A GPU Accelerated Virtual Scanning Confocal Microscope

James Kingsley†, Zhilu Chen+, Jeffrey Bibeau‡, Luis Vidali‡, Xinming Huang+, Erkan Tüzel†
†Department of Physics, Worcester Polytechnic Institute
+Department of Electrical and Computer Engineering, Worcester Polytechnic Institute
‡ Department of Biology and Biotechnology, Worcester Polytechnic Institute

***Abstract*—Fluorescence Recovery After Photobleaching (FRAP) is a commonly used technique for quantifying the movement of small biological systems. To aid in the evaluation of experimentally produced data, we used the parallel processing power offered by Graphics Processing Units (GPUs) to accelerate a computational simulation of the process. We find that the parallel process is significantly faster when implemented on the GPU, and that further speed increases can be accomplished via various optimizations, bringing the speed increase up to a factor of one hundred in some cases.**

***Index Terms*—CUDA, GPU, FRAP, parallel computing**

I. Introduction

The generation and maintenance of cellular polarization impacts an array of biological systems in plants, animals, and fungi. Some examples of the significance of cell polarity include: the establishment of morphogenetic gradients in animals [1] and plants [2], asymmetric cell division [3], axonal outgrowth [4], the immunological synapse [5], yeast budding [6], fungal hyphae invasion [7], pollen tube growth [8], root hair development [9], and moss protonemal cell development [10]. Since the endomembrane trafficking system and the plant cytoskeleton are involved in cellular polarization it is essential to characterize the dynamic interactions of these two systems.

To date, there are a handful of techniques that can be used to study protein dynamics *in vivo*. These include fluorescence correlation spectroscopy (FCS), single-particle tracking, and fluorescence recovery after photobleaching (FRAP) [11]–[15]. The most commonly used technique among the three is FRAP; this is because single particle tracking will not work at high probe concentrations and FCS requires very fast scanning rates. The FRAP experiments are based on a process called photobleaching [16]. Photobleaching occurs when exposure to high intensity light causes a fluorescent molecule to lose its fluorescence. During FRAP, a molecular species of interest is fluorescently labeled and monitored within a region of interest inside the cell. This region is then subjected to a high intensity laser pulse that locally abrogates the fluorescence within that region. Following this pulse, unbleached molecules move into the region of interest, causing an increase in average local fluorescence. This rate of increase is directly related to the mobility of the fluorescently labeled species and can be extended to quantify the diffusion coefficients and binding rates of proteins associated with cell polarization [16].

Due to the importance of cell polarized growth and the popularity of FRAP, there has been an increased need for fast and accurate quantification of FRAP results. Often these results are in physical geometries that are impractical to analyze analytically. To that aim, we have created a three-dimensional FRAP simulation for polarized cell growth in moss, *Physcomitrella patens*, an excellent model organism in which to study the dynamics of the cytoskeleton and endomembrane system during polarized growth [17], [18]. As previous fluid simulation techniques have been shown to have a dramatic speed improvement when implemented on GPUs using Nvidia's Compute Unified Device Architecture(CUDA) [19], FRAP simulations were implemented using CUDA as well.

II. Algorithm

Our model simulates a set of fluorescent particles in a Brownian fluid. We can apply photobleaching and imaging events at any time during this simulation. Typically there are two types of simulations used: direct imitation of time-series data and higher resolution 3D imaging. In the first, we limit ourselves to taking images with the same limitations as a microscope: a finite image acquisition time, a maximum frame rate, and that photobleaching cannot happen during imaging. This generates a time-series of images that directly corresponds to a physical experiment and can be used for quantitative comparison. In the second, we ignore physical limitations and image a "z-stack": a set of image slices that together provide a full 3D image of the system. This is done with instantaneous image acquisition and a high frame rate, which does not correspond to a physical process, but allows us to visualize the three-dimensional behavior of the photobleaching and recovery processes in a way not possible using a physical microscope.

A. Brownian Fluid

In order to simulate the behavior of a set of fluorescent particles, we treat them as being embedded in a simple Brownian fluid described by a single diffusion coefficient. This diffusion coefficient, D, is given by the Einstein relation

$$D = \frac{k_B T}{\zeta} \quad , \tag{1}$$

with temperature, T, and Boltzmann constant, k_B. Assuming spherical particles with radius R, the friction coefficient ζ is

978-1-4799-6233-4/14/$31.00 ©2014 IEEE

given by

$$\zeta = 6\pi\eta R \ , \tag{2}$$

for viscosity η. The fluid is assumed to have a very low Reynolds number, which means that the particles do not experience inertial effects. This lack of inertia means that the Stokes equation applies, i.e.

$$\zeta \frac{\Delta \vec{x}}{\Delta t} = \vec{\xi}(t) \tag{3}$$

where $\xi_i(t)$ is the component of random noise with the properties

$$\langle \xi_i(t) \rangle = 0 \tag{4}$$

$$\langle \xi_i(t_0)\xi_j(t) \rangle = 2\zeta k_B T \delta_{ij} \delta(t - t_0) \ . \tag{5}$$

To mimic the geometry of a growing moss tip, we imposed boundary conditions constraining the fluid to a hemisphere-capped cylinder. When a particle hits one of the boundaries, it is reflected elastically. The boundary check is repeated in the event that a particle collides with a boundary more than once in a single time step.

B. Computational Imaging

Fluorescence imaging relies on exciting a fluorescent object with light at one wavelength, and then imaging the light that it emits at a different wavelength. In a confocal microscope, both illumination and imaging are done through the same objective optics by scanning a laser beam. This means that only a single point can be imaged at a time, improving resolving power, but forcing the microscope to raster across the entire image.

We model the lens optics of the microscope as producing a Gaussian beam that depends on the wavelength, λ, and the numerical aperture, NA, of the microscope.

To render microscope output images, it is assumed that the intensity of the light emitted by a particle is linear with the intensity of the excitation light it experiences. This means that the intensity at a point R, $I(R)$, is the sum of the contributions from each particle at point P, each of which is a product of the intensity the point experiences from the excitation beam and the intensity the microscope sees from the point:

$$I(R) = \sum_{i=0}^{N} I_{\lambda_{\text{absorb}}}(P_i - R) I_{\lambda_{\text{emit}}}(P_i - R) \ . \tag{6}$$

Here, we use the fact that the Gaussian beam intensity is an even function.

The physical microscope rasters across lines over a finite period of time. This process is imitated by rendering one line at a time, with a time step between each. When processing a line, an iteration over each particle finds the nearest point on that line, calculates the image intensity contribution from that point, and then repeats this process, moving away from the starting point until the contribution from that particle falls below the image granularity.

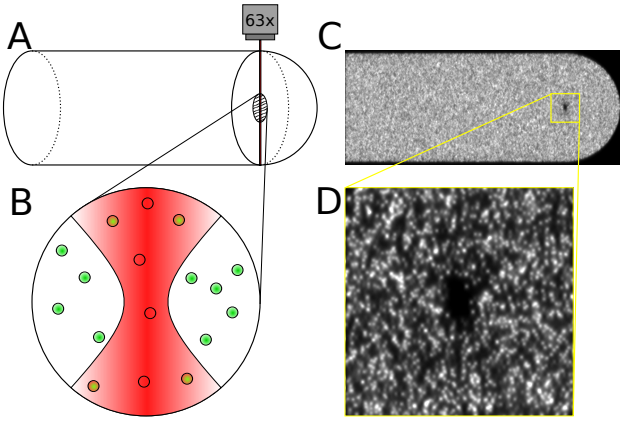

Figure 1. Overview of system: A) A schematic diagram of the "moss geometry" used for simulations. A slice is taken and expanded in B, showing the effect of the laser photobleaching particles. B) Zoomed in slice from A, showing the focused Gaussian form of the laser beam. Particles outside the beam are unaffected, while particles inside the beam are stochastically bleached depending on the intensity they experience. C) A computationally produced image of the entire moss geometry. With sufficient particles, it is impossible to distinguish a single particle. The highlighted area has a hole, from a single point of laser bleaching. D) Due to the properties of the laser beam, the effects of the photobleaching beam are well confined in both horizontal and vertical axes.

C. Modeling Photobleaching

We assume that the photobleaching laser is also a Gaussian beam, with a much higher power (See Fig. 1). We also assume that the probability of a given fluorescent particle bleaching is directly proportional to the intensity of the light it experiences. Since bleaching, like imaging, is accomplished by rastering across the bleach region in a finite time, we treat the bleaching of a single region as a set of discrete line-bleach events at different times. For efficiency reasons, we convolve the Gaussian beam along the bleach line, giving a single functional form describing the line-bleach-event.

III. GPU Implementation

A. Simulation overview

The whole simulation consists of three parts: initialization, a main loop and finalization. Before we can compute anything, initial values of variables must be generated and memory space must be allocated both on the CPU and GPU. The main loop is where the actual simulation takes place. During each iteration, the particles are moved, and (if the descriptors require it) photobleached or imaged. The finalization part is relatively simple. No special operations are needed and the only thing necessary is to free the allocated memory. Fig. 2 shows the structure and code flow of the whole simulation.

B. Initializing the simulation

We initialize the particles with a CPU function because its execution time is negligible when compared to the whole simulation. The data is then copied to GPU memory and stays there for the rest of the simulation to avoid expensive data transfer time between CPU and GPU. The FRAP descriptors and image descriptors are also initialized on the CPU for

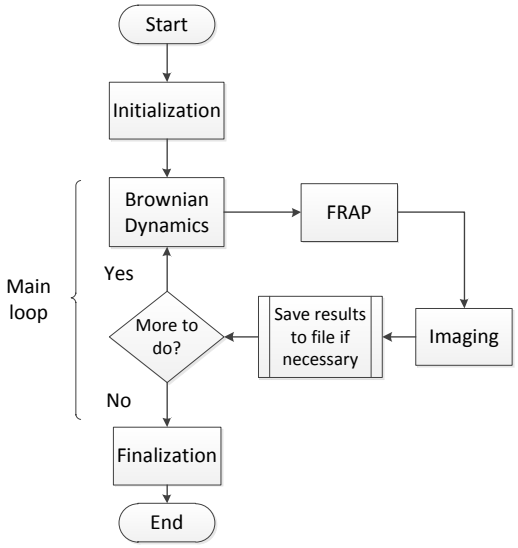

Figure 2. Simulation overview.

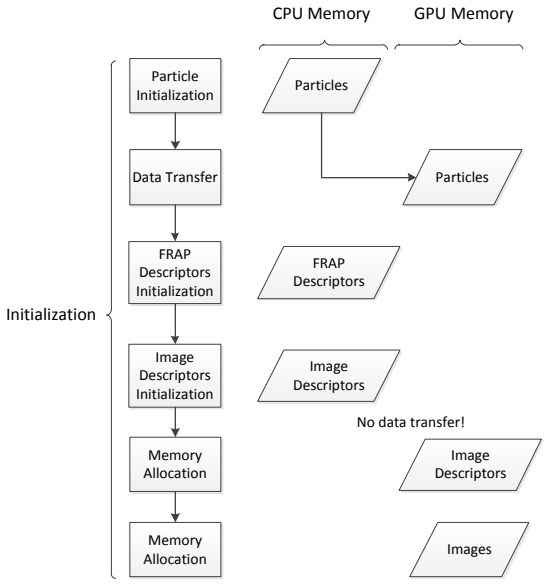

Figure 3. Data transfer and memory allocation in initialization.

similar reasons. We use a GPU kernel to initialize the GPU seeds for random number generating functions so that each thread has its own unique seed. GPU memory space for other data is also allocated, such as image descriptors and images. However, these are not immediately transferred; they will be transferred on demand. FRAP descriptors are used as control variables in CPU functions and are not need in GPU execution. Image descriptors are used in GPU kernels but they are transferred during the imaging kernel step. Fig. 3 shows the data transfer and memory allocation in initialization.

C. GPU kernels

There are three GPU kernel functions in our main loop: Brownian Dynamics kernel, FRAP kernel and Imaging kernel. GPU kernel functions are launched by the CPU and executed by the GPU. Certain CPU functions are used for controlling the code flow and kernel launches.

The Brownian Dynamics kernel is launched every time step in the main loop. It runs with one thread per particle and each thread calls two device functions. The first device function updates the pseudo-velocity of each particle, while the second moves the particle, updates its position, and checks for any collisions with walls. It may be called multiple times inside a while loop.

The FRAP kernel runs with one thread per particle. The FRAP descriptors control the launches of the FRAP kernel in a CPU function. When a descriptor calls for a FRAP event to happen, it passes the appropriate parameters to the FRAP kernel, which updates the particles on GPU without a formal memory copy.

The imaging kernel runs with one thread per particle. The image descriptors control the launches of the imaging kernel and are updated by the CPU function after each launch. Meanwhile, the imaging kernel needs to use the image descriptors for computation. Therefore, data transfer is needed every time before launching imaging kernel. This slows down the program not only because of the data transfer itself but also the synchronization before and after it. This can be further optimized using the techniques that are explained in Section IV. The imaging kernel sums up the contributions to the image from every particle. In order to avoid race conditions, the atomic add operation is used. Fig. 4 shows how FRAP kernel and imaging kernel work.

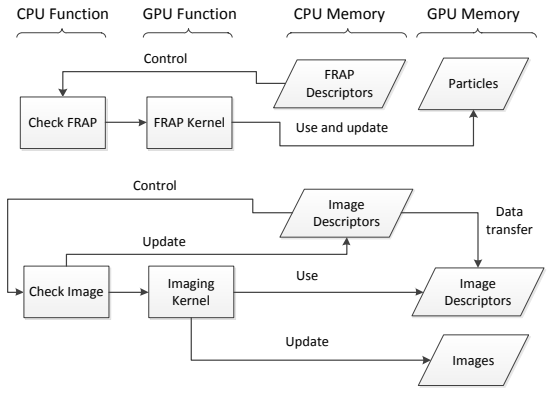

Figure 4. FRAP and imaging kernels

IV. GPU OPTIMIZATIONS

Since both a large part of our execution time and algorithmic complexity is spent on the imaging part of the algorithm, we focused on optimizing its performance.

A. Bucket Sort

While any given image line will only have contributions from a relatively small portion of the total particles, it is unknown which particles will contribute to a specific line. This means that only a small fraction of the render threads in a warp will go through their render loop, increasing branch inefficiency. NVidia's CUDA architecture runs warps of 32 threads in parallel, executing the same instruction simultaneously. When one thread has to take a branch, the other threads must wait for it to finish, a process called warp divergence.

To avoid this problem, we would like to coalesce particles to their relative locations, such that entire warps will either all need to be rendered, or all skip rendering. We do this by sorting the particles in order of their positions in the directions perpendicular to imaging. We only need the particles to be coalesced into approximate order however – a perfect ordering is unnecessary and excessively time consuming. Thus, we use a variant on the bucket sort where we skip the final sorting step. This is implemented by first counting how many elements will be in each bucket, which is then run through a cumulative sum function to get a list of offsets for each bucket. Each element can then be directly inserted into its new position in the destination array, giving a coarse-grained sort in $O(n)$ time.

B. Asynchronous kernel execution in streams

An additional avenue of optimization is the minimization of scheduling delays between kernel executions. This can be mitigated by taking advantage of CUDA streams and asynchronous operations [20]–[22].

For standard imaging, this was done by breaking the set of particles into two sets. Since every particle is independent, the GPU can start the second step on the first set of particles before it finishes the first step on the second set. However, this provides a minimal performance benefit as the GPU can fully occupy itself with a single kernel launch. Additionally, the GPU already has a very short kernel scheduling overhead, and removing it accomplishes little. For the full z-stack imaging, however, the optimization is both more simple and effective. As a large number of imaging kernels will be run in a single time step, they can be asynchronously queued in independent streams, and only synchronize with the CPU when the completed image is copied back to the host and written to disk.

C. Floating point imaging

A final GPU-specific optimization was the use of single-precision floating point math in the imaging routines. On the CPU there is effectively no difference in speed between double and single precision arithmetic, but on the GPU there is a large difference. As experimental images are eight to twelve-bit, single-precision floating point has enough accuracy to produce comparable quality images, and so quality is not sacrificed in this process. Thus, changing over all of the imaging math to single precision had no ill effects, and gave a noticeable speedup, as shown in the Tables V-A and V-B.

V. Performance Evaluation

We measure the computing performance using an NVidia Geforce GTX 780 Ti graphics card for the various versions of the GPU code, and compare it to the performance on an Intel Xeon E5620 as a CPU baseline. The CPU baseline is a well-optimized naive implementation on a single thread. It is compiled with native CPU optimizations, for optimal performance on the specific hardware on which is it executed. The nature of the algorithm and its practical use means that while it could be efficiently parallelized, there is little benefit in comparison to simply executing multiple copies of the software in parallel.

A. Experimental Time-series Simulation performance

Standard experimental simulation runs alternate between movement and either a single photobleach or imaging step. This means that all parts are of comparable import, and optimizations such as the pre-imaging sort are less effective.

Benchmarks are done with 1,000,000 particles in a $20\mu m$ long cylinder capped with a $5\mu m$ radius cylinder. The system is photobleached twice, for a total of 512 lines, and imaged in 160 images, each 256x106 pixels. Each photobleaching and imaging line is done on a separate time-step.

Table I
Execution time to run an experimental time-series job for 1,000,000 particles

Version	Time (s)	Speedup	Speedup (cumulative)
CPU	10728.5		
GPU (baseline)	1538.8	6.9	6.9
GPU (2D sorted, 64x128)	271.3	5.7	39.5
GPU (single precision)	179.9	1.5	59.6

B. Full z-stack performance

Because rendering a full z-stack for every desired time step heavily skews the computational load towards the imaging routine, the performance profile of the full z-stack render jobs depends almost entirely on the imaging part. This means that certain expensive optimizations (such as the sorting step) are more effective.

Benchmarks are done using the most optimized GPU version (floating-point imaging) with 1,000,000 particles in a $20\mu m$ long cylinder capped with a $5\mu m$ radius cylinder. The system is imaged in 106 images, each 256x106 pixels. All imaging is done in a single time step.

The performance of the simulation depends on its parameters. Fig. 5 and Fig. 6 show the effects of varying both the size and horizontal to vertical ratio of the 2D sorting grid. These show that a larger grid improves speed more than a small grid, although the upper limit is due to hardware limitations. In addition, as illustrated in Fig. 7, the efficiency increases as the number of particles is increased.

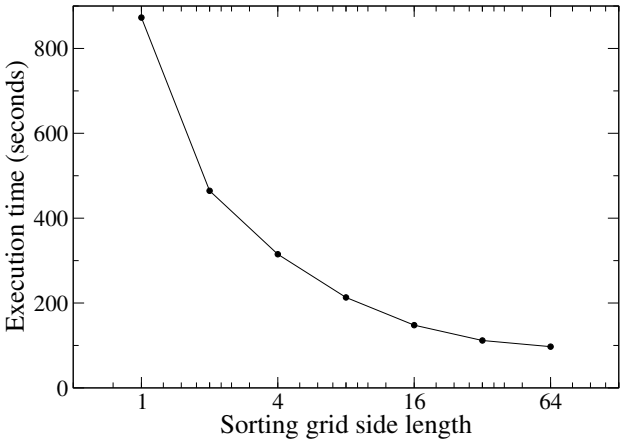

Figure 5. Execution time for a 1,000,000 particle z-stack using square grid sizes with the floating-point GPU version

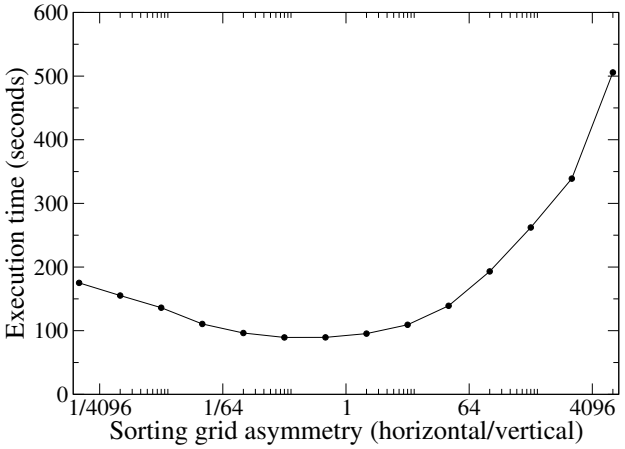

Figure 6. Execution time for a 1,000,000 particle z-stack using asymmetric grid sizes with 8192 slots using the floating-point GPU version

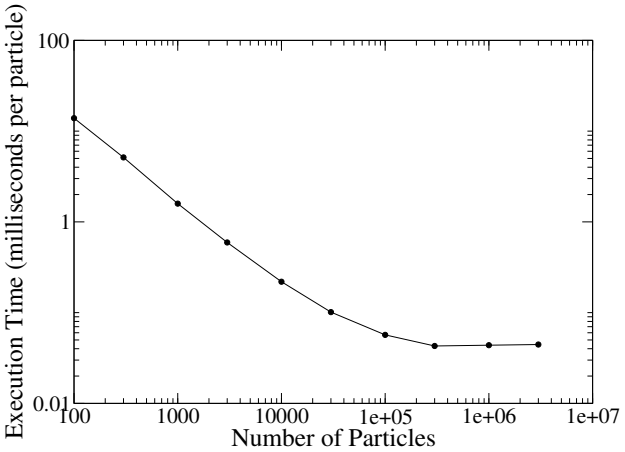

Figure 7. Execution time for varying numbers of particles in a z-stack using optimal grid size using the floating-point GPU version

Table II
EXECUTION TIME TO RUN A Z-STACK FOR 1,000,000 POINTS

Version	Time (s)	Speedup	Speedup (cumulative)
CPU	4229.6		
GPU (baseline)	497.0	8.5	
GPU (2D sorted: 64x128)	89.5	5.9	47.3
GPU (parallel imaging)	84.5	1.1	50.0
GPU (single precision)	41.7	2.0	101.4

VI. CONCLUSIONS

The computational power delivered by modern GPUs can be effectively used to provide a cost-efficient way of doing large-scale scientific simulation. We implement an experimentally relevant simulation model, namely FRAP, using NVidia CUDA and get a speed improvement of up to 100x over conventional CPU-based methods. Finally, we demonstrate the pitfalls of warp divergence and the benefits of using single-precision over double-precision floating point arithmetic, as well as additional techniques by which independently parallel GPU algorithms can be improved. While our model implements the specifics of our experimental system – moss tip cells in a confocal microscope – our approach can be easily generalized to other complex cellular geometries and experimental setups.

VII. ACKNOWLEDGMENTS

We acknowledge support from WPI startup funds, the WPI Alden Fellowship, and NSF CBET 1309933. LV acknowledges support from NSF-MCB 1253444.

REFERENCES

[1] M. Hülskamp, C. Pfeifle, and D. Tautz, "A morphogenetic gradient of hunchback protein organizes the expression of the gap genes krüppel and knirps in the early *Drosophila* embryo," *Nature*, vol. 346, no. 6284, pp. 577–580, 1990.

[2] J. Friml, A. Vieten, M. Sauer, D. Weijers, H. Schwarz, T. Hamann, R. Offringa, and G. Jürgens, "Efflux-dependent auxin gradients establish the apical–basal axis of *Arabidopsis*," *Nature*, vol. 426, no. 6963, pp. 147–153, 2003.

[3] J. Dong, C. A. MacAlister, and D. C. Bergmann, "BASL controls asymmetric cell division in *Arabidopsis*," *Cell*, vol. 137, no. 7, pp. 1320–1330, 2009.

[4] L. Luo, Y. J. Liao, L. Y. Jan, and Y. N. Jan, "Distinct morphogenetic functions of similar small GTPases: *Drosophila* drac1 is involved in axonal outgrowth and myoblast fusion." *Genes & Development*, vol. 8, no. 15, pp. 1787–1802, 1994.

[5] J. Yi, X. S. Wu, T. Crites, and J. A. Hammer, "Actin retrograde flow and actomyosin II arc contraction drive receptor cluster dynamics at the immunological synapse in Jurkat T cells," *Molecular biology of the cell*, vol. 23, no. 5, pp. 834–852, 2012.

[6] A. Adams, D. I. Johnson, R. M. Longnecker, B. F. Sloat, and J. R. Pringle, "CDC42 and CDC43, two additional genes involved in budding and the establishment of cell polarity in the yeast *Saccharomyces cerevisiae*." *The Journal of Cell Biology*, vol. 111, no. 1, pp. 131–142, 1990.

[7] S. Seiler and M. Plamann, "The genetic basis of cellular morphogenesis in the filamentous fungus *Neurospora crassa*," *Molecular biology of the cell*, vol. 14, no. 11, pp. 4352–4364, 2003.

[8] B. Kost, E. Lemichez, P. Spielhofer, Y. Hong, K. Tolias, C. Carpenter, and N.-H. Chua, "Rac homologues and compartmentalized phosphatidylinositol 4, 5-bisphosphate act in a common pathway to regulate polar pollen tube growth," *The Journal of cell biology*, vol. 145, no. 2, pp. 317–330, 1999.

[9] A. J. Molendijk, F. Bischoff, C. S. Rajendrakumar, J. Friml, M. Braun, S. Gilroy, and K. Palme, "*Arabidopsis thaliana* Rop GTPases are localized to tips of root hairs and control polar growth," *The EMBO journal*, vol. 20, no. 11, pp. 2779–2788, 2001.

[10] G. Schmiedel and E. Schnepf, "Polarity and growth of caulonema tip cells of the moss *Funaria hygrometrica*," *Planta*, vol. 147, no. 5, pp. 405–413, 1980.

[11] D. Soumpasis, "Theoretical analysis of fluorescence photobleaching recovery experiments," *Biophysical journal*, vol. 41, no. 1, pp. 95–97, 1983.

[12] R. B. Phillips, J. Kondev, J. Theriot, N. Orme, and H. Garcia, *Physical biology of the cell*. Garland Science New York, 2009.

[13] I. F. Sbalzarini, A. Mezzacasa, A. Helenius, and P. Koumoutsakos, "Effects of organelle shape on fluorescence recovery after photobleaching," *Biophysical journal*, vol. 89, no. 3, pp. 1482–1492, 2005.

[14] V. González-Pérez, B. Schmierer, C. S. Hill, and R. P. Sear, "Studying smad2 intranuclear diffusion dynamics by mathematical modelling of frap experiments," *Integrative Biology*, vol. 3, no. 3, pp. 197–207, 2011.

[15] B. L. Sprague and J. G. McNally, "Frap analysis of binding: proper and fitting," *Trends in cell biology*, vol. 15, no. 2, pp. 84–91, 2005.

[16] J. G. McNally, "Quantitative FRAP in analysis of molecular binding dynamics *In Vivo*," *Methods in cell biology*, vol. 85, pp. 329–351, 2008.

[17] L. Vidali and M. Bezanilla, "*Physcomitrella patens*: a model for tip cell growth and differentiation," *Current opinion in plant biology*, vol. 15, no. 6, pp. 625–631, 2012.

[18] F. Furt, Y.-C. Liu, J. P. Bibeau, E. Tüzel, and L. Vidali, "Apical myosin XI anticipates F-actin during polarized growth of *Physcomitrella patens* cells," *The Plant Journal*, vol. 73, no. 3, pp. 417–428, 2013.

[19] Z. Chen, J. Kingsley, X. Huang, and E. Tuzel, "Accelerating a novel particle-based fluid simulation on the gpu," in *High Performance Extreme Computing Conference (HPEC), 2013 IEEE*. IEEE, 2013, pp. 1–6.

[20] N.-P. Tran, M. Lee, S. Hong, and D. Choi, "Multi-stream parallel string matching on kepler architecture," in *Mobile, Ubiquitous, and Intelligent Computing*, ser. Lecture Notes in Electrical Engineering, J. J. J. H. Park, H. Adeli, N. Park, and I. Woungang, Eds. Springer Berlin Heidelberg, 2014, vol. 274, pp. 307–313.

[21] H. Khatter and V. Aggarwal, "Efficient parallel processing by improved cpu-gpu interaction," in *Issues and Challenges in Intelligent Computing Techniques (ICICT), 2014 International Conference on*, Feb 2014, pp. 159–161.

[22] *CUDA C PROGRAMMING GUIDE*, 5th ed., NVIDIA, July 2013.