Speeding Up Particle Trajectory Simulations under Moving Force Fields using GPUs (Accepted in JCISE 2012)

Robert Patro
Graduate Student
Inst. for Advanced Computer Studies
Department of Computer Science
University of Maryland
College Park, Maryland 20742
Email: rob@cs.umd.edu

John P. Dickerson
Research Assistant
Inst. for Advanced Computer Studies
University of Maryland
College Park, Maryland 20742
Email: johnd@umiacs.umd.edu

Sujal Bista
Graduate Student
Inst. for Advanced Computer Studies
Department of Computer Science
University of Maryland
College Park, Maryland 20742
Email: sujal@cs.umd.edu

Satyandra K. Gupta
Professor, Fellow of ASME
Inst. for Systems Research
Department of Mechanical Engineering
University of Maryland
College Park, Maryland 20742
Email: skgupta@umd.edu

Amitabh Varshney
Professor
Inst. for Advanced Computer Studies
Department of Computer Science
University of Maryland
College Park, Maryland 20742
Email: varshney@cs.umd.edu

In this paper, we introduce a GPU-based framework for simulating particle trajectories under both static and dynamic force fields. By exploiting the highly parallel nature of the problem and making efficient use of the available hardware, our simulator exhibits a significant speedup over its CPU-based analog. We apply our framework to a specific experimental simulation: the computation of trapping probabilities associated with micron-sized silica beads in optical trapping workbenches. When evaluating large numbers of trajectories (4096), we see approximately a 356 times speedup of the GPU-based simulator over its CPU-based counterpart.

1 Introduction

The simulation of particle movement under general force fields is of interest across many scientific disciplines. For example, biochemists may wish to model stochastic diffusion of glutamate receptors [1], while financial analysts may attempt to simulate point estimates of stock prices based on stochastic models [2]. In this paper, we present a framework to simulate such general particle movement under user-defined force fields. We apply this framework to the task of manipulating micro and nanoscale components using an optical force field.

Micro and nanoscale components can be used to exploit new phenomena that take place at the small scale [3]. Potential applications of such small components include bio-sensors, electronic components, photonic devices, solar cells, and batteries [4,5]. In order to construct useful devices, micro and nanoscale components need to be assembled together. Assembling small components to make functional devices remains a challenge despite rapid advances in imaging, measurement, and fabrication at the small scale. Two types of assembly processes are possible at the small scale. The first type of process is self-assembly [6]. This process is useful for large scale production. The second type of process is directed assembly. This process is useful for prototyping new design concepts, small scale production, device repair, and creating templates for certain types of self-assembly processes. In this paper, we limit ourselves to directed assembly processes. A number of manipulation techniques for directed assembly of small components have emerged.

In order to manipulate a small component, the appropriate optical, electrostatic, or magnetic field (i.e., trap) needs to be created to trap the component. The field is then controlled to move the component. In this paper, we focus on the optical force fields and traps associated with them. Optical trapping takes place in a fluidic workspace. The interaction between the optical field and the component is stochastic in nature, due to the Brownian motion of the component, as well as the presence of uncertainty in the component location as a result of sensor errors. Unfortunately, the off-line planning approach that works well at the macroscale does not work at the small scale. To cope with the stochastic nature of the problem, automated microscale assembly requires a real-time and automated planning paradigm that takes actions based on the estimated state of the workspace in a feedback loop [7].

In order to develop automated, real-time planning algorithms, we need to develop a fundamental understanding of the interaction of components with trapping fields. For example, we need to understand and characterize trapping probability, trap reliability, and trap robustness. The knowledge of the trapping probability will enable us to position the optical trap close enough to the component of interest so that it gets trapped. Moreover, any other component that drifts close enough to the optical trap, such that its trapping probability exceeds a certain threshold value, is a potential or imminent source of unintended trapping. Thus, appropriate collision prevention schemes need to be applied in such cases to avoid losing the trapped component. Under different operating conditions, components interact qualitatively differently with the trap. Unfortunately, the parameter space that describes different operating conditions is very large. Hence, simulation is the only viable option for characterizing the interactions between the trap and the component to facilitate real-time planning.

This paper presents a GPU-based method to simulate the trajectory of particles under a force field. The method described in this paper is able to handle time varying force fields. The accuracy of computation depends on the accuracy of the force field. We illustrate the usefulness of this method by showing how it can be used to estimate trapping probabilities for moving optical traps. This new method presents considerable computational improvements over our previous approach to conducting CPU-based simulations for estimating trapping probabilities. The work presented in this paper enables fast computation of trapping probabilities. Faster computation of trapping probabilities presents three main advantages. First, it enables exploration of a much larger parameter space. Hence, it improves our understanding of trapping physics. Second, fast computation of trapping probabilities eliminates the need for the use of meta-modeling during the particle transport planning. Finally, faster computation enables more simulation runs and hence we get better estimates of the trapping probability.

2 Related Work

Since the initial discovery of optical gradient forces [8] and subsequent invention of optical tweezers [9], the use of optical traps to manipulate micron-sized objects has been widespread. Simulations of the optical traps play an important role in utilizing optical tweezers as assembly tools at the micro and nanoscale [7]. Early work in this area focused on trap and particle interactions and models were developed to estimate optical forces acting on spherical particles. As a part of our recent prior work, we developed a model to simulate the trajectory of a spherical particle under optical forces [10]. This model was implemented on a CPU and used to generate trapping probability estimates by conducting off-line simulations. A meta-model was developed to quickly query trapping probability estimates during the automated path planning [7]. CPU-based simulations are very time consuming. Hence, in an offer to speed up the simulation process, we successfully implemented Brownian motion simulation on GPU [11]. The current paper builds on our previous work and presents a new GPU-based method for computing particle trajectories under optical force fields. In computing particle trajectories, it combines both the influence of Brownian motion and an optical force field in a single unified GPU-based framework.

Graphics processors have evolved over the last decade from a fixed-function (rendering-only) pipeline to a highly flexible programmable pipeline. One of the application areas that has received significant attention in this evolution has been physically-based simulations due to their application to simulation of environmental effects for 3D games [12]. Some of the earliest work in this area involved physically-based wave simulations on 2D lattices [13]. More general Lattice-Boltzmann 3D simulations for fluids and gases were later implemented on GPUs by Li et al. [14, 15]. Other fluid simulations on GPUs include work on Navier-Stokes equations for incompressible fluids (see review in this area by Harris [16]) as well as Euler-equations-based dynamics of ideal gases (Hagen et al. [17]). Work on fluid simulation has also been extended to deal with obstacles [18, 19, 20, 21, 16]. Visualization of flows on GPUs has been addressed through line-integral convolution and Lagrangian-Eulerian advection [22, 23, 24]. Recent work by Juba and Varshney [25] shows the suitability of the GPUs for stochastic calculations.

Initial work has been done porting optical trapping simulations to graphics processors; however, to our knowledge, no GPU-based trapping probability estimation work exists. [26] and [27] simulate the manipulation of glass microspheres within a holographic laser testbed. Due to accelerated two-dimensional FFT calculations, both show significant computational speedups over their CPU-based counterparts. Similar results are available for arbitrarily positioned dual-beam ("twin") traps [28]. Using a model similar to ours, Balijepalli [11] provides an accuracy and error analysis of results from both CPU and GPU, along with experimental validation of the simulation.

3 Calculating Trapping Probabilities

Both fully- and semi-autonomous operation of optical tweezers require the capability to trap a particle and plan its path through the workspace; avoiding collisions with other components in real-time. The probability that a given particle will be trapped within a predictable spatial region about the laser beam focus, or *trapping probability*, is a critical aspect of this real-time motion planning problem. As it is infeasible to determine trapping probabilities through real-world experiments due to a large parameter space (including, for example, initial particle position, neighboring workspace component existence, motion of laser), we have developed a computational framework in which trapping probabilities can be determined through optical tweezers simulation.

3.1 Simulated Particle Motion

Our initial code operates on glass and silica microspheres, as these are well-studied in the literature [29]. Furthermore, properties of these spheres can be accurately modeled using theoretically- and experimentally-verified equations.

In general, any moving component in the workspace will experience hydrodynamic forces coupled with a rapidly fluctuating force, the result of frequent and numerous collisions with surrounding liquid molecules. We can model these closely connected forces using Langevin's equation [30, 31]. We use model given in [11]. Given a velocity V, we describe the change in velocity over time as:

$$\frac{dV(t)}{dt} = -\frac{\gamma}{m}V(t) + \frac{\xi}{m}\Gamma(t) \tag{1}$$

The hydrodynamic forces over time (t), are a function of velocity (V), mass (m), and drag (γ). The drag coefficient γ is given by Stokes' Law [32] as $\gamma = 6\pi\eta r$, with η representing the fluid viscosity as a function of temperature and r representing the radius of the silica sphere, in this case. The stochastic, rapidly fluctuating force Γ is scaled by a constant ξ satisfying the fluctuation-dissipation theorem [33], which is defined as $\xi = \sqrt{2\gamma K_B T}$, with K_B representing Boltzmann's constant [34]. The stochastic term Γ prevents a direct analytic solution to Langevin's equation; as such, our framework uses a finite difference expression of Equation 1. The stochastic term Γ is substituted by a appropriately scaled normal distribution $N(0, 1/\delta t)$ where δt is the finite time step. The term $1/\sqrt{\delta t}$ is absorbed into a scaling constant as described in [11].

We combine the finite difference form of Langevin's equations with an enumeration of all other forces affecting a silica sphere in the workspace to yield Equation 2. The external force factor F_{ext} includes the constant (for a sphere) forces of gravity and buoyancy, as well as the all-important optical trapping force applied by the laser to the particle.

$$A(t+\delta t) = -\frac{\gamma}{m}V(t) + \frac{1}{m}\sqrt{\frac{2\gamma K_B T}{\delta t}}N(0,1) + \frac{F_{ext}}{m} \tag{2}$$

We explicitly integrate Equation 2 to obtain a simulation of particle movement over time. In practice, the most popular explicit integration techniques include fourth-order Runge-Kutta methods [35], the Gear predictor-corrector method [36], and second-order "velocity" Verlet integration [37]. Each is appropriate in certain situations; as such, we use Verlet integration (Equations 3 and 4) due to its conservation of energy at larger time steps and ease of computation, as discussed in [11]. Since the time step has a direct effect on the number of iterations required to run a simulation (and thus its runtime), we are interested in maximizing time step size while maintaining tight error bounds.

$$X(t+\delta t) = X(t) + V(t)\delta t + \frac{1}{2}A(t)\delta t^2 + O(\delta t^4) \tag{3}$$

$$V(t+\delta t) = V(t) + \frac{A(t+\delta t) + A(t)}{2}\delta t + O(\delta t^2) \tag{4}$$

Given velocity $V(t)$ and external force factor F_{ext}, we compute the acceleration $A(t+\delta t)$ at the next time step using Equation 2. From here, the velocity Verlet method provides the next position, $X(t+\delta t)$, and velocity, $V(t+\delta t)$, in a single pass.

The characteristic time scale of our entire model is given by the relaxation time $\frac{m}{\gamma}$, with $\gamma = 6\pi\eta r$ the drag coefficient of Stokes' equations; this is the time needed for a particle's initial velocity to converge to thermal equilibrium. In our experiments, the numerical integration time step δt is set to the nearest multiple of 100 ns such that $\delta t << \frac{m}{\gamma}$. Choosing such a small δt provides an opportunity to observe interesting nonequilibrium behavior in the simulation. Furthermore, this small time step decreases maximum error in both velocity and position, bounded by the velocity Verlet to $O(\delta t^2)$ and $O(\delta t^4)$, respectively.

Our particle simulation framework accepts either a continuous or discretely sampled force field. In our experiments, we choose to represent the optical field discretely. Force values are provided by numerically integrating the basic scattering and gradient forces of radiation (see Ashkin's seminal paper [9]). Such integration is typically done by tracing representative rays of light from the laser position through the workspace to the laser focus – and possibly through microspheres in the workspace. By tracing the reflection and refraction of these simulated rays, a reliable estimate of both the scattering and gradient forces can be computed. In our prototype implementation, we rely on code written in [10] to obtain these forces.

3.2 Trapping Probability Estimates

To estimate trapping probabilities, particle trajectory simulation is performed multiple times at a given point in the parameter space. The simulation is conducted by starting the particle at the designated location. The trapping probability is estimated as a ratio of the number the times the sphere gets trapped by the laser beam over the total number of trials.

Trapping is a complex phenomenon due to the Brownian motion of the particles. If a beam is held stationary for an indefinite period of time, then particles far away from the beam are likely to wander into the beam due to Brownian motion and eventually be trapped. Particles continue to exhibit Brownian motion even after they have been trapped. Therefore, trapped particles eventually jump out of the trap at practically useful laser powers. Many different notions of trapping probability can be defined based on the context. For the purpose of this paper, we are mainly interested in a notion of trapping probability which is relevant from the point of view of path planning. In this application, the laser beam continues to move. Hence, the time available for trapping is relatively small. For the purpose of this paper, we have done all trapping probability estimates for fixed finite period of time. Methodology presented in this paper can be easily used to compute trapping probability as a function of the available trapping time.

We have assumed that the Brownian motion inside the trap is negligible. Hence, the mean time to escape of the trap is quite large with respect to the planning horizon. Therefore, we do not account for the possibility of a trapped particle escaping the trap in our calculations. The simulation infrastructure has a built-in capability to simulate Brownian motion under the force field. Therefore, it can automatically account for the situations where a particle will escape the trap if the simulation is performed over the long period of time. For the figures produced in this paper, we assume a particle to be trapped if the probability of a particle escaping the laser within 1 ds is less than $\frac{1}{N}$, where N is the number of simulated trajectories per grid position.

4 Massively Parallel Simulations

4.1 Overview

We leverage the massively parallel architecture of the GPU to expedite the calculation of trapping probabilities over a wide range of locations relative to the focal point of the laser. In particular, we consider a discrete grid, $G = \{[y_i, z_j]\}$ of particle locations.

As detailed in section 3, the particles we are interested in trapping undergo motion governed by the Langevin equation; which contains a stochastic component modeling the Brownian motion of the particles. Thus, a reliable estimate of the trapping probability at a particular grid cell, $[y, z]$, can only be obtained by repeatedly simulating the trajectory of a particle intially placed at $[y, z]$. To obtain a 95% confidence interval of less than ± 0.03125 on our estimated trapping probability, we simulate the particle trajectories 1024 times at each grid cell. The greater the degree of confidence we require, the more trajectories we must simulate for each cell on the simulation grid G. This is a highly computationally intensive, but inherently parallel process. In particular, the estimation of the trapping probability at each grid cell can be done independently.

Though the parallel nature of the problem is straightforward, consideration is still required if we are to achieve optimal performance on current hardware. In all of the experiments detailed below, we have performed the trapping probability estimates on an Nvidia Tesla S1070; a GPU computing oriented, CUDA capable device. A diagram of the CUDA architecture – how the threads of execution and memory are arranged – is provided in Figure 1.

GPU vs. CPU implementations While this paper focuses on a GPU implemetation of the trapping probability calculation, we wish to draw attention to the fact that it is the massively parallel nature of this computation, and not the specific GPU implementation, that is of the broadest interest. In particular, the CPU implementation with which we contrast our GPU implementation, while reasonably efficient, still has many avenues for optimization. Specifically, we compare GPU performance against a single-threaded CPU implementaion that does not take advantage of vectorized instructions. While it is certainly true that the CPU implemenation could be improved to offer better performance, the most important observation is the rate at which the simulation results scale. Since each simulation can be performed independently, the overall computation of trapping probabilities is massively data parallel. Thus, while the performance gap between the two implementations can undoubtedly be shrunken somewhat, the data-parallel nature of the problem ensures that scalable parallel architecture of the GPU will continue to provide a very significant benefit.

4.2 Data Structures

There are two obvious was to parallelize the computation of trapping probabilities: among grid cells or among particles. We choose the latter of these two as we feel it offers more flexiblity and allows us to maximally utilize the GPU. There are four major sources of input to the simulation kernel:

1. an array of particles
2. a force grid
3. a set of fundamental physical constants
4. a stream of pseudorandom numbers

Care must be taken to ensure that each of these input sources is amenable to GPU computation. Each particle maintains a position, velocity, acceleration, and state variable. The position, velocity and acceleration are given as vectors in $\mathbb{R}^3$, while the state variable is a simple boolean value which is examined to determine if simulation for this particular particle should continue. Thus, we can imagine each particle as a tuple $p_i = (\mathbf{x}_i, \mathbf{v}_i, \mathbf{a}_i, s_i)$, and the array of N particles simply becomes $P = \{p_i\}_{0 \le i < N}$. While this is logically convenient, it is much more efficient to store the particles as a structure of arrays (SoA) rather than the array of structures (AoS) detailed above. Hence, we consider the array of particles as $P = \{X, V, A, S\}$, where X, V, A, and S are each arrays of length N storing in their i^{th} position the respective field for the i^{th} particle. Because of the manner in which the GPU threads access memory, the SoA approach

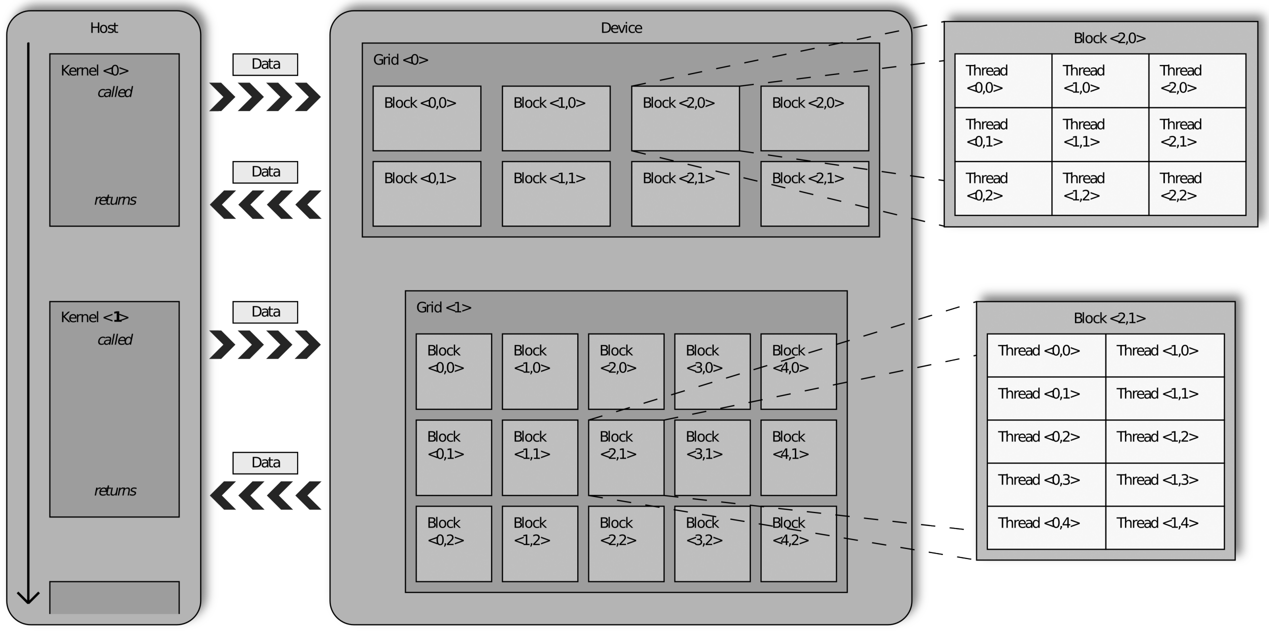

Fig. 1. The CUDA architecture provides for a logical hierarchy of parallelism that maps well to the hardware. The computational kernel is run in parallel among a large number of threads which are grouped into 1,2 or 3 dimensional thread blocks. Threads within a block may communicate using the shared memory or coordinate execution using synchronization primitives. The blocks are likewise grouped into 1,2 or 3 dimensional grids. Each thread is able to access its own grid, block and thread identifiers.

allows coalesced memory access for groups of threads, and hence, requires fewer memory reads.

The force grid is a 2D array of values, representing a discrete sampling of the force function, $\mathcal{F}:\mathbb{R}^2\rightarrow\mathbb{R}^2$, which maps input grid positions to output force vectors. In our case, since $\mathcal{F}$ is fairly well behaved, it suffices to take a discrete uniform sampling of this function at intervals of $0.25\,\mu$m for all $0\leq y\leq 20$ and $-20\leq z\leq 8$. The value at all intermediate locations is estimated by means of a bilinear interpolation of the values at the nearest sample positions. We choose to store our discrete force grid, F, as a texture. This results in two main benefits. First, access to texture memory is cached, resulting in faster average access times for looking up force values for spatially coherent particles. Second, the GPU has dedicated hardware for performing bilinear interpolation. Thus, by storing F as a texture, we benefit both from faster access to the force values as well as hardware accelerated bilinear interpolation.

We also require access to some physical constants to compute particle trajectories. Storing these constants in the global GPU memory makes access, which is required for each simulation timestep, expensive. However, storing local copies of all of these constants is highly wasteful. In fact, all CUDA capable devices have a special read-only segment of memory reserved for constants, for which access is cached. Utilizing this constant memory allows us fast yet global access to the set of physcial constants required to evalute the governing equations of the particles' dynamics.

Finally, the simulation requires a stream of pseudo-random numbers to model the Brownian motion of the particles. To provide these random numbers, we adapt the GPU implementation used by Meel et al. [38]. Each particle maintains a state which determines its current position in the pseudo-random progression. During the actual simulation, uniformly distributed pseudo-random numbers are generated on demand by a GPU kernel and subsequently transformed into random samples from the standard normal distribution by mean of the Box-Muller transform.

By making careful considerations in the layout and structure of our data, and by exploiting the special hardware provided by the GPUs, we are able to make efficient use of these devices when parallelizing our estimates of trapping probability.

4.3 Trapping Criteria

Reliably classifying a specific particle as "trapped" or "not trapped" is necessary to provide accurate overall trapping probabilities. To accomplish this goal, we provide a simple parallelizable method that returns, once per particle, a 0 (not trapped) or 1 (trapped) upon termination.

Similar to [10], the following conditions sufficiently describe the termination criteria for our method:

1. Particle falls to the bottom of the workspace $\rightarrow 0$.
2. Particle strays outside the bounds of the force field $\rightarrow 0$.
3. Total experimental time is expended without trapping $\rightarrow 0$.
4. Particle is trapped $\rightarrow 1$.

These termination criteria are supressed in certain situations. Specifically, criterion 1 is ignored if the particle remains affected by the laser forces. Furthermore, criterion 2 is supressed if either the particle is sufficiently close to the trap focus (as it will be affected by strong optical gradient forces) or if the laser has nontrivial horizontal velocity (as the cone of the laser might move to intersect with the particle).

Criterion 4 returns when, informally, the optical forces imparted on a particle by the laser effectively overpower all other acting forces (*e.g.*, Brownian motion, gravity, and buoyancy). For a stationary laser (velocity $v = \mathbf{0}$), this amounts to a trapped particle remaining within fixed distance d_z^v beneath the laser and distance d_{xy}^v along the X- and Y-axes. For a laser with nontrivial horizontal or vertical velocity, both d_z^v and d_{xy}^v change considerably as a function of both laser velocity and particle size.

The calculation of these trapping boundaries will potentially change from experiment to experiment due to, for example, an adjustment in the power of the laser; however, we assume that such environmental variables will remain static across a single experimental run. We do expect both the horizontal and vertical components of the laser's velocity to change over the course of a single experiment. As such, we adopt a strategy of one-time computations of radial and axial trapping bounds d_{xy}^v and d_z^v for different base laser velocities, relying on interpolation to provide bounds for any velocity.

For a stationary laser, we place 1024 particles at the origin and execute a version of our full parallel simulation, described in Section 4.4, for 50ms. Over this time period, all of these particles fall into the optical trap; our simulation records the variation in both radial and axial displacement of the particle. Upon termination, we select the maximum of the maximum radial and axial displacements as our final, stationary trapping bounds. This approach generalizes to a moving laser through supression of criterion 2, as defined above.

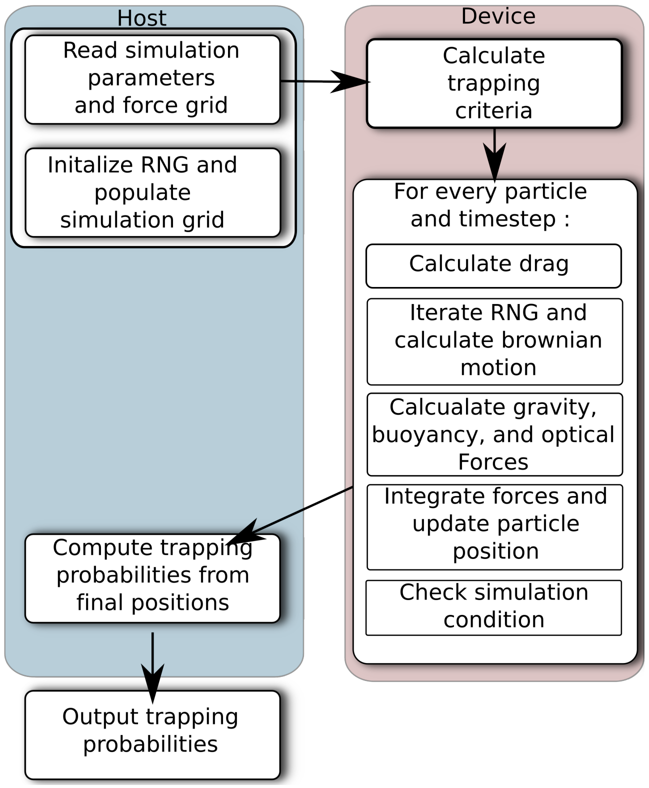
Host
Device
Read simulation parameters and force grid
Initalize RNG and populate simulation grid
Calculate trapping criteria
For every particle and timestep :
Calculate drag
Iterate RNG and calculate brownian motion
Calcualate gravity, buoyancy, and optical Forces
Integrate forces and update particle position
Check simulation condition
Compute trapping probabilities from final positions
Output trapping probabilities

Fig. 2. An overview diagram of our system showing the distribution of work and the flow of data between the host and the device (GPU). Currently, the host is used only to load the initial simulation parameters and set the initial particle positions. The device performs the simulation of the particle trajectories and returns the final positions to the host, which then simply computes the fraction of trapped particles corresponding to each initial position.

4.4 Execution

With the experimental setup fully initialized, we are ready to perform the massively parallel simulation of multiple particles on the GPU. For a specific point x_* in the workspace, we perform $N = 1024$ simulations in parallel. Note that, for each individual simulation run, the return value is a binomial random variable of 0 (not trapped) or 1 (trapped); thus, for simulation sample sizes of at least 30 runs, its outcome can be fit to a Gaussian curve with mean $\mu = P_T$, the trapping probability, and variance $\sigma^2 = \frac{P_T(1-P_T)}{N}$ [39]. For $N = 1024$, this yields a 95% confidence interval with maximum error of less than ± 0.03125.

After initializing N particles to the desired starting position x_* and all active lasers to their initial positions and velocities, we assign each particle and laser to a thread on the GPU. We then execute Algorithm 1.

Algorithm 1 computes the final positions of each of the N particles after some predetermined period of time *MAXTIME*. The first loop (Line 1) executes in parallel on the GPU. Line 3 calculates the current drag force being applied to the particle. As shown in Equation 1, the drag force is based on, among other things, a drag coefficient γ. The value of γ remains constant throughout the simulation; as such, our implementation makes use of constant memory on the GPU to realize significant speedup. The nondeterministic Brownian force is calculated at Line 4 using the streaming GPU-specific random number generator discussed in Section 4.1. Line 5 calculates a summation of forces, including the optical force F applied by the laser, through a combination of constant memory and texture memory. Numerical evolution in Line 9 updates the positions of the particles and lasers through velocity Verlet integration; this already quick method is further sped up through GPU-specifc "fast" algebraic operations.

When Algorithm 1 returns, the host gathers all N final positions of particles p_i for $i \in [1,N]$ and lasers l. It then computes the set of *trapped* particles, *TRAPPED*, as follows:

$$TRAPPED = \{p_i | trapped(p_i, l, v_l)\} \tag{5}$$

In other words, the set of trapped particles, *TRAPPED*, is the set of all particles, p_i, which, at the termination of the simulation, adhere to the trapping conditions of lasers $\ell_j \in l$ which are moving with corresponding velocities v_{ℓ_j}.

From this, the probability (with 95% confidence interval of ± 0.03125) that a particle starting from location x_* will be trapped by a laser with velocity v_l within time period

Algorithm 1 Parallel particle evolution on the GPU

```
Input: Set of active threads, each with pre-initialized particle $p$ and laser $l$ with velocity $v$. Time limit $MAXTIME$.
Output: Positions of particle and laser after $MAXTIME$.
 1: for all active threads (in parallel) do
 2:     while particle $p$ is not trapped and time $t < MAXTIME$ do
 3:         calculate drag force
 4:         calculate Brownian motion
 5:         calculate sum of gravity, buoyancy, optical forces
 6:         if $trapped(p,l,v)$ then
 7:             break
 8:         end if
 9:         second-order velocity Verlet integration to provide incremental update to velocity, position
10:         update position of laser $l$ according to velocity $v$
11:     end while
12:     return final positions of $p$ and $l$
13: end for
```

MAX_TIME is:

$$P_T = \frac{|TRAPPED|}{N} \tag{6}$$

5 Results

5.1 Trapping Probabilities

We evaluate the trapping probability for particles under the effect of both a stationary and moving laser. The trapping probabilities are estimated on a discrete grid of positions relative to the focus of the laser. In all of our experiments, we consider a grid whose extents match that of the discrete laser force grid F. We sample this grid in the y and z directions at a resolution of $0.25\,\mu$m.

5.1.1 Trapping Probabilities under a Stationary Laser

Figures 3 illustrates how the probability of trapping a particle varies with respect to the particle's distance from the laser in both the y and z directions, under the influence of a stationary laser. To verify the accuracy of our trapping probability estimates, we examine how the estimates computed using our GPU implementation differ from those computed using a CPU-based implementation of the same particle dynamics model. Since the underlying governing equations are the same, and we perform a sufficient number of simulations (i.e. 1024 per grid position) to ensure 95% confidence in our probability estimate, we expect to see very little difference between the two plots. Figure 4 shows the difference in probability estimates between the two implementations. The average error between the CPU and the GPU implementation of the probability estimate (computed using L2 norm) is 0.00091.

We also examine the difference between probability estimates computed using single and double precision. We found the average error (computed using L2 norm) to be 0.00084.

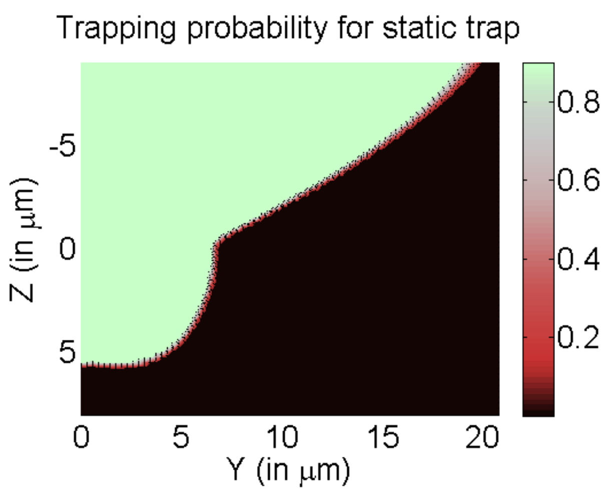

Fig. 3. This plot shows the probability of trapping a particle under the force exerted by a stationary laser with its focus at $(0,0)$, and was generated using the GPU implementation detailed in this paper.

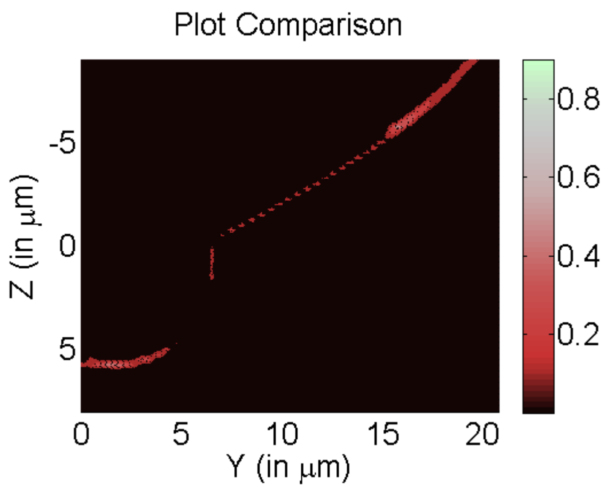

Fig. 4. This plot shows the absolute difference between the CPU and the GPU implementation of the probability of trapping a particle under the force exerted by a stationary laser with its focus at $(0,0)$.

5.1.2 Trapping Probabilities under a Moving Laser

Figures 5, and 6 illustrate how the probability of trapping a particle varies with respect to the particle's distance from the laser in both the y and z directions, under the influence of a moving laser. In particular, we consider a laser moving with a constant velocity of $0.65\,\mu\mathrm{m\,ms}^{-1}$ in the y direction as well as a laser moving with a constant velocity of $0.325\,\mu\mathrm{m\,ms}^{-1}$ in the z direction.

5.2 Timing

To obtain a timing comparison between the CPU-based and GPU-based simulations, we parametrize over the number of particles. Given a grid spacing of 0.25 microns and a grid over $[0,20] \times [-20,+8]$, we must test 9744 positions. Assuming 1024 particles per position to achieve a 95% confidence interval with maximum error of less than ± 0.03125,

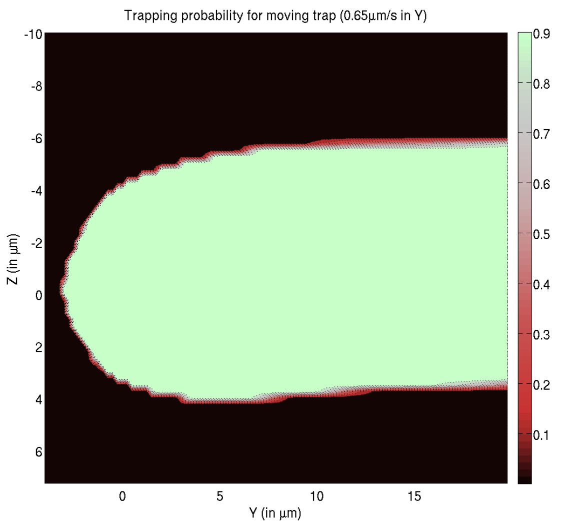

Fig. 5. This plot shows the probability of trapping a particle under the force exerted by laser moving at a constant velocity of $0.65\,\mu\mathrm{m\,ms}^{-1}$ in the direction $(1,0)$, and was generated using the GPU implementation detailed in this paper.

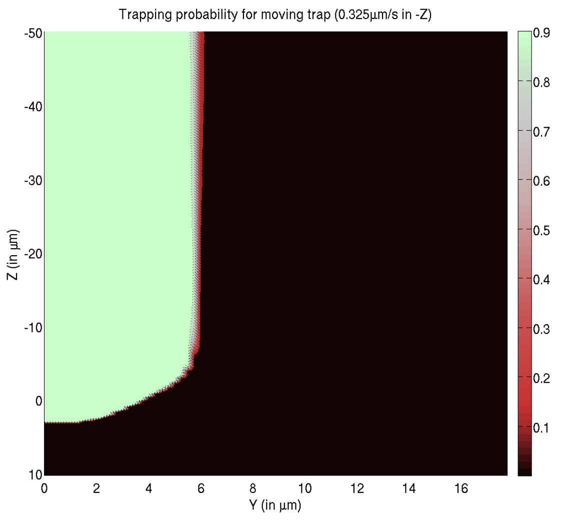

Fig. 6. This plot shows the probability of trapping a particle under the force exerted by laser moving at a constant velocity of $0.325\,\mu\mathrm{m\,ms}^{-1}$ in the direction $(0,-1)$, and was generated using the GPU implementation detailed in this paper.

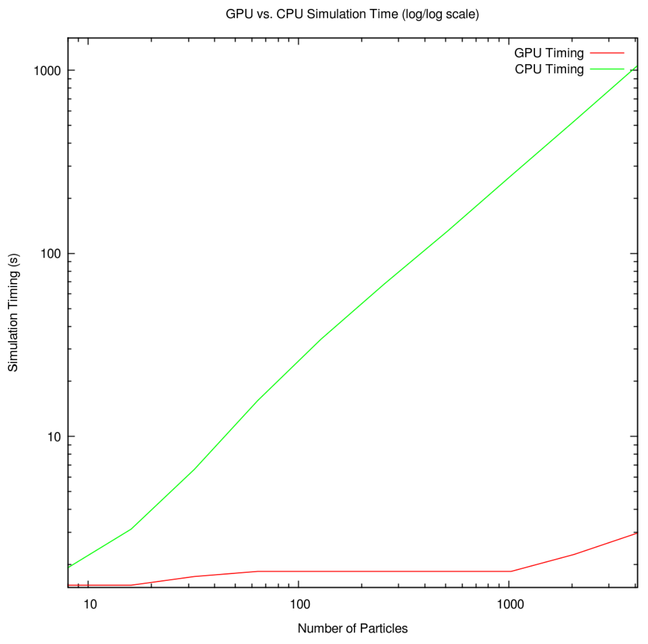

Fig. 7. The running time required as a function of the number of trajectories calculated using both the CPU and GPU simulators. The plots have been placed on a log-log scale. The CPU simulator exhibits exactly the type of linear performance curve we expect, while GPU performance slowed by less than a factor of 2 between 8 and 4096 trajectories.

this results in roughly ten million tests. The performance results can be seen in Figure 7. As the number of particles increases, so does the benefit of the GPU-based parallel simulation. At $N = 4096$ particles per grid cell, the GPU-based simulation is ~ 356 times faster than its CPU-based counterpart.

6 Conclusion and Future Work

We have introduced a GPU-based framework for massive simulation of particle motion under user-defined force fields. We applied this framework experimentally to the well-studied problem of computing the trapping probabilities associated with micron-sized silica beads in optical trapping workbenches. The simulator handles both stationary and moving laser-induced force fields and, due to the highly parallel nature of the problem and efficient use of the available hardware, exhibits a significant speedup over its CPU-based analog. In particular, when evaluating many trajectories (4096), we see approximately a 356 times speedup of the GPU based simulator over its CPU based counterpart. This speedup is more than can be accounted for by the increased processor count on the GPU, and we attribute the extra performance to the higher memory bandwidth, spatially coherent caching, and hardware accelerated bilinear interpolation of the laser force field in the GPU simulator. In particular, this last operation must be performed in software on the CPU. We believe this work indicates that GPUs hold great promise in accelerating the type of compute-intensive simulations that are required when working with optical tweezers and nanoscale assembly in general. Often times, the stochastic nature of such simulations leads to a probabilistic approach involving many independent trials, a setup whose parallelism uniquely suits the type of high-throughput computation enabled by modern GPUs.

In the future, we are interested in extending this work to deal with more general types of nanocomponents, perhaps even those which cannot be defined analytically. We are also interested in investigating other areas of the optical workbench workflow where GPUs can be used to accelerate computational bottlenecks in the process.

Achnowledgements

This work has been supported in part by the NSF grants: CCF 04-29753, CNS 04-03313, CCF 05-41120 and CMMI 08-35572. We also gratefully acknowledge the support provided by the NVIDIA CUDA Center of Excellence award to the University of Maryland and constructive discussions with David Luebke at NVIDIA research. Any opinions, findings, conclusions, or recommendations expressed in this article are those of the authors and do not necessarily reflect the views of the research sponsors.

References

[1] Tolle, D., and Le Novere, N., 2010. "Brownian diffusion of ampa receptors is sufficient to explain fast onset of ltp". *BMC Systems Biology,* **4**(1), p. 25.

[2] Broadie, M., and Kaya, O., 2006. "Exact simulation of stochastic volatility and other affine jump diffusion processes". *Operations Research,* **54**(2), pp. 217–231.

[3] Bhushan, B., ed., 2004. *Springer Handbook of Nanotechnology*. Springer-Verlag, New York, NY.

[4] Niemeyer, C., and Mirkin, C. A., eds., 2004. *Nanobiotechnology: Concepts, Applications and Perspectives*. Wiley-VCH, Germany.

[5] Wilson, M., Kannangara, K., Smith, G., Simmons, M., and Raquse, B., 2002. *Nanotechnology: Basic Science and Emerging Technologies*. Chapman and Hall/CRC, Boca Raton, FL.

[6] Requicha, A., and Arbuckle, D., 2006. "CAD/CAM for nanoscale self-assembly". *IEEE Computer Graphics and Applications,* **26(2)**, pp. 88 – 91.

[7] Banerjee, A., Pomerance, A., Losert, W., and Gupta, S., 2010. "Developing a stochastic dynamic programming framework for optical tweezer-based automated particle transport operations". *Automation Science and Engineering, IEEE Transactions on,* **7**(2), april, pp. 218 –227.

[8] Ashkin, A., 1970. "Acceleration and trapping of particles by radiation pressure". *Physical Review Letters,* **24**(4), pp. 156–159.

[9] Ashkin, A., Dziedzic, J., Bjorkholm, J., and Chu, S., 1986. "Observation of a single-beam gradient force optical trap for dielectric particles". *Optics letters,* **11**(5), p. 288.

[10] Banerjee, A., Balijepalli, A., Gupta, S., and LeBrun, T., 2009. "Generating Simplified Trapping Probability Models From Simulation of Optical Tweezers System". *Journal of Computing and Information Science in Engineering,* **9**, p. 021003.

[11] Balijepalli, A., LeBrun, T., and Gupta, S., 2010. "Stochastic Simulations With Graphics Hardware: Characterization of Accuracy and Performance". *Journal of Computing and Information Science in Engineering,* **10**, p. 011010.

[12] Owens, J. D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A. E., and Purcell, T., 2007. "A survey of general-purpose computation on graphics hardware". *Computer Graphics Forum,* **26**(1), Mar., pp. 80–113.

[13] James, G., 2001. *Operations for hardware-accelerated procedural texture animation*. Charles River Media, pp. 497–509.

[14] Li, W., Wei, X., and Kaufman, A. E., 2003. "Implementing lattice boltzmann computation on graphics hardware". *The Visual Computer,* **19**(7-8), pp. 444–456.

[15] Wei, X., Zhao, Y., Fan, Z., Li, W., Qiu, F., Yoakum-Stover, S., and Kaufman, A. E., 2004. "Lattice-based flow field modeling". *IEEE Transactions on Visualization and Computer Graphics,* **10**(6), pp. 719–729.

[16] Harris, M., 2005. "Fast fluid dynamics simulation on the GPU". In SIGGRAPH '05: ACM SIGGRAPH 2005 Courses, ACM, p. 220.

[17] Hagen, T. R., Lie, K.-A., and Natvig, J. R., 2006. "Solving the Euler equations on graphics processing units.". In International Conference on Computational Science (4), pp. 220–227.

[18] Bolz, J., Farmer, I., Grinspun, E., and Schröoder, P., 2003. "Sparse matrix solvers on the GPU: conjugate gradients and multigrid". In SIGGRAPH '03: ACM SIGGRAPH 2003 Papers, ACM, pp. 917–924.

[19] Krüger, J., and Westermann, R., 2003. "Linear algebra operators for GPU implementation of numerical algorithms". *ACM Transactions on Graphics,* **22**(3), July, pp. 908–916.

[20] Liu, Y., Liu, X., and Wu, E., 2004. "Real-time 3D fluid simulation on GPU with complex obstacles". In Pacific Conference on Computer Graphics and Applications, IEEE Computer Society, pp. 247–256.

[21] Sander, P., Tatarchuk, N., and Mitchell, J. L., 2004. *Explicit Early-Z Culling for Efficient Fluid Flow Simulation and Rendering*. Charles River Media.

[22] Heidrich, W., Westermann, R., Seidel, H.-P., and Ertl, T., 1999. "Applications of pixel textures in visualization and realistic image synthesis". In I3D '99: Proceedings of the 1999 symposium on Interactive 3D graphics, ACM, pp. 127–134.

[23] Jobard, B., Erlebacher, G., and Hussaini, M. Y., 2001. "Lagrangian-eulerian advection for unsteady flow visualization". In Proceedings of the Conference on Visualization 2001 (VIS-01), T. Ertl, K. Joy, and A. Varshney, eds., IEEE Computer Society, pp. 53–60.

[24] Weiskopf, D., Hopf, M., and Ertl, T., 2001. "Hardware-accelerated visualization of time-varying 2-D and 3-D vector fields by texture advection via programmable per-pixel operations". In Vision, Modeling, and Visualization, pp. 439–446.

[25] Juba, D., and Varshney, A., 2008. "Parallel stochastic measurement of molecular surface area". *Journal of Molecular Graphics and Modelling,* **27 No. 1**, August, pp. 82 – 87.

[26] Reicherter, M., Haist, T., Zwick, S., Burla, A., Seifert, L., and Osten, W., 2005. "Fast hologram computation and aberration control for holographic tweezers". K. Dholakia and G. C. Spalding, eds., Vol. 5930, SPIE,

p. 59301Y.

[27] Haist, T., Reicherter, M., Wu, M., and Seifert, L., 2006. "Using graphics boards to compute holograms". *Computing in Science & Engineering,* **8**(1), pp. 8–13.

[28] Hermerschmidt, A., Kruger, S., Haist, T., Zwick, S., Warber, M., and Osten, W., 2007. "Holographic optical tweezers with real-time hologram calculation using a phase-only modulating LCOS-based SL at 1064 nm". In Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, Vol. 6905, p. 7.

[29] Wright, W., Sonek, G., and Berns, M., 1994. "Parametric study of the forces on microspheres held by optical tweezers". *Applied Optics,* **33**(9), pp. 1735–1748.

[30] Gardiner, C., 1985. *Handbook of stochastic methods*. Springer Berlin.

[31] Langevin, P., 1908. "On the theory of brownian motion". *C. R. Acad. Sci.,* **146**, pp. 530–533.

[32] Girault, V., and Raviart, P., 1979. "Finite element approximation of the Navier-Stokes equations". *Lecture Notes in Mathematics, Berlin Springer Verlag,* **749**.

[33] Weissbluth, M., 1989. *Photon-atom interactions*. Academic Press San Diego, CA.

[34] Grassia, P., 2001. "Dissipation, fluctuations, and conservation laws". *American Journal of Physics,* **69**, p. 113.

[35] Jameson, A., Schmidt, W., and Turkel, E., 1981. "Numerical solutions of the Euler equations by finite volume methods using Runge-Kutta time-stepping schemes". *AIAA paper,* **1259**, p. 1981.

[36] Allen, M., and Tildesley, D., 1990. *Computer simulation of liquids*. Oxford University Press, USA.

[37] Verlet, L., 1968. "Computer experiments on classical fluids. ii. equilibrium correlation functions". *Phys. Rev,* **165**(1), pp. 201–14.

[38] van Meel, J., Arnold, A., Frenkel, D., Zwart, S. P., and Belleman, R., 2008. "Harvesting graphics power for MD simulations". *Molecular Simulation,* **34**, March, pp. 259–266.

[39] Hines, W. W., Montgomery, D. C., Goldsman, D. M., and Borror, C. M., 2003. *Probability and Statistics in Engineering*, 4 ed. Wiley, Jan.

[40] Masuda, N., Ito, T., Tanaka, T., Shiraki, A., and Sugie, T., 2006. "Computer generated holography using a graphics processing unit". *Opt. Express,* **14**(2), pp. 603–608.

[41] Ratner, M., and Ratner, D., 2002. *Nanotechnology: A Gentle Introduction to the Next Big Idea*. Prentice Hall, Upper Saddle River, NJ.

[42] Peng, T., Balijepalli, A., Gupta, S., and LeBrun, T., 2009. "Algorithms for Extraction of Nanowire Lengths and Positions From Optical Section Microscopy Image Sequence". *Journal of Computing and Information Science in Engineering,* **9**, p. 041007.