
13. Gillespie D. T.: Approximate accelerated stochastic simulation of chemically reacting systems. *J. Chem. Phys.* **115**, (2001) 1716-1733.
14. Lok, L: The need for speed in stochastic simulation. *Nature Biotechnology*, 22, (2004), August, 964
15. Gusatto, E., Mombach, J.C.M., Cercato, F.P., Cavalheiro, G.H.. An efficient parallel algorithm to evolve simulations of the cellular Potts model. *Parallel Processing Letters*, v.15, p. 199-208, 2005.