

# Preliminary Work on Data-Parallel Execution of Cellular Potts Models

Jose Juan Tapia Valenzuela, Roshan M. D'Souza

Dept. of Mechanical Engineering, Houghton, 49931  
{jjatpiv,rmdsouza}@mtu.edu

**Abstract:** The Cellular Potts Model (CPM) is a very important algorithm in systems biology to simulate cell sorting and the formation of organ structures. However its computational intensity has limited the size and scope of its applicability. Parallel computing techniques using traditional cluster computing have been marginally successful in addressing scalability of CPMs. In this paper, we explore data-parallel computing architectures such as graphics processing units (GPUs) for simulating large-scale CPMs. Our preliminary results indicate that our techniques running on a single desktop are up to 30x faster than previous methods executing on computing clusters consisting of upto 25 nodes.

**Keywords:** Cellular Potts Model, Data-Parallel Computing, GPGPU

## 1. Introduction

The Cellular Potts Model(CPM) introduced by Glazier and Graner <sup>2)</sup> is an extension of the extended large-q Potts model. This lattice based model has traditionally been used for the simulation of phenomenon as diverse as formation of metallic grains<sup>4)</sup>, soap bubbles, and foam; however in recent years there has been a lot of interest into its application in the simulation of the collective behavior of cellular structures. In the CPM, a biological cell is represented as an internally structure-less collection of pixels, where each pixel is one point of the lattice. The CPM is evolved by updating pixels according to a series of probabilistic rules, based on the cell properties in our simulation. A CPM typically includes such properties as cell-cell adhesion energy, constrains for the volume and surface of our cells, managing different type of cells to see how they interact, etc.

Due to the emergent nature of CPMs, useful simulations typically have very large sizes. In biological simulations, each lattice site must represent 2-5  $\mu m$  with lattice sizes reaching  $10^7 - 10^9$  lattice points. This makes these simulations expensive both in memory requirements and computation time. Typically the resources required are much beyond what is available on a computer containing a single CPU. Consequently, researchers have used parallel computing using a cluster of CPUs to scale CPM simulations.

## 2. Previous Work

One of the first attempts at parallelizing the CPM was that of Wright et al.<sup>4)</sup>. Since they only considered local rules, parallelization using computing clusters enabled them to simulate extremely large lattice sizes. Chen et al. <sup>1)</sup> solved many of the limitations of Wright et al's work by dividing up the lattice into blocks to be processed by different nodes in a cluster. A checkboard

execution scheme was introduced to minimize communication due to synchronization between nodes. They also showed that due to the stochastic nature of the simulation, model accuracy does not suffer much if synchronization does not takes place at every time step. This analysis is based on the assumption that lattice sites updates are rare occurrences more so at the boundaries between regions. However, for large-scale simulations, this assumption may not hold true. The Random Walker algorithm by Gussatto et al.<sup>3)</sup> tracks cell boundary sites and therefore only valid spin flips are tested. However, their implementation contains several downfalls, one of the biggest being that they require a complete lattice copy to be in each of their nodes, which limits scalability.

## 3. Approach

Our approach differs significantly from previous efforts because we use a completely new computing model, namely, stream computing which is optimized for high throughput devices such as graphics processing units(GPUs) for implementing the CPM. GPUs achieve high through put computation by using thousands of lightweight execution threads. For a given processing step, all threads simultaneously execute the same program called a kernel on different parts of the input data. This means data-interdependencies and branching have to be limited as they cause significant degradation of performance. Threads are grouped into thread-blocks and threads in a block have access to 16kB of register space called shared memory. The other memory types include global memory which is similar to CPU random access memory(RAM), and texture memory. Global memory reads and writes can be made very efficient if they can be coalesced. Texture memory is read-only and very efficient for data-structures that have spatial components.

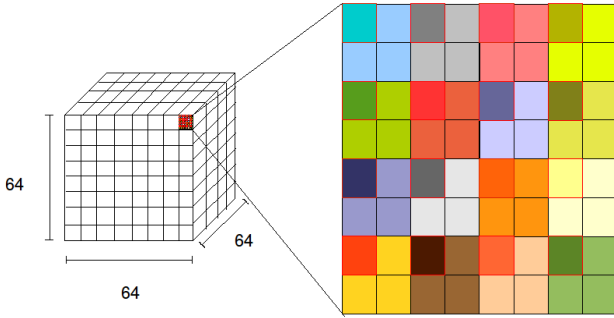


Figure 1: Example of how the spatial subdivision works. Each color represents a section analyzed by a different thread.

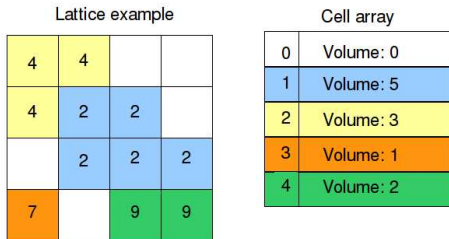


Figure 2: Data structures

We parallelize the simulation of the CPM by dividing up the lattice into blocks that will be processed by different thread blocks. Each block is further divided into sub-blocks that are to be processed by individual threads. A checkerboard scheme similar to Chen et al. is used to avoid conflicts between threads as they update lattice sites. Figure 1 illustrates this subdivision. Throughout the simulation, we keep track of lattice data and cell data. Every lattice holds a cell ID. Each cell contains the cell ID and cell volume, a global property. Figure 2 illustrates cell and lattice data storage. A double buffering scheme is used to avoid data corruption due to simultaneous read and write. Since lattice update involves spatially coherent but un-coalesced memory reads, we use texture memory to store lattice data. The output of the processing is written to global memory. At the end of each update step, the updated lattice data in global memory is efficiently block copied to texture memory. We also use atomic operations to enforce the condition that no more than one lattice site is updated per cell during a single update.

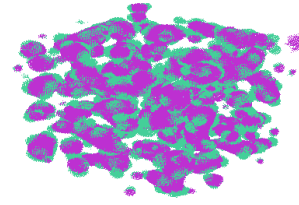


Figure 3: Cell sorting

## 4. Results

We implemented our algorithms on a desktop running Ubuntu 7.0 OS. The graphics card used was a NVIDIA 280 GTX with 1GB of main memory. Figure 3 shows the graphical output of cell sorting. We benchmarked against parallel implementation on a computing cluster consisting of 25 nodes by Chen et al.<sup>1)</sup> as well as an implementation of the random walk algorithm<sup>3)</sup>. The results are shown in Table 1. The timing results are the average of 40 runs with random simulation parameters.

## 5. Discussion

In this paper, we have demonstrated how the GPU is an ideal architecture for scaling the Cellular Potts Model. With speedups of nearly 30x, combined with the much lower economical cost of acquiring a standard computer with a high-end graphics card, when compared to the cost of acquiring and maintaining a computer cluster, the advantages of our parallel implementation over traditional approaches are clear. The performance advantages are due to the special architecture of the GPUs. By far the most significant advantage the GPUs have is the memory bandwidth. A simple GFLOPS comparison between a 25 node cluster and the NVIDIA 280 GTX would indicate that our implementation should be no more than 9x faster than the cluster implementation. The main disadvantage with GPUs is the programming model for which entirely new sets of algorithms have to be developed. We are working towards developing new data-parallel algorithms and interfaces to enable easy access to this powerful platform.

## References

- [1] N. Chen, J.A. Glazier, J.A. Izaguirre, and M.S. Alber. A parallel implementation of the Cellular Potts Model for simulation of cell-based morphogenesis. *Computer Physics Communications*, 176(11-12):670–681, 2007.
- [2] F. Graner and J.A. Glazier. Simulation of biological cell sorting using a two-dimensional extended Potts model. *Physical Review Letters*, 69(13):2013–2016, 1992.

Table 1: Comparison benchmark

Model	Grid Size	Cells( $10^6$ )	Iterations	Time(secs)
RW Cluster(4)	1E6		10000	2145±6
MC Cluster(25)	2.25E6	0.09	2000	370
MC GPGPU	2.097E6	0.1	2000	12.08±0.21
MC GPGPU	4.19E6	0.2	2000	25.47±0.61
MC Cluster(9)	9E6	0.36	200	254
MC GPGPU	8.38E6	0.4	200	4.73±0.35
MC GPGPU	8.38E6	0.4	2000	49.63±1.11
MC Cluster(16)	1.6E7	0.64	200	274.5
MC GPGPU	1.677E7	0.8	200	10.02±0.28
MC GPGPU	1.677E7	0.8	2000	98.03±2.06

- [3] E. Gusatto, J.C.M. Mombach, F.P. Cercato, and G.H. Cavaleiro. An efficient parallel algorithm to evolve simulations of the cellular Potts model. *Parallel Processing Letters*, 15(1):199–208, 2005.
- [4] S.A. Wright, S.J. Plimpton, T.P. Swiler, R.M. Fye, M.F. Young, and E.A. Holm. Potts-model grain growth simulations: Parallel algorithms and applications. *Sandia Report SAND97-1925*, 1997.