Long time-scale simulations of *in vivo* diffusion using GPU hardware

Elijah Roberts[1], John E. Stone[2], Leonardo Sepúlveda[1], Wen-Mei W. Hwu[3] and Zaida Luthey-Schulten[4]

[1]Center for Biophysics and Computational Biology
University of Illinois
Urbana, IL, USA
{erobert3,lsepulv2}@illinois.edu

[2]Beckman Institute
University of Illinois
Urbana, IL, USA
johns@ks.uiuc.edu

[3]Department of Electrical and Computer Engineering
University of Illinois
Urbana, IL, USA
w-hwu@illinois.edu

[4]Department of Chemistry
University of Illinois, Urbana, IL, USA
zan@illinois.edu

Abstract

To address the problem of performing long time simulations of biochemical pathways under in vivo *cellular conditions, we have developed a lattice-based, reaction-diffusion model that uses the graphics processing unit (GPU) as a computational co-processor. The method has been specifically designed from the beginning to take advantage of the GPU's capacity to perform massively parallel calculations by not only executing a core set of mathematical calculations, but also running much of the underlying algorithmic logic on the GPU. In this study we present our three-dimensional model for in vivo diffusion that exploits the calculation capabilities of the GPU. The implementation of the diffusion operator on the GPU is subject to architectural constraints, and we discuss its structure and the trade-offs made to accommodate the GPU hardware.*

1. Background and Significance

The cell is a crowded space [5, 18] with proteins, nucleic acids, and other macromolecules constantly in contact with and colliding into each other. In the midst of this chaotic and turbulent scene, extensive and intricate networks of biochemical reactions operate, in many cases, by random Brownian diffusion of one molecular species to its reaction counterpart. Often the concentration of one or both of the reactants is as low as a few molecules per cell, resulting in stochastic dynamics that depend upon a molecule's initial position. Additionally, the cell is not a homogeneous mixture of macromolecules, some are localized to specific sub-volumes within the cell and their localization has a dramatic effect on biochemical networks in which they participate. Computational models of cellular biochemical networks that are spatially resolved can therefore be useful when testing hypotheses developed from single molecule experimental results of a network's activity, looking for unexpected, emergent behavior in a network's temporal and spatial dynamics, and comparatively investigating the parameter space explored during a network's evolution.

One technique that can yield data on the *in vivo* positioning of macromolecules is cryoelectron tomography, which has been used to localize ribosomes in intact cells with $\sim$5 nm resolution [10]. It is anticipated that by combining results from these types of studies the global distribution of large macromolecules within the cell can be reasonably approximated for various parts of the cell cycle.

A key issue in modeling whole-cell biochemical networks is accounting for the cellular environment. Inside a cell, approximately 20-30% of the volume is occupied by macromolecules, the diffusion coefficients of which are reduced 3 to 15 fold relative to their *in vitro* values [8, 4]. Additionally, diffusive behavior that does not obey the standard relation between mean square displacement (MSD) and time has been observed in living cells. Specifically, anomalous subdiffusion has been seen in both experimental [16, 1, 6] and theoretical studies [3, 13, 12], although its exact extent (and origin) inside living cells is still debated.

We have derived a new cellular automata (CA) [14, 17, 2]

This work was supported by grants from the Department of Energy (DE-FG02-05ER-64144) and National Science Foundation (EF-0526747,MCB-0844670,PHY-0822613). Computational resources were provided by the CUDA Center of Excellence, UIUC.

based method that utilizes the graphics processing unit (GPU) to perform long time-scale simulations of whole-cell reaction-diffusion models under *in vivo* conditions. CA methods have long been used in statistical physics and computational chemistry and ours is a derivative of a multiparticle model [7]. Being a lattice model, computational complexity scales with the number of lattice sites (independent of the number of particles located on the lattice) and so can reach long time-scales under crowded conditions.

CA models are characterized by three properties: space and time are discrete, physical quantities are described by a finite set of values, and the time evolution of the system is governed by a rule using only local information. These properties make them theoretically well-suited toward GPU implementation, since calculations use integer math (no dependency on GPU floating point precision) and the entire lattice can be updated in parallel using only small amounts of local memory. Additionally, CA models are highly parallelizable, most of the code can run on the GPU.

In this work we present the techniques and strategies that enabled simulations of *in vivo* diffusion on GPU hardware. We first introduce the multiparticle model and the adaptations made to it to permit efficient GPU simulation. We then discuss the implementation details of the method and the programmatic trade-offs made to accommodate GPU architectural limitations. Finally, we present analyses of free and obstructed diffusion simulations along with timing results from whole-cell *in vivo* diffusion simulations. Performance analyses are presented and compared for two GPU models, a G80-based FX5600 and a GT200-based GTX280.

2 Methods

2.1 Multiparticle diffusion model

In the multiparticle diffusion model of Karapiperis and Blankleider [7], particles follow independent random walks between lattice sites in a stochastic manner and are characterized by permitting multiple particles per lattice site. We have modified the model to support efficient implementation on a GPU. The model is constructed on a cubic lattice ($\mathcal{L}$) with uniform spacing in the x, y, and z dimensions with distance λ. Lattice sites are located on the lattice at positions $\vec{r} = a\lambda\hat{\imath} + b\lambda\hat{\jmath} + c\lambda\hat{k}$, where a, b, and c are integers. Particles of various species (α) are positioned at the lattice sites according to some initial condition at time $t = 0$ and then move on the lattice from site to site according to the rules of the model. Time is also discrete in the model and particle movement occurs instantaneously at time steps separated by time τ.

At time t, a site at position $\vec{r}$ on the lattice is described by its occupancy, $N_\alpha(\vec{r}, t)$, giving the number of particles of species α located at the site. A diffusion operator ($\mathcal{D}$) updates a lattice site at a time step by moving particles to and from the site according to a model of their diffusive behavior. The state of a lattice site after a single time step is therefore given by

$$N_\alpha(\vec{r}, t + \tau) = \mathcal{D}N_\alpha(\vec{r}, t). \tag{1}$$

Normal Brownian diffusion can be phenomenologically modeled as a series of independent random choices for the movement of particles in a system. In previous models [7, 2], particles could move only in a single dimension during a time step, *i.e.*, a particle moved $\pm\lambda\hat{\imath}$, $\pm\lambda\hat{\jmath}$, or $\pm\lambda\hat{k}$. However, implementing such a model requires access to the entire three-dimensional neighborhood surrounding each lattice site during a time step. As will be shown later (Section 2.2), this requirement severely limits the computational performance of the model on a GPU. We instead model diffusion as three independent random choices (one for each dimension) during a single time step. Decomposing the problem in such a manner reduces the size of the neighborhood that must be searched to perform the diffusion calculation, while maintaining the exact diffusion solution under appropriate conditions.

At each time step in our multiparticle diffusion model, for each dimension, a particle has a probability of moving one lattice site in the negative direction, a probability of staying at the current lattice site, and a probability of moving one lattice site in the positive direction. For the x, y, and z dimensions, these probabilities are (p_{-1}, p_0, p_1), (q_{-1}, q_0, q_1), and (s_{-1}, s_0, s_1), respectively. Since a particle must make a single choice in each dimension, $p_{-1} + p_0 + p_1 = q_{-1} + q_0 + q_1 = s_{-1} + s_0 + s_1 = 1$.

The diffusion operator applied to a lattice site calculates the random movement choices for each nearby particle and then updates the occupancy of the site to be the sum of all particles that remain at the site and those that enter from neighboring sites,

$$\mathcal{D}N_\alpha(\vec{r}, t) = \sum_{a=-1}^{1}\sum_{b=-1}^{1}\sum_{c=-1}^{1} p_{-a}q_{-b}s_{-c}N_\alpha(\vec{r} + \vec{dr}, t), \tag{2}$$

where $\vec{dr} = a\lambda\hat{\imath} + b\lambda\hat{\jmath} + c\lambda\hat{k}$. The time evolution of the entire lattice is realized by the simultaneous application of the diffusion operator on each lattice site. If there is no net probability of moving in any dimension ($p_{-1} = p_1$, $q_{-1} = q_1$, $s_{-1} = s_1$) then there is no advection and, also assuming isotropy ($p_{-1} = q_{-1} = s_{-1}$, $p_0 = q_0 = s_0$, $p_1 = q_1 = s_1$), the model obeys the standard diffusion equation [2],

$$\frac{\partial}{\partial t}N_\alpha(\vec{r}, t) = D\nabla^2 N_\alpha(\vec{r}, t),$$

where the diffusion coefficient for particles of species α is given by $D = \frac{\lambda^2}{2\tau}(1 - p_0)$.

2.2 Multiparticle GPU implementation

In-memory lattice representation. The lattice is the central construct of the multiparticle model. Particles are located at uniformly spaced sites on the lattice and move from site to site according to the rules of the model. The simplest memory representation of the lattice consists of an array of memory locations, each of which stores the state of a site.

During a simulation, the lattice is kept in GPU global (device) memory. Lattice sites are arranged contiguously in a single block of memory and the size of each site is a single 32-bit word, in order to make optimal use of GPU memory bandwidth. The sites are ordered within the memory block according to the lattice index ($\mathcal{L}_{index} = x + y{\cdot}x_n + z{\cdot}x_n{\cdot}y_n$), as shown in Figure 1 (a)

The state of a lattice site is given by the list of particles present at the site and the type of the site; it must be fully described by a single 32-bit word. To fit all of the required information about a site into the available space, the 32-bit word is packed with fields of various bit lengths, Figure 1 (b-d). Four bits are allocated to the site type, which enables the differentiation of up to sixteen different types of lattice sites. The remaining 28 bits are divided equally into fields storing the chemical identities of the particles at the site. Because all of the particle fields must fit into 28 bits, the number of different chemical species that can be discerned decreases as the maximum number of particles that can be present at a site (n_{max}) increases, according to $\alpha = 2^{\lfloor 28/n_{max} \rfloor} - 1$. The value of zero is reserved to indicate no particle present. Three lattice implementations with a maximum of two, four, or eight particles per site are implemented. They allow the use of 16383, 127, or 7 unique chemical species, respectively.

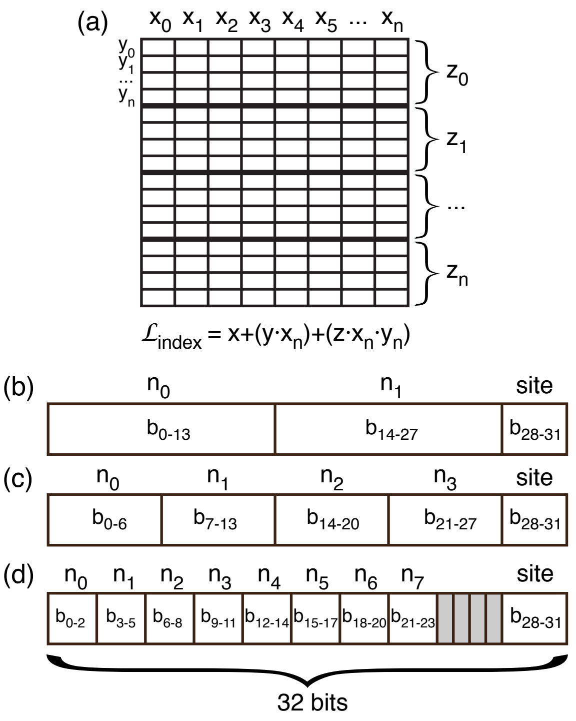
(a)
x_0 x_1 x_2 x_3 x_4 x_5 ... x_n
y_0
y_1
...
y_n
z_0
z_1
...
z_n
$\mathcal{L}_{index} = x+(y{\cdot}x_n)+(z{\cdot}x_n{\cdot}y_n)$
(b)
n_0 n_1 site
b_{0-13} b_{14-27} b_{28-31}
(c)
n_0 n_1 n_2 n_3 site
b_{0-6} b_{7-13} b_{14-20} b_{21-27} b_{28-31}
(d)
n_0 n_1 n_2 n_3 n_4 n_5 n_6 n_7 site
b_{0-2} b_{3-5} b_{6-8} b_{9-11} b_{12-14} b_{15-17} b_{18-20} b_{21-23} b_{28-31}
32 bits

Figure 1. (a) The layout of a three-dimensional lattice of size $x_n{\times}y_n{\times}z_n$ in memory. (b-d) The bit layout of a lattice site with a maximum of two, four, and eight particles per site, respectively. For each particle (n_i), the bits of the lattice site (b_{i-j}) that are used to store the particle's chemical species are shown along with the bits used to store the site type.

Processing strategy. The time evolution of the lattice under the multiparticle diffusion model is given by the diffusion operator as shown in Equations (1) and (2). The diffusion kernel[1] implements the operator by reading a lattice from global memory, calculating the position of each particle at the next time step according to the diffusion model, and then writing the new lattice back into global memory. Its general structure is as follows (see Table 1 for a timing profile):

1. Load a block of lattice sites from a lattice in global memory into shared memory.
2. Generate a random value for each particle's movement.
3. Choose whether each particle should move to a neighboring site or remain in place according to the probabilities associated with the particle and site types. Store the choice for each particle in shared memory.
4. Make a list of the particles for each site that were selected to either move into the site from a neighboring site or to remain at the current site.
5. Store the list of particles at each site into a new lattice in global memory.

Since the algorithm runs in parallel on the GPU, the original lattice can not be modified until after the entire calculation has been completed. As such, the algorithm requires two separate copies of the lattice in global memory. The maximum amount of memory that can be used by a lattice is limited to one-half of the total free GPU memory.

Shared memory constraints. The first step of the diffusion algorithm loads a block of the lattice into shared (on-chip) memory. From shared memory, the lattice sites can

[1]Code destined to be executed on the GPU is organized into execution units called kernels. A kernel is compiled from C source code by the Compute Unified Device Architecture (CUDA) compiler into set of device specific instructions. When invoked, a kernel is downloaded to the GPU and executed in parallel using a large number of threads. Threads are organized into thread blocks, in which each thread in a block has access to a common shared memory space.

Table 1. Multiparticle diffusion kernel calculation profile for a $256 \times 256 \times 256$ lattice simulation

Calculation	FX5600			GTX280			
	Time (ms)	%	Performance[†]	Time (ms)	%	Performance[†]	Speedup
Load lattice block	5.2	20	13 GB/s	2.2	16	30 GB/s	2.4X
Random number generation[‡]	7.0	27	48 GOPS	3.8	28	88 GOPS	1.8X
Particle movement decision	7.7	29	109 GOPS	4.4	33	191 GOPS	1.8X
Particle propagation	3.6	13	94 GOPS	1.6	12	209 GOPS	2.2X
Store lattice block	2.9	11	23 GB/s	1.5	11	44 GB/s	1.9X
Total	26.4	100		13.5	100		1.9X

[†]Bandwidth rates were calculated as four bytes times the number of lattice sites being transferred divided by the runtime. Operation rates were calculated as the number of logical operations per site times the number of lattice sites divided by the runtime.
[‡]Operation count was calculated using 64-bit operations, but current hardware implements 64-bit operations using 32-bit instructions. Performance calculated using the 32-bit instruction count (88) yields 210 GIPS and 387 GIPS, respectively.

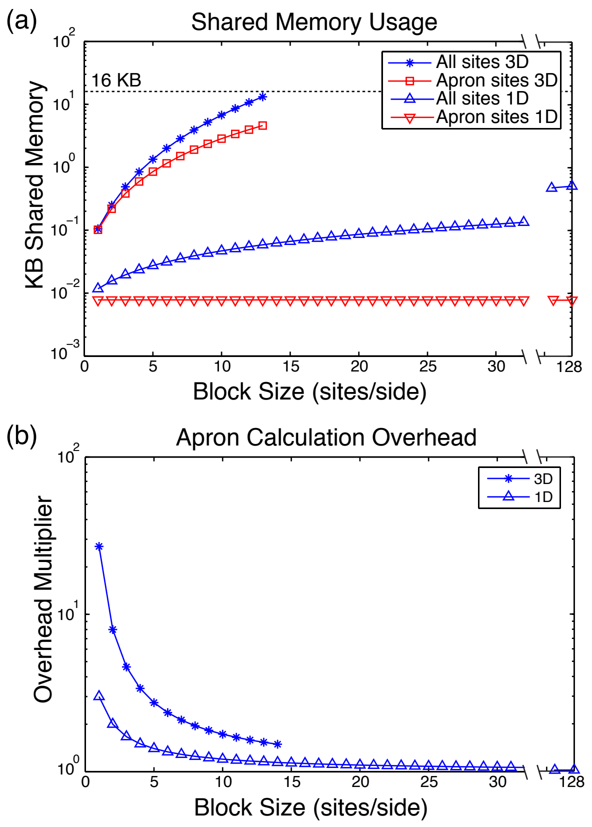

Figure 2. (a) Shared memory usage for the diffusion operator as a function of the size of the block being processed. The memory used for (blue) all sites and (red) apron sites are shown for comparison. (b) The calculation overhead due to the apron sites.

be processed by a thread block (one thread per lattice site) without the latency and bandwidth limitations associated with global memory. Significantly, not only are the lattice sites being processed loaded into shared memory, but also an apron of sites surrounding them. These additional sites are required because particles may enter a lattice site from *any* of its nearest neighbors. Even though an apron site will not be stored into the final lattice by the thread block, the movement of its particles must still be calculated so that it can be determined which of the particles (if any) move into a lattice site that *is* being processed by the thread block.

When a small three-dimensional block of lattice sites is being processed the apron sites account for a large fraction of the total sites. The amount of shared memory required to process a cubic block of lattice sites of dimensions $B \times B \times B$ is $4 \cdot (B + 2)^3$ bytes with the memory used for the apron sites being $4 \cdot ((B+2)^3 - B^3) = \mathcal{O}(B^2)$ bytes (Figure 2 (a)). The total available shared memory on current generation GPUs is 16 KB per multiprocessor (minus a small amount of overhead). The largest block that can be loaded has thirteen lattice sites per side ($4 \cdot (13 + 2)^3 = 13500$). In order for the GPU to efficiently process thread blocks, though, the actual amount of shared memory dedicated to a single thread block must only be a fraction of the total. For a block size of eight sites, the shared memory required is 3.9 KB (still a somewhat high allocation), of which roughly half is needed for apron sites.

However, shared memory usage is not the only overhead associated with the apron sites. Since apron sites are only processed by a thread block to find the particles moving from them, the sites must also be processed again to determine the particles moving to them (when the site is processed as part of another block). The diffusion operator must be calculated twice for these sites. Figure 2 (b) shows the increase in the number of diffusion calculations required as a result of the apron calculations. The number of calculations that must be performed for a block size of eight sites per side is twice the total number of lattice sites. Half of all the diffusion calculations would be redundant calculations required for the apron sites.

The combined effect of the shared memory and calculation efficiency requirements, makes calculating the diffusion operator computationally expensive for a three-dimensional block of lattice sites. Smaller blocks are less efficient to calculate (large surface area to volume ratio), but larger blocks cannot be processed because of shared memory limitations. One technique to alleviate this contradictory condition is to remove the square dependence of the number of apron sites on the block size. In Section 2.1, it was shown that the three-dimensional diffusion model can be equivalently expressed as a decomposition into each dimension independently. Implementing the diffusion operator as three successive diffusion calculations, one in each dimension, dramatically decreases the number of apron sites that must be loaded and redundantly calculated. Lattice sites are processed in a one-dimensional block of length B. The shared memory necessary to process a block is $4{\cdot}(B+2)$ and the memory used by the apron sites is $4{\cdot}((B+2)-B) = 8$ bytes (constant regardless of the block size). For a one-dimensional block size of 32 lattice sites and larger there is only minimal overhead associated with the apron sites (a 6% increase in shared memory usage and required calculations).

Random number generation. A large fraction ($\sim$25%) of the calculation time required for the multiparticle diffusion model is spent generating random numbers. For a time step, each particle requires three random numbers (one for each dimension) to realize its movement according to the site transition probabilities. To generate the random values, we use a combination of 64-bit random generators, as described in Press *et al.* [11]. Specifically, we use a linear congruential generator, followed by a 64-bit xorshift, and finally a pass through a multiplicative linear congruential generator.

Random number generation is constrained by the requirement that particles that fall into apron sites must have their diffusive motion calculated multiple times, each time returning the same result. To enforce the constraint the random value generated for each particle is a random hash based on a 128-bit value containing the site index, particle index, and time step. The combination guarantees a unique but reproducible random value for each particle at each site for each timestep. Additionally, a seed value must be specified to make each simulation a unique realization of the model, which is also incorporated into the calculation.

Any correlations produced by the random number generator would cause the simulation results to deviate from the true distribution of the underlying model. We have checked the generator for such using the "BigCrush" test suite from the TestU01 random number testing library [9]. The method passed all tests.

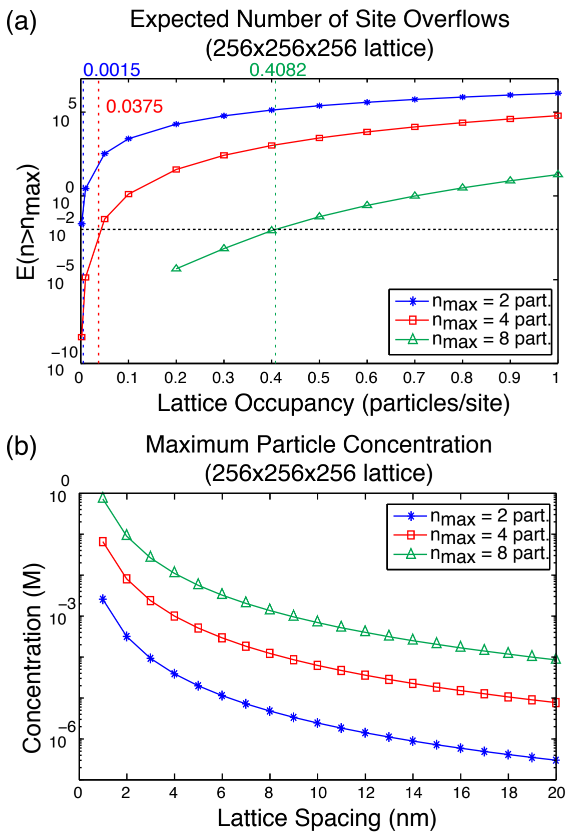

Figure 3. (a) The expected number of site overflows as a function of the lattice occupancy. Shown are the values for a lattice with a maximum of (blue) two, (red) four, and (green) eight particles per site (n_{max}). Also shown (black dotted) is the threshold of one overflow in one hundred configurations. (b) The maximum particle concentration (expected number of site overflows less than one in a hundred) as a function of the lattice spacing.

Site overflow. In the multiparticle model an unlimited number of particles can theoretically be located at a lattice site. In the implementation, though, there are a limited number of bits available for storing particles at each lattice site. Three separate kernels are implemented allowing a maximum of 2, 4, or 8 particles at each site. If, during an calculation, more than this number of particles are moved to a site the kernel must gracefully handle the overflow; particles cannot be lost. To avoid losing particles we use an overflow list, a list (stored in global memory) of all the sites that overflowed during a diffusion calculation. If the kernel detects that a lattice site has more particles than the maximum allowable, it stores the index of the lattice site in the overflow list along with the chemical identities of the extra particles. After each diffusion kernel execution, the extra particles from every site in the overflow list are randomly placed back into the lattice at a nearby site of the same type (done in CPU code). The overflow list prevents particle loss, but using it incurs a computational cost. To achieve optimal performance, simulation parameters should be chosen such that sites rarely overflow. The overflow list then becomes an exception mechanism to handle low frequency events.

In order to choose the appropriate parameters to avoid excessive sites overflows during a simulation, one must first know the chance of a site overflow occurring for the lattice configuration. As an estimation of this probability during a simulation, consider the process of adding N particles to an empty lattice L with L_s total sites. Assuming that all sites are equally probable, the probability of placing a particle at any given site is $\frac{1}{L_s}$. The probability of a site containing n particles after all N have been added ($p(n)$) is therefore the probability of placing n particles into the site during N independent choices. This probability is given by the binomial distribution,

$$p(n) = \binom{N}{n} \left(\frac{1}{L_s}\right)^n \left(1 - \frac{1}{L_s}\right)^{N-n}. \qquad (3)$$

Multiplying the probability for a single site by the total number of sites we get the expected number of lattice sites having exactly n particles, $E(n) = L_s p(n)$. Finally, the expected number of lattice sites that will overflow, *i.e.*, exceed the maximum number of particles that can be stored in a site (n_{max}), is

$$E(n > n_{max}) = L_s \left(1 - \sum_{i=0}^{n_{max}} p(i)\right). \qquad (4)$$

From Equations (3) and (4) it can be seen that the expected number of sites that will overflow depends on both the number of particles on the lattice and the lattice size. Although, for two lattices of different sizes the *probability* of a site overflowing is approximately equal if their *occupancy* (mean number of particles per site) is the same, there are more sites in a larger lattice that can overflow so the expected *number* of overflows is larger.

For a simulation to run as efficiently as possible, the number of particles on the lattice must be such that site overflows happens infrequently, perhaps once in every one hundred time steps. Approximating each timestep as an independent lattice configuration, the maximum number of particle per site and the lattice occupancy should be chosen such that the expected number of site overflows is ≤ 0.01. Figure 3 (a) shows the expected number of site overflows as a function of the lattice occupancy. For a $256{\times}256{\times}256$ lattice, the maximum allowable occupancy is 0.0015, 0.038, and 0.41 particles per site for $n_{max} = 2, 4$, and 8, respectively. In general, it is also useful to interpret the lattice occupancy as a concentration. The concentrations corresponding to the maximum occupancy (such that $E(n > n_{max}) \leq 0.01$) are shown in Figure 3 (b) as a function of lattice spacing. For a $256{\times}256{\times}256$ lattice with 1 nm spacing, the maximum particle concentration is around 2 mM for $n_{max} = 2$, 60 mM for $n_{max} = 4$, and 650 mM for $n_{max} = 8$. At smaller lattice spacing, care must also be taken so that the maximum concentration does not exceed what is physically realistic.

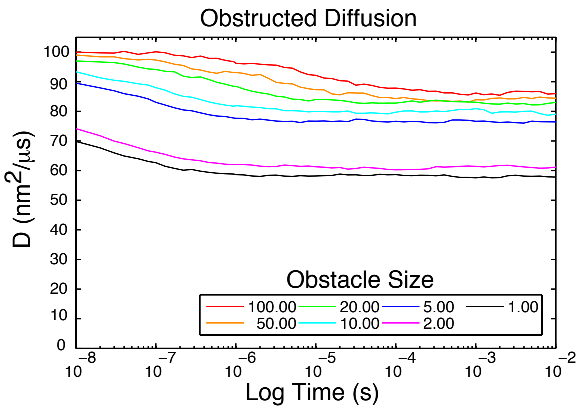

Figure 4. Observed diffusion coefficient (D) of particles with an *in vitro* D of 100 $\mathrm{nm}^2/\mu\mathrm{s}$ diffusing on a lattice populated with obstacles of radius from 100 nm to 1 nm (top curve to bottom curve) occupying 30% of the lattice sites by volume.

While the techniques described above minimize the overhead of overflow handling, they do not eliminate it. Control must still be returned to the calling program on the CPU after each kernel execution to check for overflow exceptions, incurring a $\sim$1 ms overhead for each kernel invocation. It may be possible to improve performance by implementing an entirely GPU based exception mechanism following the global GPU barrier technique introduced in [15]. In general, exception handling techniques are still an under-developed area of GPU programming.

3 Simulation results

Analysis of free and obstructed diffusion simulations. To validate the GPU implementation of the multiparticle model, we first assessed its characteristics when simulating freely diffusing particles on a periodic lattice. Under such conditions, the particle distributions should agree with the continuum model presented earlier. For a particle undergoing Brownian diffusion, the relationship between its mean square displacement (MSD) and the amount of time it has been freely diffusing is given by the well-known relation, in three dimensions, $< r^2 >= 6Dt$.

We performed 10 ms simulations of 16,380 particles with nine different diffusion coefficients (200, 100, 50, 25, 10, 5, 1, 0.1, and 0.01 $\mathrm{nm}^2/\mu\mathrm{s}$) freely diffusing on a periodic 128×128×128 lattice. The lattice's natural diffusion coefficient was the same in each simulation (200 $\mathrm{nm}^2/\mu\mathrm{s}$; 2 nm spacing, 10 ns time step) and only the transition probabilities varied ($p_{-1} = p_1 = 0.5, 0.25, 0.125, 0.0625, 0.025, 0.0125, 2.5{\cdot}10^{-3}, 2.5{\cdot}10^{-4}, 2.5{\cdot}10^{-5}$). The calculated and expected values for MSD and D agree over the entire course of each simulation.

To test our model's ability to reproduce anomalous subdiffusion in crowded environments, we constructed a periodic lattice with obstructions modeled as clusters of reflective lattice sites. Clusters were determined by mapping a sphere with the diameter of the obstruction onto the lattice and setting each site located within the sphere as reflective. Particles diffusing on the lattice have zero probability to transition to a reflective lattice site, and must diffuse around the obstacles. In this approximation lattice obstructions are stationary, a reasonable assumption for larger obstacles but which is less realistic as the diameter of an obstacle approaches the lattice spacing.

Simulations with obstacle sizes ranging in radius from 100 nm to 1 nm were simulated at three different obstructed volume fractions (10%, 20%, and 30% by volume). Analysis of the simulations shows that the multiparticle lattice model does exhibit anomalous subdiffusion in crowded systems (Figure 4). Like other computational models of crowded diffusion, it shows normal Brownian diffusion at short times, a cross-over period during which diffusion is anomalously subdiffusive, and finally a return to normal diffusion at long times. As the size of the obstruction increases, the crossover begins later in the simulation; larger obstacles will have greater mean spacing at a give volume fraction than smaller obstacles. For an obstructed volume of 30%, the crossover time with obstacles 10 nm in radius is in the microsecond range. This result is particularly relevant as 10 nm is approximately the radius of a ribosome, one of the most abundant large (compared to a protein) particles in the cell, occupying 8-10% by volume.

***In vivo* modeling.** The *in vivo* cytoplasm is more complex than can be modeled by obstacles of a single size. Ridgway *et al.* [12] used the available proteomic data to describe the cytoplasmic environment of an *E. coli* cell in terms of the populations of different size classes of particles. We used the same particle classes and populations to construct a lattice model of a stationary *in vivo* environment. Figure 5 shows the lattice representations of the particle classes along with illustrations of a periodic volume used for analyzing diffusion under the model and a full-size *E. coli* cell. We simulated diffusion of particles with various diffusion coefficients in the *in vivo* environment to test the effects of the stationary obstruction and lattice approximations. The *in vivo* diffusion of proteins is reduced by approximately 20% in the simulations (smaller decreases were observed for particles with lower diffusion coefficients). This is somewhat less than the 30% reduction seen in the Brownian dynamics models, where obstructions are mobile. However, we see the same crossover time scale, during which diffusion is anomalous (10^{-6} s).

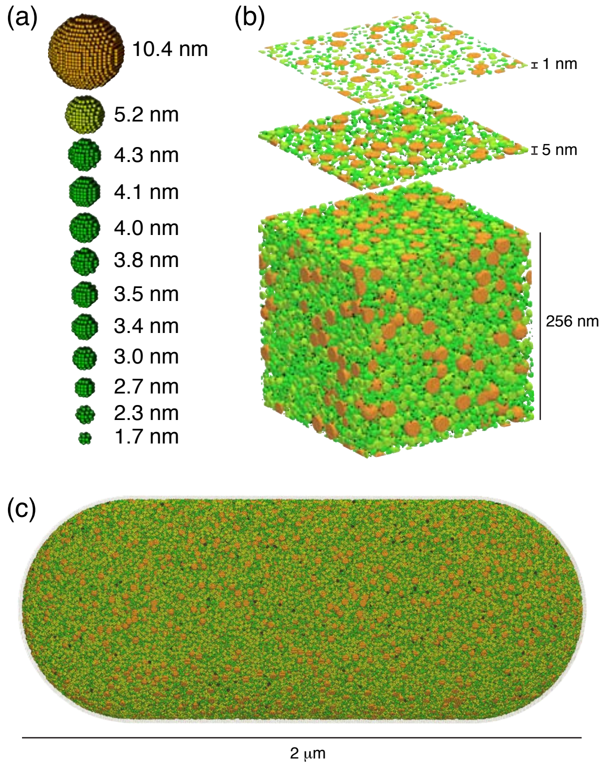

Figure 5. (a) Lattice representations of *in vivo* obstaclesith the indicated radius. (b) $256\ \mathrm{nm}{\times}256\ \mathrm{nm}{\times}256\ \mathrm{nm}$ **lattice model of an *in vivo* environment (30% obstructed volume). (c) Model of an *E. coli* cell (2 μm in length, 0.8 μm in diameter) using the same *in vivo* environment.**

Despite the approximations involved, the lattice model appears to capture the intrinsic nature of the effect of *in vivo* crowding on diffusion. Unlike Brownian dynamics models, where performance scales with the number of particles, the multiparticle diffusion model described here is invariant toward the number of obstacles. Its performance depends only on the total number of lattice sites. Using a single

GPU, *in vivo* simulations can extends well into the seconds time range (see Table 2).

4 Conclusions

Lattice-based cellular automata (CA) models are very amenable to GPU implementation. They require only local state and can be calculated in parallel over the entire lattice. We have presented what we believe to be the first example of a physical CA model designed specifically to run on the GPU, taking advantage of its unique performance characteristics. Development of the multiparticle diffusion model described in this work revealed certain general strategies that may be applicable toward GPU implementation of other CA or lattice models. While we achieved a reasonable fraction of the total theoretical performance possible (∼20-30%), we anticipate detailed analysis of bottlenecks and fine-tuning of the code will permit further speedup. Performance between two GPU models (FX5600 and GTX280) appears to indicate that the model will scale in speed across generations of GPU evolution. The computational potential of the GPU to perform long-time simulations of *in vivo* reaction-diffusion models has only begun to be utilized. Future work will allow simulations to span multiple GPUs on a single host and to utilize clusters of networked GPU compute nodes to achieve even larger simulation efficiencies. GPUs should be able to provide the computational power to, in the near future, simulate entire *E. coli* cells at mesoscopic resolutions for time scales on the order of a cell cycle.

Although the current study focused on GPUs from NVIDIA using the CUDA API, other existing and future many-core processors (such as Intel's Larrabee) are expected to share similar performance characteristics and architectural limitations. The OpenCL API, currently in the standardization process under the Khronos Group, may play a central role in bringing about a unifying framework allowing widespread use of GPUs for scientific computing.

Table 2. Performance of *in vivo* simulations

Lattice Size	Spac (nm)	Time Step (μs)	Calc Perf (10^6 sites/s)	Sim Perf (s/GPU·day)	Spd
			FX5600/GTX280	FX5600/GTX280	
$64^2\times128$	20	8.00	219/533	290/700	2.4X
$64^2\times128$	16	5.12	212/522	180/440	2.4X
$128^2\times256$	10	2.00	310/781	13/32	2.5X
$128^2\times256$	9	1.62	307/747	10/25	2.5X
$128^2\times256$	8	1.28	302/776	8.0/20	2.5X
$256^2\times512$	7	0.94	349/648	0.85/1.6	1.8X
$256^2\times512$	6	0.72	348/647	0.65/1.2	1.8X
$256^2\times512$	5	0.50	347/645	0.45/0.83	1.8X
$256^2\times512$	4	0.32	346/642	0.29/0.52	1.8X

References

[1] D. Banks and C. Fradin. Anomalous diffusion of proteins due to molecular crowding. *Biophys. J.*, 89(5):2960–2971, 2005.

[2] B. Chopard and M. Droz. *Cellular automata modeling of physical systems*. Cambridge University Press, Cambridge, UK, 1998.

[3] J. Dix, E. Hom, and A. Verkman. Fluorescence correlation spectroscopy simulations of photophysical phenomena and molecular interactions: a molecular dynamics/Monte Carlo approach. *J. Phys. Chem. B*, 110(4):1896, 2006.

[4] J. Dix and A. Verkman. Crowding effects on diffusion in solutions and cells. *Ann. Rev. Biophys.*, 37:247–263, 2008.

[5] R. J. Ellis. Macromolecular crowding: obvious but underappreciated. *Trends Biochem. Sci.*, 26(10):597–604, Oct 2001.

[6] I. Golding and E. Cox. Physical nature of bacterial cytoplasm. *Phys. Rev. Lett.*, 96(9):98102, 2006.

[7] T. Karapiperis and B. Blankleider. Cellular automaton model of reaction-transport processes. *Physica D*, 78(1-2):30–64, 1994.

[8] M. Konopka, I. Shkel, S. Cayley, M. Record, and J. Weisshaar. Crowding and confinement effects on protein diffusion in vivo. *J. Bacteriol.*, 188(17):6115–6123, 2006.

[9] P. L'Ecuyer and R. Simard. TestU01: A C library for empirical testing of random number generators. *ACM Trans. Math. Software*, 33(4):22, 2007.

[10] J. Ortiz, F. Förster, J. Kürner, A. Linaroudis, and W. Baumeister. Mapping 70S ribosomes in intact cells by cryoelectron tomography and pattern recognition. *J. Struct. Biol.*, 156(2):334–341, 2006.

[11] W. Press. *Numerical recipes: the art of scientific computing*. Cambridge University Press, 2007.

[12] D. Ridgway, G. Broderick, A. Lopez-Campistrous, M. Ru'aini, P. Winter, M. Hamilton, P. Boulanger, A. Kovalenko, and M. Ellison. Coarse-grained molecular simulation of diffusion and reaction kinetics in a crowded virtual cytoplasm. *Biophys. J.*, 17(5):493–498, 2008.

[13] M. Saxton. A biological interpretation of transient anomalous subdiffusion. I. Qualitative model. *Biophys. J.*, 92(4):1178, 2007.

[14] T. Toffoli and N. Margolus. *Cellular automata machines: a new environment for modeling*. The MIT Press, Cambridge, MA, 1987.

[15] V. Volkov and J. Demmel. Benchmarking GPUs to tune dense linear algebra. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. IEEE Press Piscataway, NJ, USA, 2008.

[16] M. Weiss, M. Elsner, F. Kartberg, and T. Nilsson. Anomalous subdiffusion is a measure for cytoplasmic crowding in living cells. *Biophys. J.*, 87(5):3518–3524, 2004.

[17] S. Wolfram. *Cellular automata and complexity: collected papers*. Westview Press, Boulder, CO, 1994.

[18] H. Zhou, G. Rivas, and A. Minton. Macromolecular crowding and confinement: biochemical, biophysical, and potential physiological consequences. *Ann. Rev. Biophys.*, 37(1), 2008.