

# Data-Parallel Algorithms for Agent-Based Model Simulation of Tuberculosis On Graphics Processing Units

Roshan M. D'Souza  
Dept. of Mechanical Engineering  
Michigan Tech. University  
Houghton, MI, 49931  
Email: rmdsouza@mtu.edu  
Simeone Marino and Denise Kirschner  
Dept. of Immunology and Microbiology  
University of Michigan  
Ann Arbor, MI  
Email: simeonem@gmail.com, kirschne@unmich.edu

**Keywords:** Agent-Based Models, Integrative Systems Biology, GPGPU

## Abstract

Agent-based modeling has been recognized as a method to bridge the translational gap in integrative systems biology. However, the computational complexity of agent-based models at biologically relevant scales makes simulation impractical on traditional CPU-based serial computing. In this paper we present a series of algorithms for simulating large scale agent-based models on graphics processing units (GPUs). GPUs have recently emerged as a powerful and economical computing platform for certain applications in scientific computing. As a test case, we have implemented an agent-based model of tuberculosis. This model simulates the interaction of the human immune system in the lung with *Mycobacterium Tuberculosis* and tracks the formation of characteristic structure called granulomas. The model represents immune cells such as T-Cells and Macrophages using mobile agents, effector chemokines using field equations, and bacteria. Our implementation is two orders of magnitude larger and runs three orders of magnitude faster than the original CPU-based implementation.

## 1. INTRODUCTION

In the past two decades, there has been much progress in understanding intra cellular processes. Research in the "omics" fields such as proteomics and genomics has generated vast amount of knowledge. However, this knowledge has not resulted in many usable clinical therapies. The traditional reductionist approach use in systems biology where complex systems are reduced to a set of linear models has not worked[1]. This is because biological systems are inherently coupled, non-linear and heterogeneous. Moreover the underlying processes span multiple scales in time and space.

Increasingly mathematical formulations are being used to

build computer models of these phenomena. Processes such as diffusion-reaction dynamics of chemicals can be modeled using deterministic partial differential equations. In case of interactions that are intrinsically random, stochastic differential equations are being used[2]. These two approaches are referred to as equation based modeling as algebraic equations are used to describe system dynamics. These approaches are valid if the continuum assumptions hold. For systems where continuum analysis fails due to the paucity in number of the molecules, there are stochastic methods such Gillespie's Stochastic Simulation Algorithm (SSA)[3][4]. SSAs have become extremely important in characterizing intra-cellular signaling events[5][6]. However, all these previous methods are not suited to the multi-scale nature of biological systems.

### 1.1. Agent-Based Modeling in Systems Biology

Agent-based modeling has been recently recognized as a method to overcome the translational gap in systems biology[7]. Agent-based models(ABM) are direct computational representations of discrete dynamics systems where the system level behaviors emerge from the local interaction of constituent entities called agents[8]. On a computer, agent are represented as concurrent objects whose state transitions are governed by rules. Agent-based modeling is well suited to capture the multi-scale nature of biological systems. ABMs are well suited to capture stochasticity, heterogeneity, and hierarchy that is present in biological systems. Agent rules can be explicitly derived from the description of biological functions obtained through bench research. Unlike equation based modeling, ABMs are descriptive and it is easier for biologists with limited mathematical background to relate to this modeling technique. It has been previously used to simulate inflammatory cell tracking[9] [10], to model tumor growth[11][12], to simulate intracellular processes[13][14][15] bio-molecular reactions, wound healing[16][17], morphogenesis[18] [19] [20], microvascular

patterning[21], pharmacodynamics[22][23][24][25], tuberculosis[26], and sepsis[7][27].

## 1.2. Limitations of Agent-Based Modeling

ABMs suffer from two major limitations. Due to their emergent nature, their behaviors cannot be characterized analytically. Consequently, techniques for phase plane, bifurcation, and stability analysis are non-existent[28]. ABMs are to be used in a manner similar to experimentation where several Monte Carlo runs have to be done to generate statistically dense data sets[27]. The emergent nature also means that the system level behaviors are sensitive to agent populations. It is therefore required to have representative agent populations in the model. In systems biology, this can range anywhere from a few hundred thousand agents to hundreds of millions of agents.

There are several ABM toolkits that have been developed over the previous two decades[29][30][31]. Most of these toolkits provide a set of libraries for canonical agent actions that can be extended to implemented model specific agent actions using OOP concepts such as inheritance. These toolkits are essentially discrete-event simulators[32] that model agent actions as discrete events. A scheduler queues all the agent actions for a given time step and executes them one at a time. They are designed to execute serially on the central processing unit (CPU). Large-scale biological models can have billions of agent actions for every time step. Such large scale simulations are infeasible on contemporary ABM toolkits. Parallel computing solutions using computing clusters are marginally more scalable as compared to single computer implementations. Large scale ABM simulation requires entire new architectures and algorithms.

## 1.3. ABM Simulation on Graphics Processing Units

GPUs were originally developed to handle computations related to the shaded display of 3D objects. The need for novel shading effects particularly in video games led vendors to enable customized shading through user-defined functions that could be programmed directly into hardware. This programmability enabled computational scientists to use the same hardware for general purpose scientific computing. Recently, the increased use of GPU in scientific computing has led to the development of specialized computing cards and direct APIs.

GPUs follow the stream programming model[33]. Data is represented as a stream of elements of the same type. The data elements could be atomic variables such as integers, floats, characters etc. or user defined complex data structures. Functions called kernels mutate the data. In a typical update step, several thousand instances of a single kernel act in parallel of different elements of the data stream. One major restriction

on the data is that the update of the  $i^{th}$  element in the stream is not dependent on the updates of other elements in the same time step, i.e., there are no precedence constraints. While a small number of precedence constraints could be handled, a strict ordering of events does not facilitate parallel execution. The main challenge lies in formulating the ABM in terms of the stream programming model.

## 2. PREVIOUS WORK

There are several research efforts related to ABM simulation on the GPU. Harris et al.[34] developed an extension of Cellular Automata called Coupled Lattice Maps to simulate boiling, react-diffusion, and convection. Kolb et al. [35] developed a particle simulator implemented almost entirely on the GPU. For spawning, they used a CPU-based allocator. However, this allocator can be prohibitively time consuming for large model sizes. Millán and Rudomín[36] developed a system to simulate crowd behaviors on the GPU. However, their work did not incorporate agent replication and collision avoidance. More recently Perumalla et al. [37] developed an extended cellular automata approach to simulate several canonical ABMs on the GPU. While their technique did include collision avoidance, it does not incorporate replication. In addition, mobile agents are bound to lattice sites with a maximum of one agent per site. Richmond et al. [38] have recently developed techniques to simulate agents in a grid-free environment. Their method is similar to many particle systems developed previously. However, their agents are able to communicate with each other in a user-defined radius.

The authors previously developed techniques for simulating large-scale ABMs on GPUs [39]. They demonstrated their techniques by implementing a large-scale version of the SugarScape ABM. They incorporated incorporated features such as environmental resource growth, agent replacement, agent movement, agent mating, pollution formation, and pollution diffusion. They develop a randomized memory allocator that executed entirely on the GPU. In addition, they developed a collision resolution algorithm based on z-culling. This work has greatly expanded the size and complexity of models that can be handled by performing all computations entirely on the GPU.

## 3. PAPER OVERVIEW

In this paper we describe our data-parallel implementation of an ABM to simulate the formation of characteristic structures called granulomas in the infection pathology of *Mycobacterium Tuberculosis (MTb)*. This model was previously developed by Segovia et al. [26]. The MTb ABM is a discrete-time, discrete-space representation of the complex dynamics resulting from the interactions of human immune cells with the *MTb* pathogen in the lung. Mobile agents are used to represent immune cells such as T-Cells and

Macrophages. The environment represents an section of the aveolar lung tissue. Effects captured in the simulation include agent motion, chemotaxis, diffusion-decay of chemokines, lysis of bacteria by macrophages, and adaptive immune response. We begin by briefly describing different elements of the MTb ABM. Next, we describe various data-parallel algorithms that have been developed to implement the ABM. The results section describe the bench mark studies against a serial implementation of the same ABM. We conclude the paper with a discussion of results and future work.

### 3.1. Environment

The environment in the MTb ABM is modeled using as a grid of static agents. These agents keep track on the level of chemokines, bacteria, and the number of mobile immune cells present at each grid location. Chemokine dynamics are modeled using a linear diffusion decay equation given by:

$$\frac{dE}{dt} = \lambda \nabla^2 E - \gamma E \quad (1)$$

where  $E$  is the field,  $\lambda$  is the diffusion constant, and  $\gamma$  is the decay constant. Infected macrophages act as sources that release chemokines at the grid point on which they are located at a given time. Since the chemokines dynamics are two orders of magnitude faster than mobile agent dynamics, a quasi static assumption is used. In between every mobile agent update, the chemokine field is advanced by hundred time steps assuming stationary agents. Subsequently, the chemokine field is updated to account for chemokine release by mobile agents using equation given by:

$$E(t + \delta t) = E(t) + C_m * \delta t \quad (2)$$

where  $C_m$  is the chemokine released by an infected macrophage per unit time, and  $\delta t$  is the time period of mobile agent update. In the current version of the model, bacteria are seeded at the center of the environment grid. Extra-cellular bacteria grow in the environment according to the equation:

$$B(t + \delta t) = \min\{B(t) + B(t) * \alpha_{BE} (1 - \frac{B(t)}{1.1 * K_{BE}}), K_{BE}\} \quad (3)$$

where  $B(t)$  is the bacteria count at a grid location,  $\alpha_{BE}$  is the bacteria growth rate,  $K_{BE}$  is the maximum bacteria count that can be held at any grid location. In addition, there is a constraint that any grid cell can contain no more than one macrophage and one t-cell simultaneously. A randomly chosen sub-set of the grid locations act as source compartments. These source compartments model blood vessels and are conduits for macrophages and t-cells to appear into the simulation.

### 3.2. Macrophages

Macrophages are the most complex of the agent types in the simulation. Every macrophage is represented using a mobile finite state machine (FSM). The macrophage FSM has five states, namely: RESTING, INFECTED, ACTIVATED, CHRONICALLY INFECTED, and DEAD. Each macrophage has a finite life span. When they enter the simulation, all macrophages are in the RESTING state. In this state, they are capable of killing small amounts of extra-cellular bacteria. They can also uptake extra-cellular bacteria and become INFECTED. Intra-cellular bacteria  $B_I(t)$  in INFECTED macrophages grow according to the equation:

$$B_I(t + \delta t) = B(t)_I + B_I(t) * \alpha_{BI} \quad (4)$$

where  $\alpha_{BI}$  is the bacteria growth rate. An INFECTED macrophage can become ACTIVATED with a small probability if it encounters a T-Cell in its vicinity. In the ACTIVATED state, the macrophage kills all internal and extra-cellular bacteria it encounters. An INFECTED macrophage can also become CHRONICALLY INFECTED if the intra-cellular bacteria grow and cross a pre-defined threshold. Intracellular bacteria in CHRONICALLY INFECTED macrophage grow according to:

$$B_I(t + \delta t) = \min\{B(t)_I + B_I(t) * \alpha_{BI} (1 - \frac{B_I(t)}{K_I + 30}), K_{BI}\} \quad (5)$$

CHRONICALLY INFECTED macrophages burst when the intra-cellular bacteria reach another threshold  $K_{BI}$  and spread the bacteria in the vicinity. CHRONICALLY INFECTED macrophages can also die if they encounter T-Cells. Macrophages also die when the age exceeds the lifespan.

### 3.3. T-Cells

T-Cells are immune effectors which start appearing in the simulation on the tenth day of the infection. They have two states, namely, LIVE and DEAD. T-Cells move about in the environment and perform the task of activating INFECTED macrophages and killing CHRONICALLY INFECTED macrophages.

### 3.4. Agent Motion

In the absence of a chemokine field, both T-Cells and Macrophages perform random motion. In the presence of a chemokine field, motion occurs in the direction of the highest gradient. All motion is governed by speed which is dependent on the state of the agent. An agent moves to a grid point if and only if the grid point is not occupied by an agent of the same type. Otherwise, the agent remains stationary for that time step. In case the grid point is occupied by another type of agent, motion occurs with a small probability.

### 3.5. Cell Recruitment

At the beginning of the simulation, the environment is seeded randomly with macrophages. As the simulation proceeds, macrophages and t-cells are recruited from outside the environment through the source compartments. Cell recruitment occurs only if there is a perceptible chemokine field and the source compartment is not already occupied by a cell of the same type.

## 4. DATA-PARALLEL TECHNIQUES FOR MTB ABM SIMULATION

The main challenge lies in formulating the ABM in terms of stream computation. Model data is to be represented as stream, while as model update function are to be represented as kernels. In our implementation, we use flat arrays to represent model data. Various agent actions such motion, .. are formulated as kernels.

### 4.1. Data Representation

There are five arrays to represent model data. The first is an array for representing macrophage state data. The macrophage data structure is given by:

```
struct Macrophage
{
position pos; //environment cell location
unsigned int state; // agent state
float bacteria; //bacteria load
float age; //remaining life span
};
```

The second array is for representing T-cell state data. The T-cell data structure is given by:

```
struct TCell
{
int x, y; //environment cell location
float fx, fy; //actual position
unsigned int state; // agent state
float age; //remaining life span
};
```

The third array holds the environment state data. The environment data structure is given by:

```
struct env
{
float bacteria;
int sourceIndex;
}Env[];
```

Note that the environment data structure does not have space for spatial location. Since environment agents are static,

their location is implicitly defined by the location in memory. The variable "sourceIndex" indicates if the environment grid point is a source location and if so, the index of the source compartment. The value of sourceIndex ranges from 0 to MAX\_SOURCE\_COMPARTMENTS. Source location are places where cell recruitment occurs. The chemokine field is represented using a separate array "E" of floats. This is to facilitate manipulation using Fourier Transforms. Finally, there are two bit arrays "Tcmap" and "Mcmap", one for resolving T-Cell collision and the other for resolving Macrophage collisions respectively.

### 4.2. Agent State Update

Agent state update is accomplished using a set of kernels. The CPU is used to allocate memory on the GPU for various agent state arrays as well as to initialize these arrays before they are copied to the GPU (left side of figure 1). On the right hand side in figure 1 is the illustration of the order in which various kernels are invoked. In the following sections, each of the kernels is described briefly.

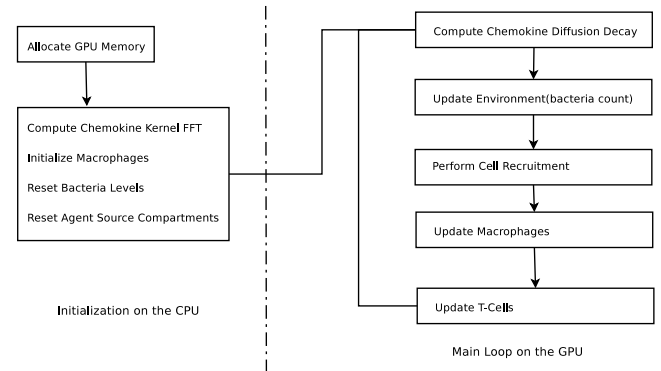


Figure 1. Data-parallel simulation loop

#### 4.2.1. Chemokine Update

As described previously, the chemokine dynamics are two orders of magnitude faster than mobile agent dynamics. Therefore, the model requires 100 updates of the chemokine diffusion-decay equation before the mobile agents are updated. The application of the diffusion decay kernel on the chemokine field can be thought of as a convolution given by:

$$E(i, j) = \sum_{l=0}^m \sum_{k=0}^m f(i-k, j-l)E(k, l) \quad (6)$$

The size of  $f$  is a function of the number of time steps  $m$ . The overall complexity of this operation is given by  $O(m^2n^2)$ , where  $n$  is the width of the chemokine field. If  $m \ll n$ , then it is best to do a direct convolution. However, for large  $m$ , computing this convolution in the Fourier Domain improves the complexity to  $O(m^2 \log(m) + n^2 \log(n))$ . Computation in the

Fourier domain is possible because the governing equation is linear time-invariant and the domain has periodic boundary conditions. At the beginning of the simulation, the diffusion-decay kernel for 100 time steps in the spatial domain is computed on the CPU. It is then transferred to the GPU and the FFT is computed and stored on the GPU. Algorithm 3 illustrates this procedure.  $\hat{f}$  is the FFT of the convolution kernel  $f$ .

---

**Algorithm 1** Computer Diffusion Decay Kernel

---

```

1: procedure compute-diffusion-decay-kernel()
2:   On the CPU
3:    $f \leftarrow$  compute time domain kernel for 100 times steps
4:   Transfer  $f$  to the GPU
5:    $\hat{f} \leftarrow$  FFT( $f$ )

```

---

Algorithm 3 illustrates the procedure to update the chemokine field in the Fourier domain. This procedure is run along with the mobile agent update. This essentially eliminates the "stiffness" associated with the fast chemokine dynamics. Lines 2-4 solve the diffusion decay equation in the frequency domain. Lines 6-9 account for chemokine release by INFECTED, CHRONICALLY INFECTED, and ACTIVE macrophages.

---

**Algorithm 2** Update Chemokine Field

---

```

1: procedure update-chemokine-field(E,M)
2:    $\hat{E}(t) \leftarrow$  FFT( $E$ )
3:    $\hat{E}(t + \delta t) \leftarrow \hat{f} * \hat{E}(t)$ 
4:    $E(t + \delta t) \leftarrow$  invFFT( $\hat{E}(t + \delta t)$ )
5:   for all  $m \in M$  in parallel do
6:      $loc \leftarrow$  find_loc( $m.pos$ )
7:     if  $m.state ==$  INFECTED OR CHRONICALLY INFECTED OR ACTIVATED then
8:        $E[loc] += C_I$ 
9:     end if
10:  end for

```

---

#### 4.2.2. Environment Kernel

The environment kernel is the simplest. It only handles extra-cellular bacteria growth according to equation . Algorithm 1 illustrates the pseudo-code for environment updates.

---

**Algorithm 3** Calculating Extra-Cellular Bacteria Growth

---

```

1: procedure update-extra-cellular-bacteria(Env)
2:   for all  $e \in Env$  in parallel do
3:      $b \leftarrow e.bacteria$ 
4:      $b \leftarrow \min\{b + b * \alpha_{BE}(1 - \frac{b}{1.1 * K_{BE}}), K_{BE}\}$ 
5:      $e.bacteria \leftarrow b$ 
6:   end for

```

---

#### 4.2.3. Macrophage Update

Macrophages are the most complicated agents in the *MTb* ABM. The behaviors of the macrophages are determined by the agent state. Each of the agent states has different behavioral rules. The variable ID is the index of the macrophage in the array  $M$ . It is implicitly calculated by the thread of execution. The main kernel function listed in Algorithm 4 first accounts macrophage death due to aging (lines 4-8). Line 9 handles agent motion. Next based on the current state of the macrophage, different sub-kernels are called (lines 10-22).

---

**Algorithm 4** Update Macrophages

---

```

1: procedure update-macrophages(M)
2:   for all  $m \in M$  in parallel do
3:     ID  $\leftarrow$  threadID
4:      $m.age \leftarrow m.age - \Delta t$ 
5:     if  $m.age < 0$  then
6:        $m.state = DEAD$ 
7:       return
8:     end if
9:     update_position(ID, $m.speed,m.pos,Mcmap$ )
10:    if  $m.state == RESTING$  then
11:      restingRules( $m$ )
12:    end if
13:    if  $m.state == INFECTED$  then
14:      infectedRules( $m$ )
15:    end if
16:    if  $m.state == CHRONICALLY\_INFECTED$  then
17:      chronicallyInfectedRules( $m$ )
18:    end if
19:    if  $m.state == ACTIVATED$  then
20:      activatedRules( $m$ )
21:    end if
22:  end for

```

---

Algorithm 5 accounts for RESTING behaviors. If the macrophage is at an environment grid where there is sufficient bacteria ( $N_{RK}$ , with a small probability  $p_k$  it can get infected. During infection, it uptakes  $N_{rk}$  amount of bacteria. Its speed also changes (lines 5-10).

---

**Algorithm 5** Resting Rules

---

```
1: procedure restingRules(m)
2: loc ← find_loc(m.pos)
3: bac ← Env[loc].bacteria
4: r ← rand[0,1]
5: if bac >  $N_{rk}$  then
6:   bac ← max(bac- $N_{rk}$ , 0)
7:   if r <  $p_k$  then
8:     m.bacteria =  $N_{rk}$ 
9:     m.state = INFECTED
10:    m.speed = INFECTED_SPEED
11:  else
12:    return
13:  end if
14: end if
```

---

Algorithm 6 accounts for INFECTED behaviors. Infected macrophages grow intra-cellular bacteria (line 2). Lines 3-6 implement state change to CHRONICALLY\_INFECTED if the intra-cellular bacteria grow beyond a threshold  $N_c$ . Lines 8-14 handle macrophage activation by T-Cells in the Moore neighborhood.

---

**Algorithm 6** Infected Rules

---

```
1: procedure infectedRules(m)
2: m.bacteria = m.bacteria *(1+ $\alpha_{BI}$ )
3: if m.bacteria >  $N_c$  then
4:   m.state = CHRONICALLY_INFECTED
5:   m.speed = CHRONICALLY_INFECTED_SPEED
6:   return
7: end if
8: n ← number of T-Cells in the Moore neighborhood
9: r ← rand[0,1]
10: if r <  $n * t_{actm}$  then
11:   m.state = ACTIVATED
12:   m.bacteria = 0.0
13:   m.speed = ACTIVATED_SPEED
14: end if
```

---

Algorithm 7 accounts for CHRONICALLY INFECTED behaviors. Line 2-4 handle intra-cellular bacteria growth. Lines 5-10 handle the case where the intra-cellular bacteria load is less than a threshold  $K_{BI}$  and the macrophage is killed by a T-Cell. Lines 14-16 handle the case where macrophages die because of bacteria overload.

---

**Algorithm 7** Chronically Infected Rules

---

```
1: procedure chronicallyInfectedRules(m)
2: bac ← m.bacteria
3: bac = min{ bac+ bac* $\alpha_{BI}$  (1- $\frac{bac}{K_T+30}$ ),  $K_{BI}$  }
4: m.bacteria ← bac
5: if bac <  $K_{BI}$  then
6:   n ← min number of T-Cells in the Moore neighborhood
7:   r ← rand[0,1]
8:   if n > 0 AND r <  $pT_k$  then
9:     m.bacteria *= 0.5
10:    burst(m)
11:   else
12:     return
13:   end if
14: else
15:   burst(m)
16: end if
```

---

Algorithm 8 handles macrophage death. When a macrophage dies, the intra-cellular bacteria load is evenly distributed between the environment cells in the Moore neighborhood (lines 2-5).

---

**Algorithm 8** Killing Macrophages

---

```
1: procedure burst(m)
2: for p ∈ Moore neighborhood of m.pos do
3:   loc ← find_loc(p)
4:   Env[loc].bacteria += m.bacteria/9.0f
5: end for
6: m.state = DEAD
7: return
```

---

#### 4.2.4. T-Cell Update

The T-Cell update is much simpler compared to macrophages. In the current model, T-Cells have just two states, namely, LIVE, and DEAD. The T-Cell update kernel increments the age of the kernel and calls the motion sub-kernel. The motion sub-kernel for the T-Cells is the same as that of macrophages.

---

**Algorithm 9** Updating T-Cells

---

```
1: procedure update T-Cells() FOR all t ∈ T in parallel
2: ID ← threadID
3: t.age ← t.age- $\Delta t$ 
4: if t.age < 0 then
5:   t.state = DEAD
6:   return
7: end if
8: update_position(ID,t.speed,t.pos,Tcmap)
```

---

### 4.2.5. Handling Motion

Both T-Cells and Macrophages move about in the environment. While T-Cells have a single speed, macrophage speed changes based on the state. Additionally, the motion is constrained by the fact that no two cells of the same type can be co-located at the same grid location. Algorithm.. illustrates motion handling. The input arguments include speed, the position  $p$ , and the collision map array. The arrays  $Deltap$  contain possible motion vectors. The variable "loc" is an index into the chemokine and collision map array corresponding to the location of the agent. Lines 5-7 select a random motion direction if the chemokine level at the given location is less than the threshold  $E_{thresh}$ . Lines 10,11 generates the possible new location and index. If the new index is the same as the old, only the position is updated. If the new index already has an agent that the motion is rejected. Lines 19-22 handle the conflict if more than one agent try to move to the same location. We use the atomic compare and swap functionality to accomplish collision resolution. The variable

---

#### Algorithm 10 Handling Agent Motion

---

```

1: procedure update-position(ID,speed,p,collisionMap)
2: loc ← find_loc(p)
3:  $\Delta p[]$  ← array of unit vectors to moore neighborhood
4:  $e \leftarrow E[loc]$ 
5: if  $e < E_{thresh}$  then
6:    $k \leftarrow \text{rand}[0,8]$ 
7: else
8:    $k \leftarrow$  direction of maximum chemokine gradient
9: end if
10:  $p_{new} \leftarrow p + \Delta p[k]*\text{speed}$ 
11: new_loc ← find_loc( $p_{new}$ )
12: if collisionMap[loc] == OCCUPIED then
13:   return
14: end if
15: if loc == old_loc then
16:    $p \leftarrow p_{new}$ 
17:   return
18: end if
19: if atomicCAS(collisionMap[loc],EMPTY,ID) ==
    EMPTY then
20:   collisionMap[old_loc]← EMPTY
21:    $p \leftarrow p_{new}$ 
22: end if

```

---

### 4.2.6. Cell Recruitment

Cell recruitment involves dynamic memory allocation during run-time. However, memory allocation, typically viewed as a serial operation is not possible on the GPU. It is also not efficient to do allocation on the CPU and transfer the updated agent state array to the GPU every time step. We have developed a new algorithm to handle memory allocation based on

sorting. The agent state array is sorted based on whether the agent is alive or dead (Algorithm 11,12). This groups live and dead agents together in the array. Next, the number of live agents are counted  $n$  using parallel pre-fix sum. The sorted agent array and the total number of live agents are inputs to the kernel that actually handles agent recruitment

---

#### Algorithm 11 Prepare Agent State Array for Cell Recruitment

---

```

1: procedure prepare-agent-state-array(M)
2:  $M \leftarrow \text{sort}(M,\text{agent\_state})$ 
3:  $n \leftarrow$  count live agents in M using parallel pre-fix sum
4: return

```

---

For every environment compartment which is source compartment, based on the chemokine level, and the occupancy of the collision map, a new cell is created in the agent state array at an offset given by  $N-n+idx+1$ , where  $N$  is the size of the agent state array, and  $idx$  is index of the source compartment. Note that this algorithm requires that the number of empty slots in the agent state array be less than the number of source compartments for successful allocation. However, we have found that due to the stochastic nature of the simulation, this limitation does not effect the end result.

---

#### Algorithm 12 Recruit Cells

---

```

1: procedure recruit-cells(M,n)
2:
3: for all  $e \in \text{Env}$  in parallel do
4:   pos ← grid location of  $e$ 
5:   loc ← find_loc(pos)
6:   idx ← Env[loc].sourceIndex
7:   chm ← E[loc]
8:   if idx AND chm >  $C_{Thresh}$  AND collisionMap[loc] ==
     EMPTY AND  $M[N - n + idx + 1].\text{state} == \text{DEAD}$  then
9:      $M[N - n + idx + 1] \leftarrow$  new agent at pos
10:    collisionMap[loc] == OCCUPIED
11:   end if
12: end for
13: return

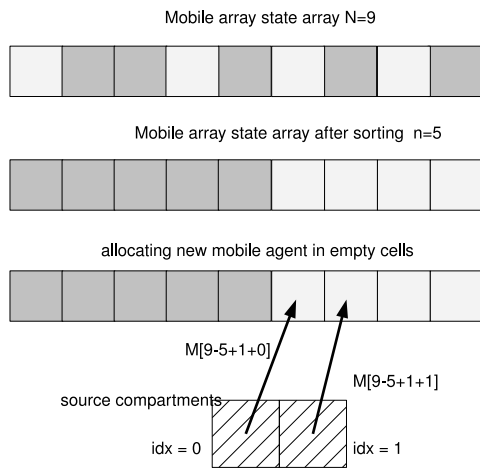
```

---

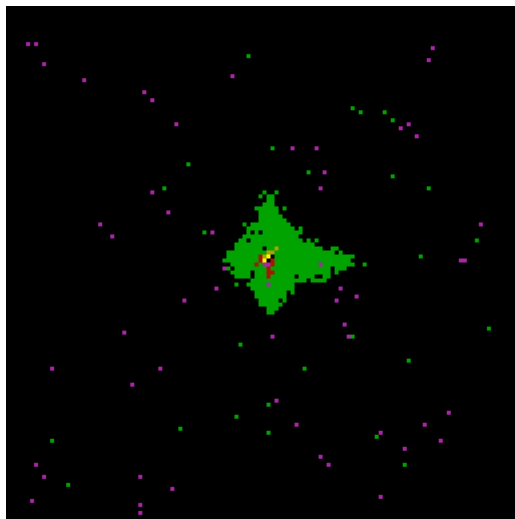
Figure illustrates the process. In this agent state array, the maximum capacity is  $N=9$ . The number of live agents is  $n=5$ . The sorting routine places live agents at the front and the dead agents at the rear end of the array. The allocation kernel given in Algorithm 5 then places the new cells in the empty agent state array cells. In this illustration, there are only two source compartments.

## 5. RESULTS

The algorithms described above were implemented using CUDA on a GeForce 8800M GTX GPU. This processor has

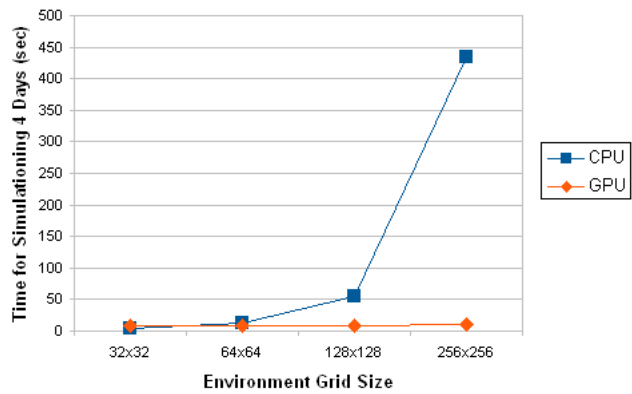


**Figure 2.** Parallel memory allocation for cell recruitment

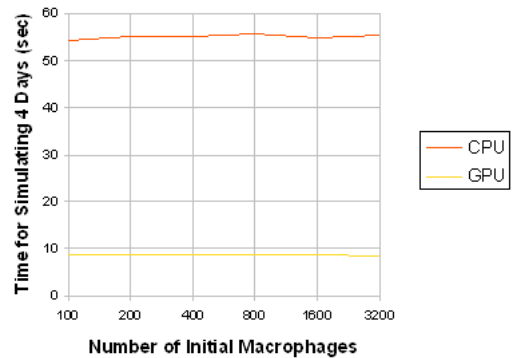


**Figure 3.** Screen shot of simulation. In this figure, green pixels represent resting macrophages, orange pixels represent infected macrophages, red pixels represent chronically infected macrophages, and purple pixels represent T-Cells. All these cells are chemotactically moving towards the center where infected macrophages are releasing chemokines.

96 core and has 360 GFlops processing rate. The GPU implementation was benchmarked against a CPU implementation running on a Intel Centrino processor. For the purpose of the benchmarks, the simulation was run to account for 4 days of real time with T-Cells arriving at the beginning of the second day. Figure 1 show the performance scaling with respect to the increase in the environment size. It is clear that time for simulation for the GPU implementation barely changes. At a environment resolution of 256x256, the GPU implementation is .. faster than the CPU implementation. Figure 2 shows



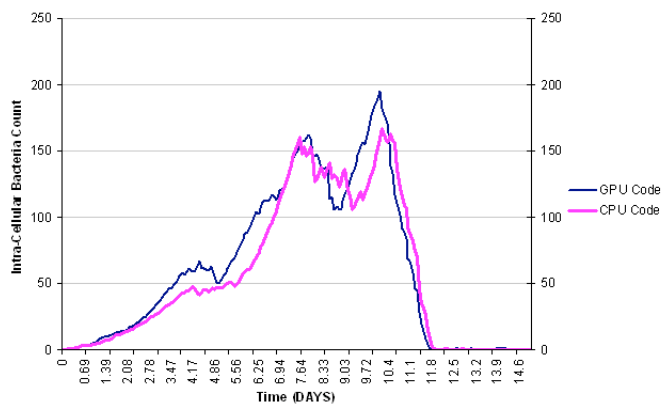
**Figure 4.** Performance scaling with environment size. The initial macrophage population is set to 100. The simulation is run for 4 days worth of updates with T-Cells being recruited on the 2<sup>nd</sup> day



**Figure 5.** Performance scaling with initial macrophage population. The environment size is set to 128x128

the scaling with the number of initial macrophages. The environment resolution for this benchmark was set to 100x100. It is apparent that the performance for both the CPU and GPU does not degrade much in the ranges that we tested. Here too, the GPU performance is much better. This clearly indicates that most of the computational load is due to the calculation of the diffusion decay equation in the environment. Our implementation on the GPU has two advantages. First, the multi-core GPU provides much better computing power as compared to the CPU. Second, and more importantly, the use of spectral methods(FFT) for solving the diffusion decay equation provides an enhanced speed up due to the size of the diffusion decay kernel. Figure shows the comparison between the output of the two implementations for the same initial parameters. Here we track the amount of intra-cellular bacteria over time. The two curves were generated by averaging the results of 5 simulation runs. Clearly, the output of both simulation show the same trend.





**Figure 6.** Tracking intra-cellular bacteria with respect to time for the CPU implementation and the GPU implementation. The simulation is run for 15 days with T-Cells being recruited on the 10<sup>th</sup> day. The environment is set to 128x128 with initial macrophage population set to 100.

## 6. CONCLUSIONS

We have successfully developed and implemented a series of data-parallel algorithms for simulating the *Mycobacterium Tuberculosis* ABM. Key innovation that have made this possible include the collision handling algorithm based on atomic compare and swap, and the parallel memory allocation algorithm based on sorting. Our implementation executes entirely on the GPU with no communication with the CPU and therefore avoids the bottleneck posed by the slower CPU-GPU memory bus. Our current simulation only models the dynamics of *MTb* in the lung. The performance gains obtained through data-parallel execution opens up the possibility of scaling ABMs to realistic levels. We are working on building an extended model that will include other organs such as lymphnodes that are involved in the pathology. Our ultimate goal is to build the complete organs system model with each organ modeled as a separate ABM with interconnects that will model the lymphatic and circulatory system. This model will then be used to virtually test drug protocols.

## Biography

## REFERENCES

- [1] Y Vodovotz, M Csete, J Bartels, S Chang, and G An. Translational systems biology of inflammation. *PLoS Computational Biology*, 4(4):1–6, 2008.
- [2] Bernt Oksendahl. *Stochastic differential equations: an introduction with applications*. Springer - Verlag, 2002.
- [3] D T Gillespie. General method for numerically simulating the stochastic time evolution of coupled chemical reactions. *J. Comput. Phys.*, 22:403–434, 1976.
- [4] D T Gillespie. Exact stochastic simulation of coupled chemical reactions. *J. Phy. Chem.*, 81:2340–2361, 1977.
- [5] Quan Liu and Ya Jia. Fluctuations-induced switch in the gene transcriptional regulatory system. *Phys. Rev. E*, 70(4):041907, Oct 2004.
- [6] Mukund Thattai and Alexander van Oudenaarden. Stochastic gene expression in fluctuating environments. *Genetics*, 167:523–530, 2004.
- [7] G An. Agent-based computer simulation and sirs: building a bridge between basic science and clinical trials. *Shock*, 16(4):266–273, 2001.
- [8] E Boanbeau. Agent-based modeling: methods and techniques to simulate human systems. *PNAS*, 99(3):7280–7287, 2002.
- [9] A M Bailey, B C Thorne, and S M Peirce. Multi-cell agent-based simulation of the microvasculature to study the dynamics of circulating inflammatory cell trafficking. *Ann. Biomed. Engg.*, 35(6):916–936, 2007.
- [10] J Tang and et al. Simulating leukocyte-venule interactions—a novel agent-oriented approach. In *Conf. Proc. IEEE Eng. Med. Biol. Soc.*, volume 7, pages 4978–4981, 2004.
- [11] L Zhang, C A Athale, and T S Deisboeck. Development of a three-dimensional multiscale agent-based tumor model: simulating gene-protein interaction profiles, cell phenotypes and multicellular patterns in brain cancer. *J. Theor. Biology*, 244(1):96–107, 2007.
- [12] T S Deisboeck and et al. Pattern of self-organization in tumour systems: complex growth dynamics in a novel brain tumour spheroid model. *Cell Prolif.*, 34(2):115–134, 2001.
- [13] D Y Wong, A Qutub, and C A Hunt. Modeling transport kinetics with starlogo. In *Conf. Proc. IEEE Eng. Med. Biol. Soc.*, volume 2, pages 845–848, 2004.
- [14] G Broderick, M Ruaini, E Chan, and M J Ellison. A life-like virtual cell membrane using discrete automata. *In Silico Biol.*, 5(2):163–178, 2005.
- [15] M Pogson, R Smallwood, E Owarnstrom, and M Holcombe. Formal agent-based modelling of intracellular chemical interactions. *Biosystems*, 85(1):37–45, 2006.
- [16] D C Walker. Agent-based computational modeling of wounded epithelial cell monolayers. *IEEE Trans. Nanobioscience*, 3(3):153–163, 2004.

- [17] Q Mi, B Riviere, G Clermont, D L Steed, and Y Vodovotz. Agent-based model of inflammation and wound healing: insights into diabetic foot ulcer pathology and the role of transforming growth factor-beta1. *Wound Repair Regen.*, 15(5):671–682, 2007.
- [18] D Longo, S M Peirce, T C Skalak, L Davidson, M Marsden, B Dzamba, and D W DeSimone. Multicellular computer simulation of morphogenesis: blastocoel roof thinning and matrix assembly in *xenopus laevis*. *Dev. Biol.*, 271(1):210–222, 2004.
- [19] M R Grant, S H Kim, and C A Hunt. Simulating in vitro epithelial morphogenesis in multiple environments. In *Comput. Syst. Bioinformatics Conf.*, pages 381–384, 2006.
- [20] S Chen, S Ganguli, and C A Hunt. An agent-based computational approach for representing aspects of in vitro multi-cellular tumor spheroid growth. In *Conf. Proc. IEEE Eng. Med. Biol. Soc.*, volume 1, pages 691–694, 2004.
- [21] S M Peirce, E J van Gieson, and T C Skalak. Multicellular simulation predicts microvascular patterning and in-silico tissue assembly. *FASEB J.*, 18(6):731–733, 2004.
- [22] C A Hunt, G E P Ropella, L Yan, D Y Hung, and M S Roberts. Physiologically based synthetic models of hepatic disposition. *J. Pharmacokinetics and Pharmacodynamics*, 33(6):737–772, 2006.
- [23] L X Garmire, D G Garmire, and C A Hunt. An in silico transwell device for the study of drug transport and drug-drug interactions. *Pharm. Res.*, 24(12):2171–2186, 2007.
- [24] T N Lam and C A Hunt. Applying models of targeted drug delivery to gene delivery. In *Proc. IEEE Eng. Med. Biol. Soc.*, number 5, pages 3535–3538, 2004.
- [25] Y Li and C A Hunt. Studies of intestinal drug transport using an in silico epithelio-mimetic device. *Biosystems*, 82(2):154–167, 2005.
- [26] J L Segovia-Juarez, S Ganguli, and D. Kirschner. Identifying control mechanisms of granuloma formation during m. tuberculosis infection using an agent-based model. *J. Theor. Biol.*, 231(3):357–376, 2004.
- [27] G An. In silico experiments of existing and hypothetical cytokine-directed clinical trials using agent-based modeling. *Critc. Care Med.*, 32(10):2050–2060, 2004.
- [28] H K Khalil. *Nonlinear systems*. Prentice Hall, 2002.
- [29] M J North, N T Collier, and J R Vos. Experiences in creating three implementations of the repast agent modeling toolkit. *ACM Transactions on Modeling and Computer Simulations*, 16(1):1–25, 2006.
- [30] R Burkhart. The swarm multi-agent simulation system. In *OOPSLA '94 Workshop on "The Object Engine"*, 1994.
- [31] S Luke, C-R Claudio, K Sullivan, and G Balan. Mason: A multiagent simulation environment. *Simulation*, 81(7):517–527, 2005.
- [32] B P Zeigler, H Praehofer, and T G Kim. *Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems*. Academic Press, 2000.
- [33] U J Kapasi, S Rixner, W J Dally, B Khailany, J H Ahn, P Matson, and J Owens. Programmable stream processors. *IEEE Computer*, pages 54–62, 2003.
- [34] M Harris, G. Coombe, T Scgeuermann, and A Lastra. Physically-based visual simulation on graphics hardware. In *Proceedings Eurographics/SIGGRAPH Workshop on Graphics Hardware*, 2002.
- [35] A Kolb, L Latta, and C Rezk-Salama. Hardware-based simulation and collision detection for large particle systems. In *Proceedings of the ACM 2004 Graphics Hardware Conference*, Grenoble, France, 2004.
- [36] E Millán and I Rudomín. Impostors and pseudo-instancing for gpu crowd rendering. In *GRAPHITE '06: Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia*, pages 49–55, New York, NY, USA, 2006.
- [37] K Perumalla and B Aaby. Data parallel execution challenges and runtime performance of agent simulations on the gpu. In *Proceedings of the 2008 ACM Spring simulation multi conference*. ACM, 2008.
- [38] P Richmond and R Daniela. A high performance framework for agent based pedestrian dynamics on gpu hardware. In *Proceedings of EUROESIS ESM 2008*, Université du Havre, Le Havre, France, 2008.
- [39] R M D'Souza, M Lysenko, and K Rahmani. Sugarscape on steroids: simulating over a million agents at interactive rates. In *Proceedings of the Agent2007 Conference*, 2007.