UNIVERSITY OF CALIFORNIA
Santa Barbara

Spatial Stochastic Simulation of Biochemical Systems

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Brian J. Drawert

Committee in Charge:

Professor Linda R. Petzold, Chair

Professor Mustafa Khammash, Co-Chair

Professor John R. Gilbert

March 2013

The Dissertation of
Brian J. Drawert is approved:

Professor John R. Gilbert

Professor Mustafa Khammash, Committee Co-Chairperson

Professor Linda R. Petzold, Committee Chairperson

December 2012

Spatial Stochastic Simulation of Biochemical Systems

by

Brian J. Drawert

Acknowledgements

I would like to thank my Ph.D advisor, Professor Linda Petzold, for her guidance, support, and mentorship. I also thank my dissertation co-chair, Professor Mustafa Khammash and committee member Professor John Gilbert, for their helpful insight and support.

I would like to thank my collaborators Michael J. Lawson, Andreas Hellander, Tau-Mu Yi, Chris Bunch, Marc Griesemer, Cherie Briggs, Stefan Engblom, and P-O Östberg for all of their hard work.

I would also like to thank Dan Gillespie, for both his pioneering work, and for our stimulating work together.

I am indebted to many of my colleagues for providing a stimulating and fun environment in which I learned and grew tremendously. I am especially grateful to Bernie Daigle, Min Roh, Kevin Sanft, Ben Bales, Rone Kwei Lim, and Stefan Hellander.

I gratefully acknowledge financial support from the National Science Foundation through an IGERT Fellowship in Computational Science and Engineering, the Department of Computer Science at UCSB, and the UCSB Institute for Collaborative Biotechnologies, the National Science Foundation, the National Institute of Health, and the Department of Energy.

Finally, I would especially like to thank Maggie Drawert for her love, patience, understanding, and encouragement.

Curriculum Vitæ
Brian J. Drawert

Education

2013 (expected)	Doctor of Philosophy in Computer Science, University of California, Santa Barbara
	Emphasis: Computational Science and Engineering
2007	Masters of Science in Physics, Depaul University
2001	Bachelor of Science in Computer Science, Illinois Institute of Technology

Experience

2002-2007	Software Contractor, MD Records (Chicago, IL)
2003-2004	Software Engineer, Leapfrog Online (Evanston, IL)
2001	Software Engineer, Period 7 (Chicago, IL)
2001	Software Engineer, Vicode (Chicago, IL)
1999-2001	Software Engineer, L90 (Chicago, IL)
1998-1999	Web Engineer, MyPoints.com (Schaumburg, IL)

Publications

B. Drawert, M. Lawson, L. Petzold, M. Khammash "The Diffusive Finite State Projection Algorithm for Efficient Simulation of the Stochastic Reaction-Diffusion Master Equation" *Journal of Chemical Physics* (Feb 2010)

C. Bunch, B. Drawert, N. Chohan, C. Krintz, L. Petzold, K. Shams "Language and Runtime Support for Automatic Configuration and Deployment of Scientific Computing Software over Cloud Fabrics" *Journal of Grid Computing* (Mar 2012)

B. Drawert, S. Engblom, A. Hellander "URDME: a modular framework for stochastic simulation of reaction-transport processes in complex geometries" *BMC Systems Biology* (Jun 2012)

M. Lawson, B. Drawert, M. Khammash, L. Petzold, T-M. Yi "Stochastic Spatial Dynamics Enable Robust Cell Polarization" *PLOS Computational Biology* (2013) *(under review)*

P-O Östberg, A. Hellander, B. Drawert, E. Elmroth, S. Holmgren, L. Petzold "Abstractions for Scaling eScience Applications to Distributed Computing Environments; A StratUm Integration Case Study in Molecular Systems Biology" In Proceedings of: *Bioinformatics 2012, International Conference on Bioinformatics Models, Methods, and Algorithms* (Feb 2012)

P-O Östberg, A. Hellander, B. Drawert, E. Elmroth, S. Holmgren, L. Petzold "Reducing Complexity in Management of Scientific Computations" In Proceedings of: *CCGrid 2012, The 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing* (May 2012)

C. Bunch, B. Drawert, M. Norman "MapScale: A Cloud Environment for Scientific Computing" Technical Report, *University of California, Computer Science Department* (Jun 2009)

B. Drawert, S. Engblom A. Hellander "URDME v. 1.1: Users manual". Technical Report 2011-003, *Department of Information Technology, Uppsala University* (Mar 2011)

Invited Talks and Colloquia

"Polarized Spatial Stochastic Amplification During Mating in *Saccharomyces cerevisiae*" *The Society for Mathematical Biology* Knoxville, TN (July 2012)

"Polarization in Yeast Mating: Modeling and Simulation of Spatial Stochastic Phenomena" *Scientific Computing Seminar - Uppsala University* Uppsala, Sweden (Mar 2011)

"Polarized Spatial Stochastic Amplification During Mating in *Saccharomyces cerevisiae*" *International Conference on Systems Biology* Edinburgh, Scotland (Oct 2010)

"Spatial Stochastic Simulation of Biochemical Systems" *Theoretical Ecology Seminar* UCSB (Mar 2010)

Presentations

"Efficient Stochastic Simulation of Spatially Inhomogeneous Biochemical Systems" B. Drawert, M. J. Lawson, A. Hellander, M. Khammash, L. Petzold, *Southern California Systems Biology Conference* (Jan 2012)

"Efficient Stochastic Simulation of Spatially Inhomogeneous Biochemical Systems" B. Drawert, M. J. Lawson, A. Hellander,

M. Khammash, L. Petzold, *Southern California Systems Biology Conference* (Jan 2011)

"Efficient Stochastic Simulation of Spatially Inhomogeneous Biochemical Systems" B. Drawert, M. J. Lawson, A. Hellander, M. Khammash, L. Petzold, *Gordon Research Conference on Stochastic Physics in Biology* (Jan 2011)

"Polarized Spatial Stochastic Amplification During Mating in *Saccharomyces cerevisiae*" B. Drawert, M. J. Lawson, L. Petzold, M. Khammash, T-M. Yi, *ICSB* (Oct 2010)

"Stochastic Modeling and Simulation of Cell Polarization During Mating in Budding Yeast" B. Drawert, M. J. Lawson, T-M. Yi, L. Petzold, M. Khammash, *IGERT Project Meeting* (May 2010)

"Stochastic Modeling and Simulation of Cell Polarization During Mating in Budding Yeast" B. Drawert, M. J. Lawson, T-M. Yi, L. Petzold, M. Khammash, *Q-Bio* (Aug 2009)

Honors and Awards

2007	NSF IGERT Fellowship in Computational Science and Engineering
2001	Outstanding Community Contribution Award, IIT
1996	Heald merit scholarship, IIT

Abstract

Spatial Stochastic Simulation of Biochemical Systems

Brian J. Drawert

Recent advances in biology have shown that proteins and genes often interact probabilistically. The resulting effects that arise from these stochastic dynamics differ significantly than traditional deterministic formulations, and have biologically significant ramifications. This has led to the development of computational models of the discrete stochastic biochemical pathways found in living organisms. These include spatial stochastic models, where the physical extent of the domain plays an important role; analogous to traditional partial differential equations.

Simulation of spatial stochastic models is a computationally intensive task. We have developed a new algorithm, the Diffusive Finite State Projection (DFSP) method for the efficient and accurate simulation of stochastic spatially inhomogeneous biochemical systems. DFSP makes use of a novel formulation of Finite State Projection (FSP) to simulate diffusion, while reactions are handled by the Stochastic Simulation Algorithm (SSA). Further, we adapt DFSP to three dimensional, unstructured, tetrahedral meshes in inclusion in the mature and widely usable systems biology modeling software URDME, enabling simulation of the complex geometries found in biological systems. Additionally, we extend DFSP

with adaptive error control and a highly efficient parallel implementation for the graphics processing units (GPU).

In an effort to understand biological processes that exhibit stochastic dynamics, we have developed a spatial stochastic model of cellular polarization. Specifically we investigate the ability of yeast cells to sense a spatial gradient of mating pheromone and respond by forming a projection in the direction of the mating partner. Our results demonstrates that higher levels of stochastic noise results in increased robustness, giving support to a cellular model where noise and spatial heterogeneity combine to achieve robust biological function. This also highlights the importance of spatial stochastic modeling to reproduce experimental observations.

Contents

Acknowledgements **iv**

Curriculum Vitæ **vi**

Abstract **ix**

List of Figures **xiv**

1 Introduction **1**
1.1 Outline 4

2 Simulation of Spatially Inhomogeneous Discrete Biochemical Systems **5**
2.1 Discrete Biochemical Simulation 7
2.1.1 Chemical Master Equation and the SSA 7
2.1.2 The Finite State Projection Algorithm 9
2.2 Spatially Inhomogeneous Systems 11
2.2.1 Reaction-Diffusion Master Equation 11
2.2.2 Spatial Stochastic Simulation Algorithms 14

3 The Diffusive Finite State Projection Algorithm for Efficient Simulation of the Stochastic Reaction-Diffusion Master Equation **16**
3.1 The Diffusive FSP Method 17
3.1.1 DFSP 18
3.1.2 Adaptive Step Splitting 21
3.1.3 Detailed Algorithm Descriptions 24
3.2 Examples and Analysis 28

3.2.1 Diffusion Example 28
3.2.2 G-protein Cycle Example 39
3.3 Conclusions 47

4 URDME: a Modular Framework for Stochastic Simulation of Reaction-Transport Processes in Complex Geometries 48
4.1 Background 50
4.2 Implementation 54
4.3 Results 62
4.3.1 Simulating Min oscillations in *E. Coli* 63
4.3.2 Developing and benchmarking a new algorithm for spatial stochastic simulation 67
4.3.3 Active transport in a neuron 70
4.4 Discussion 77

5 Adaptive Accelerated Spatial Stochastic Simulation on NVIDIA graphics processing units 91
5.1 Overview of the DFSP Algorithm 92
5.2 Computation of DFSP lookup tables via Uniformization 95
5.3 Estimation of the Operator-Splitting Error 100
5.4 Adaptive DFSP algorithm 102
5.4.1 Implementation in URDME 108
5.4.2 Parallel GPU implementation 109
5.5 Numerical experiments 118
5.5.1 Polarization of *S. cerevisiae* 118
5.5.2 Min oscillations in *E. coli* 121
5.6 Discussion 129

6 Spatial Stochastic Modeling of Cell Polarization 133
6.1 Background 135
6.2 Model Description 140
6.2.1 Model Structure 141
6.2.2 Parameter Estimation 146
6.3 Results 150
6.3.1 Spatial Stochastic Amplification 150
6.3.2 Tracking of a Dynamic Input 151
6.3.3 Robustness to Parameter Perturbation 155
6.3.4 Stochastic Simulations Reproduce the Mutant Phenotype . 159
6.4 Discussion 161
6.5 Materials and Methods 166

7 Conclusions **169**

Bibliography **174**

List of Figures

3.1 Adaptive step splitting error 24
3.2 Solution to a pure diffusion problem with a step function as the initial condition. Plotted is the state of the system at t = 0s (dashed blue), 500s (dotted black) and 7000s (solid red) for a stochastic trajectory. The domain is a circle with radius $2\mu m$, subdivided equally into 200 voxels. 30
3.3 Plot of the Normalized L_∞ error (maximum deviation from analytical solution) versus time in the mean (a) and variance (b) for varying ensemble sizes for both DFSP ($\tau_D = 0.1$s) and ISSA (voxel size of $0.06\mu m$), for an ensemble size of 10^3 trajectories. The error increases with time (as expected for a discretized solution) at the same rate for both DFSP and ISSA. Additionally, the error decreases (to the discretization error limit) with increasing ensemble size at the same rate for DFSP and ISSA. 32
3.4 Plot of the Normalized L_∞ error versus voxel size (both on a log-scale) calculated at $t = 100$s (a transient state) for an ensemble size of 10^3 trajectories. As voxel size decreases, the error in the mean decreases at the same rate for both DFSP and ISSA. The error in the variance shows a similar trend, however it also shows increased error for small voxel sizes. This is mostly likely sampling error due to a constant system population distributed into an increasing number of voxels. 33
3.5 The distribution distance for the diffusion example, the $Kmean$ at every 100s for 7000s for ISSA-vs-ISSA (green line with dots), ISSA-vs-DFSP with $\tau_D = 0.1$s (dashed red line) and $\tau_D = 1.9$s (blue line). In this plot the ensemble size is 10^5, at which point the DFSP solution becomes distinguishable from the ISSA solution for larger time steps. For ensemble sizes $\leq 10^4$, the DFSP solution is indistinguishable from the ISSA solution for both step sizes (results not shown). 35

3.6 Maximum number of molecules moved in a single diffusion step of DFSP versus τ_D for an error tolerance of 10^{-5}. For values of $\tau_D > 0.925$s, DFSP will try to move less than one molecule, thus violating the error tolerance. 37
3.7 Error_{τ_D} vs τ_D for DFSP applied to the diffusion example (line with circles) and G-protein cycle example (line with x's), linear fits (red dashed lines). For the diffusion example, the error is constant and only a function of the error tolerance. This is because there is no contribution to the error from the reaction operator. For the G-protein cycle example, the error increases linearly with τ_D and converges to the diffusion error as τ_D goes to zero. This is because the error in the reaction operator is linear with the timestep (τ_D); as the timestep goes to zero the reaction error goes to zero. 38
3.8 Spatial concentration of Ligand (top) and mean and variance of $G_{\beta\gamma}$ population (bottom) at $t = 100$s for a trajectory of the G-protein cycle example. The Ligand gradient is the input to this model and is constant in time. $G_{\beta\gamma}$ is the output and is time varying. 40
3.9 The distribution distance for the G-Protein cycle example, the $Kmean$ from (3.3) at every 100s for 7000s for ISSA-vs-ISSA (green line with dots), ISSA-vs-DFSP with $\tau_D = 0.1$s (dashed red line) and $\tau_D = 1.9$s (blue line). In this plot the ensemble size is 10^5, at which point the DFSP solution becomes distinguishable from the ISSA solution for larger time steps. For ensemble sizes $\leq 10^3$, the DFSP solution is indistinguishable from the ISSA solution for both stepsizes (results not shown). 42
3.10 Speedup of DFSP over ISSA and both MSA stencils (subscript 1 denotes the stencil including adjacent voxels, and 2 the stencil including the two nearest neighbors on either side) for varying values of τ_D for the G-protein cycle example. DFSP achieves significant performance increases over ISSA and both MSA stencils for reaction-diffusion simulation. The speedup is due in part to the number of times the reaction propensity function needs to be updated due to diffusive transfers. DFSP updates less often than ISSA or MSA. As τ_D increases, the number of updates decreases and performance increases. 45
3.11 Computation time for DFSP with varying values of τ_D and varying numbers of voxels in the system with error tolerance of 10^{-5} for the diffusion example. ISSA computation times for each of the system sizes is provided for comparison. 46

4.1 The URDME framework consists of three loosely coupled layers. Solvers reside at the bottom level and are most often written in a compiled language like ANSI-C. The middle layer provides for interfaces between the solvers and the top-level mesh-generation infrastructure. Both the top- and the bottom-layer may be replaced by other software as long as the middle level is extended appropriately. 56
4.2 **(A)** URDME flow diagram for the complete simulation process. **(B)** Process flow diagram for the stochastic simulation step of (A) using the NSM solver. **(C)** DFSP solver flow diagram, an alternative to (B) for the stochastic simulation step. 58
4.3 **(A)** Geometry and mesh modeling of an *E. Coli* cell. **(B)** Temporal average concentration of MinD protein as a function of position along the long axis of the *E. Coli* cell (top), and the time series plot of the oscillations. **(C)** Six *E. Coli* cells of increasing lengths, as specified in the parameter sweep described in Table 4.1. The color intensity shows the temporal average concentration of MinD protein along the membrane. **(D)** Parameter sweep shows how the relative concentration of MinD changes as the bacterium grows. 66
4.4 DFSP benchmark results. **(A) Performance of DFSP** shows a comparison of simulation times for DFSP at varying τ_D values (red) and NSM (blue), and the DFSP speedup factor (green). For this model, DFSP outperforms NSM for $\tau_D > 0.01$. **(B) Error in DFSP** shows the relative error in MinCDE oscillation period (red) and the oscillation patterns for three simulations. Simulations with $\tau_D < 0.1$ produces coherent oscillation patterns and result in a negligible error. The system was simulated to a final time 900s. Simulations were performed on a 1.8 Ghz Intel Core i7 processor. 71

4.5 The neuron geometry **(A)** is based on a artistic CAD rendering generated with the public domain version of the software Blender (http://www.blender.org). In order to conduct simulations in this geometry, the model was exported in the STL surface mesh format, imported into the open-source meshing package Gmsh [47], where the boundary was re-parametrized and the domain subsequently meshed with a volume mesh in 3D. The resulting mesh was then converted into a Comsol Multiphysics 3.5a model to serve as a geometry description for the URDME model. Assembly of active transport jump rate constants were conducted by URDME on the unstructured mesh shown in **(B)**. For a mathematical background on how to obtain these constants on the unstructured mesh, see [60]. URDME's capability to use an unstructured mesh made up of tetrahedral and triangular elements is of vital importance in order to be able to resolve the complex geometry of the neuron. 73
4.6 Normalized concentration of cargo V in the soma (green), axon (blue) and dendrites (red) as a function of time. Initially, the parameters satisfy $\sigma_{kd} = 10\sigma_{dk}$ and cargo localizes to the axon due to the larger fraction of time spent in the kinesin binding state. At time $t = 0.5$ the situation is reversed, and the localization of V shifts from axon to dendrites. The red regions in the inlays depicting the neuron show the areas where V is present. 75
4.7 Performance comparison of the three software packages for an increasing number of voxels. Each point shows the mean and the error bars show the standard deviation of a ensemble of $N = 5$ runs. For URDME the number of voxels represents the number of mesh vertices, for MesoRD it represents the number of cubic subvolumes, and for STEPS it represents the number of tetrahedrons. All simulations were performed on a 1.8 GHz Intel core i7 processor with 4GB of memory. 85

5.1 Process flow of the Adaptive DFSP algorithm. The first step is taken with NSM to move the system out of the initial state. Next the local splitting error is calculated in each voxel (as described above), and the maximum value of τ_D is calculated so that it is accurate for the given error tolerance. Then SSA reaction steps are executed until time $t + \tau_D$ and DFSP diffusion steps are executed until time $t + \tau_D$, and a lookup table is generated if necessary. Next the local splitting error and value of τ_D for the next step are calculated. The loop continues until the simulation end time. 104

5.2 Process flow for the GPU solver. Boxes above the dashed line correspond to the components that are implemented in the CPU, and below the line are the components implemented in the GPU. Compare to Figure 5.1. 110
5.3 Performance analysis of the reaction and diffusion kernels. The number of events is linear in timestep for both the reaction and diffusion operators. The execution time of the SSA reaction kernel (solid red line) depends linearly on the time step, while the DFSP diffusion kernel is nearly constant. On the other hand, at least for this problem, the DFSP kernel is dominating the execution time. 112
5.4 (left) Uniformization memory usage, (right) Uniformization speed comparison. GPU implementation of the uniformization matrix exponential algorithm for generation of the DFSP lookup tables. This shows the limitations of the GPU architecture, as the GPU card that we used (GeForce GTX 560) has 1GB of memory. Thus, ADFSP is unable to use the static uniformization method (blue line) as it uses more memory than is available. Dynamic memory allocation uses far less total device memory (red). The total memory used is even less with the additionally allocated working memory (black). However, use of dynamic memory is significantly slower than static memory, and even slower than the serial CPU method for large problem sizes (right). 115
5.5 Comparison of GPU and CPU implementations of the operator splitting error estimate. The CPU implementation is more efficient than both GPU implementations, and thus is used in the final ADFSP method. 117
5.6 Min oscillations in *E. Coli*. Left: 3D tetrahedral mesh of the *E. coli*. Right: Time averaged concentration of MinD on the membrane. Note that the concentration is low in the center and high on the poles of the organism. This is due to the oscillatory nature of the MinCDE cycle. MinD acts as a contractile ring formation inhibitor, thus the contractile ring forms at the center of the cell where the concentration of MinD is lowest. This allows the *E. coli* cell to reliably undergo mitosis into two equal halves. 121
5.7 The figure shows the time steps selected by ADFSP (red dashed line) and the oscillation of the MinD protein (solid blue line). Note that the time steps oscillate with a period that is approximately half the period of the MinD oscillation. When MinD is at its peak, most of the MinD protein in the system has bound to the membrane at the left pole of the *E. coli*. Similarly, when the pattern is at its trough, most of the MinD has bound to the membrane at the right pole. 123

5.8 Performance versus error tolerance for the ADFSP implementations on CPU and GPU, along with NSM for comparison. As the error tolerance is loosened, the run time for both the ADFSP (CPU) and ADFSP_GPU methods decreases. These results are from the MinCDE model with medium discretization (1009 voxels). For an error tolerance of 1e-1, ADFSP is 3 times faster than NSM and ADFSP_GPU is 4.3 times faster. 124
5.9 Relative error in oscillation period versus error tolerance for ADFSP implementation on the CPU and GPU. As the error tolerance is loosened the error in oscillation period for both the ADFSP (CPU) and ADFSP_GPU methods increase. 126
5.10 Comparison of the oscillation in ADFSP and NSM. The ADFSP run used an error tolerance of 5e-2. For this value, the ADFSP algorithm is approximately the same speed as NSM for this problem. These results are for the MinCDE model with medium discretization (1009 voxels). 127
5.11 Comparison of the oscillation pattern in ADFSP, ADFSP_GPU and NSM for the MinCDE model with medium discretization (1009 voxels). The phase of each trajectory is adjusted so that the center peaks (closest to t=200) are aligned. The ADFSP runs use an error tolerance of 1e-3 (top left), 1e-2 (top right), and 1e-1 (bottom). Note that the coherence of the oscillation patterns degrades as the error tolerance is loosened. 128

6.1 **Spatial amplification in cell polarity during yeast mating (A)** Spatial amplification occurs in stages during cell polarization in yeast. The external spatial gradient of α-factor is shallow (gray), and it generates a comparable gradient of free G$\beta\gamma$ on the cell membrane. This initial internal gradient induces a polarized cap of active Cdc42 (green) which in turn localizes the tightly condensed polarisome (red) to the front of the cell. In this manner, a shallow external gradient is amplified to a steep internal gradient. **(B)** A schematic and microscopy image of two mating yeast cells with aligned punctate polarisomes. The polarisomes are labeled with Spa2-GFP (**a**-cell) and Spa2-RFP (α-cell). During mating the polarisomes at the tip of the mating projection are tightly localized and seek out one another until they are aligned and adjacent. When the projections meet the membranes and polarisomes fuse, and mating occurs. 137

6.2 **Diagram describing yeast polarisome model. (A)** Input driven recruitment of cytoplasmic Bni1 by membrane bound active Cdc42 (Cdc42a). **(B)** Bni1 on the membrane nucleates and polymerizes actin cables. **(C)** Actin cables direct transport of Spa2 from the cytoplasm to the membrane. **(D)** Spa2 provides positive feedback as it recruits cytoplasmic Bni1 to the membrane and inhibits actin depolymerization. 142
6.3 **Parameter estimation from experimental data including FRAP. (A)** Experimental FRAP recovery curves for Bni1 (Solid blue line: average of 5 experiments. Light blue area: 95% confidence interval around average. Dashed red: fit to exponential). Notice that the WT curve has a much shorter time to half recovery than the LatA-treated curve (time to half recovery indicated by dashed black lines). **(B)** FRAP simulation time to half recovery for varying diffusion rates and B_{off} (left) and B_{on} (right). There is no change with B_{on}, while for B_{off} as D goes to zero the curves approach the theoretical no-diffusion limit (dashed black). **(C)** Left: Cartoon illustrating the Bni1 temperature sensitive mutant experiment performed in [39] and simulated in this paper. Right: Phase plane of actin cable half-life (color-coded) as a function of A_{off} and K_{m} (simulation of the experiment in [39]), with the curve representing 45s (dotted black) and our model fit (dashed black). This phase plane represents the average of those generated for initial conditions corresponding to 10 different observed cells. 147
6.4 **Punctate Polarization (Green: Cdc42a, Red: Spa2). (A)** Yeast cells treated with α-factor show the wider Cdc42a (marked by Ste20-GFP) and tighter Spa2 polarization. **(B)** Visualization of a stochastic realization of the polarisome model (white indicates regions with actin cables). **(C)** Normalized fluorescence intensity membrane profiles of Ste20-GFP and Spa2-mCherry from a yeast cell undergoing polarisome formation (Green: Ste20 (Cdc42a). Dashed black: Ste20 fit. Red: Spa2). **(D)** Normalized membrane intensity profile from stochastic and deterministic realizations of polarisome model, the Cdc42a input, and the mean output of a stochastic ensemble ($n = 500$). Inset: Absolute membrane intensity profile from stochastic and deterministic realizations of polarisome model, and Spa2 ensemble mean. (Red: Spa2 Stochastic. Dashed blue: Spa2 deterministic. Black diamond: Spa2 ensemble mean. Green: Cdc42a (input)). 152

6.5 **Polarisome tracking of directional change in Cdc42a (Green: Ste20 (Cdc42a), Red: Spa2, Blue: Bni1). Left:** *In vivo* data. **Right:** *In silico* data. **Top row:** In both the cell **(A)** and the simulation **(B)**, the Cdc42a profile shifts first, followed by the the polarisome (indicated by Spa2). **Middle and bottom rows:** Spatial dynamics of Bni1 **(C, D)** and Spa2 **(E, F)** during polarisome tracking of Cdc42a. Note that the time scale of polarisome switching is similar between *in vivo* and *in silico* experiments, especially in the $\sim$ 10 minute overlap time when two polarisomes are present. 154
6.6 **Six polarization phenotype space plots of $B_{\mathbf{fb}}/B_{\mathbf{on}}$ ratio versus $K_{\mathbf{m}}$.** The first five panels show results from the stochastic model with varying Actin_t values of 20, 40, 60, 80, 100 (left to right); the final panel shows results from the deterministic model. K_{m} values (x-axis) range from 0 to 4000, $B_{\text{fb}}/B_{\text{on}}$ ratio (y-axis) ranges from 0 to 10. Blue indicates accurate tracking (>70% probability), red indicates narrow width (<22° FWHM), and purple indicates that both criteria are met. As the number of actin cables is increased, the stochastic phenotype plots converge to the deterministic plot. Lower actin cable number confers a larger region where both criteria are satisfied, indicating that increased stochasticity leads to more robustness to parameter variation. For each plot, the S_{on} parameter was adjusted to maintain a constant flux of Spa2 to the membrane. 158
6.7 **The multi-polarisome phenotype in *spa2*Δ cells. Columns:** WT phenotype (**left**), *spa2*Δ phenotype (**right**). **(A)** *In vivo* microscopy images of polarizing yeast cells marked with Sec3-GFP (top row) and Bni1-GFP (bottom row). Note the difference between the single punctuate polarisome (left) and the multi-polarisome phenotype (right). **(B)** *In silico* snapshots of yeast polarisome simulations for both stochastic (top row) and deterministic (bottom row) models showing Bni1. Note that only the stochastic *in silico* model is able to match the *in vivo* multi-polarisome phenotype. 160

6.8 **Stochastic versus deterministic polarization schematic time course:** Green is the input Cdc42a profile, Red is the output (Spa2 or Bni1). **Top:** Initially, in both the stochastic and deterministic simulations, all of the output protein is in the cytoplasm. **Middle:** After a short time period one molecule has been recruited to the membrane. In the stochastic simulation this addition takes place in one discrete location, whereas in the deterministic simulation the addition is in a continuous concentration gradient along the membrane. **Bottom:** This difference in allocation of molecules results in differing final profiles. In the stochastic case, feedback has recruited most of the output protein to the location of the first addition, whereas in the deterministic simulation output protein has been added smoothly along the membrane, resulting in a smooth final distribution. 163

Chapter 1

Introduction

Mathematical modeling and computer simulation have emerged as indispensable components in the scientific process. The construction of computer models to explain the behavior of natural processes often illuminates assumptions or is able predict previously unknown behaviors of these systems. This leads investigators to design laboratory experiments to test these limits, which in turn creates more complete models and advances the frontiers of knowledge.

The field of computational systems biology harnesses the synergy of the modeling driven experiment to understand the dynamical nature of biological processes. Systems biology is a field of study that focuses on the complex interactions within biological systems to develop a mechanistic understanding. Computational systems biology seeks to advance the understanding of biological processes through

modeling-driven experiment design. Our work focuses on stochastic chemical kinetics and spatial stochastic simulation in the context of biological systems.

In recent years it has become increasingly clear that stochasticity plays an important role in many biological processes. Some examples include bistable genetic switches (both endogenous [90, 91] and synthetically constructed [46, 57]), noise-enhanced robustness of oscillations [130, 30], and fluctuation-enhanced sensitivity or "stochastic focusing" [106]. In many cellular systems, small local populations can create stochastic effects even if total cellular levels are high [41]. A review in Nature noted that numerous cellular systems, including development, polarization and chemotaxis, rely on spatial stochastic noise for robust performance [133]. Additional examples include end-to-end oscillations in MinCDE in *E. coli* [66], spontaneous polarization of *S. cerevisiae* [2], and actin mediated directed transport[87].

Spatial localization plays a critical role in many cellular processes. An example of spatial localization that underlies many aspects of cell and developmental biology, from stem cells to the brain [29], is cell polarity, whereby cellular components that were previously uniformly distributed become asymmetrically localized, creating complexity of form and function. Cell polarity is fundamental to the diverse specialized functions of eukaryotic cells. In epithelial cells, for example, the differentiation of apical and basal specializations results in apical and baso-lateral

regions of plasma membrane that differ markedly in lipid and protein composition. In the *C. elegans* embryo, polarity is determined by opposing protein complexes that create distinct anterior and posterior domains, establishing and maintaining both cortical and cytoplasmic differences [101]. In yeast, cell polarity is necessary in division and mating. Beyond yeast, cell polarity is essential to the partitioning of cell fate in embryonic development. It is also essential in the creation of axons and their guidance during neuronal development, as well as the intimate communication between lymphocytes within the immune system [17]. From a modeling point of view, cell localization cannot be understood without proper modeling of the spatial dynamics that govern the creation and time evolution of such localization.

In this dissertation we present contributions to the field of computational systems biology in the form of advanced algorithms and software for spatial stochastic simulation of discrete biochemical systems. We also present work that illustrates the fundamental importance of spatial stochastic dynamics to biological systems by modeling the polarization mating response in *S. cerevisiae*.

1.1 Outline

The remainder of this dissertation is organized as follows: in the next chapter we describe the mathematical and computational formalism for stochastic simulation. In Chapter 3 we introduce the Diffusive Finite State Projection (DFSP) algorithm for spatial stochastic simulation. In Chapter 4 we present the software package URDME for simulation of Reaction Diffusion Master Equation models on Unstructured tetrahedron based spatial grids. In Chapter 5 we extend DFSP to include automatic error control, and explore the benefits of parallel execution on NVIDIA graphics processing units. In Chapter 6 we explore the spatial stochastic dynamics of cellular polarization in yeast mating. We conclude this dissertation with a summary and directions for future work.

Chapter 2

Simulation of Spatially Inhomogeneous Discrete Biochemical Systems

Discrete stochastic models of chemical reactions have been successful in describing numerous noise-induced phenomena in the cell. In discrete stochastic simulation, the state of the system is described by the number of molecules of each reacting species present at a given time in a reacting volume. The probability of finding the system in a given state is governed by the Chemical Master Equation (CME). This is the equation that describes the evolution of the probability density functions of the system. The CME is a set of as many coupled ordinary differential equations as there are combinations of molecules that can exist in the system.

From a simulation perspective, the most common approach for simulating the CME relies on Gillespie's Stochastic Simulation Algorithm (SSA) [49], which is a Kinetic Monte Carlo method for generating sample paths of the underlying stochastic chemical process. A key assumption in Gillespie's original SSA is that the system is well-mixed, i.e. the probability of finding a molecule in a given volume is given by the ratio of that volume divided by the total volume of the entire system (e.g. the cell). This assumption is valid for problems where the diffusion rates are fast compared to the reaction rates, so that the system can be considered homogeneous and well-mixed after each reaction. While this is a reasonable assumption in some situations, biological cells are obviously not spatially homogeneous. The SSA has been adapted to solve inhomogeneous problems in a formulation referred to as the inhomogeneous SSA (ISSA), whereby the cell volume is partitioned into an array of small sub-volumes (voxels) in which reactions take place among reactant species in each voxel while the reactant species diffuse over time from each voxel to its neighbors.

2.1 Discrete Biochemical Simulation

2.1.1 Chemical Master Equation and the SSA

Consider a system involving N molecular species $\{S_1, ..., S_N\}$, represented by the state vector $X(t) = [X_1(t), ..., X_N(t)]^T$, where $X_i(t)$ is the number of molecules of species S_i at time t. The M reaction channels are labeled $\{R_1, ..., R_M\}$. Assume the system is well-mixed and in thermal equilibrium. The dynamics of reaction channel R_j are characterized by the *propensity function* a_j and by the *state change vector* $\nu_j = [\nu_{1j}, ..., \nu_{Nj}]^T$: $a_j(x)dt$ gives the probability that, given $X(t) = x$, one R_j reaction will occur in the next infinitesimal time interval $[t, t + dt]$, and ν_{ij} gives the change in X_i induced by one R_j reaction.

The system is a Markov process whose dynamics are described by the Chemical Master Equation (CME) [50]

$$\begin{aligned}\frac{\partial P\left(x,t|x_0,t_0\right)}{\partial t} &= \mathcal{M}\, P\left(x,t|x_0,t_0\right) \\ &= \sum_{j=1}^{M}\left[a_j(x-\nu_j)P(x-\nu_j,t|x_0,t_0) - a_j(x)P(x,t|x_0,t_0)\right], \quad (2.1)\end{aligned}$$

where the function $P(x, t|x_0, t_0)$ denotes the probability that $X(t)$ will be x, given that $X(t_0) = x_0$ and $\mathcal{M}$ denotes the generating matrix for the Markov chain that describes the chemical reactions. For all but the most simple systems, the

chemical master equation is made up of an extremely large or infinite number (dimension) of coupled ordinary differential equations (ODEs). Rather than evolve the CME directly, it is common practice to compute an ensemble of stochastic realizations whose probability density function converges to the solution of the CME. In chemical kinetics, the SSA [49] is used for this purpose.

By far the most widely used methods for simulating the CME are based on Gillespie's Stochastic Simulation Algorithm (SSA) [49], which generates sample paths of the underlying stochastic process. On each time step, the SSA generates two random numbers which determine, based on the probabilistic reaction rates (called *propensities*), which reaction will fire next and what time it will fire. Then the system state is updated by firing that reaction, and the propensities, which depend on the system state, are updated. A great many stochastic realizations are needed to generate accurate probability density functions for the state variables of the system.

At each step, the SSA generates two random numbers, r_1 and r_2 in $U(0,1)$ (the set of uniformly distributed random numbers in the interval (0,1)). The time for the next reaction to occur is given by $t + \tau$, where τ is given by

$$\tau = \frac{1}{a_0} \ln\left(\frac{1}{r_1}\right). \tag{2.2}$$

The index μ of the occurring reaction is given by the smallest integer satisfying

$$\sum_{j=1}^{\mu} a_j > r_2 a_0, \tag{2.3}$$

where $a_0(x) = \sum_{j=1}^{M} a_j(x)$. The system states are updated by $X(t+\tau) = X(t)+\nu_\mu$. The simulation then proceeds to the time of the next reaction. Because the SSA simulates all reaction events in the system, it can be computationally intensive. Much recent effort has gone into speeding up the SSA by reformulation [15], [48], [119], use of advanced computer architecture [82], and by aggregating reaction events to take larger time steps (tau-leaping)[51].

2.1.2 The Finite State Projection Algorithm

The Finite State Projection (FSP) [98] method directly calculates an analytical approximation to the solution of the CME, as opposed to simulating an ensemble of trajectories by SSA. It does this by forming a computationally tractable projection of the full state space and computing the time evolution of the probability density function in this projection space. The FSP was formulated to solve spatially homogeneous stochastic models, but can be adapted to solve the diffusion master equation (DME). Techniques for taking advantage of time scale separation

in spatially homogeneous chemically reaction system were explored in [107] and [92].

The FSP method determines the approximate probability density vector (PDV) of the populations in a chemically reacting system by solving the CME in a truncated state space. Two theorems provide the foundation for the FSP. The first shows that the solution of the projected system increases monotonically as the size of the projection increases. The second guarantees that the approximate solution never exceeds the actual solution, and provides a bound on the error. It is important to note that while the evolution of a trajectory is random, the evolution of the PDV for a given initial condition is deterministic.

For a truncated state transition matrix A_J (see [98] for its construction) and initial truncated PDV $P_J(t = 0)$, the FSP finds $P_J(t)$ at any time t within any given accuracy ϵ using the truncated CME

$$\dot{P}_J = A_J\, P_J(t). \tag{2.4}$$

Since (2.4) is a linear constant-coefficient ODE, its solution is given by

$$P_J(t) = \exp(A_J t) P_J(0). \tag{2.5}$$

Recent work has focused on optimizing FSP through more effective dynamic state space truncation [100] and more efficient algorithms for solving the resulting equation [13].

2.2 Spatially Inhomogeneous Systems

2.2.1 Reaction-Diffusion Master Equation

The dynamics of spatially inhomogeneous stochastic systems are governed by the Reaction-Diffusion Master Equation (RDME) which was originally proposed and derived in [45]. More recently it was shown that biologically observed self-organized criticality emerges only when diffusion and reactions are treated as discrete stochastic processes [118]. This led to the adaptation of Gillespie's SSA to spatially inhomogeneous problems, called the Inhomogeneous SSA, or ISSA. In this formulation, the domain is discretized into subvolumes or voxels. Each voxel is well-mixed so that intra-voxel reactions are unchanged from the homogeneous case. Diffusive transfers between voxels are modeled by unimolecular decay and creation events occurring simultaneously in adjacent voxels. The state of the system is then the number of molecules of each species in each voxel at a given time.

It is important to note that the spatially inhomogeneous stochastic model is formulated on the mesoscopic scale. The voxel size is bounded by the well-mixed assumption of its mathematical formulation. The length ℓ of a voxel must be chosen to be small enough to capture the desired features of our system, but large enough so that the system can be considered to be well mixed in each voxel. Specifically, ℓ should satisfy $\ell \gg Kn/D$, where K is the reaction rate constant, n is the number of molecules in a given voxel and $D = D_A + D_B$ is the combined diffusion rates of the reactants [70, 37]. Recent work has further characterized the minimum voxel size and explored computational methods for accurate simulation below this limit [61, 40].

For spatial discretization, assume that the domain Ω is partitioned into voxels V_k, $k = 1, ..., K$. For simplicity of presentation, we will consider for the moment a one dimensional domain. Each molecular species is represented by the state vector $X_i(t) = [X_{i,1}(t), ..., X_{i,K}(t)]$, where $X_{i,k}(t)$ is the number of molecules of species S_i in voxel V_k at time t. Molecules in the domain are able to react with molecules within their voxel, as described in Section 2.1.1, and diffuse between neighboring voxels. The dynamics of diffusion of species S_i from voxel V_k to V_j is characterized by the *diffusion propensity function* $d_{i,k,j}$ and the *state change vector* $\mu_{k,j}$. $\mu_{k,j}$ is a vector of length K with -1 in the kth position, 1 in the jth position and 0 everywhere else: $d_{i,k,j}(x)dt$ gives the probability that, given $X_{i,k}(t) = x$,

one molecule of S_i will diffuse from voxel V_k to V_j in the next infinitesimal time interval $[t, t+dt]$. Note that if $k = j \pm 1$ then $d_{i,j,k}(x) = D/l^2$, where D is the diffusion rate and l is the characteristic length of the voxel, and otherwise it is zero. The Diffusion Master Equation (DME) can then be written in a form similar to the CME:

$$\begin{aligned}\frac{\partial P\left(x,t|x_0,t_0\right)}{\partial t} &= \mathcal{D}\,P\left(x,t|x_0,t_0\right) \\ &= \sum_{i=1}^{N}\sum_{k=1}^{K}\sum_{j=1}^{K}\left[d_{i,j,k}(x_i-\mu_{k,j})P(x_1,...,x_i-\mu_{k,j},...,x_N,t|x_0,t_0)\right. \\ &\quad \left.-d_{i,j,k}(x_i)P(x,t|x_0,t_0)\right], \end{aligned} \tag{2.6}$$

where $\mathcal{D}$ denotes the generating matrix for the Markov chain that describes the diffusion of molecules in the system.

Combining (2.1) and (2.6) yields the RDME

$$\frac{\partial P\left(x,t|x_0,t_0\right)}{\partial t} = \mathcal{M}P\left(x,t|x_0,t_0\right) + \mathcal{D}P\left(x,t|x_0,t_0\right). \tag{2.7}$$

The RDME is a linear constant-coefficient ODE, however it has many more possible states than the corresponding CME and thus is more difficult to solve. Rather than solve the RDME directly, it is common practice to compute an en-

semble of stochastic realizations whose histogram converges to the PDV of the RDME.

2.2.2 Spatial Stochastic Simulation Algorithms

Spatial stochastic simulation via the ISSA begins by partitioning the cell volume into an array of small sub-volumes (voxels). We will be primarily concerned with two essential processes: reaction and diffusion. Reactions take place among reactant species occupying each of the voxels. At the same time, the reactant species diffuse over time from one voxel element to its neighbors. Species are assumed to diffuse independently. Diffusion is modeled as unimolecular decay and creation events occurring simultaneously in adjacent voxels, with the transition rate dictated by the diffusion coefficient and possibly by other factors.

Some recent work has dealt with efficient formulations of the ISSA. The Next Subvolume Method (NSM)[31] is a clever and widely used formulation of the ISSA to provide better efficiency for reaction-diffusion systems. It is used in the MesoRD [58] and SmartCell [3] software. However, its efficiency is limited by the fact that, as an exact method, it must simulate every event in the system, including all of the diffusive transfers. The Multinomial Simulation Algorithm (MSA)[79] aggregates the fast diffusive transfers. Instead of executing each diffusive event individually, it calculates the inter-voxel flux of particles by sampling from a binomial distri-

bution. The binomial tau-leap spatial stochastic simulation algorithm [88] seeks to improve performance by combining the ideas of aggregating diffusive transfers with the priority queue structure found in the NSM. Under some circumstances it is possible to treat diffusion deterministically, thus eliminating the tracking of fast diffusive transfers almost entirely. Reactions are typically handled by the SSA. The Hybrid Multiscale Kinetic Monte Carlo Method [139] and the Gillespie multi-particle method [113] are examples of this approach. The adaptive hybrid method for stochastic reaction-diffusion processes described in [42] integrates multiple methods for stochastic and deterministic diffusion adaptively for different components of a model.

Chapter 3

The Diffusive Finite State Projection Algorithm for Efficient Simulation of the Stochastic Reaction-Diffusion Master Equation

Spatial stochastic simulation is an extremely computationally intensive task. This is due to the large number of molecules which, along with the refinement of the discretized spatial domain, results in a large number of diffusive transfers between voxels (sub-volumes). In this chapter, we present a novel formulation of the Finite State Projection (FSP) method [98], called the Diffusive FSP (DFSP) method, for the efficient and accurate simulation of diffusive processes. Using

the FSP method's ability to provide a bound on the error, we are able to take large diffusion timesteps with confidence in our solution. We then show how to construct a fractional step method for spatial stochastic simulation of reaction-diffusion processes which treats diffusion with DFSP and reactions with SSA. This work was performed in collaboration with Michael Lawson, and was originally presented in [28].

3.1 The Diffusive FSP Method

The DFSP method is based on two observations. First, diffusion of any one molecule is independent of the diffusion of all other molecules in the system. Using this independence, we note that the diffusion of molecules originating in one voxel is independent of the diffusion of all molecules originating in other voxels. Thus, we can decompose the problem of diffusing molecules in K voxels into K sub-problems, one for each voxel.

The second observation is that the DME describes a stochastic process, but the DME itself is a system of ODEs. That is, the evolution of a particular trajectory is stochastic, but the evolution of the Probability Density Vector (PDV) is deterministic. Thus, if we can solve the DME for a given sub-problem with n molecules for a time step Δt, then we can re-use this solution for all other

sub-problems with n molecules and time step Δt. Next we will describe more rigorously a sub-problem and show how to set up and solve a FSP for such a sub-problem. Note that to solve the full problem, one needs only to sum the molecule distributions from each sub-problem.

3.1.1 DFSP

As above, we will consider a problem on a 1D periodic domain that is sub-divided into K equally sized voxels, each with length l. The kth sub-problem defines a diffusion problem that is initialized with empty voxels, except for the kth voxel, which contains n_k molecules of a given species. This initial condition is considered a state. The states of the system are defined by unique configurations of molecules in voxels, with the total number of molecules in the system always summing to n_k. The possible number of states is finite, though extremely large. The PDV enumerates these states and gives the probability of being in any state at a given time. For the initial condition, it is clear that the PDV for the system is $P(0) = [1, 0, 0...0]^T$. That is, at time zero, the probability of being in the state of the initial condition is one, and the probability of being in all other states is zero.

To solve the DME directly for a sub-problem, the DFSP method retains a finite set of states that carry a high probability, and truncates states of little probabilistic

importance. To determine which states to retain, we will walk through the process of diffusing molecules. The initial condition forms the first tier. The second tier is defined by the states that can be reached with one diffusion event from the initial condition. The third tier is defined by the states that can be reached with one diffusion event from any state in the second tier and is not redundant with states in higher tiers.

In defining each of these states, there is an additional parameter, MAX, that is defined as the maximum number of voxels a particle can diffuse away from its originating voxel in one time step. The value of MAX is one less than the number of tiers. All of the states in the last tier are one diffusive step away from violating the MAX condition. MAX puts a limit on the allowable number of particles for a sub-problem without violating the error condition, (error $< \epsilon$). It is important to note that MAX dictates the amount of memory storage required by the algorithm.

For illustration, consider the situation where a voxel contains 20 molecules at the beginning of a time step, and $MAX = 2$; that is, we are tracking diffusive jumps of at most 2 voxels away from the originating voxel per time step. The initial state is given by $x_1 = \{0, 0, 20, 0, 0\}$. x_1 is the only state in the first tier. Since we are on a one-dimensional domain, the states reachable in a single diffusive jump event from x_1 are $x_2 = \{0, 1, 19, 0, 0\}$ and $x_3 = \{0, 0, 19, 1, 0\}$. These two states

make up the second tier. The third tier is comprised of $x_4 = \{1, 0, 19, 0, 0\}$, $x_5 = \{0, 0, 19, 0, 1\}$, $x_6 = \{0, 2, 18, 0, 0\}$, $x_7 = \{0, 0, 18, 2, 0\}$ and $x_8 = \{0, 1, 18, 1, 0\}$. Note that x_1 is reachable from the states in the second tier, but since that state is found in a higher tier, it is not included in the third tier.

As each tier is added, the corresponding state transitions are included in A_J. After each tier is added, the truncated system can be solved and the truncated PDV ($P_J(\Delta t)$) calculated. Thus, after adding a tier, we can determine a bound on our error for the current projection (ϵ). The addition of states ends when the error bound is below a predetermined tolerance.

To calculate the final state of the system due to diffusion over an interval of Δt, we sample the PDV by selecting K uniformly distributed random numbers $R_k \in U(0, 1)$ and finding the smallest integer μ_k such that $\sum_{j=1}^{\mu_k} PDV[j] > R_k$, where $PDV[j]$ is the probability weight of state j. Let $X_{s,k}(t) = n_k$ be the number of molecules of species s in voxel k at time t and let $T_\kappa(j \,|\, n)$ be the number of molecules in voxel κ of state x_j, given n molecules initially (e.g. $x_1 = \{0, 0, n, 0, 0\}$ if MAX=2). Then the discrete time evolution of the system is given by

$$X_{s,k}(t + \Delta t) = \sum_{i=-MAX}^{MAX} T_i \left(\mu_{k+i} \,|\, X_{s,k+i}(t)\right). \tag{3.1}$$

For a sub-problem with n molecules and a time step Δt, we can store its PDV and re-use it for all other sub-problems containing n molecules and time step Δt. As a result, if we keep a constant time step, simulating a diffusion process becomes a matter of selecting K random numbers and performing a look-up and comparison.

To simulate the full RDME, we take a reaction step and then a diffusion step, each of size τ_D. Following the SSA, we take a reaction step by evolving the system through reaction events until the time of the next reaction exceeds τ_D. We then perform diffusion of the molecules at the end of the reaction step via the DFSP as described above. At the end of the diffusive step, the simulation time is $t_0 + \tau_D$. We continue interleaving reaction and diffusion steps until the final time.

3.1.2 Adaptive Step Splitting

In the case where an initial population for a sub-problem is large enough to exceed the error condition (ϵ) for a given MAX, we need to split the step. Rather than split the step in time, we take advantage of the independence of diffusing molecules and split the sub-problem into several sub-sub-problems. For example, suppose that the maximum number of particles one can diffuse in τ_D without violating the error condition is 10. In this case, we would treat this sub-problem of 20 particles as two sub-sub-problems of 10 each. The states for each

sub-sub-problem are $x_1 = \{0, 0, 10, 0, 0\}$, $x_2 = \{0, 1, 9, 0, 0\}$, $x_3 = \{0, 0, 9, 1, 0\}$, $x_4 = \{1, 0, 9, 0, 0\}$, $x_5 = \{0, 0, 9, 0, 1\}$, $x_6 = \{0, 2, 8, 0, 0\}$, $x_7 = \{0, 0, 8, 2, 0\}$ and $x_8 = \{0, 1, 8, 1, 0\}$. We can then reconstruct the solution for the sub-problem by picking a uniformly distributed random number (as above) for each sub-sub-problem, selecting the corresponding state and then summing these two sets. It is clear that the states of the sub-problem are all possible combinations of x_1, x_2, x_3, x_4, x_5, x_6 and x_7. While some of these combinations may be redundant, the number of unique states for the sub-problem of 20 particles has been increased from the original 7. By the first FSP theorem, the solution of the projected system increases monotonically as the size of the projection increases; as a corollary, the size of the error must decrease as we add states.

We continue splitting the sub-problems until the error from each sub-problem is less than $\epsilon/2^{\mathcal{L}}$, where $\mathcal{L}$ is the recursion level. In the extreme case of splitting the sub-problem into sub-sub-problems of one molecule, the combination of states would provide all possible combinations of the original n_k particles in the $2 \times MAX + 1$ voxels of the sub-problem.

The advantage of splitting the sub-problems in this way (as opposed to splitting the time step) is that we can keep Δt constant, which allows us to re-use our lookup table. Calculation of the lookup table is the most computationally expensive part

of the algorithm. In order to maximize speed, we seek to avoid changing the time stepsize whenever possible.

Next we perform an analysis of the adaptive step splitting error control. Consider the case where we want to calculate a final state of a sub-problem containing 100 molecules of a chemical species after $\tau_D = 0.1$ using a local error tolerance of 10^{-5}. If all 100 molecules are moved simultaneously, then the resulting single step FSP error will be 0.38 and our truncated state space contains 62% of the probability density. Utilizing the fact that the FSP error has a non-linear relationship with the number of molecules moved (Figure 3.1 shows the error as a function of the number of molecules moved in one timestep), we can split the molecules into smaller groups where the sum of the error of diffusing the smaller groups is less than the original error. We recursively split a group of molecules in half if the error to move it in one step is greater than the error tolerance (adjusted for the recursion level). For 100 molecules, we first split them into two groups of 50 (error of $3.86e-2$), then four groups of 25 (error of $1.46e-3$), and so on. In total, we will move twelve groups of six molecules each with error $2.4e-7 < 10^{-5}/2^4 = 6.3e-7$ (four levels of recursion), four groups of four molecules each with error $1.4e-8 < 10^{-5}/2^5 = 3.1e-7$, and four groups of three molecules each with error $1.6e-9 < 10^{-5}/2^5 = 3.1e-7$ (both with five levels of recursion). The total error is $3.0e-6$ which is the sum of the error in all of

the recursion steps. Using this method, we are able to satisfy the error tolerance, while continuing to utilize the efficiency of the lookup tables.

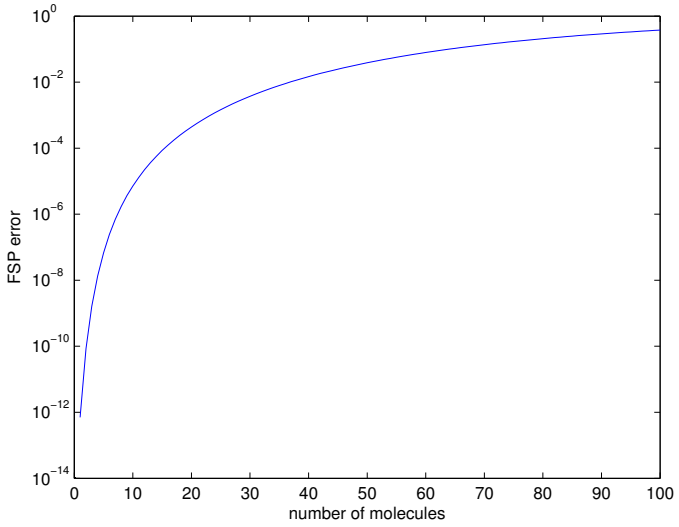

Figure 3.1: Projection error (ϵ) for varying number of molecules given $\tau_D = 0.1s$, $\ell = 0.62\mu m$, $D = 0.001\mu m^2 s^{-1}$. Adaptive step splitting allows us to take advantage of the independence of diffusing molecules and the nonlinear relationship of projection error to the number of molecules diffused to reduce the total error of diffusion step over τ_D by splitting it into sub-sub-problems of fewer molecules, as opposed to splitting the time step.

3.1.3 Detailed Algorithm Descriptions

State Space Exploration The algorithm to determine the truncated state space is presented in detail in Algorithm 1. The input parameters are the number n_k of particles in the originating voxel k, and the maximum number of diffusive transfers MAX that a particle can move away from the originating voxel in one

diffusion time step. The state representing the initial condition is that all n_k particles are in the originating voxel. The algorithm is presented for an anisotropic, one dimensional Cartesian mesh with periodic boundary conditions, and assumes that the number of voxels in any dimension is large relative to MAX.

INPUT: n_k, MAX, Initial State
OUTPUT: $\texttt{TransitionMatrix}_n$, $\texttt{StateList}_n$

1: Initialize: `NextTierQueue` $\leftarrow$ Initial State, `Queue` $\leftarrow \emptyset$
2: Initialize: `StateList` $\leftarrow$ Initial State, `TransitionMatrix` $\leftarrow \emptyset$
3: **for** Tier $\in (2, MAX)$ **do**
4: `Queue` $\leftarrow$ `NextTierQueue`
5: `NextTierQueue` $\leftarrow \emptyset$
6: **for all** states $s \in$ `Queue` **do**
7: **for all** non-empty voxels $v \in s$ **do**
8: **for all** inter-voxel transitions d {with probabilities $p(d)$} originating from v **do**
9: find state $t \leftarrow s + d(v)$
10: **if** $t \notin$ `StateList` **then**
11: add t to `StateList`, add t to `NextTierQueue`
12: **end if**
13: `TransitionMatrix`$(s,t) \leftarrow p(d)$
14: **end for**
15: **end for**
16: **end for**
17: **end for**
18: Update Diagonal elements in `TransitionMatrix`
19: Truncate `TransitionMatrix` so that it is of dimension |`StateList`|
20: Create absorbing state in `TransitionMatrix`

Algorithm 1: State Space Exploration

We then store the $\texttt{TransitionMatrix}_n$ and $\texttt{StateList}_n$ for later use. For all cases where the number of particles in the originating voxel n, such that n is greater than MAX, the structure of the $\texttt{TransitionMatrix}_n$ is constant, and

the values in the matrix are linear functions of n. This matrix is obtained by performing the state space exploration algorithm with n as an unspecified parameter constrained to a value greater than MAX. For $n < MAX$ it is still necessary to go through the state space exploration, because for these values the $\texttt{TransitionMatrix}_n$ will not conform to the general structure.

DFSP diffusion step This is the algorithm for taking a single time step of length τ_D for a single voxel k containing n_k particles. We assume that $\texttt{TransitionMatrix}_n$ and $\texttt{StateList}_n$ have already been calculated and stored, and that MAX and τ_D are constant. Model parameters for the diffusion coefficient D and voxel length ℓ are also used.

Algorithm 2 shows the details of this process. The output of this algorithm is a vector map where the positions correspond to voxel indices and the values correspond to the number of particles that have traveled to that voxel from the originating voxel n_k via diffusion after an interval of length τ_D seconds.

Reactions In our computational framework for reaction-diffusion problems, we use a fractional step method which simulates the diffusive transfers by DFSP and the reaction events by SSA. We begin at t_0 and calculate the first reaction event. We simulate reactions until the time to the next reaction would advance the simulation beyond $t_0 + \tau_D$, at which point we forego the last reaction and

INPUT: n_k, τ_D, $\texttt{TransitionMatrix}_n$, $\texttt{StateList}_n$, $\texttt{ErrorTolerance}$
OUTPUT: Output State

Initialize once: $\texttt{PDVLookupTable}$, $n_{max} \leftarrow \infty$, $\mathcal{L} \leftarrow 0$
if $n_k >= n_{max}$ **then**
  return Output State $\leftarrow$ $\texttt{DFSP\_Diffusion}(\lfloor n_k/2 \rfloor, \mathcal{L}+1)$ + $\texttt{DFSP\_Diffusion}(\lceil n_k/2 \rceil, \mathcal{L}+1)$
else
  if $\texttt{PDVLookupTable}$ contains n_k **then**
    $\texttt{PDV} \leftarrow \texttt{PDVLookupTable}[n_k]$
  else
    $\texttt{PDV} \leftarrow \exp(\ \texttt{TransitionMatrix}_{n_k} \times D/\ell^2 \times \tau_D\ ) \times [1, 0, 0, \cdots, 0]^T$
    if $\texttt{PDV}[\text{end}] > \texttt{ErrorTolerance}\ /\ 2^{\mathcal{L}}$ **then**
      $n_{max} \leftarrow n_k$
      return Output State $\leftarrow$ $\texttt{DFSP\_Diffusion}(\lfloor n_k/2 \rfloor)$ + $\texttt{DFSP\_Diffusion}(\lceil n_k/2 \rceil)$
    end if
    $\texttt{PDVLookupTable}[n_k] \leftarrow \texttt{PDV}$
  end if
  Generate a random number $X \in U(0, 1)$
  Find the smallest integer μ such that $\sum_{j=1}^{\mu} \texttt{PDV}[j] > X$
  return Output State $\leftarrow \texttt{StateList}_{n_k}[\mu]$
end if

Algorithm 2: DFSP diffusion step with splitting

perform a diffusion step using DFSP. After the diffusion step, the simulation is at time $t_0 + \tau_D$. This process is repeated until the simulation is complete.

This process is detailed in Algorithm 3. Inputs to this algorithm are τ_D, the stoichiometric matrix ν and the initial state of the system. The calls to `DFSP_Diffusion` use a `StateList` and `TransitionMatrix` that correspond to the geometry and jump propensities of the problem as well as a specified `ErrorTolerance`.

3.2 Examples and Analysis

We examine two models to explore the validity, accuracy and speed of DFSP. The first is a model of pure diffusion. The second is a biologically inspired reaction-diffusion spatial stochastic model.

3.2.1 Diffusion Example

The first example is composed of a single chemical species diffusing in one dimension. The domain is periodic ($\Omega = 12.4\mu m$) and we discretized it into 200 voxels of length $\ell = 0.062\mu m$. This domain is equivalent to a circle with radius 2μm, so we plot the results on the interval $[-2\pi, 2\pi)$. The initial condition is a step-function such that each voxel in the interval $[-2\pi, 0)$ has 100 molecules and the remaining voxels are empty. The chemical species move with a diffusion

Initialize system state: X, $t = 0$
Calculate the propensity functions $a_{jk}(X)$ and $a_0 \leftarrow \sum_{k=1}^{K} \sum_{j=1}^{M} a_{jk}(X)$ {where M is the number of reactions and K is the number of voxels}
Generate two random numbers $r_1, r_2 \in U(0,1)$
$t_{next\_rxn} \leftarrow t + \frac{1}{a_0} \ln\left(\frac{1}{r_1}\right)$
$t_{next\_diff} \leftarrow t + \tau_D$
while $t < t_{final}$ **do**
  if $t_{next\_rxn} < t_{next\_diff}$ **then**
    Find μ_r, μ_x smallest integers to satisfy $\sum_{k=1}^{\mu_x} \sum_{j=1}^{\mu_r} a_{jk} > r_2 a_0$
    Update $X_{\mu_x}(t_{next\_rxn}) = X_{\mu_x}(t) + \nu_{\mu_r}$
    Generate two random numbers $r_1, r_2 \in U(0,1)$
    $t \leftarrow t_{next\_rxn}$
  else
    $X_{next} \leftarrow \emptyset$
    for $k \in (1...K)$ **do**
      for $i \in (1...N)$ **do**
        $X_{next} \leftarrow X_{next} +$ `DFSP_Diffusion`$(X_{k,i})$ {diffusion of species i in voxel k}
      end for
    end for
    $X \leftarrow X_{next}$
    $t \leftarrow t_{next\_diff}$
    $t_{next\_diff} \leftarrow t + \tau_D$
  end if
  Update propensity functions $a_{jk}(X)$ and $a_0 \leftarrow \sum_{k=1}^{K} \sum_{j=1}^{M} a_{jk}(X)$
  $t_{next\_rxn} \leftarrow t + \frac{1}{a_0} \ln\left(\frac{1}{r_1}\right)$
end while

Algorithm 3: RDME simulation algorithm using DFSP for diffusion and SSA for reactions

coefficient of $0.001\mu m^2 s^{-1}$. Numerical experiments show that the relaxation time of this system is approximately 7000 seconds (data not shown). In this example, we use the adaptive step splitting with $MAX = 5$. Figure 3.2 shows the initial condition (dashed blue), a transient state (dotted black) and a final state (solid blue) for a single sample trajectory of this model.

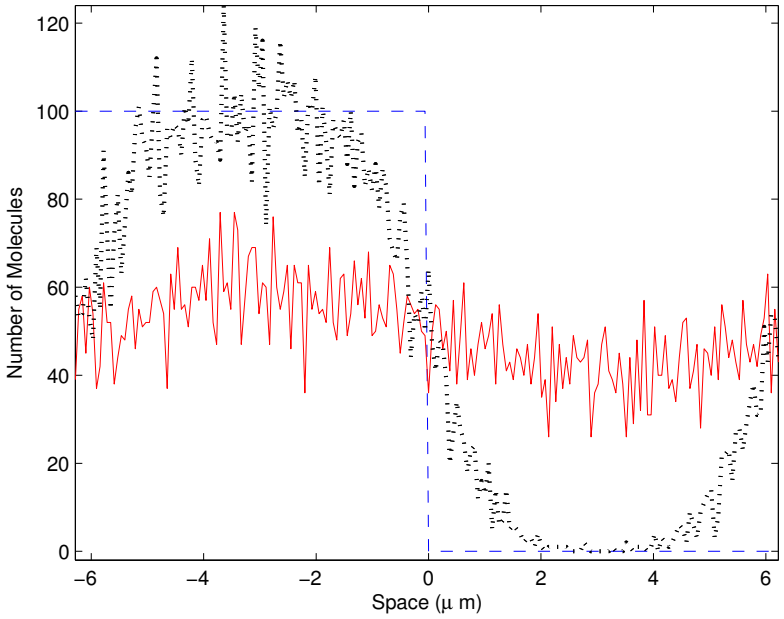

Figure 3.2: Solution to a pure diffusion problem with a step function as the initial condition. Plotted is the state of the system at t = 0s (dashed blue), 500s (dotted black) and 7000s (solid red) for a stochastic trajectory. The domain is a circle with radius $2\mu m$, subdivided equally into 200 voxels.

Validation

To test the validity of solving the diffusion example with ISSA or DFSP we solve for the moments analytically (see [28] for derivation). Figure 3.3 shows the error in the mean and variance as a function of time for three different sized ensembles of ISSA and DFSP trajectories. The error is calculated using the L_∞ norm (across space) of the difference between the ensemble moments and the analytically derived moments, divided by the norm of the analytical moment.

$$\text{Normalized } L_\infty \text{ error}(t) = \frac{\|\text{analytical\_moment}(x,t) - \text{ensemble\_moment}(x,t)\|_\infty}{\|\text{analytical\_moment}(x,t)\|_\infty} \tag{3.2}$$

As the ensemble size increases, the error in both the mean and the variance decreases at the same rate for ISSA as for DFSP. Figure 3.4 shows the error in the mean and variance as a function of voxel size. As the voxel size decreases, the error decreases. This demonstrates convergence of RDME solution methods to the analytical solution to the stochastic diffusion equation. Since the ISSA is an exact simulation method to the RDME while DFSP is an approximate method, this analysis shows that DFSP is just as valid as the ISSA for these parameter values.

To assess the accuracy of DFSP, we treat an ensemble of ISSA simulations as the baseline distribution because the ISSA is a true realization of the RDME

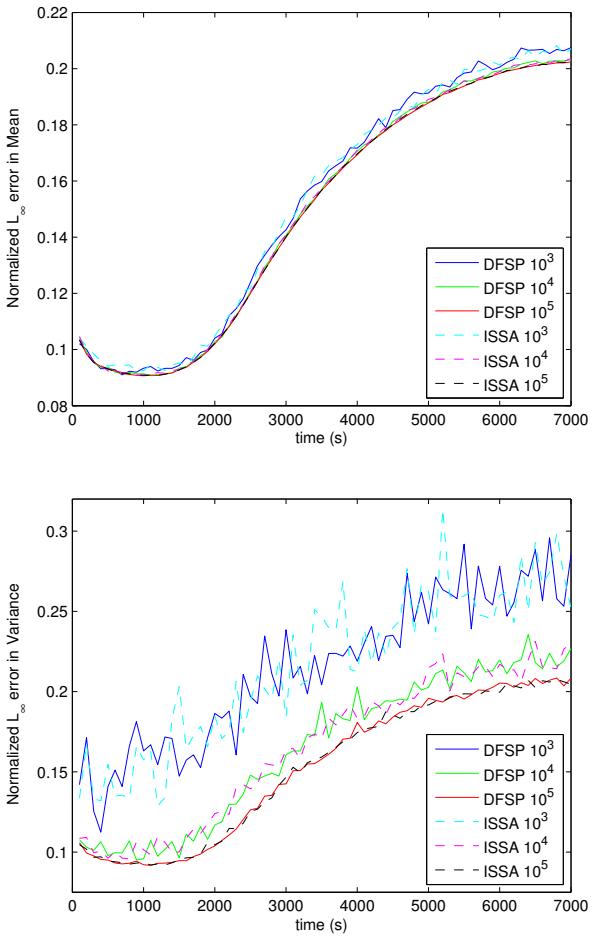

Figure 3.3: Plot of the Normalized L_∞ error (maximum deviation from analytical solution) versus time in the mean (a) and variance (b) for varying ensemble sizes for both DFSP ($\tau_D = 0.1$s) and ISSA (voxel size of $0.06\mu m$), for an ensemble size of 10^3 trajectories. The error increases with time (as expected for a discretized solution) at the same rate for both DFSP and ISSA. Additionally, the error decreases (to the discretization error limit) with increasing ensemble size at the same rate for DFSP and ISSA.

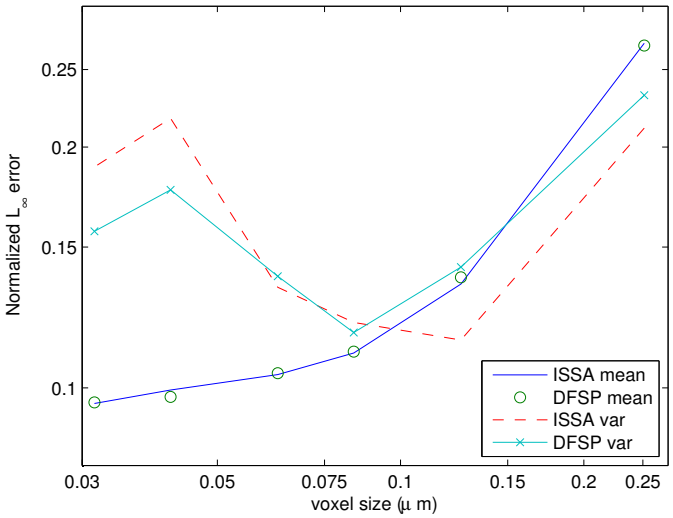

Figure 3.4: Plot of the Normalized L_∞ error versus voxel size (both on a log-scale) calculated at $t = 100$s (a transient state) for an ensemble size of 10^3 trajectories. As voxel size decreases, the error in the mean decreases at the same rate for both DFSP and ISSA. The error in the variance shows a similar trend, however it also shows increased error for small voxel sizes. This is mostly likely sampling error due to a constant system population distributed into an increasing number of voxels.

and its ensemble converges to the exact solution of the RDME. The Kolmogorov distance [77] is a standard measurement of the difference between two cumulative distribution functions (CDF). It is defined as the largest deviation between two CDFs. We choose this measure because it compares all the moments of two distributions and is thus a stronger tool for analysis than methods that use individual moments. We plot the average Kolmogorov distance across space ($Kmean$) sampled at each point in time. This is given by

$$Kmean(a,b,t) = \frac{1}{N}\sum_{n=0}^{N-1} \|CDF_a(n\ell,t) - CDF_b(n\ell,t)\|_\infty \tag{3.3}$$

where N is the number of voxels. The $CDF_a(x,t)$ is calculated from an ensemble of trajectories generated by algorithm a (e.g. ISSA or DFSP) sampled at spatial location x at time t. We compare the $Kmean$ of two independent ISSA ensembles (this is known as the self-distance) at each sampled point in time with the $Kmean$ of an ISSA ensemble and a DFSP ensemble. If the two $Kmean$ values are similar, then DFSP is statistically indistinguishable from ISSA for this ensemble size.

Figure 3.5 shows $Kmean$ values across time for the ISSA self distance and ISSA versus DFSP. We show results for ISSA versus DFSP for two sets of simulation parameters: the first uses $\tau_D = 0.1s$, `ErrorTolerance` $= 10^{-5}$ and the second uses $\tau_D = 1.9s$, `ErrorTolerance` $= 10^{-3}$. These results are for an ensemble size of 10^5

trajectories. We note that for an ensemble size $\leq 10^4$, DFSP is indistinguishable from ISSA (data not shown). These results show that for sufficiently small τ_D and `ErrorTolerance`, DFSP is a good approximation for ISSA. For the results with $\tau_D = 1.9$s, the adaptive step splitting fails to meet the error tolerance; therefore, as the ensemble size grows the error accrued by DFSP is no longer negligible. Thus, it is clear that for increasing values of τ_D and `ErrorTolerance` the error in the simulation grows. We will show that it is possible to utilize this feature of DFSP to trade accuracy for computational performance.

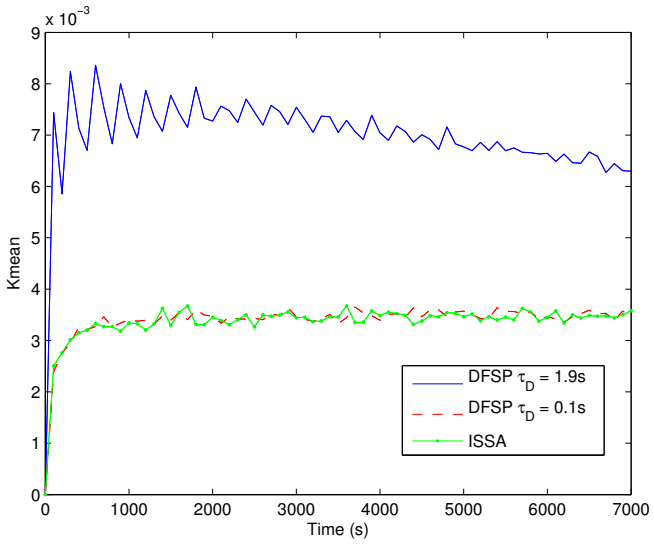

Figure 3.5: The distribution distance for the diffusion example, the $Kmean$ at every 100s for 7000s for ISSA-vs-ISSA (green line with dots), ISSA-vs-DFSP with $\tau_D = 0.1$s (dashed red line) and $\tau_D = 1.9$s (blue line). In this plot the ensemble size is 10^5, at which point the DFSP solution becomes distinguishable from the ISSA solution for larger time steps. For ensemble sizes $\leq 10^4$, the DFSP solution is indistinguishable from the ISSA solution for both step sizes (results not shown).

Error Analysis

To study the error properties of DFSP, we must first find the limits of our adaptive step splitting error control method. The contribution from diffusion to the total error should be constant for all values of τ_D as long as we are able to move at least one molecule per DFSP diffusion step without violating our error tolerance. Figure 3.6 shows a plot of the maximum possible number of molecules moved per diffusion step of DFSP for various values of τ_D, and a fixed error tolerance of 10^{-5}. To find the maximum number of molecules we can move for a given τ_D we compute DFSP matrix exponentials for increasing molecule counts. The maximum number that can be moved is one less than the number at which the estimate error first exceeds the tolerance. From this study, we determined that the maximum value of τ_D is 0.925s.

To measure the error in the simulated ensembles, we integrate the deviation between the Kolmogorov distance of DFSP and ISSA and the self-distance of ISSA over space and time, normalized by the size of the domain.

$$\text{Error}_{\tau_D} \;=\; \frac{\int\int |\text{DFSP}_{\tau_D}(x,t) - \text{ISSA}(x,t)|\; dx\, dt}{\int\int\; dx\, dt} \tag{3.4}$$

where $\text{ISSA}(x,t)$ is the Kolmogorov distance over space and time between two ensembles of 10,000 runs of the ISSA, and $\text{DFSP}_{\tau_D}(x,t)$ is the Kolmogorov distance

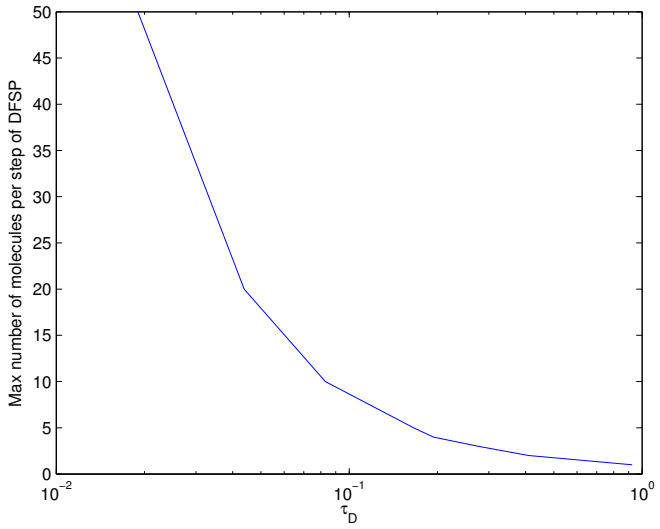

Figure 3.6: Maximum number of molecules moved in a single diffusion step of DFSP versus τ_D for an error tolerance of 10^{-5}. For values of $\tau_D > 0.925$s, DFSP will try to move less than one molecule, thus violating the error tolerance.

over space and time between 10,000 runs of the DFSP algorithms (with diffusion step τ_D) and 10,000 runs of the ISSA algorithm. We examine this error metric for varying values of τ_D with a fixed error tolerance of 10^{-5}. Figure 3.7 shows the error as a function of τ_D for the diffusion example as well as the G-protein example (discussed in Section 3.2.2). Thus for this range of values of τ_D, the error for the diffusion example is constant, and is a function only of the `ErrorTolerance` parameter.

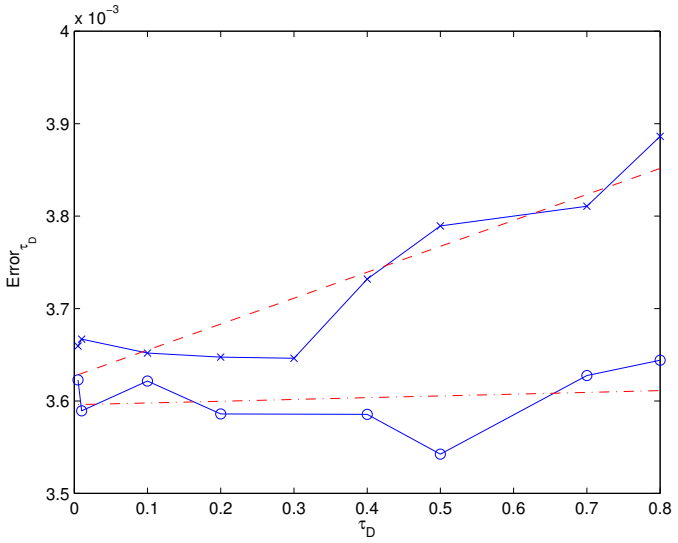

Figure 3.7: Error$_{\tau_D}$ vs τ_D for DFSP applied to the diffusion example (line with circles) and G-protein cycle example (line with x's), linear fits (red dashed lines). For the diffusion example, the error is constant and only a function of the error tolerance. This is because there is no contribution to the error from the reaction operator. For the G-protein cycle example, the error increases linearly with τ_D and converges to the diffusion error as τ_D goes to zero. This is because the error in the reaction operator is linear with the timestep (τ_D); as the timestep goes to zero the reaction error goes to zero.

3.2.2 G-protein Cycle Example

The second example is the pheromone induced G-protein cycle in *Saccharomyces cerevisiae*. We have converted the PDE model from [18] into a stochastic model and for simplicity reduced it to ligand, receptor and G-protein species. The ligand level is constant in time but varies spatially (a cosine function), with parameters determined experimentally. The ligand binds stochastically with an initially isotropic field of receptor proteins. The bound receptor activates the G-protein, causing the G_α and $G_{\beta\gamma}$ sub-units to separate. G_α acts as an auto-phosphotase and upon dephosphorylation, rebinds with $G_{\beta\gamma}$ to complete the cycle. The spatial domain is identical to the previous model and the simulation time is set to 100 seconds, as deterministic simulation shows that steady state is achieved by that time (data not shown). $G_{\beta\gamma}$ is the component farthest downstream from the ligand input and acts as signal to the downstream Cdc42 cycle and will therefore be the output for this model. Figure 3.8 shows the constant ligand gradient (left) and the spatial distribution of $G_{\beta\gamma}$ over 1000 runs (right, mean and standard deviation). See the Appendix of [28] for complete description of the reactions.

Validation

For the G-protein example, Figure 3.9 shows the $Kmean$ for ISSA versus DFSP (using $\tau_D = 0.1s$, `ErrorTolerance` $= 10^{-5}$) for an ensemble size of 10^5 trajecto-

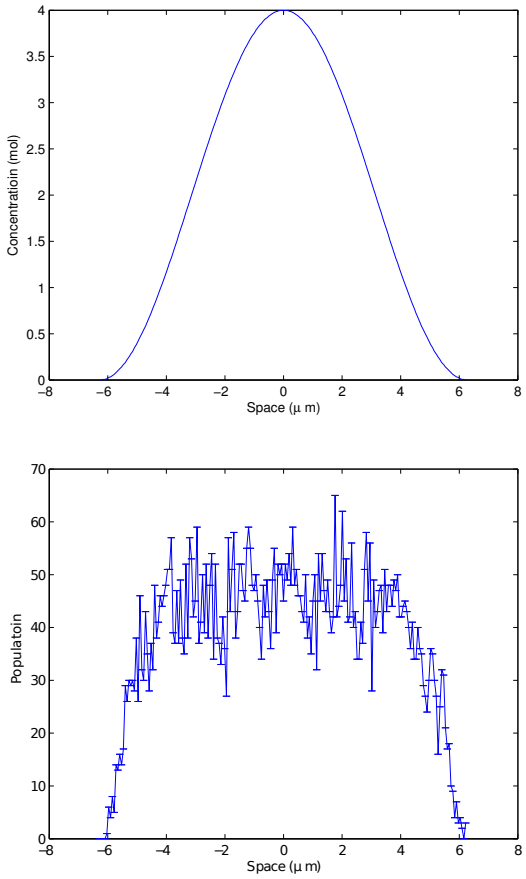

Figure 3.8: Spatial concentration of Ligand (top) and mean and variance of $G_{\beta\gamma}$ population (bottom) at $t = 100$s for a trajectory of the G-protein cycle example. The Ligand gradient is the input to this model and is constant in time. $G_{\beta\gamma}$ is the output and is time varying.

ries. Note that for an ensemble size of $\leq 10^3$ trajectories, the results of DFSP are indistinguishable from those of ISSA. For these simulation parameters, the difference between ISSA and DFSP values of $Kmean$ is constant over time, and the DFSP $Kmean$ is consistent across the time span of the simulation. This indicates that the simulation is stable, but there is an error in the results that shows up as a difference between the DFSP and ISSA curves. We will discuss the source of this error and provide an analysis in the following section. We also show results for τ_D increased to the CFL limit [19], which is $\sim 1.9s$, and with `ErrorTolerance` set to 10^{-3} in an attempt to determine the limits of DFSP's ability to handle full reaction-diffusion models. For these parameters the specified `ErrorTolerance` cannot be met, though the adaptive splitting moves a single molecule per step. The difference between this curve and the ISSA curve is significantly greater than for the previous parameters, and is oscillatory in time. This indicates that the simulation results are inconsistent with the underlying mathematical process.

Error Analysis

DFSP uses operator splitting over a given timestep of length τ_D, first applying the reaction operator (SSA in this case) to the system. Then the diffusion operator (using FSP) is applied to the resulting state of the system. Since these operators are decoupled, an additional splitting error is incurred by the method

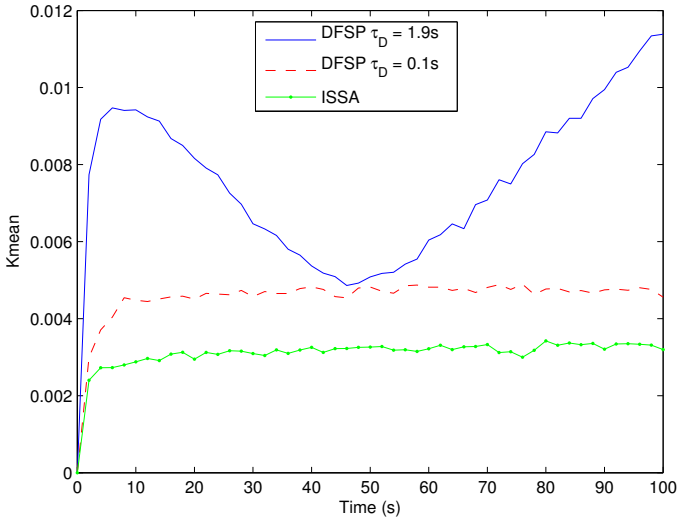

Figure 3.9: The distribution distance for the G-Protein cycle example, the $Kmean$ from (3.3) at every 100s for 7000s for ISSA-vs-ISSA (green line with dots), ISSA-vs-DFSP with $\tau_D = 0.1$s (dashed red line) and $\tau_D = 1.9$s (blue line). In this plot the ensemble size is 10^5, at which point the DFSP solution becomes distinguishable from the ISSA solution for larger time steps. For ensemble sizes $\leq 10^3$, the DFSP solution is indistinguishable from the ISSA solution for both stepsizes (results not shown).

when reactions are included. Molecules that react in the timestep are not diffused, and molecules produced by a reaction in the timestep are diffused for the full length of the timestep.

DFSP applied to the RDME is a first-order operator split method [121], and as such it is expected that the error should increase approximately linearly with τ_D. Figure 3.7 shows the error as a function of τ_D. We see that the error in the G-protein example is increasing approximately linearly with respect to τ_D and collapses to the error in the diffusion only system as τ_D goes to zero, confirming our expectation.

Performance

Figure 3.10 shows the speedup of DFSP over ISSA and MSA for the G-protein example. The performance increase for DFSP over ISSA and MSA is due in part to the difference in the number of times the reaction propensities must be updated as a result of diffusion events. For one realization, the expected number of diffusion events in an ISSA simulation is 1.2×10^6. Thus the reaction propensities must be updated approximately 2.4×10^6 times (source and destination voxels for each diffusion event). By numerical experimentation, the average number of reaction events for any of the methods is $\geq 170,000$. The time to the next diffusion event for MSA is given as the minimum of the time to the next reaction step

and a predetermined time step, therefore there must be at least as many diffusion events in an MSA simulation (regardless of stencil) as reaction events. For MSA, diffusion is done in all voxels, therefore updates need to be done in every voxel at each time step. Thus, the expected number of reaction propensity updates in MSA due to diffusion steps is $\geq 3.4 \times 10^7$. For DFSP with $\tau_D = 0.1$s, 1000 diffusion steps are taken, and the reaction propensities are updated in every voxel on each DFSP step, resulting in 2×10^5 reaction updates. Thus, it is reasonable to expect that for this problem DFSP will be $\sim 10^2$ times faster than MSA_2 (MSA using the two nearest neighbors in each direction) and ~ 10 times faster than ISSA for $\tau_D = 0.1$ for this problem. Figure 3.10 validates this claim.

Next we examine the effect of different spatial discretization schemes on performance. Figure 3.11 shows the computation time as a function of τ_D for three levels of mesh refinement, and the computation time for ISSA at each level for comparison. The computation time for DFSP does not vary as greatly as ISSA for different mesh sizes. In solving for the diffusion step of the algorithm, DFSP iterates over the voxels in the system and thus should scale linearly with the number of voxels. For comparison, in ISSA the number of diffusion jumps scales as $2/\ell^2$. Further calculations show that as we double the number of voxels the runtime for DFSP doubles, while for ISSA it increases by a factor of eight. This confirms our expectations.

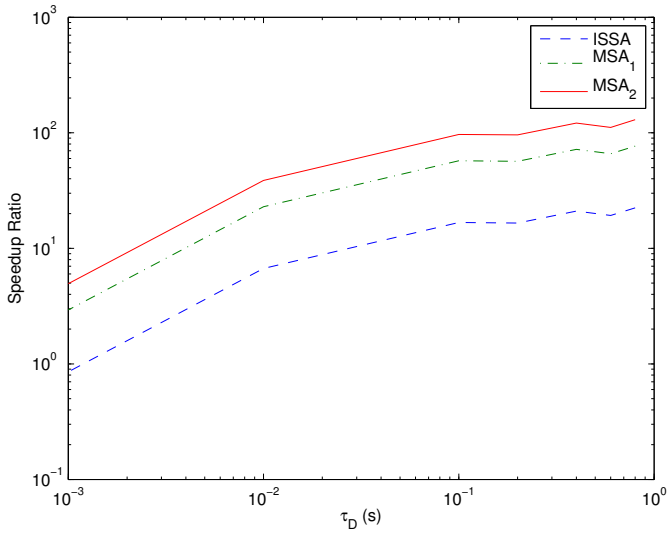

Figure 3.10: Speedup of DFSP over ISSA and both MSA stencils (subscript 1 denotes the stencil including adjacent voxels, and 2 the stencil including the two nearest neighbors on either side) for varying values of τ_D for the G-protein cycle example. DFSP achieves significant performance increases over ISSA and both MSA stencils for reaction-diffusion simulation. The speedup is due in part to the number of times the reaction propensity function needs to be updated due to diffusive transfers. DFSP updates less often than ISSA or MSA. As τ_D increases, the number of updates decreases and performance increases.

For N=100, ISSA outperforms DFSP for the range of τ_D values shown. However, for this discretization level τ_D can be as large as 3.6s. This would result in speedups for this discretization level that are equivalent to those of the N=200 and N=400 mesh sizes. For reaction-diffusion systems the operator splitting error is proportional to τ_D. Thus for coarse meshes where a large τ_D is possible, global accuracy constraints may force a selection of the τ_D and `ErrorTolerance` parameters such that ISSA performs better than DFSP.

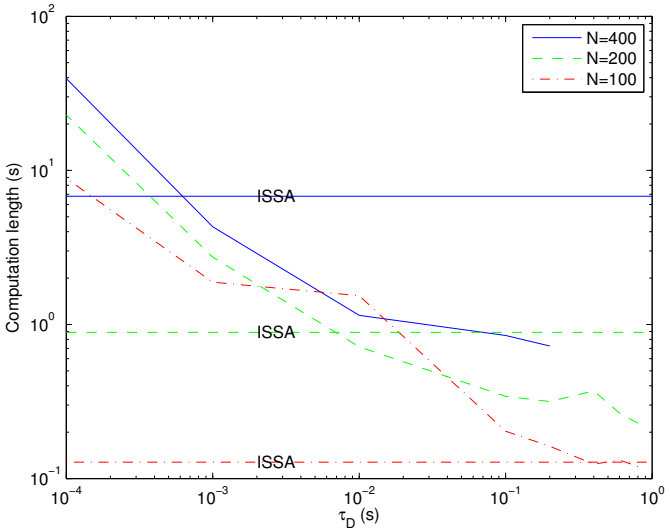

Figure 3.11: Computation time for DFSP with varying values of τ_D and varying numbers of voxels in the system with error tolerance of 10^{-5} for the diffusion example. ISSA computation times for each of the system sizes is provided for comparison.

3.3 Conclusions

DFSP is a powerful new algorithm that yields impressive performance improvements over ISSA. It provides a means to quantify and control the error, allowing a precise trade-off between accuracy and performance. Additionally, unlike many hybrid algorithms, DFSP conserves mass.

The speedup offered by DFSP enables the simulation on a workstation of ensemble sizes that were previously feasible only on high performance clusters, and it extends the scope of problems that are computable on high performance clusters. To produce our validation data for the G-protein cycle model, we needed an ensemble of 100,000 runs for statistical accuracy. The DFSP algorithm generated this data set in 6.2 hours (for $\tau_D = 0.1$s and error tolerance of 10^{-5}) and 3.8 hours (for $\tau_D = 1.9$s and error tolerance of 10^{-3}) on a commodity desktop workstation with a quad-core processor (computing four trajectories simultaneously). The ISSA data sets were generated on a high performance computer cluster, so direct comparison is not possible. However, we estimate that each of the ISSA data sets would take approximately 472 processor hours, or 118 real hours (approximately 5 days) to calculate on the desktop workstation.

Chapter 4

URDME: a Modular Framework for Stochastic Simulation of Reaction-Transport Processes in Complex Geometries

Experiments *in silico* using stochastic reaction-diffusion models have emerged as an important tool in molecular systems biology. Designing computational software for such applications poses several challenges. First, realistic lattice-based modeling for biological applications requires a consistent way of handling complex geometries, including curved inner- and outer boundaries. Second, spatiotemporal stochastic simulations are computationally expensive due to the fast time scales of individual reaction and diffusion events when compared to the biological phenomena of actual interest. We therefore argue that simulation software needs to

be both computationally *efficient*, employing sophisticated algorithms, yet in the same time *flexible* in order to meet present and future needs of increasingly complex biological modeling.

In collaboration with Andreas Hellander and Stefan Engblom of Uppsala University, we have developed URDME, a flexible software framework for general stochastic reaction-transport modeling and simulation [26]. URDME uses **U**nstructured triangular and tetrahedral meshes to resolve general geometries, and relies on the **R**eaction-**D**iffusion **M**aster **E**quation formalism to model the processes under study. An interface to a mature geometry and mesh handling external software (Comsol Multiphysics) provides for a stable and interactive environment for model construction. The core simulation routines are logically separated from the model building interface and written in a low-level language for computational efficiency. The connection to the geometry handling software is realized via a Matlab interface which facilitates script computing, data management, and post-processing. For practitioners, the software therefore behaves much as an interactive Matlab toolbox. At the same time, it is possible to modify and extend URDME with newly developed simulation routines. Since the overall design effectively hides the complexity of managing the geometry and meshes, this means that newly developed methods may be tested in a realistic setting already at an early stage of

development.

In this chapter we present URDME, originally published in [26]. We demonstrate, in a series of examples with high relevance to the molecular systems biology community, that the proposed software framework is a useful tool for both practitioners and developers of spatial stochastic simulation algorithms. Through the combined efforts of algorithm development and improved modeling accuracy, modeling of increasingly complex biological models has become possible. URDME is freely available at http://www.urdme.org.

4.1 Background

Stochastic simulation of reaction kinetics has emerged as an important computational tool in molecular systems biology. In cases for which mean-field analysis has been shown to be insufficient, stochastic models provide a more accurate, yet computationally tractable alternative [126, 106, 9]. For example, a frequently studied topic is the mechanisms for robustness of gene regulatory networks relative to intrinsic and extrinsic noise [123, 33, 112]. In a stochastic mesoscopic model the time evolution of the number of molecules of each species is described by a continuous-time discrete-state Markov process. Realizations of this process

can be generated using techniques such as the Stochastic Simulation Algorithm (SSA) [52].

If the system can be assumed to be spatially homogeneous, or well-stirred, simulations are simplified considerably compared to a spatially varying setting. However, there are many phenomena inside the living cell for which spatial effects play an important role [41, 124]. In such cases, a mesoscopic spatial model can be formulated by first discretizing the computational domain into subvolumes, or voxels. Molecular transport processes are then modeled as combined decay- and creation events that take a molecule from one voxel to an adjacent one [129, 44]. For appropriate discretizations [70, 37], the assumption of spatial homogeneity holds approximately within each voxel, where reactions can be simulated as in the well-stirred case. The governing equation for the probability density function is called the Reaction Diffusion Master Equation (RDME) and methods to generate realizations in this framework have been used previously to study reaction-diffusion systems in the context of molecular cell biology [11, 41, 31, 122].

Modern experimental techniques can provide information not only on the total copy numbers but also on the spatial localization of individual molecules [110, 32]. As such techniques are further developed and spatial models can be calibrated to biological data, methods and software for flexible and efficient simulation of spatial stochastic models will likely continue to grow in importance. As a coarse-grained

alternative to detailed microscopic models based on Smoluchowski reaction dynamics [140, 4], or other similar microscale simulators such as MCell [75], simulations in the RDME framework are orders of magnitude faster than microscopic alternatives [62].

For most applications, a large number of sample realizations need to be generated to allow for a useful statistical analysis. Exploring parameter regimes or estimating responses to different stimuli adds to the complexity so that the generation of tens of thousands of independent realizations is not uncommon. Computational efficiency is therefore an important concern and has motivated research in many types of approximate or optimized methods (see for example [79, 114, 88, 10, 42]).

Despite advances in the development of approximate methods, spatial stochastic simulation in realistic geometries is still challenging. One of the main reasons is the complexity involved in handling the 3D geometry and the associated mesh. The purpose with this chapter is to introduce URDME, a modular software framework for spatial stochastic simulation. The goal of URDME is twofold: first, it provides applied users with a powerful and user-friendly modeling environment that supports realistic geometries. Second, URDME facilitates the development of new computational methods by taking care of the technical details concerning the geometry, the mesh generation, and the assembly of local rate constants. By providing a well-defined interface to the modeling environment, new algorithms

can be incorporated into the URDME framework as plug-in solvers. We anticipate that this modular structure will facilitate the development and dissemination of advanced simulation methodologies to real-world molecular biology applications.

URDME differs from other public domain software for mesoscopic simulations such as MesoRD [58] or SmartCell [3], in that it uses unstructured tetrahedral meshes to discretize the domain, offering a much greater geometrical flexibility and better resolution of curved surfaces compared to Cartesian meshes. URDME shares its utilization of tetrahedral meshes with another reaction-diffusion simulation software, STEPS [62], which we will discuss later in the chapter. One of the defining features of URDME is that it is structured to be highly modular in order to be useful as a platform for developers of the associated computational tools. This design also allows for flexible work-flows for result generation. When used interactively, URDME's Matlab interface provides for convenient model construction and evaluation. Since the solvers are automatically compiled into optimized stand-alone executables, URDME can also be used to define batch jobs using the very same Matlab interface. In this way, URDME is a convenient platform both in the initial modeling phase as well as when performing high-performance and/or high-throughput computational analysis.

4.2 Implementation

In this section we describe how the URDME framework is structured, how it is used to simulate a model, and how to interface with it to add new simulation algorithms. For more details, refer to the software manual [25].

Overview The URDME framework consists of three logical layers connected by well-defined interfaces (see Figure 4.1). At the top level, a third-party software for mesh-generation is used to define the geometry and to generate the mesh. Currently, URDME interfaces with Comsol Multiphysics 3.5a for this functionality. The middle layer routines in Matlab serve as an interactive environment for model construction, and connects the geometry and mesh-handling facilities of Comsol with the core simulation algorithms (bottom layer).

With this modular structure, the top level can be replaced by other mesh generation software such as for example Gmsh [47], provided that the appropriate interface routines are added to the middle level interface. Relying on Comsol Multiphysics for the geometry definition and mesh-generation provides for a convenient interactive environment for the model construction, allowing advanced models to be formulated quite easily.

The default core solver at the bottom level is an optimized implementation of the Next Subvolume Method (NSM) [41]. Since the solver layer is kept separate

from the model building interface, new solvers can easily be added to URDME while taking advantage of all of the infrastructure related to model management and post-processing. The data passed to the solvers is well-defined and documented (see [25] for more information). It is our goal for URDME to grow through the contribution of solvers from the community. One such solver has already been contributed and distributed in this way: the diffusive finite state projection (DFSP) algorithm [28], described in Chapter 3 of this dissertation. Additionally, the URDME framework has been utilized in the development of new algorithms [36, 42, 59] and in a master equation formulation of active transport on microtubules [60].

Using URDME for model development and simulation The process of analyzing a reaction-diffusion model with URDME begins with the creation of a Comsol model file that defines the geometry of the domain, including (optionally) the subdomains where specific *localized* reactions are to be defined (e.g. membrane, cytosol, and nucleus). At this stage, the biochemical species and their associated diffusion rates are also defined. Once the model is set up, the mesh generation facilities of Comsol are used to create a tetrahedral discretization of the domain. Next, this information is exported to Matlab via an API connection as illustrated in Figure 4.2A (top). The interface routines of URDME are then

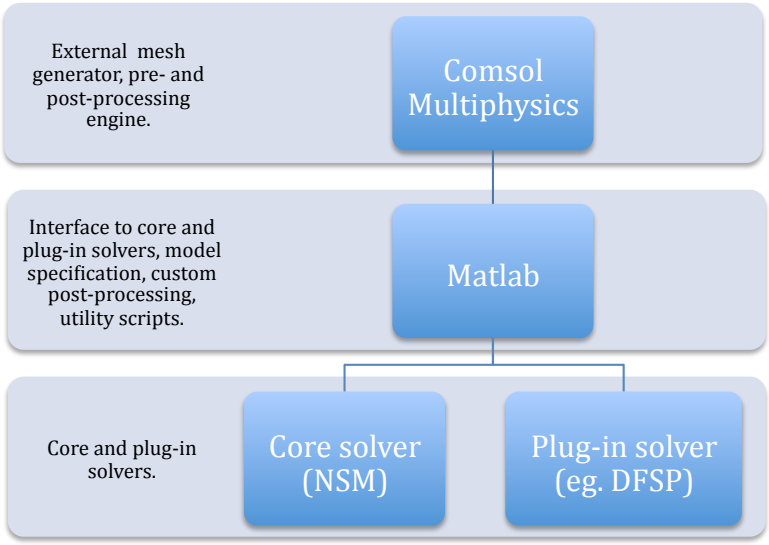

Figure 4.1: The URDME framework consists of three loosely coupled layers. Solvers reside at the bottom level and are most often written in a compiled language like ANSI-C. The middle layer provides for interfaces between the solvers and the top-level mesh-generation infrastructure. Both the top- and the bottom-layer may be replaced by other software as long as the middle level is extended appropriately.

used to assemble the data structures needed by the core simulation routines. The overall process is summarized in Figure 4.2A (bottom).

Apart from defining the geometry, the user also needs to create two additional program files to be used by URDME. The first is a Matlab function (referred to as the *model file*), that defines the data related to the actual simulation. This includes the initial distribution of molecules, the stoichiometric matrix defining the topology of the reaction network, a certain dependency graph for events in the model, and the simulation interval (for a detailed list, see [25]). This model file can also be used to define custom configurations for the model, including restricting a species to a specific subdomain, adding modified transport terms, and evaluating expressions over the geometry such that this information can be passed on to the core solver. In this way, URDME supports custom modeling that would be very hard to achieve with a less flexible software architecture. This, we argue, is one of the defining and unique features of the URDME framework.

The second program file a user must create is a templated C-program file that defines the propensity functions for the chemical reactions of the model. This file defines one function for each chemical reaction in the system, which are called by the core solver routines to calculate the propensity for each reaction in each voxel. The propensity function template requires the output to depend only on the system state at the current time, but is unique to a voxel and allows for

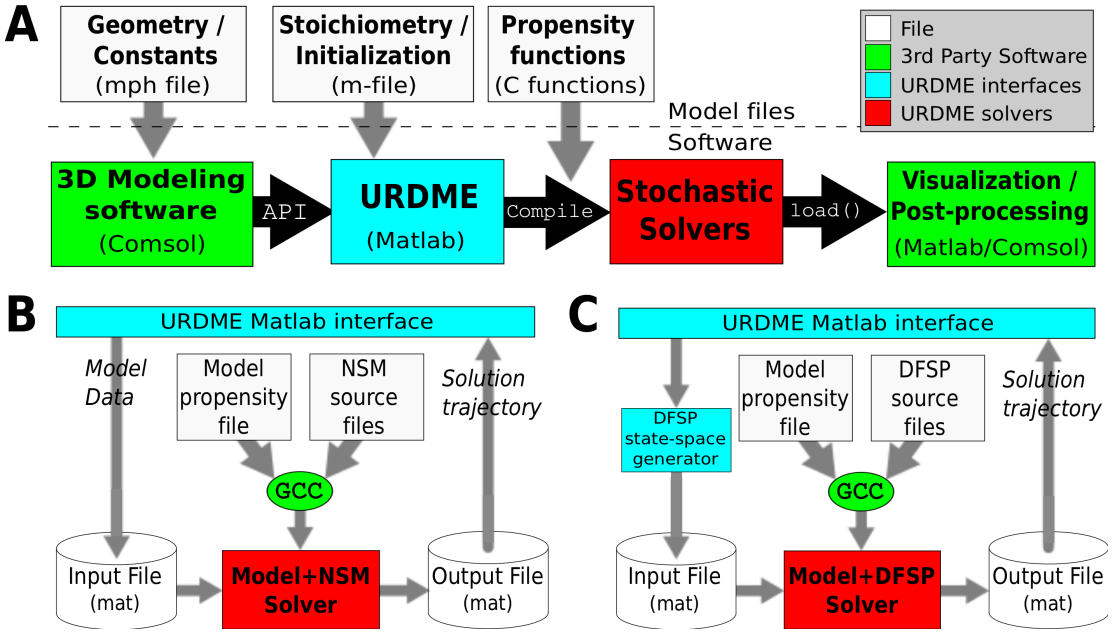

Figure 4.2: **(A)** URDME flow diagram for the complete simulation process. **(B)** Process flow diagram for the stochastic simulation step of (A) using the NSM solver. **(C)** DFSP solver flow diagram, an alternative to (B) for the stochastic simulation step.

additional data to be passed on to the function. The propensity function file is later automatically compiled and linked with the core solver, resulting in a highly efficient solution procedure.

Once the model data structure has been exported to Matlab and the model and propensity functions have been defined, the next step is to let URDME execute a simulation of the model. From the users' perspective, simulation now only requires to invoke the `urdme` function in Matlab with the proper arguments,

```
>> model = urdme(model,@model_file,...
         {'Propensities','propensity_file'});
```

The arguments passed are the Comsol data structure, the model function, the propensity functions, and various optional arguments. URDME now invokes GCC to compile the propensity function file with the specified solver (defaulting to NSM) to create a dedicated executable for the model. This executable is then invoked using the model and geometry data structure as inputs. Note that compilation and execution of the low-level components of the system is fully automatic, and requires no additional action from the user. Following a successful execution of the core solver the `urdme` function returns a modified model data structure with a single stochastic solution trajectory attached to it.

Since the layers of URDME are decoupled, it is also possible to execute the solvers in non-interactive batch mode to allow for more flexible result generation and distribution of computations on a multicore platform. For example, to conduct the simulation in background mode and write the resulting trajectory to the file 'output.mat' one simply invokes `urdme` with a few additional arguments,

```
>> model = urdme(model,@model_file,...
            {'Propensities','propensity_file','Mode','bg',...
            'Output','output.mat'});
```

Here, control returns to Matlab directly after execution of the solver executable, without waiting for it to complete.

Visualization and post-processing are important components in most simulation software. Once a URDME simulation is complete, users can easily visualize

the spatially varying concentration of biochemical species in their model by using Matlab's interface to the Comsol graphics routines. Examples of this will be presented in the Results section. Similarly, most modeling and simulation projects require custom data analysis once the simulation data has been generated. To facilitate this, URDME supports the creation of post-processing scripts in Matlab using its native high-level scripting language and computational libraries. Examples of complex post-processing routines implemented as Matlab functions and scripts are available as part of the example directories in the URDME software distribution package.

Structure and implementation of core simulation algorithms Taken together, the components of URDME that were introduced in the previous section create a flexible and expandable platform. While an applied user need not know any details about how a core solver is implemented, the developer of a new simulation algorithm can use the infrastructure to implement a plug-in solver to URDME. Figure 4.2C illustrates the structure of the plug-in solver that implements the DFSP algorithm [28]. Note the similarities with the flow diagram of the core NSM solver in Figure 4.2B. URDME plug-in solvers have three main components: a Makefile, the solver source files, and (optionally) a pre-execution script intended to be invoked by the middle-level scripting interface. The solver Make-

file is used for compiling and building the solver automatically from the Matlab interface. The name of this file tells URDME what solver it builds; when `urdme` is invoked with the option to run a simulation using a specific solver, it will look for a Makefile with the correct naming pattern. This Makefile then compiles the solver along with the propensity functions associated with the model being simulated into a stand-alone binary executable. Hence a different and unique executable is automatically produced for each new combination of model and solver.

The source code of the solver itself can formally consist of any number of files in any language as long as the Makefile can create the final executable called by the middle-level interface. To enable a seamless integration with the URDME Matlab interface, the URDME C API contains library routines to read and parse the data structures generated by the URDME model files. These API routines will parse all data-structures required by the core NSM solver. A plug-in solver that needs additional input will have to make sure that these are parsed correctly as part of the solver main routines. To pass such additional data to the solver, it need only be appended to the '`model.urdme`' field, either by the Matlab model file, or by a pre-execution script (compare Figure 4.2C). URDME will then write this data to the solver input file. Such a pre-execution script is an optional component of the solver integration. Simply put, when executing a model, URDME always looks for a Matlab function defined in the file 'urdme_init_<solver>.m'.

All current solvers are written in ANSI-C and use GNU-style Makefiles. The process of integrating a simulation algorithm in the URDME framework is described in more detail in [25] and is also exemplified by the source code for the DFSP plug-in that is included in the URDME distribution.

In conclusion, when all the components of a solver are in place as described above, the only difference to an end-user of URDME is a single additional argument

```
>> model = urdme(model,@model_file,...
           {'Propensities','propensity_file','Solver','dfsp'});
```

The use of the URDME framework to implement and analyze the performance of a simulation algorithm will be further described in the Results section.

4.3 Results

In this section we will use three different examples to illustrate how the design of URDME makes the software framework a useful tool to accomplish different simulation tasks. In the first example we show how an established model from the molecular systems biology literature is simulated in URDME. This example illustrates the powerful nature of the URDME scripting environment in setting up and conducting a parameter sweep. In the second example we demonstrate how URDME can aid in the development of efficient simulation algorithms by

explaining how the Diffusive Finite State Projection (DFSP) method [28], was integrated into URDME as a plug-in solver. As a final example we simulate a model of molecular transport in a neuron. Here, the unstructured mesh is an absolutely necessary feature in order to be able to resolve the complex geometry. We also show with this example how a model of active, molecular motor driven transport as proposed in [60] can be implemented in URDME to simulate molecular transport in the different parts of the neuron.

4.3.1 Simulating Min oscillations in *E. Coli*

In *E. Coli*, the Min family of proteins are believed to play a key role in the regulation of symmetric cell division. In a mechanism thought to be self-organized and to function in a manner similar to the formation of Turing-patterns, the MinD protein oscillates from pole to pole with a period close to 40 seconds. Another Min protein, MinC, co-localizes with MinD and acts as a repressor for the formation of the cell division site by destabilizing Ftz polymerization [67]. On average, MinD (and hence MinC) will spend less time near the center of the cell, allowing the division ring to assemble there. Both deterministic and stochastic models of this system have been studied previously in the literature [67, 41].

To illustrate how to use URDME to conduct a parameter sweep we will simulate the Min-system for increasing lengths of the bacterium and observe the

behavior of the oscillations. The example is representative for how experiments using different sets of parameters can be defined and organized with the current version of URDME. A detailed account of how to create all model files to run simulations of the model from [41] can be found in the software manual [25] in the form of a tutorial. There, the model is run interactively from the Matlab prompt as detailed in the previous sections. In order to conduct the experiment outlined here in the same fashion we would have to manually rebuild the geometry and execute the simulations for the different parameter cases. This would be time-consuming and error prone. Instead, we explain how to automate such a task by using the Matlab scripting environment and the URDME Matlab interface. Table 4.1 shows how the parameter sweep can be specified in a simple script in the Matlab language. The function 'coli_model' was automatically generated from the Comsol interface using the model of an *E. coli* bacterium shown in Figure 4.3A. It was then slightly modified by manipulating the original consecutive solid geometry (CSG) description. The geometry of the bacterium was parametrized by creating a copy of the original geometry and then translating it along the x-axis. The union of these two objects is the final geometry and the variable 'xsep' specifies the extent of the translation. Note that, as shown in Figure 4.3C, the bacterium will ultimately split into two separate geometric objects.

The results of the parameter sweep are summarized in Figure 4.3. Figure 4.3A shows the geometry of a model of an *E. Coli* bacterium with length 4.5 μm and radius 0.5μm discretized with a tetrahedral mesh. Figure 4.3B shows the temporal average of membrane bound MinD obtained in a simulation of the model from [67] with URDME, as well as a time series of pole-to-pole oscillations of the membrane bound fraction of MinD. As can be seen, the model predicts a minimum of MinD near the center of the cell. Figure 4.3C shows a visualization of the *E. Coli* bacterium at six different lengths, including the temporal average of the relative concentrations of the MinD protein. Figure 4.3D shows the stability of oscillations when increasing the 'xsep' parameter.

For values of the parameter 'xsep' less than about 2μm, coherent oscillations are observed and the MinD protein is concentrated at the poles of the bacterium. For larger values, the oscillations cease and MinD is distributed evenly in the cell. Hence, in order to maintain oscillations also for longer cells, the model needs to be modified in some way. For example, the total copy number of MinD is currently kept constant as the cell grows. Different initial conditions such as constant concentration can of course be tested with equal ease by making the appropriate changes to the model file.

In this example, URDME is invoked in background mode allowing for several parameter cases to be run in parallel on a multicore workstation. Instead of

returning the results directly in the workspace, we direct URDME to store the result files and the input files on disk for later post-processing.

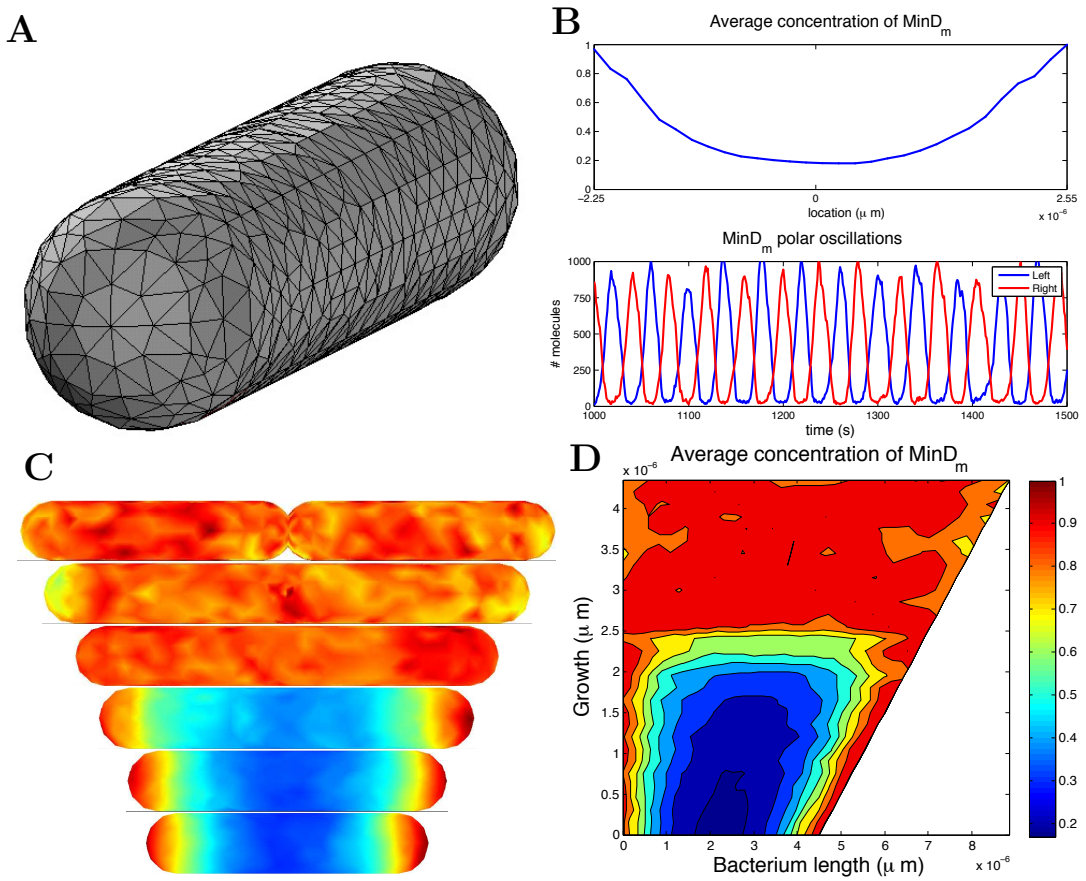

Figure 4.3: **(A)** Geometry and mesh modeling of an *E. Coli* cell. **(B)** Temporal average concentration of MinD protein as a function of position along the long axis of the *E. Coli* cell (top), and the time series plot of the oscillations. **(C)** Six *E. Coli* cells of increasing lengths, as specified in the parameter sweep described in Table 4.1. The color intensity shows the temporal average concentration of MinD protein along the membrane. **(D)** Parameter sweep shows how the relative concentration of MinD changes as the bacterium grows.

```
% Define the parameter space
Nval = 30;
xsep = linspace(0,4.5e-6,Nval+1);
xsep(end) = []; % (avoid creating two distinct bacteria)
save results/info.mat xsep

for i = 1:Nval

  % Generate the E. coli cell by merging two cells with
  % separation 'xsep(i)' along the positive x-axis
  fem = coli_model(xsep(i));

  % run an instance of URDME in background mode
  fem = urdme(fem,@huang,{'Propensities','huang', ...
                    'Mode','bg', ...
                    'Outfile',sprintf('results/out%d.mat',i)});

  % save input separately for later use
  save(sprintf('results/in%d.mat',i),'fem');
end
```

Table 4.1: Matlab script that executes simulations of the Min-model in a geometry modeling an *E. Coli* cell with varying length. URMDE executes the core simulation algorithm in the background and saves the results and input files for later post-processing, allowing for many points in parameter space to be simulated in parallel on a multicore workstation.

4.3.2 Developing and benchmarking a new algorithm for spatial stochastic simulation

Generally, a large fraction of the effort in developing simulation tools goes into software infrastructure as opposed to code pertaining to the underlying solver algorithms. URDME is designed to provide that infrastructure. The first two layers

of the framework provides handling of geometry and meshing, assembly of diffusion jump-rate constants, model integration, pre- and post-processing and data visualization. In this section we illustrate how to use URDME's infrastructure to enhance the development and benchmarking of a new stochastic simulation algorithm, DFSP [28], described in Chapter 3 of this dissertation. We describe the components of this solver and how they are integrated with URDME. This example may therefore serve as a design pattern for algorithm integration into the URDME framework.

Integration of a new solver into the URDME framework is designed to be a simple process, with the largest fraction of the required new code being specific to the underlying solver algorithm. URDME solvers have three main components: the solver source code, a Makefile, and an optional pre-execution script. The Makefile creates a standalone Unix executable from the source code. The DFSP solver uses a pre-execution script in Matlab to calculate data specific to the algorithm. This data is then added to the input file that URDME creates upon execution of the solver. Table 4.2 describes the files that are part of the DFSP solver.

In addition to the lower integration overhead of implementing a new algorithm in the URDME framework, URDME allows developers to easily benchmark their solvers. Table 4.3 shows a Matlab script that sets up a benchmarking experi-

Directory	File	Description
urdme/build	Makefile.dfsp	Solver Makefile.
urdme/src/dfsp	dfsp.c	Solver entry point and data initialization.
	dfsp.h	DFSP header file.
	dfspcore.c	Main entry point for the solver.
	dfsp_reactions.c	Simulates reaction events.
	dfsp_diffusion.c	Simulates diffusion events.
urdme/msrc	urdme_init_dfsp.m	Matlab pre-execution script.

Table 4.2: Overview of the files that make up the DFSP plug-in solver. This structure follows the general design pattern suggested for solver integration in the URDME framework and is very similar to the structure of the default NSM core solver (see [25]).

```
% DFSP Performance and Error benchmark code
tic;
solution = urdme(fem,@fange,{'Solver','nsm',...
                  'Propensities','fange'});
nsm_simulation_time = toc
nsm_period = find_mincde_period(solution)
for tau_D = [ 0.001, 0.005, 0.01, 0.05, 0.1, 0.5 ]
   tic;
   solution = urdme(fem,@fange,{'Propensities','fange',...
       'Solver','dfsp','tau',tau_D,'max_jump',10,...
       'DFSP_cache',dfsp_cache_filename});
   dfsp_simulation_time = toc
   dfsp_period = find_mincde_period(solution)
   error = abs(dfsp_period-nsm_period)/nsm_period
end
```

Table 4.3: Matlab code for benchmarking the DFSP solver using the MinCDE model of oscillations in *E. coli*.

ment to assess the performance and error of the DFSP solver when simulating the model for Min-oscillations described in the first example in this chapter. This code also illustrates the calling signature for the `urdme` function when used with the NSM and DFSP solvers. The DFSP solver takes the additional arguments 'tau' as the time-step, 'max_jump' as the maximum spatial jump distance, and 'DFSP_cache' as the cache file used to store the data specific to the DFSP algorithm. The utility function `find_mincde_period` finds the peak period of the oscillations through straightforward spectral analysis using built-in routines in the Matlab scripting environment, again illustrating the advantage of using the scripting layer's post-processing capabilities. Figure 4.4 shows the results of the benchmarking experiment. We find that the DFSP method with $0.01 < \tau_D < 0.1$ produces simulation results faster than NSM and with good accuracy in the oscillation period.

4.3.3 Active transport in a neuron

Diffusion is the dominating mechanism of molecular transport in prokaryotes such as *E. Coli*, and it was in that context the NSM was first applied [31, 41]. However, diffusion is not the only mechanism for molecular transport in eukaryotic cells. Intra-cellular cargo can be transported by motor proteins along cytoskeletal structures made up of microtubule and actin polymers [65, 86, 128]. Molecular

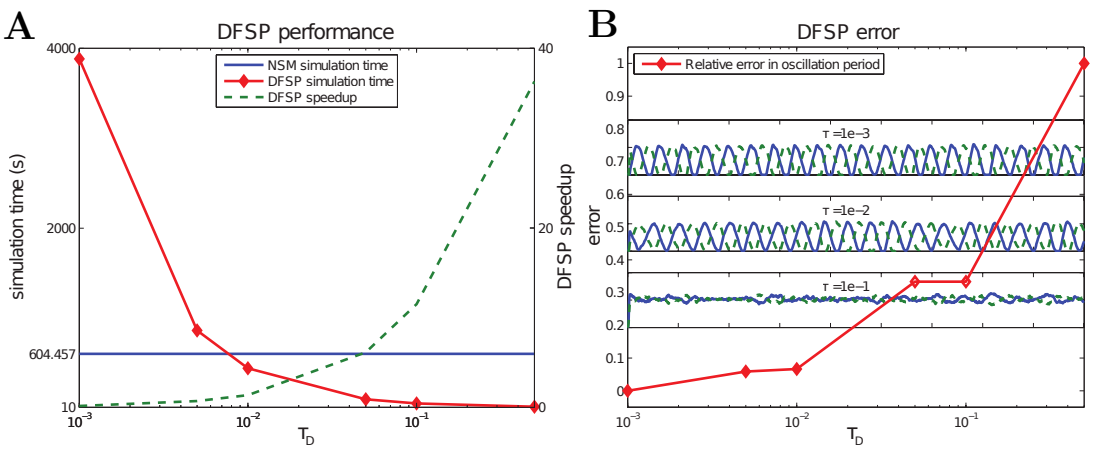

Figure 4.4: DFSP benchmark results. **(A) Performance of DFSP** shows a comparison of simulation times for DFSP at varying τ_D values (red) and NSM (blue), and the DFSP speedup factor (green). For this model, DFSP outperforms NSM for $\tau_D > 0.01$. **(B) Error in DFSP** shows the relative error in MinCDE oscillation period (red) and the oscillation patterns for three simulations. Simulations with $\tau_D < 0.1$ produces coherent oscillation patterns and result in a negligible error. The system was simulated to a final time 900s. Simulations were performed on a 1.8 Ghz Intel Core i7 processor.

motor proteins bind to the cargo and to the filaments and move the cargo along the fiber, always in a specific direction depending on the type of motor and fiber. This transport is usually much faster than diffusion but requires additional energy. Vesicles, organelles, mRNA and proteins involved in signaling are examples of cargo that are transported in this way inside living cells.

Due to the ubiquity of active transport in biological systems, it is important that simulation software have the capability to handle mesoscopic models with general transport mechanisms. In [60], the RDME was extended to include an advection term that models cargo transport on the microtubule network. A sim-

ple model of signaling in a yeast cell was considered and URDME was used for model development and simulation. To illustrate both the geometrical flexibility of URDME as well as its capability to model more general transport mechanisms, we show here how to simulate active transport in a model of a neuron with a detailed geometry.

Active transport of cellular cargo is of fundamental importance to maintain the highly polarized state of a healthy neuron. In the axon, microtubules are uniformly oriented with plus-end towards the soma and minus-end towards the synapse. Kinesin transports cargo in the anterograde direction, from the cell body to the synapse. For example, kinesin drive the transport of synaptic vesicles from the cell body through the axon where they are subsequently docked to the plasma membrane in the presynaptic terminus. Dynein drives transport in the opposite direction (retrograde transport) in the axon, and may aid in transporting for example RNA from the cell body to the dendrites [89]. In the dendrites, the situation is more complex than in the axon, since the microtubules form an array of mixed orientation. While the particular motor protein transports cargo in a specific direction on the fibers, a single cargo such as a vesicle can have many different motors bound to it simultaneously and therefore may move in a bidirectional manner [135, 56, 35]. The details of how kinesin and dynein-driven transport is coordinated and regulated to achieve differential targeting and

localization of cargo is still a largely unresolved issue [54, 116]. As an example of a possible mechanism of regulation, the microtubule binding protein Tau effects the binding affinity of kinesin to the microtubule, while dynein is less sensitive to elevated Tau concentrations [22].

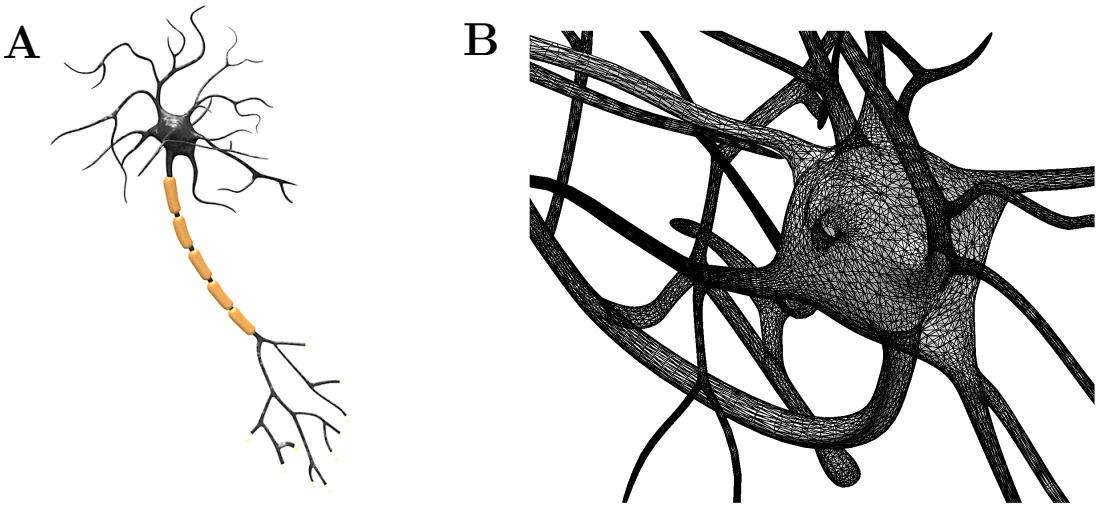

Figure 4.5: The neuron geometry **(A)** is based on a artistic CAD rendering generated with the public domain version of the software Blender (http://www.blender.org). In order to conduct simulations in this geometry, the model was exported in the STL surface mesh format, imported into the open-source meshing package Gmsh [47], where the boundary was re-parametrized and the domain subsequently meshed with a volume mesh in 3D. The resulting mesh was then converted into a Comsol Multiphysics 3.5a model to serve as a geometry description for the URDME model. Assembly of active transport jump rate constants were conducted by URDME on the unstructured mesh shown in **(B)**. For a mathematical background on how to obtain these constants on the unstructured mesh, see [60]. URDME's capability to use an unstructured mesh made up of tetrahedral and triangular elements is of vital importance in order to be able to resolve the complex geometry of the neuron.

To illustrate how diffusion and active transport can simultaneously be modeled with URDME in the neuron geometry we consider a straightforward model where a

cargo species is transported to different regions of the neuron. The motor proteins are modeled *implicitly*, that is, we assume that a population of motor proteins is associated to the cargo species at all times. Although an approximation, there are recent experimental evidence that the distributions of motors on vesicles are relatively stable [35]. Table 4.4 summarizes the model. The cargo species V is created uniformly in the cell body (R1). V can diffuse and bind reversibly to microtubule filaments, either with a kinesin motor as V^k or with a dynein motor as V^d (R2–R5). When bound to a filament, V is actively transported in a direction dictated by the kind of motor that is currently active. The cargo can reverse its direction on the fiber in bidirectional transport by letting the currently active motor protein change with some probability (R6,R7). The quotient σ_{kd}/σ_{dk} then dictates the direction of net transport. Finally, V is uniformly degraded (R8) in the whole neuron so that the total number of cargo V reaches a steady-state level.

	Reaction	Description	Cellular location
(R1)	$\emptyset \xrightarrow{\mu_1} V$	Creation of cargo	Cell body
(R2–R5)	$V \underset{\sigma_d}{\overset{\sigma_b}{\rightleftharpoons}} V^{k,d}$	Binding of V to microtubule	All domains
(R6,R7)	$V^k \underset{\sigma_{dk}}{\overset{\sigma_{kd}}{\rightleftharpoons}} V^d$	Reversal of direction	Microtubule
(R8)	$V \xrightarrow{\mu_2} \emptyset$	Degradation of V	All domains

Table 4.4: Model of active transport of a cargo species V that is transported on microtubule filaments in a direction determined by the orientation of the fibers (as modeled by a velocity field) and the current motor protein bound to the fiber (kinesin or dynein). See text for more details.

To illustrate the ability of cargo to localize to different compartments of the cell depending on the dominating motor protein, we consider the following scenario. First, we let $\sigma_{dk} = 10\sigma_{kd}$, so that on average, kinesin will spend more time bound to the microtubule than will dynein. In this case, the cargo will travel through the axon and eventually localize to the axon terminus. After half of the total simulation time has elapsed, the situation is reversed and $\sigma_{kd} = 10\sigma_{dk}$ such that the cargo will localize to the dendrites.

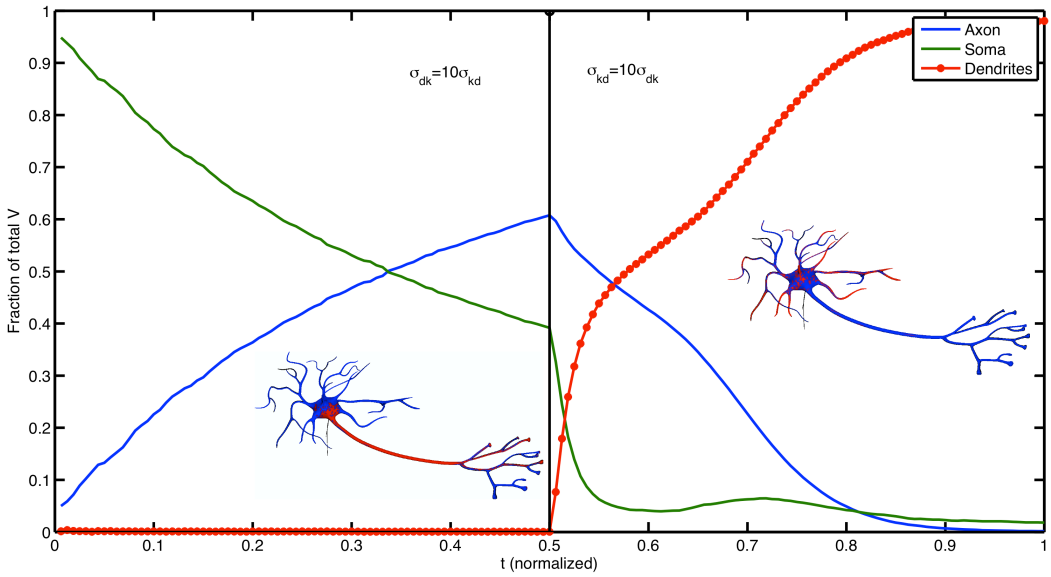

Figure 4.6: Normalized concentration of cargo V in the soma (green), axon (blue) and dendrites (red) as a function of time. Initially, the parameters satisfy $\sigma_{kd} = 10\sigma_{dk}$ and cargo localizes to the axon due to the larger fraction of time spent in the kinesin binding state. At time $t = 0.5$ the situation is reversed, and the localization of V shifts from axon to dendrites. The red regions in the inlays depicting the neuron show the areas where V is present.

Figure 4.6 shows a typical output of a simulation with URDME. The concentration V of cargo is plotted in the different regions of the neuron geometry (axon, soma, and dendrites). Since the purpose of this example is to illustrate the capability of URDME to model both diffusion and active transport in a complex geometry, the values of the various parameters have not been chosen to fit any particular neuron geometry. Hence the velocity of dynein in the axon is conveniently set to be half that of kinesin. Also, the net rate of transport of dynein in the axon is set to be one hundredth of the rate of transport of kinesin, to reflect the effects of mixed polarity of fibers [63].

In order to set up this simulation in URDME, a Matlab function for the velocity field modeling the average orientation of the fibers at any point in the domain needs to be provided. Obviously, specification of this velocity field requires biological knowledge. The ability to work in the Matlab environment greatly simplifies parametrization of the velocity field. Since this geometry was given as a surface mesh, which is also often the case when the domain is obtained from cell imaging, we have no analytical expression for the parametrization of the geometry to rely on. In this example we want the velocity field to trace the axon and dendrite structures. To achieve this, we first compute surface normals to all triangles on the surface of the neuron. An interpolation table containing vectors with base in the centroids in the triangles of the surface mesh and pointing in the direction

of suitably chosen reference points was thus constructed. For simplicity, we used two different reference points, one near the center of the cell body and the other beyond the axon terminus along the long axis of the axon. The smoothness of the velocity field can easily be improved by adding more reference points. For any point inside the domain, we evaluate the velocity by nearest neighbor interpolation using the interpolation table. From this description of the microtubule network and the information about the mesh, utility routines available as add-ons to the basic URDME package can be used to assemble jump rate constants to be used in the definition of the stochastic transport process in much the same way as for diffusion [60]. This procedure may seem complicated at a first glance, but can be performed quite easily in Matlab using built-in utility routines. The model files required to run this example can be found in [26].

4.4 Discussion

The design of URDME is motivated by both modeling and algorithm development. Systems biology investigations are typically computationally intensive, and often require large ensembles of trajectories spanning parameter space to match data, or to conduct a sensitivity and robustness analysis. Development of more efficient simulation methods is needed to make such large scale investigations

feasible. However, due to the overhead of handling complex geometries, mesh generation and visualization of results, algorithm developers often tend to consider only simple test models in simple geometries, often restricted to one or two spatial dimensions. While this can be enough to illustrate the potential benefits of a new method, the resulting software is often not general enough for use on complex biological models. URDME aims to bridge this gap by providing a large part of the infrastructure needed for simulation of realistic models. We exemplified this in this Chapter by the application of the DFSP algorithm in a full 3D simulation.

The theory and methodology for spatial stochastic simulation is still undergoing extensive development, and no single mathematical modeling framework or method has emerged as a *de facto* standard. The utility of the URDME framework is not restricted to mesoscopic RDME simulations; we have used URDME to develop solvers based on the Smoluchowski model and a microscopic–mesoscopic hybrid methods [59].

Another benefit of the modular architecture is that it simplifies the use of different execution models for the simulations. As part of work on methods for enactment of computation in grid environments, we have developed a URDME server module that enables remote execution in distributed computing environments [102, 103]. This enables highly task-parallel investigations to utilize dis-

tributed computational resources such as clusters, grids, and clouds to greatly increase productivity for the end-user.

Comparison of spatial stochastic software packages To further illustrate the design of our software, we have compared its features to two other publicly available packages for mesoscopic spatial stochastic simulation. Table 4.5 shows a comparison between URDME 1.1, MesoRD 1.0, and STEPS 1.3. MesoRD was one of the first software projects aimed at simulation of the RDME. STEPS was developed for simulation of detailed models of dendrites and synapses, but is generally applicable to a lager set of reaction-diffusion models.

There are three significant ways in which a user interacts with a spatial stochastic software package: the environment for model development, execution of a simulation, and post-processing and analysis of the data generated by the simulation. The interface and model development environment used by URDME and STEPS are similar in that both are closely tied to a programming language environment: Matlab in the case of URDME and Python for STEPS. URDME provides a single function entry point, and models are developed in external programming files. This design pattern follows that of the Matlab ODE suite. STEPS provides an object oriented Python interface for creation, simulation and post-processing of models. STEPS claims that a programmatic interface offers significant advan-

	URDME 1.1	**MesoRD 1.0**	**STEPS 1.3**
Interface	Matlab & Comsol	Command line Simple GUI (Windows)	Python
Visualization	Matlab & Comsol	OpenGL tool Matlab toolbox	PyOpenGL tool
Post-processing	Matlab	3rd party	Python
SBML support	Conversion tool (no geometry)	SBML L2v4 + CSG geometry	Import module (no geometry)
Edit Geometry	Comsol	SBML	3rd party
Mesh Type	Vertex centered Tetrahedrons	Uniform Cartesian	Body centered Tetrahedrons
Algorithms	NSM, DFSP + extendable	NSM +non-local extension	Spatial-SSA
Propensity types	All	SBML (MathML)	Mass-action
Model Features	compartments surfaces volume diffusion surface diffusion directed transport	compartments volume diffusion	compartments surfaces volume diffusion

Table 4.5: A comparison of features of RDME simulation software.

tages over a non-interactive software interface [62] (in contrast to the command line and input file interface), and we share this opinion. The major differences between URDME and STEPS are the feature set and the performance. The execution platform of URDME is the Matlab-Comsol environment, thus URDME has full access to the scientific libraries of Matlab as well as the advanced geometry and mesh handling interface of Comsol. Another major difference is one of aim. URDME is developed by a team of biological model developers as well as of algorithm developers, and it aims itself at both communities. This is reflected in its expandable solver interface and performance centric design. STEPS is aimed at simulation of neuron signaling pathway models. In contrast to the design pattern used in URDME and STEPS, MesoRD functions as a command line program that uses an input file in the Systems Biology Markup Language (SBML) [68] format to describe the model. SBML is a community effort with the aim to standardize descriptions of biochemical reaction network models. MesoRD extends the format with a custom Consecutive Solid Geometry (CSG) description of the domain geometry of the model. SBML has been widely adopted as a standard to exchange non-spatial models, but the limitations in its capability to describe spatial models has restricted its adoption for RDME simulations. The post-processing environment of URDME is closely integrated into Matlab. MesoRD provides a Matlab toolbox for analyzing the simulation data files. STEPS utilizes the Python pro-

gramming environment and packages such as NumPy, SciPy, and Matplotlib for post-processing and analysis.

Compared to static XML input files, the programmatic paradigm used by URDME and STEPS provides a more powerful but also more complex modeling environment. Constructing model files using a complete programming language reduces the restrictions imposed on the software by the model format. For example, the model of the neuron presented in the Results section could not have been described by an SBML document, nor the extended SBML format used by MesoRD. Since propensities in URDME are defined in a program file, any type of functional propensity can be used in URDME models, including Michaelis-Menten and Hill term style propensities, and even arbitrary logical expressions can be employed. This offers great flexibility in terms of the models that can be simulated, but also places more responsibility on the end-user. MesoRD uses MathML as part of the SBML definition, which allows the use of any mathematical expression in the propensities and facilitates handling of units and error checking. This is a powerful and robust, but also computationally expensive strategy. The STEPS reaction object supports only mass action kinetics, which results in an efficient but less flexible strategy.

In addition to having the most efficient and expandable design of the model propensity, URDME also provides the largest set of geometry and mesh model

features of the three software packages. URDME supports volume compartments with internal and external 2D surfaces embedded in the 3D geometry, as well as diffusion and reactions on surfaces and in the 3D volume. URDME also supports directed transport (convection) in 3D through an add-on module. STEPS 1.3 supports 3D compartments and volume diffusion. It is capable of localizing species to a curved surface embedded in 3D, but does not support surface diffusion. MesoRD 1.0 supports 3D compartments and volume diffusion only. To represent cellular membranes, MesoRD typically uses a small 3D volume on the exterior of the domain.

In summary, as a consequence of the design of the model environment, MesoRD is simpler to learn and use than both URDME and STEPS and also offers a better support for e.g. handling units, but the latter two offer a much more flexible and efficient modeling and simulation environment. In addition to the programmatic environment, both URDME and STEPS provide limited support for SBML. URDME has an experimental conversion utility that will create templates for the model and propensity file from an SBML description of the chemical reactions, see [26]. This utility will be fully included in the next version of URDME. STEPS provides a function to convert an SBML file into Python model objects. In addition to the SBML document defining the biochemical reaction network, both URDME

and STEPS require a mesh describing the model domain geometry be provided. Neither software is able to use SBML as a comprehensive model description.

Simulation performance To compare the performance of the software packages, we implemented the model of Min oscillations in *E. Coli* as described in [41] in each of the three software environments. Figure 4.7 shows simulation time as a function of the number of voxels in the mesh. The simulation was run for 900 seconds (simulation time), with the system state recorded every second. A detailed description of the model setup in the different packages and the scripts used for producing these benchmarks are provided in [26]. The URDME framework has a strong emphasis on efficient simulation algorithms, which is also visible in the figure. URDME clearly outperforms the other packages. We believe that this is in large part due to URDME's modular design and the fact that the solver source files and the propensity functions file are *compiled* into a dedicated executable for each separate model (see the Implementation section for details).

The numerical treatment of mesoscopic diffusion. URDME emphasizes the use of unstructured tetrahedral and triangular meshes to discretize the geometry. Unstructured meshes offer distinct advantages over Cartesian meshes for resolving complex geometries with non-trivial boundaries and they are more flexible than cut-cell approaches when it comes to describing processes occurring on a

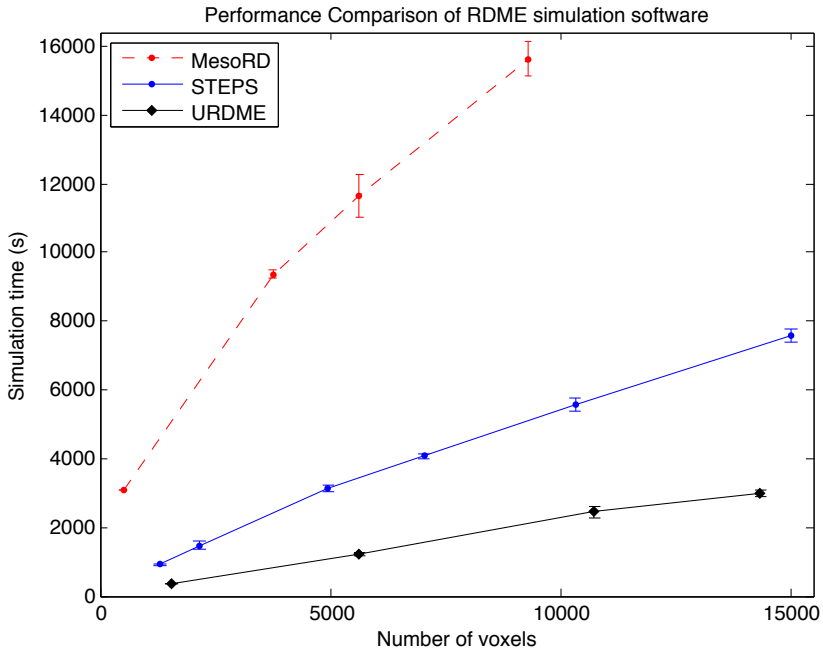

Figure 4.7: Performance comparison of the three software packages for an increasing number of voxels. Each point shows the mean and the error bars show the standard deviation of a ensemble of $N = 5$ runs. For URDME the number of voxels represents the number of mesh vertices, for MesoRD it represents the number of cubic subvolumes, and for STEPS it represents the number of tetrahedrons. All simulations were performed on a 1.8 GHz Intel core i7 processor with 4GB of memory.

curved boundary embedded in 3D space, such as the cell membrane of a spherical cell or the nuclear membrane [71]. The first version of URDME was developed as a product of theoretical work on how to obtain mesoscopic diffusion jump constants on triangular and tetrahedral meshes [36]. In short, the methodology used by URDME is based on the fact that a numerical discretization scheme for the standard diffusion equation will give jump coefficients that result in mesoscopic simulations that are consistent with both the behavior of mean values of a large ensemble of particles and the probability density function for a single particle diffusing according to Brownian motion. The latter is true since the Fokker-Planck equation for the one-particle probability density function is mathematically equivalent to the macroscopic diffusion equation. URDME currently uses a discretization with the Finite Element method to obtain the diffusion jump coefficients.

The quality of the tetrahedral mesh is an important aspect of a numerical discretization. An in-depth discussion of the requirements on the mesh for use in the mesoscopic model is given in [36]. Tetrahedra should not be too irregular, and between regions in the domain with much different resolution, the size of the elements should not grow too fast. This is also true for the solution of PDEs, and mesh generation software is aware of these issues and attempts to optimize the meshes accordingly. Surface meshes in 2D from state-of-the art mesh generation software such as Comsol tend to be of very high quality. In 3D, many meshes

will violate the assumptions in [36] to some degree. Generation of high quality unstructured meshes is an active area of research due to their importance in industrial applications. The modular design of URDME ensures that we can accommodate new results in this area without major restructuring of the code.

The influence of mesh quality on RDME simulations with unstructured meshes in 3D was studied for several different discretization schemes in [76] using particularly revealing and highly sensitive model problems. They show that unless the meshes are of high quality, discretization errors may lead to small but persisting errors for both the Finite Element and the Finite Volume methods, i.e. the convergence properties of the schemes are affected negatively. In some of these cases, simulations using a structured Cartesian mesh will have better numerical properties if the geometry permits resolution of the domain with a feasible number of subvolumes. On the other hand, it is not difficult to think of cases for which this is very difficult and for which sensitive processes occur on the parts of the domain which are hard to resolve.

Using MesoRD, surfaces in a 3D model are modeled as volume geometry objects by ensuring that the thickness of the membrane is small compared to its size, approaching a true 2D model as the thickness of the membrane becomes small. Unless one desires to resolve some dynamics on such high level of detail as to consider vertical movement of molecules in the membrane, this will be unneces-

sarily expensive since the mesh elements has to be sufficiently small to resolve the narrow 3D volume. The mesh generation in MesoRD needs several grid points in the extent of a membrane to give a fully connected diffusion volume [34, Fig. 11]. With a uniform grid, this will lead to expensive simulations since the size of the voxels necessary to accurately resolve the membrane must be used everywhere in the domain. In order to demonstrate this, we conducted a simple diffusion-only numerical experiment, described in detail in [26]. We let molecules diffuse freely on the surface of a unit sphere, and be absorbed by a small circular patch at one of the poles. Simulations using URDME are in excellent agreement with the exact solution, even for fairly coarse meshes. For example, using an ensemble size of 10^5 molecules to compute the mean absorption time, the error was $\approx 0.2\%$ for a mesh with 4343 voxels. The computing time to generate the solution was 21 seconds. By contrast, for a membrane thickness of 100 nm and a voxel size of 20 nm, MesoRD 1.0 produces a solution with about 14% error using 157128 voxels and a simulation time of 1 hour and 50 minutes on the same 2.66 Ghz Intel Core i7 with 8GB of RAM.

For complex models with both volume diffusion, surface diffusion, and reactions, it is difficult to predict what impact different sources of error in the diffusion will have on the output metric of interest. For example, for the Min system used to benchmark the different software packages in Figure 4.7, URDME, STEPS,

and MesoRD give quite similar period times of oscillations. In addition to errors caused by the discretization, errors intrinsic to the RDME mathematical model arise for highly diffusion limited reactions when the voxels become very small [70]. To some extent, this can be alleviated using modified, mesh dependent bimolecular reaction rates [37, 40], but there is a critical size of the voxels under which no correction to the traditional RDME can make it consistent with more fine scaled particle based methods [61]. Since unstructured meshes can more accurately resolve complex geometries, their spatial accuracy is often higher for equivalently sized voxels when compared to Cartesian meshes. This can help in avoiding geometrical features of the model to force us to approach the critical regime for the voxel sizes. The combined effects of diffusion discretization error and error caused by small subvolumes were investigated for several additional models in [76]. For the examples studied there, it was concluded that the error introduced by small subvolumes in 3D could be a bigger source of error than any numerical discretization errors of the diffusion operator.

Conclusions

As demonstrated by the examples in this chapter, the URDME infrastructure offers great flexibility at the stage of model construction and execution. Using

a simple script in Matlab, URDME was used to set up and conduct a series of experiments in which the geometry of an *E. Coli* bacterium was automatically varied. In another example, the basic reaction-diffusion modeling framework was extended to include active transport in a highly complex geometry obtained from external CAD and meshing software.

The URDME software framework offers unique features for both model and methods developers in computational systems biology. The support of unstructured meshes provides the capability to create models with a complex geometry that closely match the physical descriptions of the systems under study. URDME integrates easily with widely used scientific computing software to provide a versatile platform for mathematical and computational modeling, allowing for the implementation of complex and customized models and pre- and post-processing routines. The modular design ensures extensibility and interchangeability of the third-party tools used for model specification and mesh generation, as well as of the core simulation algorithms.

Chapter 5

Adaptive Accelerated Spatial Stochastic Simulation on NVIDIA graphics processing units

One of the biggest challenges faced by scientists utilizing the DFSP algorithm is correct selection of the appropriate operator splitting timestep. A correctly chosen timestep will produce results at the optimum speed without violating the specified error tolerance. Choosing an appropriate timestep requires estimation of the error due to operator splitting. The error is a function of the spatial

discretization of the domain, the diffusion coefficients of the molecules within the system, and the stiffness of the chemical reaction channels.

In this chapter we present an extension to the DFSP algorithm that automatically and adaptively selects the appropriate timestep for performance and error control. We demonstrate the utility of this method with a traditional CPU implementation. In addition we demonstrate the parallel efficiency inherent in the adaptive DFSP algorithm (ADFSP) with an implementation on the NVIDIA Graphics Processing Unit (GPU). This work was done in collaboration with Andreas Hellander and Michael Lawson.

5.1 Overview of the DFSP Algorithm

The key observation underlying the DFSP method is that the diffusion of each molecule is independent of all other molecules in the system. Starting with an initial population of a given species in a voxel, the DFSP method [28] uses the Finite State Projection method [99] to compute the probability of transition of that species into neighboring voxels. DFSP approximates the DME (2.6) for one species s and one voxel i at a time:

$$\frac{\mathrm{d}}{\mathrm{dt}} p_{i,s}(\mathbf{x}, t) = D_s p_{i,s}(\mathbf{x}, t), \tag{5.1}$$

where the diffusion matrix D_s has the same structure for all species in the domain, but is scaled by the appropriate diffusion coefficient. The solution to this linear differential system is $P_{i,s} = e^{\Delta t D_s} p_{i,s}(x, 0)$. The final spatial distribution of molecules after a timestep is then found by sampling new molecular positions using the solution vectors $P_{i,s}$ and executing the resulting transitions.

Since the diffusion of each molecule is independent, DFSP partitions the system into sub-problems of diffusing individual molecules. This requires no further approximation, but makes the resulting matrix exponential far more tractable and allows for a higher level of parallelism. Therefore, in this work we will diffuse molecules one at a time. Below we discuss the details of how we solve for the single molecule PDFs, but first we outline approximations to these subproblems and how we use them to compose a solution to the full diffusion problem.

To solve the DME directly for a sub-problem, the DFSP method retains a finite set of states that carry a high probability, and truncates states of lower probability. In the case of the DME for one molecule, states are defined by the location of that molecule. The states closest to the originating voxel will have the highest probability associated with them; those further away will have lower probability. Thus, there is a clear systematic way to search for the states with the highest probability, and the states of lower probability are lumped into one absorbing state, ϵ, that provides an error bound. In our previous work we defined

a parameter MAX as the farthest distance molecules were allowed to diffuse in one step (and thus the farthest reachable state), and the remaining transitions were captured in $\tilde{D}_s$. The approximation is given by $\tilde{P}_{i,s} = e^{\Delta t \tilde{D}_s} p_{i,s}(x,0) \approx e^{\Delta t D_s} p_{i,s}(x,0)$ and the resulting error bound is $\|\tilde{P}_{i,s} - P_{i,s}\|_1 \leq 2\epsilon$. We note that ϵ is directly calculated and can be made arbitrarily small by adding states.

In this work we use the uniformization method [73] to compute the full matrix exponential. We start with the portion of the state space with the highest probability, and end when the probability of the truncated states drops below ϵ. This results in the same 2ϵ bound as in our previous work, only now the bound is tight: $\|\tilde{P}_{i,s} - P_{i,s}\|_1 = 2\epsilon$. Similar to MAX, we have a value N_{max} that denotes the maximum number of voxels kept. In practice we fix the time step and error criterion and then vary N_{max}, but it is an important parameter to introduce because it will be used in some of the analysis that follows. This method of solving the single molecule problem requires a search on a sorted list, but it does not require a matrix exponentiation at each step of a guess and check.

To calculate the final state of the system due to diffusion over an interval of length Δt, we sample the PDF by selecting uniformly distributed random numbers $R \in U(0,1)$ and finding the smallest integer μ such that $\sum_{j=1}^{\mu} \tilde{P}_{i,s}[j] > R$, where $\tilde{P}_{i,s}[j]$ is the probability weight of state j. $\tilde{P}_{i,s}$ is normalized so that conservation

of probability is achieved:

$$\tilde{P}_{i,s}[j] = \frac{\tilde{P}_{i,s}[j]}{\|\tilde{P}_{i,s}\|_1}.$$

Repeating this process $\sum_{i,s} X_{i,s}(t)$ times identifies the final location of all the molecules in the system at time $t + \Delta t$.

For molecules of the same type originating from the same voxel, we can re-use the PDF. In addition, we can re-use the PDF for future time-steps of length Δt. As a result, for a constant time step, simulating a diffusion process becomes a matter of selecting $\sum_{i,s} X_{i,s}(t)$ random numbers and performing a lookup and comparison. Since we are able to diffuse each molecule independently, this algorithm is highly parallelizable within a time-step.

5.2 Computation of DFSP lookup tables via Uniformization

The computationally expensive part of the DFSP algorithm is the generation of the lookup tables necessary to redistribute the molecules to neighboring voxels on each diffusion step. Since each molecule is independent, this can be thought of as generating the one-particle PDF for each voxel in the system. In other words, given that a molecule starts in a particular voxel, for every voxel in the system

what is the probability that it will be located in that voxel at the end of the time step? For small problem sizes, explicitly forming local, truncated matrices and solving for the PDFs using matrix exponentials is an efficient approach that allows for reasonably large time steps (the procedure used in our previous work [28]). However, for larger problem sizes in 2D and 3D and for unstructured meshes, this strategy becomes prohibitively expensive and can negate any performance benefits of DFSP unless the ensemble size is very large and the lookup-tables can be reused efficiently between independent realizations.

We compute the one-particle PDFs by uniformization of the Markov process [73]. Uniformization can be used to analyze a continuous-time Markov process by converting it to a discrete-time chain subordinate to a Poisson process. Define $\lambda_{max} = \sup_{\mathbf{x}\in\mathcal{Z}^+} \mathcal{D}(\mathbf{x})$ as a bound on the maximum intensity of the generator of the Markov process. In the general case λ_{max} is unbounded, but for the case of a single particle jumping on the mesh, the state space is finite and $\lambda_{max} = \max_i |D_{ii}|$, i=0,...,Ndofs. The discrete time chain defined by the transition matrix $S = I - D/\lambda_{max}$ subordinate to the Poisson process $Po(\lambda_{max})$ is the equivalent to the original continuous chain [73], where I denotes the identity matrix. The computation of the time dependent PDF for a particle starting in $\mathcal{V}_i$ amounts to computing the average

$$P_i(\Delta t) = \sum_{k=0}^{\infty} p_k S^k \mathbf{e}_i, \tag{5.2}$$

where $\mathbf{e}_i$ is a vector with all elements zeros except for $e_i = 1$, and

$$p_k = \frac{(\lambda_{max}\Delta t)^k}{k!} \exp\left(-\lambda_{max}\Delta t\right)$$

is the Poisson probability that the final time Δt was reached in k steps of the discrete chain. In practice, the sum (5.2) is truncated when the tail of the Poisson distribution is small enough according to some tolerance ϵ, $\sum_{k=0}^{N_u} p_k \geq 1 - \epsilon$. In the context of DFSP, ϵ corresponds to the probability for the particle to be in the absorbing state and N_{MAX} corresponds roughly to the MAX parameter [28]. This follows trivially from the fact that since S describes a Markov chain, $S^k\mathbf{e}_i \geq 0$, $\|S^k\mathbf{e}_i\|_1 = 1\ \forall k$ and hence

$$\| \sum_{k=N_u+1}^{\infty} p_k S^k \mathbf{e}_i \|_1 \leq \sum_{k=N_u+1}^{\infty} p_k \|S^k\mathbf{e}_i\|_1 \leq \epsilon. \tag{5.3}$$

In practice, when the tolerance ϵ is met, the resulting one-particle PDF is renormalized as described in the previous section in order for the algorithm to conserve the total copy number of molecules.

With the procedure (5.3) to compute the one-particle PDFs, the following bound on the error in PDF holds by construction, where A is the truncated

solution and D is the full matrix for the DME:

$$\|e^{\Delta t D} - A\|_1 \leq \epsilon. \tag{5.4}$$

For the expected value, we then have the trivial bound

$$\|\Delta E[X^{n+1}]\|_1 = \|(e^{\Delta t D} - e^{\tau D_{fsp}})\mathbf{x}^n\|_1 = \|(e^{\Delta t D} - A)\mathbf{x}^n\|_1 \leq \epsilon\|\mathbf{x}^n\|_1. \tag{5.5}$$

Hence, the error tolerance parameter ϵ bounds the conditional relative error in mean in one timestep of DFSP,

$$\frac{\|\Delta E[X^{n+1}]\|_1}{\|\mathbf{x}^n\|_1} \leq \epsilon. \tag{5.6}$$

In many cases, this can be expected to be an overly conservative bound. In our case however, given ϵ, Δt will be chosen to fulfill the bound (5.6) by construction provided that the one-particle PDFs are computed to ϵ tolerance using uniformization.

The cost of uniformization increases rapidly with the stiffness of the diffusion operator. Intuitively, DFSP moves the problem of stiffness in the stochastic simulation to a problem of solving the diffusion equation for every voxel and chemical

species in the mesh. Conceptually, uniformization is related to the explicit Euler method for solving the diffusion PDE. In the PDE literature, it is well known that implicit methods or exponential integrators are advantageous in the stiff regime because they allow for larger time steps. Taking large time steps (relative to the CFL condition), however, quickly makes the computation of the lookup tables too expensive to be worthwhile over a pure NSM simulation. Hence, while DFSP improves on the stiffness issue of the RDME, it does not offer a clear cut solution.

There are a number of different ways to solve for the one particle PDFs, including Krylov subspace methods, Padé methods, and using the solution of a system of ordinary differential equations. While some of these techniques are used more commonly in general to compute the matrix exponential [96] than the uniformization method, we found that they were more complex to implement, and had greater computation times, while providing no additional accuracy benefit. Uniformization also provided the additional benefit that it is very amenable to parallelization and has a low memory requirement, which is beneficial for GPU implementations. We will comment further on the cost and complexity of uniformization in Section 5.4.

5.3 Estimation of the Operator-Splitting Error

Here we will summarize the results derived in [27]. DFSP uses a first order operator splitting method, or Lie-Trotter splitting, to decouple the reaction operator $\mathcal{M}$ and the diffusion operator $\mathcal{D}$ in the RDME:

$$p(\mathbf{x}, t) = e^{t(\mathcal{M}+\mathcal{D})} p(\mathbf{x}, 0). \tag{5.7}$$

The first order operator splitting approximates (5.7) by

$$p_s(\mathbf{x}, t + \Delta t) = e^{\Delta t \mathcal{M}} e^{\Delta t \mathcal{D}} p_s(\mathbf{x}, t), \tag{5.8}$$

which is accurate to $\mathcal{O}(\Delta t^2)$. Simulation of (5.8) in done by applying each operator to the state sequentially:

$$\begin{aligned} &1.\ \ p_s^{1/2} = e^{\Delta t \mathcal{M}} p(\mathbf{x}, t) \\ &2.\ \ p_s(\mathbf{x}, t + \Delta t) = e^{\Delta t \mathcal{D}} p_s^{1/2}. \end{aligned} \tag{5.9}$$

With the commutator error analysis of [72], we can find a bound on the local error in the approximation using

$$\| p(\mathbf{x}, \Delta t) - p_s(\mathbf{x}, \Delta t) \| \leq C \Delta t, \tag{5.10}$$

where C is an arbitrary constant. Thus the local error is bounded by the difference between applying the reaction operator first versus applying the diffusion operator first.

The error in a single step of the ADFSP algorithm is given by:

$$\mathcal{E} = (e^{\Delta t(\mathcal{M}+\mathcal{D})} - e^{\Delta t\mathcal{M}} e^{\Delta t\mathcal{D}})p(x,t) = \frac{\Delta t^2}{2}[\mathcal{D},\mathcal{M}]p(x,t) + \mathcal{O}(\Delta t^3), \tag{5.11}$$

where the commutator $[A, B]$ is defined as $[A, B] = AB - BA$.

We can approximate (5.11) by finding the expected change of state $\Delta E[X]$ for a given state $\mathbf{x} = p(x, t)$:

$$\Delta E[X] = 0.5\Delta t^2 \left(\mathcal{D}(\mathcal{M}(\mathbf{x})) - \mathcal{M}(\mathcal{D}\mathbf{x})\right). \tag{5.12}$$

Then we can select the next time step based on the current estimate of local error (for voxel i and species s) using

$$\tau_{suggested} = \min_s \left[\sqrt{\frac{2(ErrorTolerance)}{\sum_{i=1}^{N} |\mathcal{V}_i||\Delta E[X]_{is}|}} \right], \tag{5.13}$$

where $|\mathcal{V}_i|$ is the volume of voxel i.

5.4 Adaptive DFSP algorithm

In this section we discuss the implementation of the adaptive DFSP algorithm on both the CPU and GPU compute environments. First, we describe the implementation of the whole algorithm on a single core of the host, then for each component of the algorithm, we provide a performance study of the parallel GPU implementation in order to motivate the decision of whether to execute a component on the GPU or CPU in the multicore implementation.

The algorithm has six components. *The SSA reaction operator* advances the state of the system by executing reactions using the SSA direct method within each voxel over a time step τ_D. *The DFSP diffusion operator* advances the state of the system by executing diffusion between voxels over the same time step. *Lookup-table generation*; the DFSP diffusion operator uses a lookup table that is generated by the uniformization method discussed previously. For each value of τ_D, a distinct table is generated. A cache of lookup tables is stored to minimize the number of tables generated. *Local error estimation.* After we have advanced the state of the system, we estimate the error due to operator splitting by the formula (5.12). *Compute proposed time step.* Using the estimated error, we can compute the largest time step we can advance the system by, while maintaining the error tolerance specified by the user using formula (5.13). *Accept or reject the*

proposed time step. Due to the fast fluctuations in system states during stochastic simulation, the exact value of the estimated time step fluctuates as well. Since a lookup table is generated for each new value of time step, we choose the next time step so that it differs from the previous time step by a factor of 2^j, where $j \in \mathbb{Z}$. Thus, if the proposed time step is larger than the current time step, then the accepted time step is the current time step multiplied by the largest factor of two for which the product is smaller than the estimated time step. Conversely, if the proposed time step is smaller than the current time step, then the accepted time step is the current time step multiplied by largest factor of two for which the product is smaller than the proposed time step.

Algorithm 4 describes the algorithm in pseudo-code, and Figure 5.1 shows a diagram of the process flow of the overall algorithm. The first step of the algorithm is always a small step with NSM. The reason for this is that often the initial condition of a system to be simulated is uniformly distributed in space, or is an empty domain. In these cases, the error estimation and time step selection formulas do not produce good initial guesses for the time step.

SSA Reaction Operator ADFSP uses a simple, straightforward implementation of the direct SSA method. The chemical reactions in each voxel are simulated independently and sequentially. First, the propensity for each reaction channel in

INPUT: Input State
OUTPUT: Output States

Take step with NSM
while $t \leq t_{end}$ **do**
  estimate splitting error in each voxel
  calculate min $[\tau_{suggested}]$ over all voxels
  accept new τ_D based on $\tau_{suggested}$ and current τ_D
  if diffusion lookup table does not exist for τ_D **then**
    calculate diffusion lookup table for τ_D
  end if
  take DFSP diffusion step of length τ_D
  take DFSP reaction step of length τ_D
  add τ_D to t
end while

Algorithm 4: ADFSP algorithm

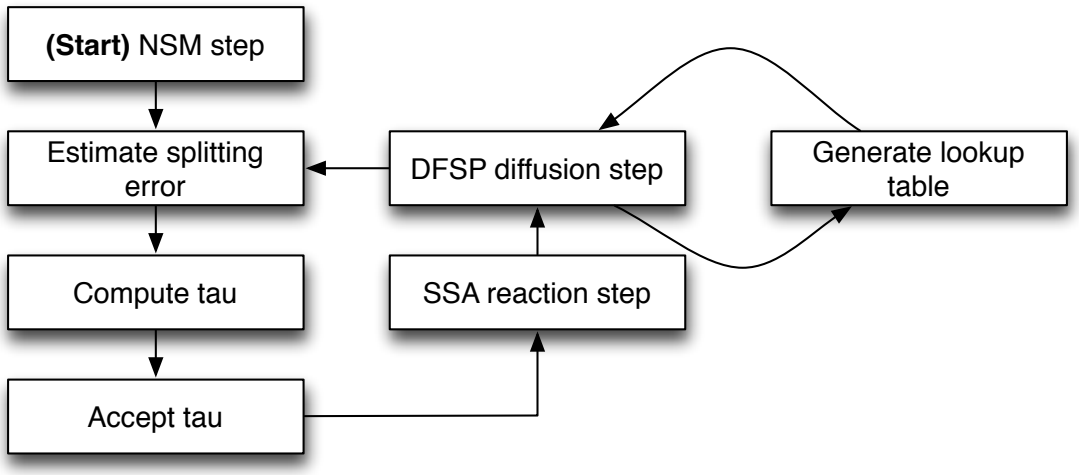

Figure 5.1: Process flow of the Adaptive DFSP algorithm. The first step is taken with NSM to move the system out of the initial state. Next the local splitting error is calculated in each voxel (as described above), and the maximum value of τ_D is calculated so that it is accurate for the given error tolerance. Then SSA reaction steps are executed until time $t + \tau_D$ and DFSP diffusion steps are executed until time $t + \tau_D$, and a lookup table is generated if necessary. Next the local splitting error and value of τ_D for the next step are calculated. The loop continues until the simulation end time.

each voxel is computed. The time to the next reaction in the voxel is calculated. If that time is larger than $t + \tau_D$, the reaction step for that voxel is complete. Otherwise the reaction is selected, the local state for the voxel is updated by the state change vector, and the relevant reaction propensities are updated. Finally, the time to the next event is calculated, and this procedure continues until the time step is complete.

DFSP Diffusion Operator ADFSP uses a simpler method than in our previous work to implement the DFSP diffusion operator. In this implementation, we move each molecule independently as opposed to the finding final spatial configurations of multiple molecules, as the earlier procedure did not scale well due to the complex connectivity of unstructured meshes. The procedure to perform a diffusion step of τ_D requires input of the current state of the system and the DFSP lookup table. The lookup table may provide redistribution PDFs for a timestep τ_{table} that is some factor of two less than the input τ_D, due to convergence requirements of the uniformization method. For each molecule in each voxel, a destination voxel is selected randomly from the DFSP lookup table. Once all molecules in the domain have been moved, the procedure is repeated $\frac{\tau_D}{\tau_{table}}$ times.

DFSP Lookup-table generation ADFSP uses uniformization to calculate the one-particle PDF lookup tables used by the DFSP diffusion operator. We will

summarize the computational procedure, however further explanation is found in Section 5.2. First we compute λ_{max}, the largest absolute value in the diagonal of the D matrix. This value is used to determine the maximum diffusion timestep, τ_{table}, where the uniformization iteration will converge within the maximum allowed iterations (we use 50 iterations for this implementation). Using λ_{max} we determine that the iteration will converge if the following inequality holds:

$$\frac{ErrorTolerance}{2} > 1 - \sum_{i=0}^{50} \mathcal{P}_i \left(\lambda_{max}\tau_{table}\right),$$

where $\mathcal{P}_i(x)$ is the ith Poisson PDF value for a distribution with mean x. Initially we set $\tau_{table} = \tau_D$. If the inequality does not hold, we reduce τ_{table} by a factor of two, and repeat the procedure until it does. We also record the minimum number of iterations necessary for convergence. Next, we perform the uniformization iteration for each species in each voxel separately. The iteration starts with a vector that is all zero except the position corresponding to the species in the voxel that this iteration is calculating, which has the value 1. This corresponds to the initial position probability of the diffusing molecule, which is 1 in the initial location and 0 elsewhere. Then, we perform a sparse matrix-vector multiplication with the D matrix (normalized by λ_{max}) and multiply the result by the Poisson PDF value corresponding to the iteration number $\mathcal{P}_i \left(\lambda_{max}\tau_{table}\right)$. The resulting vector is used

as input for the next matrix-vector multiplication, and the iteration repeats for the previously determined number of iterations. When this procedure is complete, the resulting vector is used as probabilities and is sorted and transformed into a cumulative distribution function for efficient selection of the destination state by the diffusion operator, and stored in the lookup table.

Local Error Estimation This procedure is explained more fully in Section 5.3, specifically equation (5.12). First, the reaction propensities for the current state are evaluated for each reaction channel in each voxel. That value is then multiplied by the state change vector for each species effected by that reaction. Then, the sum of the products from the previous step are stored for each species in each voxel. This is known as the *expected change of state due to reactions*. Next, for each species in each voxel we multiply the population times the probability of jumping to an adjacent voxel (multiply the state vector times the D matrix). For each destination voxel, we sum the products of the previous step. This is the *expected change of state due to diffusion*. Our next step is to combine the previous two steps, thus we take the expected state change due to diffusion and for each species in each voxel multiply that value times the probability of jumping to an adjacent voxel. Again, for each destination voxel we sum the products of that step as the *expected change of state due to reaction-then-diffusion*. Next, we

combine the first two steps in reverse order by first calculating the propensity function of each reaction channel using as the input the state of the expected change of state due to diffusion. Then for each species in each voxel we sum the products of all the reaction propensities from the previous step that effect that species, times the state change vector of that reaction channel to find the *expected change of state due to diffusion-then-reaction*. Finally, we calculate the expected change of state by finding the difference for each species in each voxel between the expected change of state due to diffusion-then-reaction from expected change of state due to a reaction-then-diffusion. The ADFSP method calculates the operator splitting error after the initial NSM step is complete, and again every time 10 reaction-diffusion steps are taken.

5.4.1 Implementation in URDME

We used URDME [26] as the development platform for the implementation of both the CPU and GPU version of the adaptive DFSP algorithm. URDME is designed with a dual purpose: to enable efficient development and simulation of molecular systems biology models, and to enable efficient development and deployment of RDME simulation solvers. URDME provides a unified modeling, pre-processing and geometry handling infrastructure. This allowed us to focus our efforts on efficient implementation of the algorithm without the burden of creating

a reaction-diffusion simulation framework. We use the highly optimized NSM core solver in URDME to compare the efficiency of our algorithm. NSM in URDME has been shown to outperform other popular reaction-diffusion simulators [26] and is to the best of our knowledge the fastest current general purpose implementation of exact RDME simulations.

The ADFSP solver is compatible with the newly released 1.2 version of URDME. Specifically, URDME 1.2 adds support for the model propensity to be defined via "inline" propensities (reactants and reaction rate specified in an array in the matlab model file, usable for mass-action only). The ADFSP_GPU solver provides a plug-in to the URDME framework (specifically "urdme_inline_convert_adfsp_gpu.m") that automatically converts the inline propensity to a set of device and host CUDA functions. This allows end users to easily specify their models without writing complex and fragile CUDA code.

5.4.2 Parallel GPU implementation

The parallelization strategy used throughout the ADFSP_GPU algorithm is to assign one GPU thread to each voxel in the system. The strategy works well for the reaction, diffusion, uniformization and error estimation components of the algorithm. The exception is the calculate time step component, as it performs an L1 norm on the estimated error vector. The GPU architecture subdivides

the total number of threads into blocks of threads that run simultaneously. The blocks are scheduled according to available resources. For this implementation we set the number of threads per block using the CUDA occupancy calculator and by performance testing. Figure 5.2 shows the process flow for the ADFSP algorithm with GPU implementation.

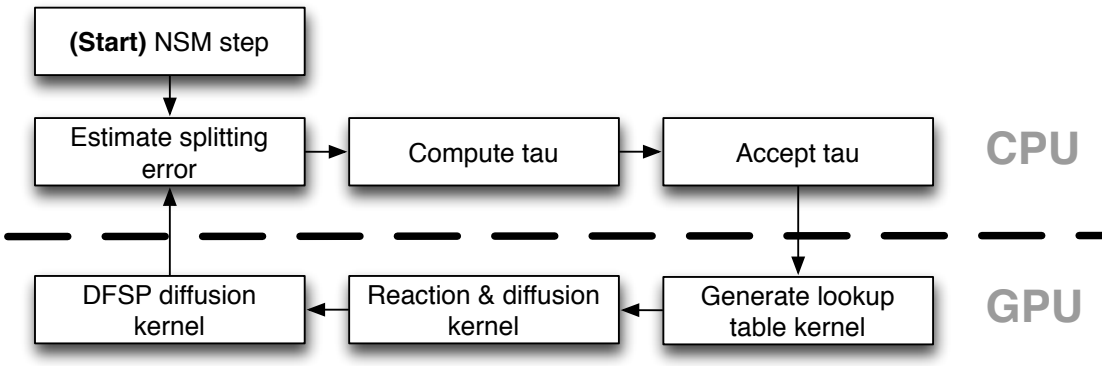

Figure 5.2: Process flow for the GPU solver. Boxes above the dashed line correspond to the components that are implemented in the CPU, and below the line are the components implemented in the GPU. Compare to Figure 5.1.

SSA Reaction Operator Kernel GPU devices have two major types of memory. The first is global device memory which is accessible to programs running on both CPU and GPU. The second type is on-chip local shared memory, which is only accessible from GPU kernels. Each block of threads has access to the same bank of shared memory, and shared memory is not persistent between block executions. It is often thought of as an explicitly managed cache.

The SSA reaction kernel makes use of local shared memory, as access to local shared memory is much faster than to global memory. Each GPU thread first reads the state of the voxel assigned to the thread into local shared memory. Then, the SSA algorithm is executed. First, the reaction propensity for each reaction is calculated. Then, the time to the next reaction is found. If that time exceeds $t + \tau_D$ the state of the system in shared memory is written to global memory and the kernel is complete. Otherwise the reaction is selected, the local state is updated by the state change vector, the relevant reaction propensities are updated, and the time to the next event is calculated.

DFSP Diffusion Operator Kernel The DFSP diffusion operator implemented on the GPU first reads the state of the voxel assigned to the thread into local shared memory. Then for each molecule in the voxel, the outbound destination is randomly selected using the lookup table, and the destination is stored in a flux vector in local shared memory. When all diffusion events have been calculated, the total flux to each outbound voxel is written to global memory using the *atomicAdd()* function to ensure that no race conditions are present in this communication step.

Figure 5.3 shows the performance of the SSA reaction kernel and the DFSP diffusion kernel as a function of time stepsize for the G-protein 1D model (see

Section 5.5.1 for complete description). The number of reaction events and diffusion events increases linearly with the time step size. The execution time of the SSA reaction kernel also depends linearly on the timestep. However, the DFSP diffusion kernel performance is nearly constant with respect to stepsize, increasing only 13% over two and a half orders of magnitude of stepsize. On the other hand, at least for this problem, the DFSP kernel is dominating the execution time.

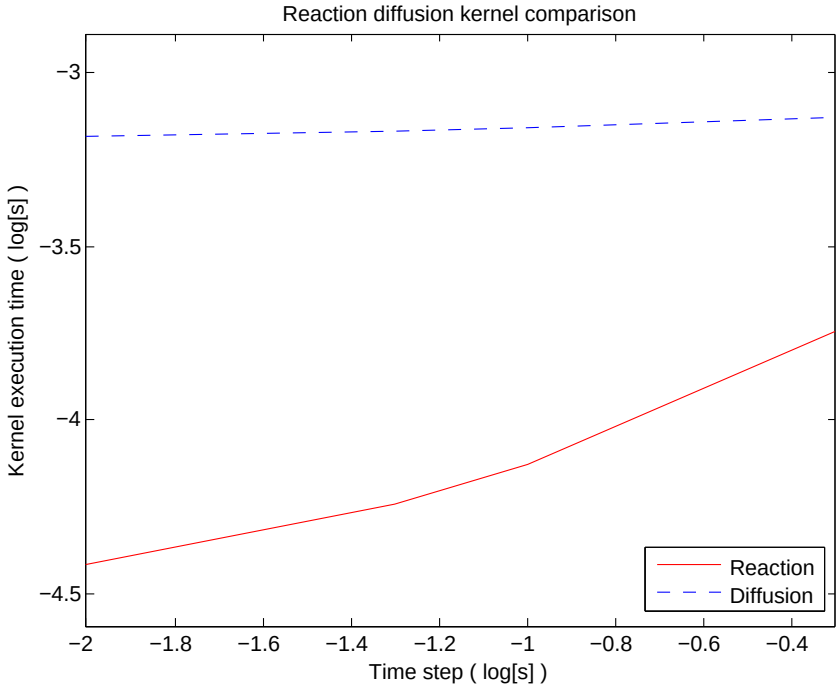

Figure 5.3: Performance analysis of the reaction and diffusion kernels. The number of events is linear in timestep for both the reaction and diffusion operators. The execution time of the SSA reaction kernel (solid red line) depends linearly on the time step, while the DFSP diffusion kernel is nearly constant. On the other hand, at least for this problem, the DFSP kernel is dominating the execution time.

Combined Reaction-Diffusion Operator Kernel To optimize performance, we created a GPU kernel that combines the reaction and diffusion steps. In this kernel, the state is first read from global memory to local shared memory. Then all reactions in the voxel are performed for the timestep, then the outbound diffusion destination for each molecule in the voxel is calculated. Then those states are written back to global memory. This combination ensures that a minimum number of global memory read/write operations are performed. Additional diffusion steps may also be performed if the lookup table timestep τ_{table} (determined by the uniformization) is less than τ_D.

Uniformization Lookup Table Generation Kernel When the ADFSP algorithm takes a diffusion step, it searches its cache for a lookup table that matches the currently selected time step. If no appropriate table is found, it uses the uniformization method to generate the lookup table for that time step. This step potentially is one of the most expensive components of the computation. This motivates the accept-time-step component which ensures that new time steps are a power of two different from previous time steps, reducing the total number of possible lookup tables to a smaller discrete number and ensuring that new tables are calculated only if there is a significant effect on performance or accuracy. Implementation of the uniformization method on the GPU imposed significant

challenges. The method needs several arrays for the calculation of the matrix-vector products. When the method is parallelized, each thread needs its own private copy of these arrays. The simplest method is to statically allocate the arrays. For small and medium sized problem this strategy provided good performance and reasonable memory usage. However, when used with a large problem we found that the total device memory usage was so large that the remaining free space was not enough to allocate the lookup table. We used two strategies to mitigate this problem. The first was to move existing lookup tables to CPU memory to free additional space on the GPU. Whenever memory needed to be allocated on the GPU, the free space was first checked. If the space was insufficient, all lookup tables were flushed to the CPU and freed on the GPU. The lookup tables were then moved back to GPU memory as needed. The second strategy was to use dynamically allocated memory on the GPU for the working array. The advantage of this method is that the memory usage is much more efficient and the working arrays are freed after the method is complete, allowing for more space for lookup tables to be resident in GPU memory. The disadvantage is that uniformization with dynamically allocated memory is significantly slower than the same method with statically allocated memory. Figure 5.4 shows memory and performance comparisons for static and dynamically allocated arrays in the GPU

implementation of the Uniformization DFSP lookup table generation method for the MinCDE model (see Section 5.5.2 for complete description).

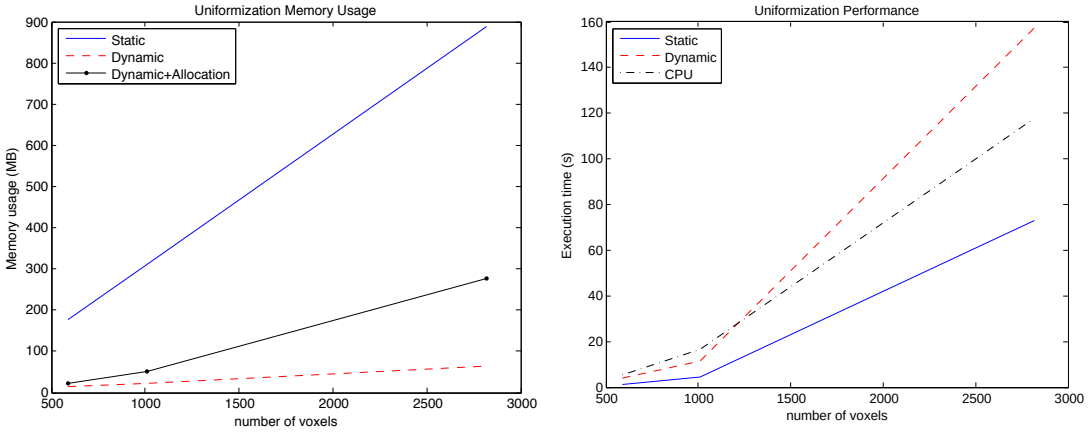

Figure 5.4: (left) Uniformization memory usage, (right) Uniformization speed comparison. GPU implementation of the uniformization matrix exponential algorithm for generation of the DFSP lookup tables. This shows the limitations of the GPU architecture, as the GPU card that we used (GeForce GTX 560) has 1GB of memory. Thus, ADFSP is unable to use the static uniformization method (blue line) as it uses more memory than is available. Dynamic memory allocation uses far less total device memory (red). The total memory used is even less with the additionally allocated working memory (black). However, use of dynamic memory is significantly slower than static memory, and even slower than the serial CPU method for large problem sizes (right).

Error Estimation Kernel Figure 5.5 shows performance comparisons for the MinCDE model between GPU and the CPU implementations for estimating the operator splitting error. Note that the CPU implementation is faster than both implementations on the GPU. The CPU implementation has multiple cross voxel data dependencies, and was not suitable fort the one thread per voxel parallelization. The implementation was reorganized such that each voxel error calculation

is independent, however this required a net increase in the number of propensity function evaluations and a linear search step of complexity $\mathcal{O}[\ (\#\text{ of connections})^2]$. This method (Figure 5.5 dashed red line) was found to be significantly slower than the CPU method (Figure 5.5 solid blue).

We implemented an optimized version on the GPU which increased the performance at the expense of additional memory to store the transposed diffusion matrix (D^T). The matrices used by the ADFSP algorithm are very sparse, thus we use the compressed column storage (CCS) method. By creating and storing a transposed diffusion matrix (D^T) on the GPU, we were able to remove the linear search step of the original implementation, and drive the time to access that data to $\mathcal{O}[1]$. This significantly increased the speed of the algorithm.

However, the performance of this method (Figure 5.5 black line with dots) was found to be closer to, but still slower than, the CPU method. As a result, the CPU error estimation method is used in the final implementation. We note, however, that the transposed D matrix GPU method has better scaling properties than the CPU method, and could possibly outperform it on larger problem sizes than the memory limitations allowed us to explore.

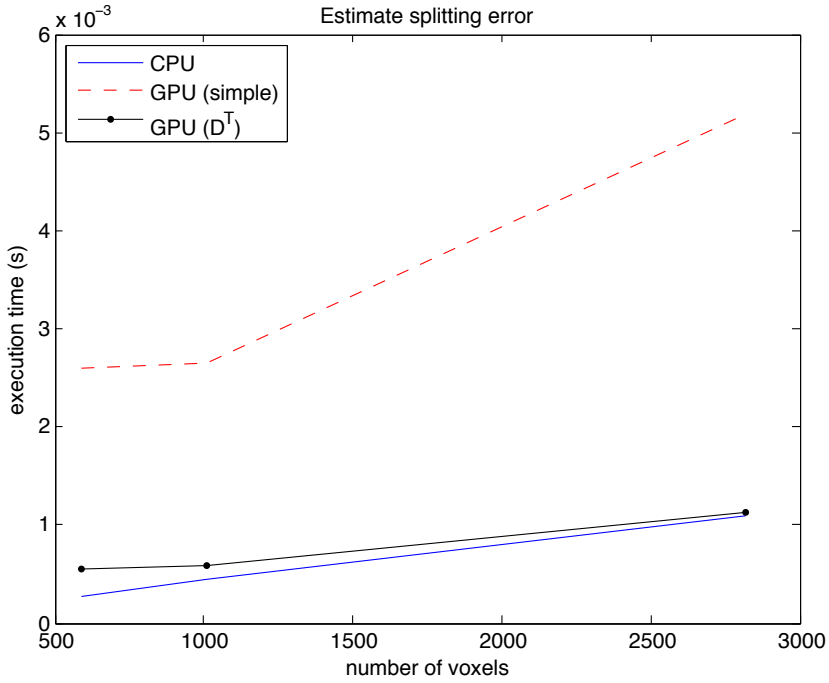

Figure 5.5: Comparison of GPU and CPU implementations of the operator splitting error estimate. The CPU implementation is more efficient than both GPU implementations, and thus is used in the final ADFSP method.

5.5 Numerical experiments

In this section we demonstrate the accuracy and efficiency of the proposed method by examining the performance, error and time step selection of the adaptive algorithm in a 1D and a 3D simulations, for varying error tolerances.

5.5.1 Polarization of *S. cerevisiae*

The first example is the pheromone induced G-protein cycle in Saccharomyces cerevisiae. We have converted the PDE model from [18] into a stochastic model and reduced it to ligand, receptor, and G-protein species (see Chapter 3 for details). The ligand level is constant in time but it varies spatially (a cosine function), with parameters determined experimentally. The ligand binds stochastically with an initially isotropic field of receptor proteins. The bound receptor activates the G-protein, causing the G-alpha (Ga) and G-beta-gamma (Gbg) subunits to separate. Ga acts as an autophosphotase and upon dephosphorylation, rebinds with Gbg to complete the cycle. The model is set in a 1D periodic domain. The simulation time is set to 1000 seconds, as steady state is achieved by that time. Gbg is the component furthest downstream from the ligand input and acts as a signal to the downstream Cdc42 cycle, and will therefore be the output for this model.

Table 5.1 provides performance comparisons for the G-protein model on the 1D domain. ADFSP is a serial implementation of our adaptive algorithm on a 1.8 Ghz Intel Core i7 CPU. ADFSP_GPU is a parallel implementation of our adaptive algorithm on a GeForce GTX 560 GPU. It is clear that the parallel implementation of ADFSP is 10 to 100 times faster than the serial implementation, which shows excellent parallel efficiency. Additionally, we found that the parallel GPU implementation of ADFSP (ADFSP_GPU) provides a 2.77x speedup over NSM, which is the fully optimized standard simulation algorithm of the field.

Algorithm	**time (s)**	
ADFSP_GPU	1.86 ± 0.46	tol=1e-1
ADFSP	53.47 ± 31.13	tol=1e-1
NSM	5.17 ± 0.18	

Table 5.1: Comparison of algorithm speed for the 1D polarization model. For this problem and parameters, the GPU implementation of ADFSP is 28 times faster than the CPU implementation and 2.7 times faster than NSM.

Component	**Time (s)**	**Fraction**	**# of calls**	**E[time/call] (s)**
reaction-diffusion	22.85458	93.74%	94330	2.42E-04
estimate error	1.175021	4.82%	8576	1.37E-04
generate lookup table	0.072708	0.3%	4	1.82E-02
compute timestep	0.021148	0.09%	8576	2.47E-06
accept timestep	0.001859	0.01%	8575	2.17E-07
TOTAL TIME: 24.3798 (s)				

Table 5.2: Computation profile for ADFSP algorithm on the G-protein model with an error tolerance of 1e-1. This table provides profiling information for the ADFSP algorithm. It shows the time taken for each kernel, along with the fraction (percentage) of the total time. It also shows the number of times that kernel was executed (# of calls) as well as the average time of execution for each call of the kernel. E.g. the "generate lookup table" kernel was executed 4 times with an average time of 0.0182 seconds per call, which accounted for 0.3% of the total execution time. Similarly, the reaction-diffusion kernel took 94330 steps with an average time of 2.42e-04 seconds per step, which accounted for 93.74% of the total execution time.

Component	**Time (s)**	**Fraction**	**# of calls**	**E[time/call] (s)**
reaction-diffusion	1.454347	85.83%	1645	8.84E-04
generate lookup table	0.210692	12.43%	3	7.02E-02
estimate error	0.013811	0.82%	150	9.21E-05
compute timestep	0.000527	0.03%	150	3.51E-06
accept timestep	0.000051	0%	149	3.42E-07
TOTAL TIME: 1.6945 (s)				

Table 5.3: Computation profile for ADFSP_GPU algorithm on the G-protein model with an error tolerance of 1e-1. Shown is the total time taken for each kernel, along with the fraction (percentage) of the total time. It also shows the number of times that component was executed (# of calls) as well as the average time of execution for each call of the component.

5.5.2 Min oscillations in *E. coli*

A spatial reaction-diffusion system that has been studied frequently in the literature both in a deterministic setting [67] and using a stochastic description [41] is the periodic oscillations of Min proteins in the bacterium *E. coli*. By oscillating from pole to pole, MinC suppresses the formation of a cell division site at the poles, indirectly positioning it in the middle of the cell. See Section 4.3.1 for more details on this model. Figure 5.6 provides visualizations of this model. This is an ideal model to test spatial stochastic algorithms, as it has a complex geometry with both membrane and cytoplasm domain, and has both mono-molecular and bi-molecular reactions.

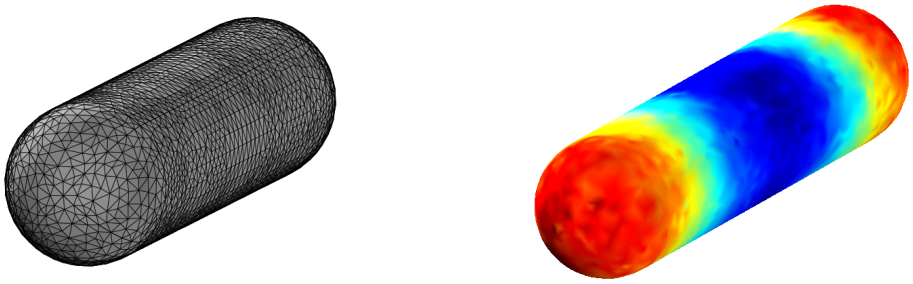

Figure 5.6: Min oscillations in *E. Coli*. Left: 3D tetrahedral mesh of the *E. coli*. Right: Time averaged concentration of MinD on the membrane. Note that the concentration is low in the center and high on the poles of the organism. This is due to the oscillatory nature of the MinCDE cycle. MinD acts as a contractile ring formation inhibitor, thus the contractile ring forms at the center of the cell where the concentration of MinD is lowest. This allows the *E. coli* cell to reliably undergo mitosis into two equal halves.

Figure 5.7 shows an overlay of the pole-to-pole oscillation pattern of membrane-bound MinD in *E. coli*, along with the time steps selected by the adaptive step size selection algorithm in ADFSP. The MinD protein binds to the membrane near the poles of the cell, becomes cytoplasmic and diffuses to the opposite pole, where it binds to the membrane again. When the population of MinD is mostly membrane bound, ADFSP is able to take a larger time step, resulting in greater performance. When MinD is mostly cytoplasmic, ADFSP reduces the timestep. In this trajectory the time steps oscillate between τ= 1.95e-3, 9.77e-4, and 4.33e-4 seconds.

To evaluate the performance of our algorithm we timed the execution of the MinCDE model with a medium discretization (1009 voxels), while varying the error tolerance. Results from these tests are shown in Figure 5.9. As the error tolerance is loosened, the execution performance of the ADFSP algorithm increases, as expected. The GPU implementation is consistently faster than the CPU implementation of the ADFSP algorithm. Figure 5.10 shows the oscillation patterns of the MinD protein for both the ADFSP and NSM methods. Figure 5.11 shows the oscillation patterns of the CPU and GPU implementations of ADFSP as well as of NSM. These results show the effects of the simulation error, as the coherence of the oscillations degrades as the error tolerance is loosened. Table 5.4 and Table 5.5 provide profiling information on the computation.

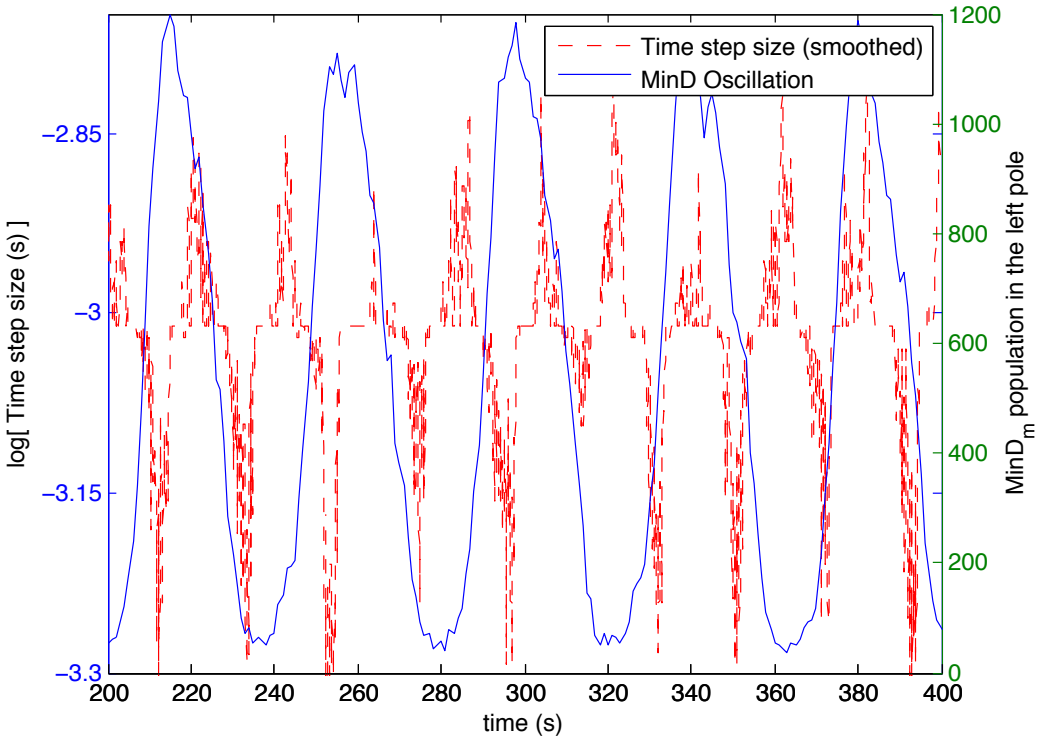

Figure 5.7: The figure shows the time steps selected by ADFSP (red dashed line) and the oscillation of the MinD protein (solid blue line). Note that the time steps oscillate with a period that is approximately half the period of the MinD oscillation. When MinD is at its peak, most of the MinD protein in the system has bound to the membrane at the left pole of the *E. coli*. Similarly, when the pattern is at its trough, most of the MinD has bound to the membrane at the right pole.

Component	**Time (s)**	**Fraction**	**# of calls**	**E[time/call] (s)**
reaction-diffusion	83.91669	71.84%	323386	2.59E-04
estimate error	19.99549	17.12%	29399	6.80E-04
generate lookup table	11.41732	9.77%	4	2.85E+00
compute timestep	0.327569	0.28%	29399	1.11E-05
accept timestep	0.01212	0.01%	29398	4.12E-07
TOTAL TIME: 116.8069 (s)				

Table 5.4: Profiling information for ADFSP algorithm on the MinCDE model with medium discretization (1009 voxels) and error tolerance of 1e-1.

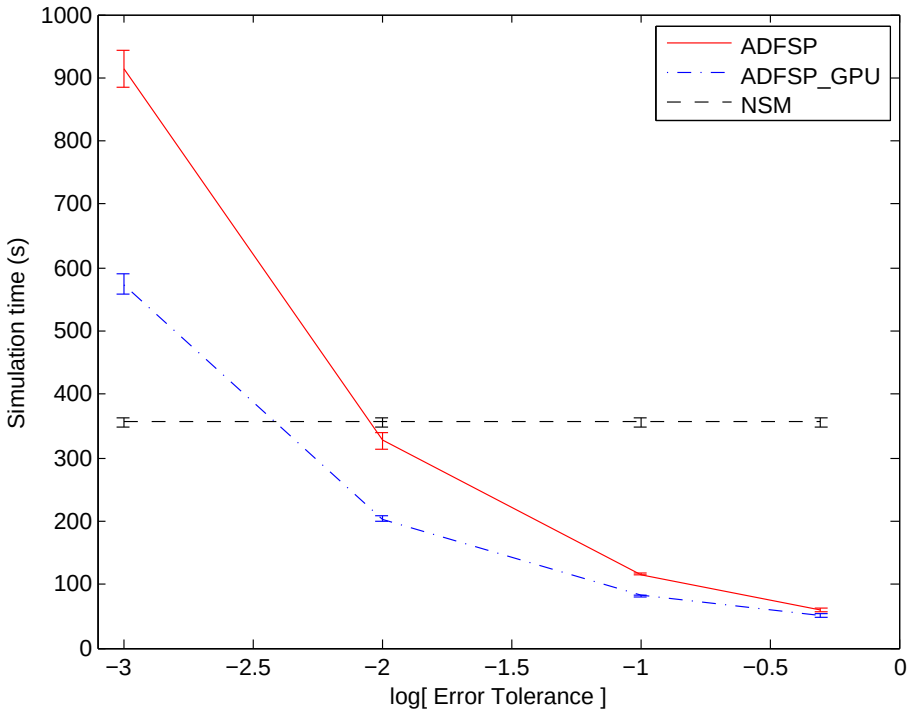

Figure 5.8: Performance versus error tolerance for the ADFSP implementations on CPU and GPU, along with NSM for comparison. As the error tolerance is loosened, the run time for both the ADFSP (CPU) and ADFSP_GPU methods decreases. These results are from the MinCDE model with medium discretization (1009 voxels). For an error tolerance of 1e-1, ADFSP is 3 times faster than NSM and ADFSP_GPU is 4.3 times faster.

Component	**Time (s)**	**Fraction**	**# of calls**	**E[time/call](s)**
reaction-diffusion	51.9517	64.94%	328391	1.58E-04
estimate error	12.4333	15.54%	29854	4.16E-04
generate lookup table	14.95643	18.7%	4	3.74E+00
compute timestep	0.415993	0.52%	29854	1.39E-05
diffusion kernel	0.035804	0.04%	33	1.08E-03
accept timestep	0.008629	0.01%	29853	2.89E-07
TOTAL TIME: 80.0015 (s)				

Table 5.5: Computation profile for ADFSP_GPU algorithm on the MinCDE model with medium discretization (1009 voxels) and an error tolerance of 1e-1. Note that the ADFSP_GPU implementation has both the reaction-diffusion and diffusion-only kernels (See Figure 5.2 for more details).

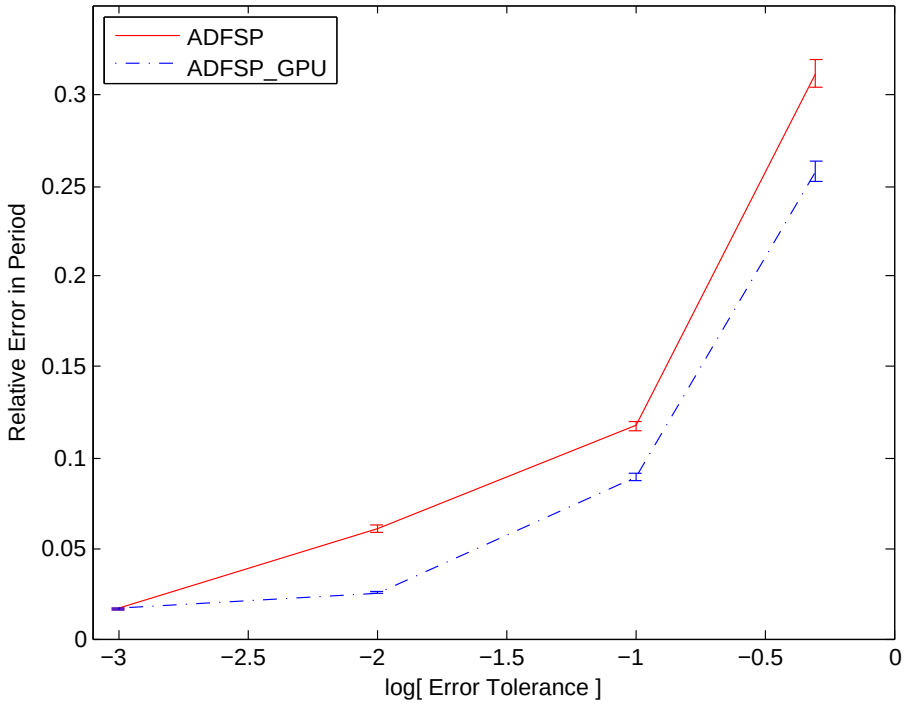

Figure 5.9: Relative error in oscillation period versus error tolerance for ADFSP implementation on the CPU and GPU. As the error tolerance is loosened the error in oscillation period for both the ADFSP (CPU) and ADFSP_GPU methods increase.

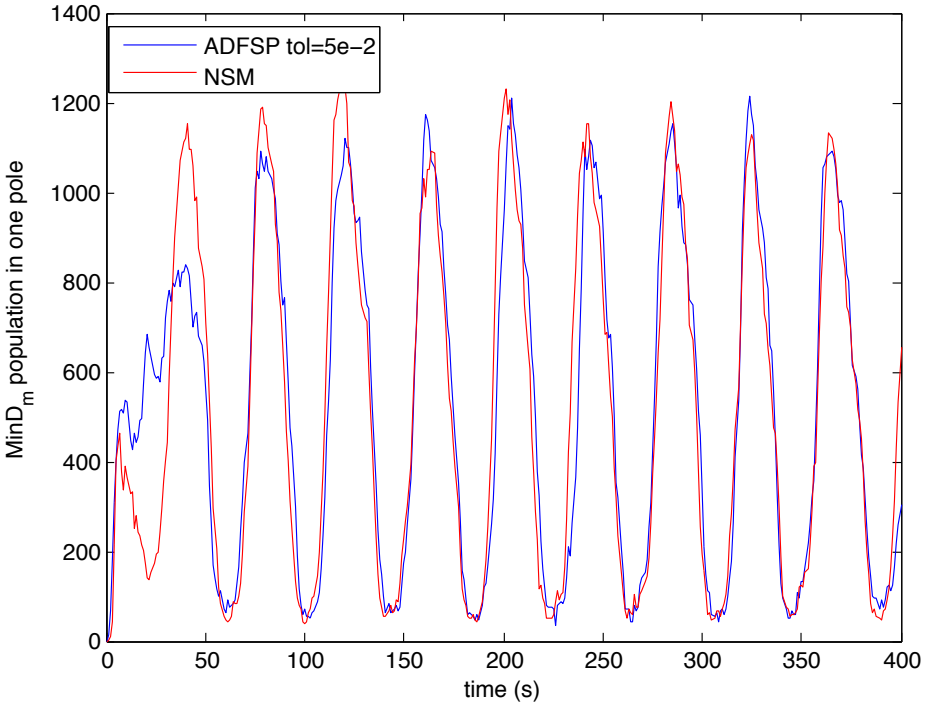

Figure 5.10: Comparison of the oscillation in ADFSP and NSM. The ADFSP run used an error tolerance of 5e-2. For this value, the ADFSP algorithm is approximately the same speed as NSM for this problem. These results are for the MinCDE model with medium discretization (1009 voxels).

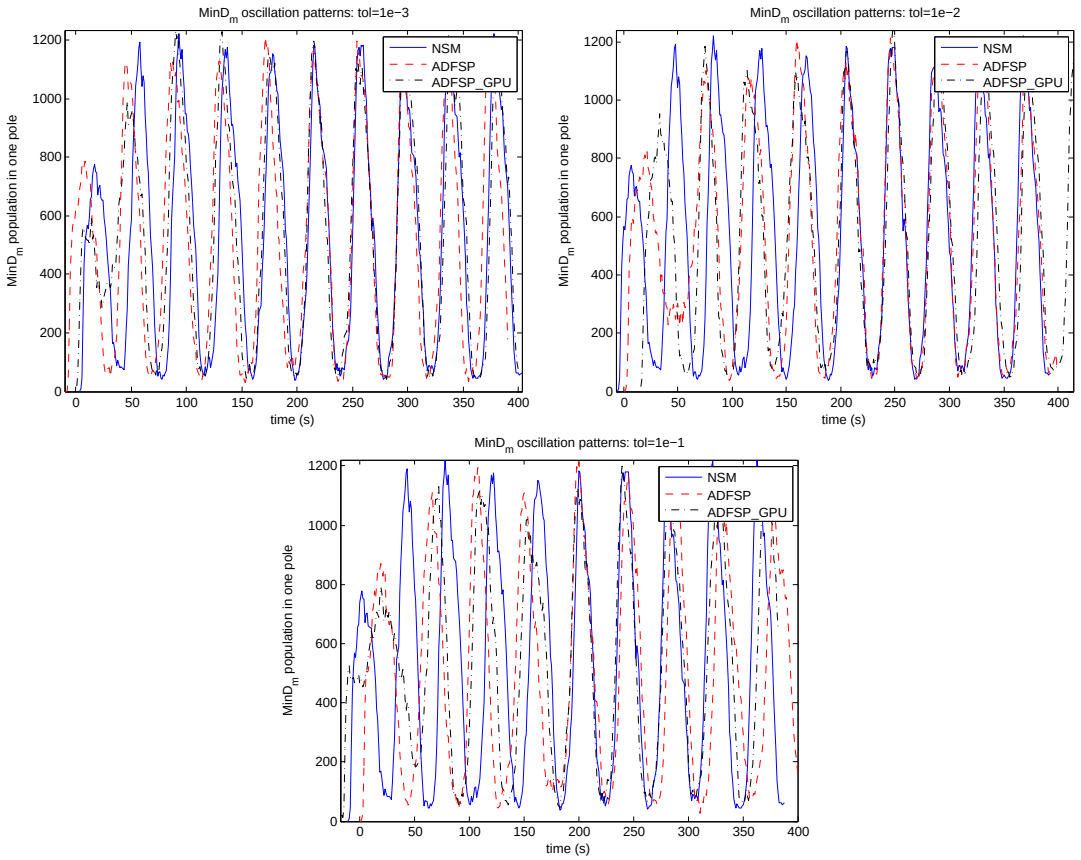

Figure 5.11: Comparison of the oscillation pattern in ADFSP, ADFSP_GPU and NSM for the MinCDE model with medium discretization (1009 voxels). The phase of each trajectory is adjusted so that the center peaks (closest to t=200) are aligned. The ADFSP runs use an error tolerance of 1e-3 (top left), 1e-2 (top right), and 1e-1 (bottom). Note that the coherence of the oscillation patterns degrades as the error tolerance is loosened.

5.6 Discussion

In this Chapter we introduced ADFSP as an extension of the DFSP algorithm that provides an error control mechanism that automatically and adaptively selects time steps to control the local error due to operator splitting. This allows the user to directly trade accuracy in simulation results for increased simulation performance, without a prior knowledge of the system. In addition, the inherently parallel nature of the DFSP algorithm is demonstrated with an implementation of ADFSP on the high-performance GPU compute environment.

Implementation of the ADFSP algorithm in the GPU compute environment presents many challenges. One of these challenges is efficient global synchronization of threads. We use a parallel model where each GPU thread is assigned to the computation of a single voxel. For most of the kernels this is the ideal arrangement, as there is no communication between threads. Global communication occurs in the ADFSP algorithm in two kernels: the diffusion kernel and the compute time step kernel. In the diffusion kernel each thread reads the local system state from global memory, calculates outbound diffusion jump events, and writes the resulting values to the global memory. Race conditions are a major concern, as multiple threads must write to the same location in global memory.

Ensuring global consistency while synchronizing multiple threads in multiple blocks is challenging. We explored several strategies to address this issue in the diffusion kernel. The first was simply to use the atomic primitive *atomicAdd()* when writing to the global state. This ensured consistency at the cost of serializing write operations on collision, when two threads write to the same location. The second strategy was to combine the thread states on a per-block basis and then use *atomicAdd()* for global writes, in an effort to reduce the number of collisions. The third strategy was to use a global memory matrix of size the system state squared, and use two kernels in sequence to synchronize the read/write communication. The first kernel reads the system state vector, calculates diffusion, then writes a column of the matrix with each row corresponding to the outbound diffusion events. The second kernel sums across each row of the matrix, writing the result to the system state vector. In our implementation we used the first strategy. The second strategy was found to be not amenable to our computation due to the unstructured mesh discretization and the broad spatial reach of the DFSP diffusion kernel. Neighboring voxels rarely diffuse molecules to the same destination voxel, and since each voxel may still be written to by multiple blocks, the *atomicAdd()* function is still required for consistency when writing to global memory. Thus, the reduction strategy does not significantly reduce the number of atomic operations required. The third strategy did not provide any performance

improvement due to the large number of global memory access operations. Benchmarking the diffusion kernel on the MinCDE model showed that the first strategy had an average execution time of 1.5e-04 seconds, while the third strategy had an average execution time of 1.9e-02 seconds.

The other kernel that requires global communication is the compute timestep kernel, which computes the L1 norm across space for each species in the system. This is not amenable to the one thread per voxel parallelism strategy. Two methods of implementing this kernel on the GPU were attempted. The first was a strategy that performed the calculation in two phases using a single block of 512 threads and a local shared memory buffer of size number-of-voxels times number-of-species. First, each thread reads the expected change of state from global memory, multiplies it by the volume and writes the product to local shared memory. In the second phase, one thread is assigned to each species in the system, while the rest of the threads wait. The active threads read and sum the values for each voxel from local shared memory, and return the square root of the sum. We found that the limited amount of shared memory rendered this method feasible only for small problem sizes. The second method also used a single block of threads, but used a single GPU thread per species in the system and no shared memory. Each thread reads the expected change of state from global memory and multiplies it by the volume, that value is summed for each voxel in the system, and the square

root of the sum is returned. While this method was feasible for larger problem sizes, it was found to be significantly slower than the CPU implementation. Thus we used the CPU compute time step kernel in our final implementation.

We analyzed both the performance and the error control characteristics of the ADFSP algorithm on two biological model systems. The first was the G-protein cycle in yeast, the reaction-diffusion problem studied in Chapter 3. The parallelism of the ADFSP algorithm was demonstrated. In the first model, the GPU implementation was shown to be 28 times faster than the CPU implementation, and nearly 3 times faster than the fastest implementation (to our knowledge) of NSM. The second model was the MinCDE oscillation in *E. coli*, a spatial stochastic system that cycles between slow membrane diffusion or fast cytoplasmic diffusion regimes. We demonstrated the performance versus accuracy tradeoff inherent in the DFSP method by showing performance and error in oscillation period as a function of the error tolerance parameter. For our nominal error tolerance parameter (1e-1), ADFSP was more than 4 times faster than NSM, while producing solutions with minimal error.

Chapter 6

Spatial Stochastic Modeling of Cell Polarization

Cell polarity is the fundamental process of breaking symmetry to create asymmetric cellular structures. Although polarity is an essential feature of living cells, it is far from being well-understood. In this Chapter we focus on the ability of yeast cells to sense a spatial gradient of mating pheromone and respond by forming a projection in the direction of the mating partner. Specifically, we examine the formation of a key element in the mating process: the polarisome, a protein complex located at the tip of the mating projection.

In this Chapter we present work done in collaboration with Michael Lawson and Tau-Mu Yi, originally presented in [80]. Using a combination of computational modeling and biological experiments we closely examine the pheromone-induced formation of the yeast polarisome. Focusing on the role of noise and spatial heterogeneity, we develop and investigate two mechanistic spatial models

of polarisome formation, one deterministic and the other stochastic. We compare the contrasting predictions of these two models against experimental phenotypes of wild-type and mutant cells, and find that the stochastic model can more robustly reproduce two fundamental characteristics observed in wild-type cells. The first is a highly polarized phenotype, via a mechanism that we refer to as spatial stochastic amplification. The second is the ability of the polarisome to track a moving pheromone input. Additionally, we find that only the stochastic model can simultaneously reproduce these characteristics of the wild-type phenotype and the multi-polarisome phenotype of a deletion mutant of the scaffolding protein Spa2.

It is an open question how the stochasticity of biochemical interactions in the cell hinders or helps cell polarity. Our analysis demonstrates that higher levels of stochastic noise levels result in increased robustness of polarization to parameter variation. Furthermore, our work suggests a novel role for a polarisome protein in the stabilization of actin cables. These findings elucidate the intricate role of spatial stochastic effects in cell polarity, giving support to a cellular model where noise and spatial heterogeneity combine to achieve robust biological function.

6.1 Background

Cell polarity is a classic example of symmetry-breaking in biology. In response to an internal or external cue, the cell asymmetrically localizes components that were previously uniformly distributed. Polarization underlies behaviors such as the chemotaxis of motile cells up chemoattractant gradients, and asymmetric cell division during development [29, 133].

In *Saccharomyces cerevisiae*, a haploid cell (**a** or α) senses a spatial gradient of mating pheromone from its partner and responds by producing a mating projection toward the source. The peptide pheromone binds to a G-protein coupled receptor which activates the heterotrimeric G-protein. Free G$\beta\gamma$ recruits Cdc24 to the membrane where it activates Cdc42. The spatial gradient of activated Cdc42 (Cdc42a) is used to position the polarisome, which generates the mating projection [7, 23, 108]. The role of actin-mediated vesicle transport in the establishment and maintenance of Cdc42a polarity is an area of active research [81, 115], however we focus only on the downstream components.

The polarisome, located at the front of the cell, helps to organize structural, transport, and signaling proteins [109], and guides polarized transport and secretion along actin cables. The function of the polarisome is conserved in eukaryotes,

and analogous scaffold complexes are responsible for such diverse structures as focal adhesions and synapses [1].

A striking feature of the polarisome is its narrow localization at the tip of the mating projection. We call the process of transforming a shallow external gradient into a steep internal gradient (i.e. all-or-none) of protein components spatial amplification; it is a significant challenge to understand and model in cellular polarization [20, 69]. In yeast, this polarization occurs in steps through successive stages of the mating pathway from the extracellular gradient of α-factor (gray in Figure 6.1A) to the more pronounced polarization of G$\beta\gamma$ (blue) to the crescent cap of active Cdc42 (green) to the punctate polarisome at the front of the cell (red) [97].

The two main components of the polarisome are Spa2 and Bni1. Spa2 is an abundant scaffold protein important for structural cohesion of the polarisome; Bni1 is a formin that initiates the polymerization of actin cables, which direct vesicles to the front of the cell [108]. In the absence of Bni1, the mating projection forms slowly and is misshapen [38]. In the absence of Spa2, the mating projection adopts a broad appearance and the polarisome is no longer a single punctate entity [6, 84, 12]. In both loss-of-function mutants, mating efficiency is drastically reduced. One hypothesis is that proper mating requires the alignment of punctate polarisomes (Fig. 1B). Indeed, mutants that exhibit abnormal polari-

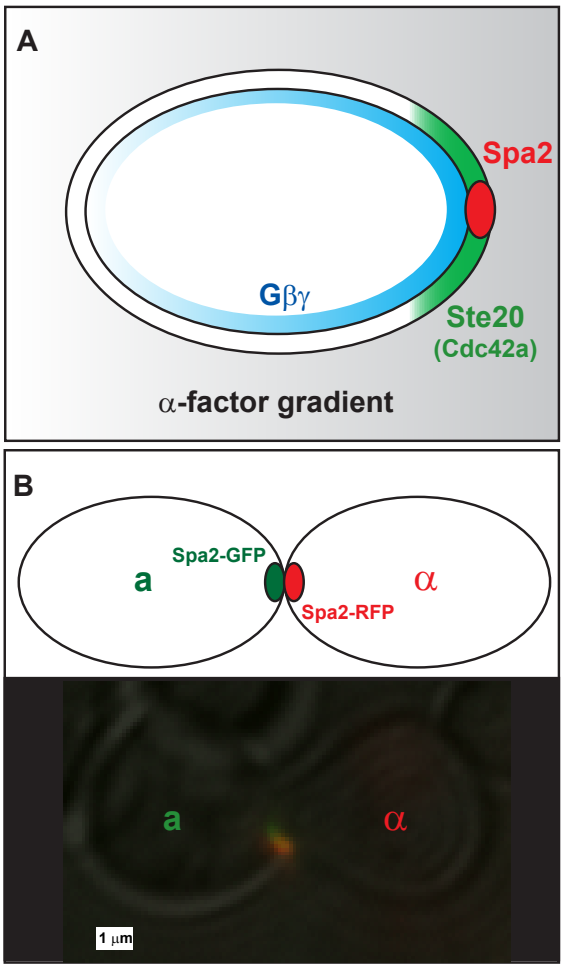

Figure 6.1: Spatial amplification in cell polarity during yeast mating **(A)** Spatial amplification occurs in stages during cell polarization in yeast. The external spatial gradient of α-factor is shallow (gray), and it generates a comparable gradient of free G$\beta\gamma$ on the cell membrane. This initial internal gradient induces a polarized cap of active Cdc42 (green) which in turn localizes the tightly condensed polarisome (red) to the front of the cell. In this manner, a shallow external gradient is amplified to a steep internal gradient. **(B)** A schematic and microscopy image of two mating yeast cells with aligned punctate polarisomes. The polarisomes are labeled with Spa2-GFP (**a**-cell) and Spa2-RFP (α-cell). During mating the polarisomes at the tip of the mating projection are tightly localized and seek out one another until they are aligned and adjacent. When the projections meet the membranes and polarisomes fuse, and mating occurs.

some dynamics often also exhibit decreased mating performance [38, 6, 24]. These data are consistent with the view that a tightly localized polarisome is critical for efficient mating.

A second key feature of the mating projection and polarisome is the ability to track a gradient that may be changing direction. Mathematical modeling of cell polarization highlights the potential tradeoff between amplification, which produces the tight polarization, and tracking of a moving signal source [93, 95]. Positive feedback is one way to achieve amplification, but this feedback can impede the ability to follow a shift in signal direction. Recent studies have shown that fine-tuned modulation of positive feedback can lead to proper polarization and chemotaxis [69, 78], whereas disruption of the positive feedback results in defective polarization [104]. Most of these studies have relied on deterministic models of spatial dynamics. An important question is how stochastic spatial dynamics affect cell polarity, and more specifically how noise affects the amplification/tracking tradeoff.

The impact of noise and stochastic dynamics on signal transduction, protein interaction networks, and gene regulation has gained broad recognition [5]. Examples range from the stochastic lysis-lysogeny decision in phage lambda [90] to transcriptional noise in yeast [85] to morphogen gradient noise in *Drosophila* [64] to stochastic dynamics in the human brain [21]. As a result, many stochastic

models of biological systems have been developed [111] including one of the yeast mating pheromone pathway [131]. In fact, continuous deterministic models (governed by ordinary differential equations) represent a limiting case of more accurate discrete stochastic models (governed by the chemical master equation) [53].

However, most of these models have been non-spatial, in which the system is considered well-mixed. Altschuler et al. [132] modeled cell polarity in yeast using stochastic spatial dynamics. In this system, polarization was induced by overexpressing constituitively-active Cdc42. Their stochastic models of cell polarization involving self-recruitment [2] and actin nucleation with directed transport [87] have highlighted the important role of spatial stochastics in initiating and maintaining spontaneous polarization. In these studies, the authors focused on polarization in the absence of a cue, and did not investigate the amplification or tracking of a gradient.

We present a mathematical model of Cdc42a-gradient induced polarisome formation. To our knowledge this is the first such model. The model is well-supported by experimental data, and we discuss the process of obtaining the parameters from experimental data. There are only two free parameters in the model, and we explore this space via extensive parameter sweeps. Comparing the results of stochastic and deterministic models with equivalent structure, stochastic simulations reveal better and more robust tradeoffs between spatial amplification and

signal tracking. In particular, spatial stochastic effects contribute to tighter polarization, an effect we refer to as *spatial stochastic amplification*. In addition, only the stochastic model can reproduce both of these characteristics of the wild-type (WT) phenotype, as well as the muti-polarisome phenotype of the *spa2*Δ mutant. Finally, our work suggests a novel role for a polarisome protein in the stabilization of actin cables.

6.2 Model Description

We have constructed a mathematical model of the formation of the yeast polarisome. Focusing on the final stage of the mating system, our model takes the broad Cdc42a distribution on the membrane as the input and seeks to produce a narrow polarisome as the output. We sought a simple model that captures the essential dynamics while limiting the size of the parameter space for model analysis. The chemical reactions that make up our model structure were simulated both stochastically and deterministically. The following two subsections describe the model structure and the parameter estimation based on our biological data. For a complete description of the model, see [80].

6.2.1 Model Structure

Figure 6.2 describes our model of the formation of the punctate yeast polarisome in response to a cue: the broader polarization of activated Cdc42 (Cdc42a). The polarisome is a complex structure consisting of at least five different proteins [108]. The two primary functions of the polarisome proteins are structural (scaffolding) and catalytic (nucleation of actin cables). Spa2 is the most abundant scaffold protein, thus to simplify our model we aggregated the scaffold species into Spa2. Bni1 is the formin responsible for actin cable formation during the mating response [38], and so we aggregated the actin nucleation dynamics into Bni1. Table 6.1 lists the biochemical species in our model.

The input to the model is the experimentally measured membrane profile of Cdc42a, and the output is the spatial profile of Spa2. Note that in the simulations, Cdc42a is not a dynamic state variable. However, the direction of the Cdc42a polarization can be shifted as a change in input. To estimate the input profile, we averaged the fluorescence intensity of a Cdc42a reporter, Ste20-GFP, over multiple cells. In the model, Cdc42a recruits Bni1 to the membrane, where it nucleates actin cable assembly [38], producing a positive feedback loop via Spa2, which is delivered to the membrane by transport along the actin cables. For simplicity, we do not explicitly model the actin polymerization process, but instead model actin

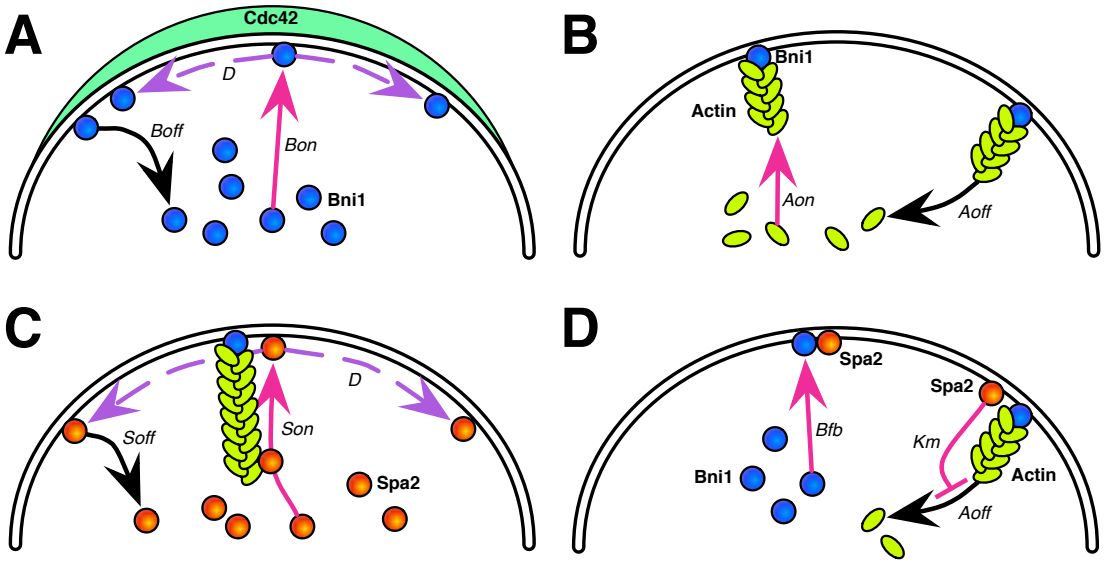

Figure 6.2: Diagram describing yeast polarisome model. **(A)** Input driven recruitment of cytoplasmic Bni1 by membrane bound active Cdc42 (Cdc42a). **(B)** Bni1 on the membrane nucleates and polymerizes actin cables. **(C)** Actin cables direct transport of Spa2 from the cytoplasm to the membrane. **(D)** Spa2 provides positive feedback as it recruits cytoplasmic Bni1 to the membrane and inhibits actin depolymerization.

cables attaching to and detaching from the membrane (Fig. 6.2), thus combining actin initiation and polymerization into a single event.

There are two positive feedback loops in the model. First Spa2 recruits Bni1 to the membrane. There is experimental evidence for Spa2 binding Bni1 [43]. Second, a polarisome component inhibits actin depolymerization so that more Spa2 can be transported to the membrane along actin cables. This is accomplished via a Michaelis-Menten term (see Table 6.2). We constructed two versions of the model structure because the exact mechanism of actin stabilization is not known: one in which Bni1 inhibits actin depolymerization (B-model), and one in which Spa2 inhibits actin depolymerization (S-model). We will focus on the S-model; analogous results can be found for the B-model in [80]. The strength of the first positive feedback loop can be adjusted via the parameter B_{fb}, and the strength of the second positive feedback loop by the inhibition constant K_m. The second positive feedback loop is a model hypothesis motivated by prior modeling results [87]. However, we note that Yu et al. [137] have shown that during yeast budding, polarization is accompanied by more stable actin cable dynamics. In addition, formins in other organisms can facilitate actin bundling at higher concentrations [55]. Finally, Spa2 interacts with a number of accessory proteins including Myo2 which exhibits synthetic lethality with Tpm1 (mutants with both genes deleted are inviable), which binds and stabilizes actin cables [83]. Table 6.2

lists the equations and Table 6.3 lists the parameters that we use to describe the model. For each of the two possible model structures, we employed two modes of simulation. The first uses discrete stochastic kinetics for diffusion and biochemical reactions. The second uses a continuous deterministic formulation, i.e. the familiar partial differential equations for reaction-diffusion systems.

Species	**Description**
$Bni1_{\mathrm{c}}$	Bni1 in the cytoplasm
$Bni1_{\mathrm{m}}$	Bni1 on the plasma membrane
$Spa2_{\mathrm{c}}$	Spa2 in the cytoplasm
$Spa2_{\mathrm{m}}$	Spa2 on the plasma membrane
$Actin_{\mathrm{c}}$	Unpolymerized actin in the cytoplasm
$Actin_{\mathrm{m}}$	Polymerized actin on the plasma membrane

Table 6.1: Biochemical species for the polarisome model.

Reaction	Reaction Rate
$Bni1_c + Cdc42_m \rightarrow Bni1_m + Cdc42_m$	$B_{on} \times [Bni1_c] \times [Cdc42_m]$
$Bni1_m \rightarrow Bni1_c$	$B_{off} \times [Bni1_m]$
$Actin_c + Bni1_m \rightarrow Actin_m + Bni1_m$	$A_{on} \times [Bni1_m] \times [Actin_c]$
$Actin_m + Spa2_m \rightarrow Actin_c + Spa2_m$	$A_{off} \frac{K_m}{K_m + [Spa2_m]} \times [Actin_m]$
$Spa2_c + Actin_m \rightarrow Spa2_m + Actin_m$	$S_{on} \times [Spa2_c] \times [Actin_m]$
$Spa2_m \rightarrow Spa2_c$	$S_{off} \times [Spa2_m]$
$Bni1_c + Spa2_m \rightarrow Bni1_m + Spa2_m$	$B_{fb} \times [Bni1_c] \times [Spa2_m]$

Table 6.2: Reactions and reaction rates for the polarisome model. The reaction rates are all first-order or second-order (bilinear) kinetics except for the actin depolymerization reaction which contains an inhibition term.

Parameter	Value	Description/References
B_{on}	$1.6 \times 10^{-6}\ mol^{-1}s^{-1}$	Active Cdc42 recruits Bni1 to the membrane [38].
B_{off}	$0.25\ s^{-1}$	Bni1 dissociates from the membrane [14].
B_{fb}	$1.9 \times 10^{-5}\ mol^{-1}s^{-1}$	Spa2 on the membrane recruits Bni1 [117].
A_{on}	$7.7 \times 10^{-5}\ mol^{-1}s^{-1}$	Bni1 initiates polymerization of actin cables [55].
A_{off}	$0.018\ s^{-1}$	Actin cables depolymerize.
K_m	3500	Spa2 inhibits actin depolymerization.
S_{on}	$0.16\ mol^{-1}s^{-1}$	Spa2 is transported along actin cables [117].
S_{off}	$0.35\ s^{-1}$	Spa2 dissociates from the membrane.
$Bni1_t$	1000 molecules	Total population of Bni1 in the cell.
$Spa2_t$	5000 molecules	Total population of Spa2 in the cell.
$Actin_t$	40 molecules	Total population of actin cables in the cell.
D	$0.0053\ \mu m\ s^{-2}$	Diffusion coefficient for membrane bound protein.

Table 6.3: Parameters and parameter values for the polarisome model. In Section 6.2.2 we outline the estimation procedure for D, B_{off}, S_{off}, A_{on}, S_{on} and S_{off}. This leaves two free parameters: the B_{fb}/B_{on} ratio and the (K_m, A_{off}) relationship (derivation in [80]).

6.2.2 Parameter Estimation

We used our simple model structure to estimate most of the reaction rates and diffusion constants from our *in vivo* data. This section provides a summary of the full calculations for the S-model. A complete description as well as calculations for the B-model are found in [80].

Similar to the approach of Marco et al. [87], we performed fluorescence recovery after photobleaching (FRAP) experiments (Fig. 6.3A) in the presence and absence of the actin depolymerization agent Latrunculin A (LatA). With LatA treatment we observed a greatly extended recovery time for Bni1, implying that actin-dependent recycling was increasing the recovery rate (compare Fig. 6.3A FRAP recovery curves (curves represent average of 5 experiments with 95% confidence interval)). From the LatA FRAP recovery curves, we directly estimated the membrane diffusion coefficient for Bni1 to obtain $D = 0.005\frac{\mu m}{s^2}$. Because we did not exclude actin-independent mechanisms of Bni1 membrane removal, the measured diffusion coefficient represents an upper bound, however this value is consistent with the slowest of those measured for membrane proteins in [127]. The membrane localization of Spa2 depends on actin, thus we could not perform the identical analysis on Spa2. Instead, we set the diffusion coefficient for Spa2 to be the same as Bni1, a protein similar in size.

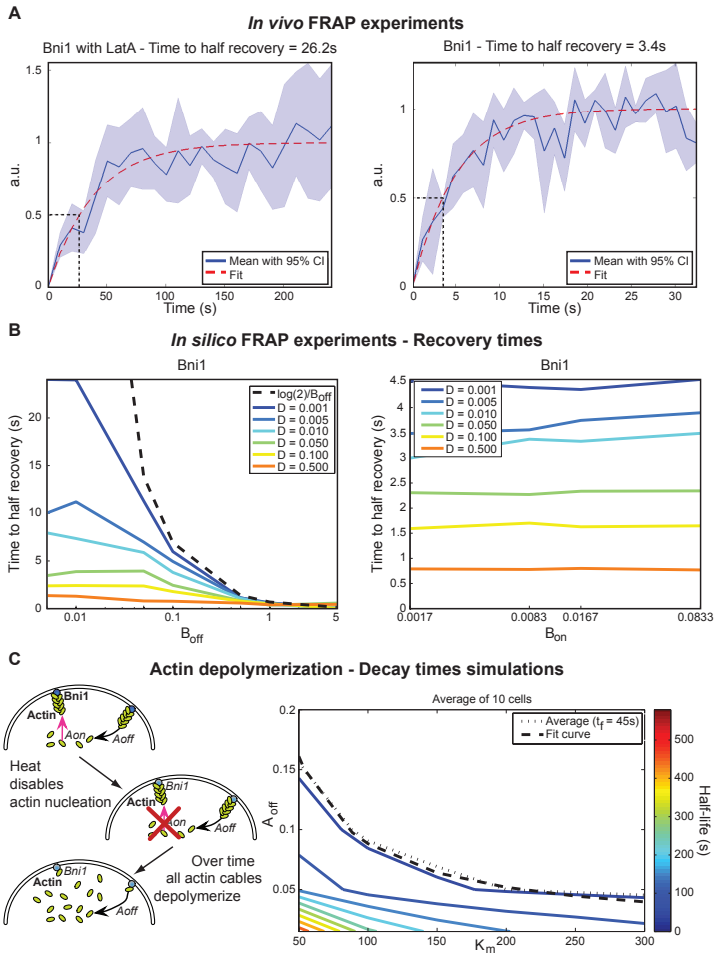

Figure 6.3: Parameter estimation from experimental data including FRAP. (A) Experimental FRAP recovery curves for Bni1 (Solid blue line: average of 5 experiments. Light blue area: 95% confidence interval around average. Dashed red: fit to exponential). Notice that the WT curve has a much shorter time to half recovery than the LatA-treated curve (time to half recovery indicated by dashed black lines). **(B)** FRAP simulation time to half recovery for varying diffusion rates and B_{off} (left) and B_{on} (right). There is no change with B_{on}, while for B_{off} as D goes to zero the curves approach the theoretical no-diffusion limit (dashed black). **(C)** Left: Cartoon illustrating the Bni1 temperature sensitive mutant experiment performed in [39] and simulated in this paper. Right: Phase plane of actin cable half-life (color-coded) as a function of A_{off} and K_{m} (simulation of the experiment in [39]), with the curve representing 45s (dotted black) and our model fit (dashed black). This phase plane represents the average of those generated for initial conditions corresponding to 10 different observed cells.

We derived an analytic expression relating the recovery half-time to the off-rate using a simple version of our model in the limit of no diffusion [120]: $t_{\max/2} = \frac{\log(2)}{k_{\text{off}}}$. Then we calculated $t_{\max/2}$ from FRAP simulations of the full model using a range of values for B_{off} and the diffusion constant D (left panel of Fig. 6.3B). As D approaches zero, the simulations converge toward the no-diffusion limit analytic expression (Fig. 6.3B). These results do not depend on a specific value for B_{on} (right panel of Fig. 6.3B). From the curves in Figures 6.3A and 6.3B and the measured value of D, we were able to estimate $B_{\text{off}} = 0.25\text{s}^{-1}$. The same procedure was performed for Spa2, with the result $S_{\text{off}} = 0.35\text{s}^{-1}$. These values correspond to a region of parameter space in which the polarisome reaction kinetics are faster than the diffusion kinetics for typical yeast membrane proteins.

From the steady-state levels of Spa2 (approximately 90% on the membrane) and S_{off}, we estimated $S_{\text{on}} = 0.32\,\text{molecules}^{-1}\text{s}^{-1}$. The fact that approximately 20% of Bni1 is on the membrane cannot uniquely determine the B_{on} parameter, as Bni1 is delivered to the membrane in two ways: recruitment by Cdc42a (B_{on}) and binding by Spa2 (B_{fb}). Instead, our analysis produced a linearly constrained relationship between B_{on} and B_{fb}.

Finally, we estimated the rate constants for actin polymerization and depolymerization. We made use of data from Evangelista et al. [39], in which the authors induced Bni1 loss-of-function using a temperature-sensitive allele, and then mea-

sured the time course of decay in the number of cells possessing actin cables. Assuming that actin depolymerization is an exponential decay process, the half-life of a single cable is the same as the half-life of a population of cables. Therefore, we can use the decay curve in [39] to estimate a depolymerization rate for individual cables. We fit the data with an exponential decay curve and determined that the time to half actin depolymerization was approximately 45s. *In vitro* experiments by Carlier et al. [16] agree with this timescale for actin depolymerization. In our model, depolymerization depends on both a basal rate A_{off} and an inhibition term containing the constant K_{m}, representing Spa2 inhibition of actin depolymerization. After substituting the actin cable half-life, we obtained the following equation for the total actin depolymerization rate: $\left\{A_{\text{off}} * \frac{K_{\text{m}}}{K_{\text{m}}+Spa2}\right\} = \frac{log(2)}{45\text{s}}$. We note that a similar expression holds for the B-model. Using the above equation we estimated the polymerization rate constant A_{on} to be 7.7e-5 molecules^{-1}s^{-1}.

We performed simulations of the actin depolymerization experiment in [39] to derive a simpler expression for the relationship between A_{off} and K_{m}. Starting with probability distributions of Spa2 and Bni1 taken from fluorescence data as initial conditions, we varied A_{off} and K_{m} and calculated the actin cable half-life. This was repeated on data from 10 cells. Figure 6.3D shows a phase plane of decay times as a function of the two parameters. From this graph, we were able to obtain a direct relationship between A_{off} and K_{m} (Fig. 6.3C, dashed black).

In summary, we used experimental data to identify 6 parameters in the model, reducing the free parameter space to two dimensions: the $B_{\mathrm{on}}/B_{\mathrm{fb}}$ ratio and the (A_{off}, K_{m}) relationship. A third undetermined parameter is the total number of actin cables. In the deterministic simulations, variation of this parameter did not affect polarization, thus it was explored separately. We have selected values for the two free parameters based on the model's ability to reproduce the spatio-temporal characteristics of our *in vivo* data (K_{m}=3500, $B_{\mathrm{fb}}/B_{\mathrm{on}} = 7.5$). We use these values as our nominal parameter set.

6.3 Results

6.3.1 Spatial Stochastic Amplification

A striking feature of the yeast polarisome is its tight localization compared to the broader polarization of Cdc42a (Fig. 6.4A). To characterize this polarization experimentally, we used Ste20-GFP as a fluorescent reporter for Cdc42a (an alternative reporter Gic2-208-GFP produced similar results, see [80]). In pheromone-induced cells, it spanned a full width at half maximum (FWHM, see Section 6.5) of approximately 48° on the membrane (averaged over multiple cells, see Fig. 6.4C). The punctate polarisome was marked by Spa2-mCherry, which localized to a region of FWHM approximately 18°. Cdc42a directs the localiza-

tion of the polarisome by binding polarisome components such as Bni1 [108]. As described above, we refer to spatial amplification as the transformation from a broader input polarization (Cdc42a) to a narrower output polarization (Spa2).

The parameter estimation described in the previous section left our model with two remaining degrees of freedom: $B_{\mathrm{fb}}/B_{\mathrm{on}}$ and (A_{off}, K_{m}). Exploring this space we found that, for any given parameter set, the stochastic model always produced tighter polarization than the deterministic model. We refer to this cue-directed, noise-driven emergent behavior as *spatial stochastic amplification*. This is illustrated in Figure 6.4D for our nominal parameter set, and this behavior is observed across parameter space. The sharp stochastic peaks sample a range of directions similar to what is observed experimentally, whereas the deterministic peak is stationary. As a result, we found that the deterministic simulation of the model overlaid the ensemble average of stochastic trajectories (see Fig. 6.4D). Moreover, as we demonstrate below, increasing the stochasticity of the dynamics results in increased amplification.

6.3.2 Tracking of a Dynamic Input

A second key performance objective that must be balanced against tight polarization is that the yeast polarisome must track a change in the direction of the input cue (Cdc42a). In yeast cells, a change in the α-factor gradient direction

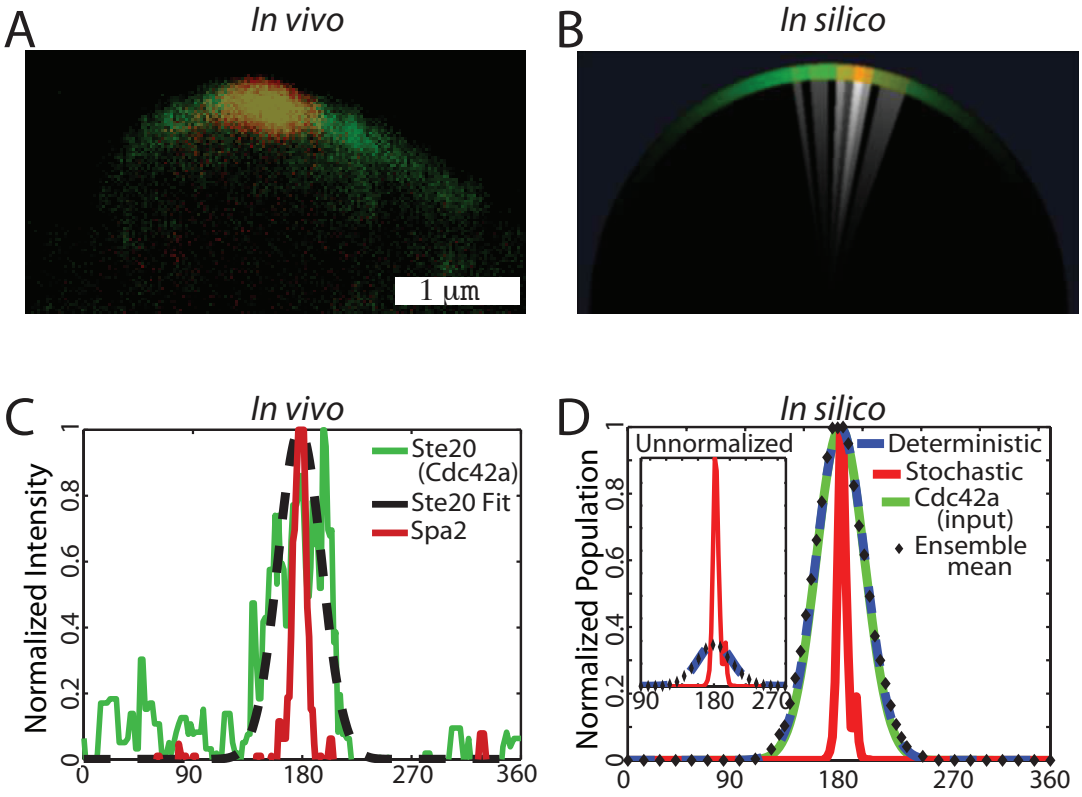

Figure 6.4: Punctate Polarization (Green: Cdc42a, Red: Spa2). (A) Yeast cells treated with α-factor show the wider Cdc42a (marked by Ste20-GFP) and tighter Spa2 polarization. **(B)** Visualization of a stochastic realization of the polarisome model (white indicates regions with actin cables). **(C)** Normalized fluorescence intensity membrane profiles of Ste20-GFP and Spa2-mCherry from a yeast cell undergoing polarisome formation (Green: Ste20 (Cdc42a). Dashed black: Ste20 fit. Red: Spa2). **(D)** Normalized membrane intensity profile from stochastic and deterministic realizations of polarisome model, the Cdc42a input, and the mean output of a stochastic ensemble ($n = 500$). Inset: Absolute membrane intensity profile from stochastic and deterministic realizations of polarisome model, and Spa2 ensemble mean. (Red: Spa2 Stochastic. Dashed blue: Spa2 deterministic. Black diamond: Spa2 ensemble mean. Green: Cdc42a (input)).

results in a corresponding change in Cdc42a polarization. Similarly, extended exposure to isotropic pheromone will also induce a change in Cdc42a localization because of the oscillatory dynamics underlying the formation of multiple projections [12, 105, 125]. We imaged dual-labeled Ste20-GFP/Spa2-mCherry cells in 100 nM α-factor. Under both directional gradient and isotropic α-factor conditions (Fig. 6.5A), we observed that Cdc42a shifts its position to a new polarization site. After a delay (middle panel of Fig. 6.5A) this change was followed by the relocalization of the polarisome (right panel of Fig. 6.5A). We refer to the relocalization of the polarisome following a shift in Cdc42a orientation as successful tracking.

We also imaged cells with Bni1-GFP/Spa2-mCherry to observe the spatio-temporal dynamics of these two polarisome constituents during tracking. A typical time trace for Bni1 and Spa2 is shown in Figures 6.5C and 6.5E respectively. We note that there is an approximately 10 minute transition period during which the nascent second polarisome has begun to form while the initial polarisome still persists. The median characteristic time of this overlap from *in vivo* measurements of five cells was 10 minutes (mean = 15$\pm$11).

In both Cdc42a/Spa2 (Fig. 5B) and Bni1/Spa2 (Fig. 6.5D and 6.5F) dynamics we found that our *in silico* experiments matched the characteristic spatio-temporal features of our *in vivo* experimental results. Figure 6.5B shows that there is a

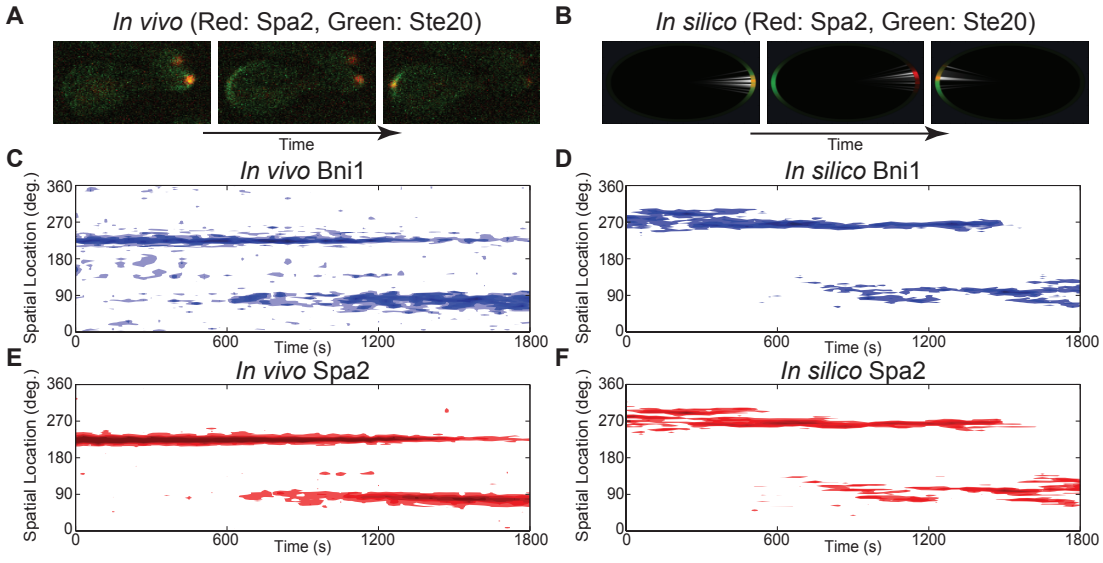

Figure 6.5: Polarisome tracking of directional change in Cdc42a (Green: Ste20 (Cdc42a), Red: Spa2, Blue: Bni1). Left: *In vivo* data. **Right:** *In silico* data. **Top row:** In both the cell **(A)** and the simulation **(B)**, the Cdc42a profile shifts first, followed by the the polarisome (indicated by Spa2). **Middle and bottom rows:** Spatial dynamics of Bni1 **(C, D)** and Spa2 **(E, F)** during polarisome tracking of Cdc42a. Note that the time scale of polarisome switching is similar between *in vivo* and *in silico* experiments, especially in the $\sim$ 10 minute overlap time when two polarisomes are present.

delay in polarisome relocation after a switch in Cdc42a orientation. This is an indication of the need to be in a parameter regime that balances tight polarization (determined in part by B_{fb}) and the ability to successfully track the input signal (determined by B_{on}), a tradeoff that will be discussed further in the following section.

Figures 6.5D and 6.5F show that Bni1 and Spa2 populations in our *in silico* experiments reproduced the spatio-temporal polarisome characteristics noted in our *in vivo* experiments: an approximately 10 minute transition period in which two polarisomes are present. Analysis of an ensemble of 500 trajectories gave an median overlap time of 9.5 (mean = 11.5 ± 8) minutes. This confirms that the polarisome dynamics produced by our model qualitatively and quantitatively agrees with what we observed experimentally.

6.3.3 Robustness to Parameter Perturbation

Previously, it has been hypothesized that there is a tradeoff between the amplified polarization and the ability to follow changes in signal direction [18, 94, 69]. We explored this hypothesis in the context of the polarisome system and investigated the effects of stochastic dynamics on the tradeoff. To accomplish this, we generated phase planes in parameter space for the deterministic and stochastic formulations of the S-model with $B_{\mathrm{fb}}/B_{\mathrm{on}}$ on the y-axis and K_{m} on the x-axis.

We measured polarisome width and the ability to track a directional change. These plots demonstrate this tradeoff for both the stochastic and deterministic simulations (Fig. 6.6). Stronger positive feedback ($B_{\mathrm{fb}} > B_{\mathrm{on}}$) and a more stable polarisome (small K_{m}) produced tighter localization. Because tracking requires the ability to sense a new input direction and shift the polarisome to the new site, greater input influence ($B_{\mathrm{on}} > B_{\mathrm{fb}}$) and less polarisome stability (large K_{m}) yielded better tracking.

To elucidate the effect of stochasticity on the polarisome system, we varied the total number of actin cables in the cell. Figure 6.6 shows phase planes for various levels of total actin cables in the cell, from 20 to 100. We also adjusted S_{on} (rate of actin-mediated Spa2 delivery) to maintain the same flux of Spa2 to the membrane. In our model approximately half of the total population of actin is on the membrane, thus the lower range of actin_t (total number of actin cables) values is consistent with the measurements made by Yu et al. [138]. For the deterministic model, varying the number of actin cables had no effect on polarisome dynamics because partial differential equations treat protein populations as continuous, thus the same profile was produced for all total actin populations. The dynamics of the stochastic model, on the other hand, was strongly dependent on actin cable number.

For the lower range of actin cables there were two striking differences between the performance of the stochastic and deterministic simulations: the stochastic model exhibited a larger overlapping region in parameter space in which both the amplification and tracking criteria were met (see Fig 6.6, compare the purple region of the first and last panels). Additionally, the deterministic model displays an abrupt transition to tracking failure, whereas the stochastic model has a smoother tradeoff between polarisome width and the probability of successful tracking. To illustrate this tradeoff, we examined the minimum possible width given successful tracking. For deterministic models the minimum width was 11.3°. For stochastic models (actin_t=40), the minimum width depended on the strictness of the tracking criteria: at 75% tracking probability the minimum was 12.0°, at 70% it was 10.8°, and at 50% it was 10.7°. This provides further evidence that the stochastic dynamics of this system play a non-trivial role in polarisome formation.

As the number of actin cables was increased, we observed that the gap between stochastic and deterministic performance decreased. As we increase the number of actin cables, the stochastic phase plane increasingly resembles the deterministic one (see Fig. 6.6). The effect is striking both in terms of the individual regions of sufficient amplification (red) and tracking (blue) as well as in the region of overlap (purple). The convergence to deterministic behavior with larger populations is not surprising, given that deterministic models represent the large population limit of

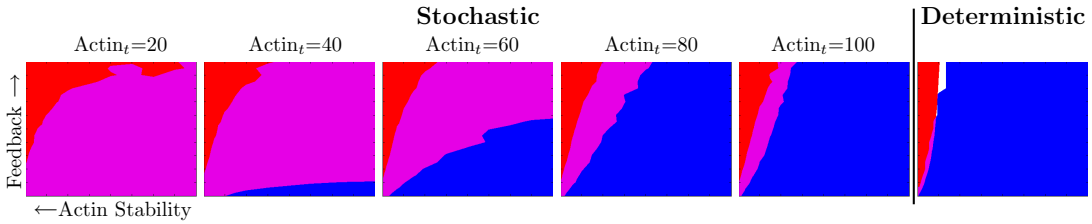

Figure 6.6: Six polarization phenotype space plots of $B_{\mathbf{fb}}/B_{\mathbf{on}}$ ratio versus $K_{\mathbf{m}}$. The first five panels show results from the stochastic model with varying Actin$_t$ values of 20, 40, 60, 80, 100 (left to right); the final panel shows results from the deterministic model. K_{m} values (x-axis) range from 0 to 4000, $B_{\mathrm{fb}}/B_{\mathrm{on}}$ ratio (y-axis) ranges from 0 to 10. Blue indicates accurate tracking (>70% probability), red indicates narrow width (<22° FWHM), and purple indicates that both criteria are met. As the number of actin cables is increased, the stochastic phenotype plots converge to the deterministic plot. Lower actin cable number confers a larger region where both criteria are satisfied, indicating that increased stochasticity leads to more robustness to parameter variation. For each plot, the S_{on} parameter was adjusted to maintain a constant flux of Spa2 to the membrane.

stochastic models. Large total actin cable populations reduce the intrinsic noise in the system, making the stochastic model behave deterministically. Thus, these data suggest that the number of actin cables determines the level of stochasticity in the polarisome system, and that increased noise in the system confers robustness to parameter perturbations. Finally, we found that all of the above observations hold in both the S- and B-models (see [80]) in which either Spa2 or Bni1 inhibits actin depolymerization. Indeed, the simulations showed that some level of polarisome protein mediated inhibition (i.e. $K_{\mathrm{m}} < \infty$) was necessary for proper amplification and tracking, suggesting a novel role for a polarisome protein in the stabilization of actin cables.

6.3.4 Stochastic Simulations Reproduce the Mutant Phenotype

We characterized the *in vivo* dynamic behavior of pheromone-induced *spa2*Δ cells [6] compared to WT cells. In both, the polarisome was marked by the protein Bni1 tagged with GFP (bottom row of Fig. 6.7A). Because the low total population of Bni1 made visualization difficult, we also included mutant and WT images of cells containing the more abundant polarisome marker Sec3 tagged with GFP (top row of Fig. 6.7A). After a two hour treatment with 100 nM α-factor, WT cells possessed a polarisome that exhibited only moderate variability in spatial and temporal behavior (left column of Fig. 6.7). On the other hand, *spa2*Δ cells displayed a distinct phenotype in which multiple polarisome foci appeared (multi-polarisome phenotype), possessing a more dramatic noisy behavior and broader polarization (right column of Fig. 6.7A). These results provide support for the Spa2-dependent positive feedback in the model.

We performed spatial stochastic and deterministic simulations for both WT and *spa2*Δ cells (Fig. 6.7B). In the stochastic simulations we were able to observe the dynamic behavior of the WT polarisome. More strikingly, the stochastic simulations were able to capture important aspects of the *spa2*Δ multi-polarisome phenotype in terms of multiple foci (top right panel of Fig. 6.7B), noisy behavior,

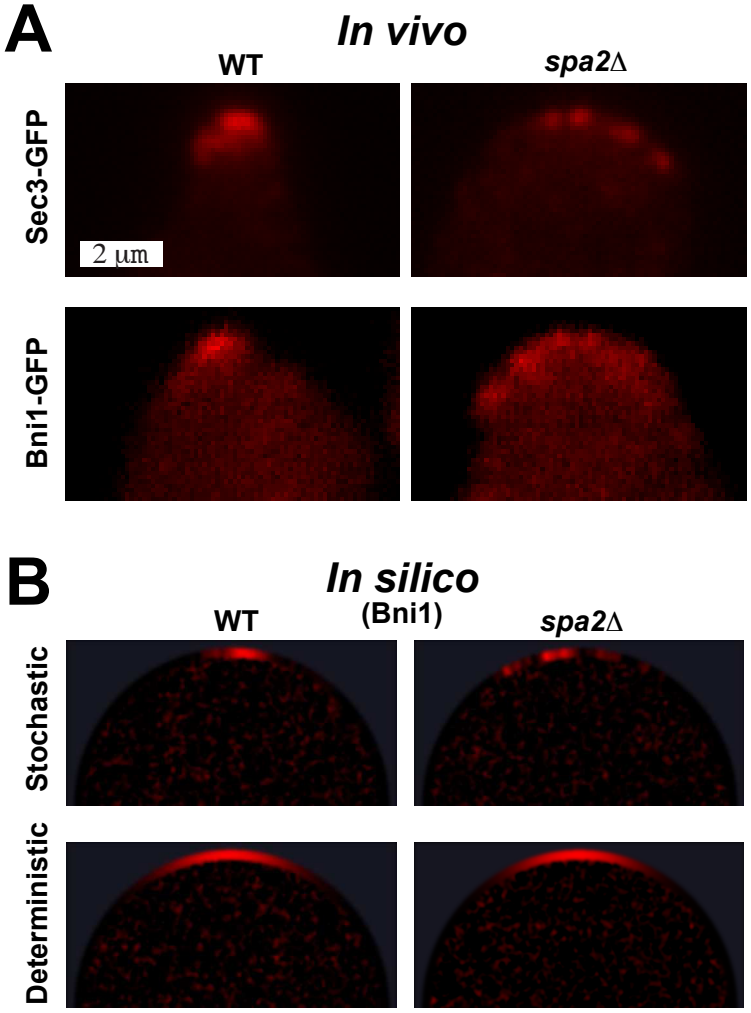

Figure 6.7: The multi-polarisome phenotype in *spa2*Δ cells. Columns: WT phenotype (**left**), *spa2*Δ phenotype (**right**). **(A)** *In vivo* microscopy images of polarizing yeast cells marked with Sec3-GFP (top row) and Bni1-GFP (bottom row). Note the difference between the single punctuate polarisome (left) and the multi-polarisome phenotype (right). **(B)** *In silico* snapshots of yeast polarisome simulations for both stochastic (top row) and deterministic (bottom row) models showing Bni1. Note that only the stochastic *in silico* model is able to match the *in vivo* multi-polarisome phenotype.

and broader polarization. In contrast, deterministic simulations of the model could not reproduce the WT or *spa2*Δ polarisome behaviors (bottom right panel of Fig. 6.7B). Specifically in the *spa2*Δ case, the deterministic simulations showed neither the presence of multi-polarisomes nor the noisy dynamic behavior.

6.4 Discussion

In this work we have constructed a simple model of the yeast polarisome, a classic example of cell polarity, focusing on the dynamics of the proteins Bni1 (a formin) and Spa2 (a scaffold protein). The parameters in the model were fit to experimental data including FRAP experiments performed on living cells. We note that this is the first mathematical model of the polarisome, and as such provides a valuable foundation for future studies of this system. In addition, our model suggests a novel role for a polarisome protein (i.e. Spa2 or Bni1) in the stabilization of actin cables, which we plan to test in the future.

Our *in silico* experiments have shown that stochastic dynamics produced qualitatively different results from deterministic dynamics. First, we found that *spatial stochastic amplification* provided tighter polarization across a range of parameters. Second, the intrinsic noise enabled better tracking given tight amplification, provided increased robustness to parameter perturbations, and better reproduced

the qualitative searching behavior of the polarisome (see below). Finally, only the stochastic model was able to reproduce the *spa2*Δ multi-polarisome phenotype.

This work builds upon and extends the previous work of Marco et al. [87] and Altschuler et al. [2] The key difference is that we focus on the polarisome and the physiological process of sensing and responding to an input gradient of Cdc42a, versus spontaneous polarization in the absence of a cue. However, in all cases, the research demonstrates the power of spatial stochastic dynamics to initiate, amplify, and adjust the polarity.

Similar to the work of Fange and Elf [41], we demonstrate that stochastic but not deterministic simulations can reproduce the phenotype of a mutant in which random spatial clusters appear. In yeast, the wild-type polarisome is a punctate structure that senses an input signal, but forms multiple foci when the Spa2-mediated positive feedback is diminished. In *E. coli*, the MinD protein is normally dispersed and undergoes oscillations, but forms random clusters when the positive feedback is increased (rate of spontaneous association with the membrane is decreased). In both cases, spatial clusters arise from the amplification provided by stochastic spatial dynamics.

The two keys to understanding *spatial stochastic amplification* are the discreteness of molecules and the noisy nature of chemical systems. The discreteness of molecules dictates an integer number of proteins in a given location, so that the

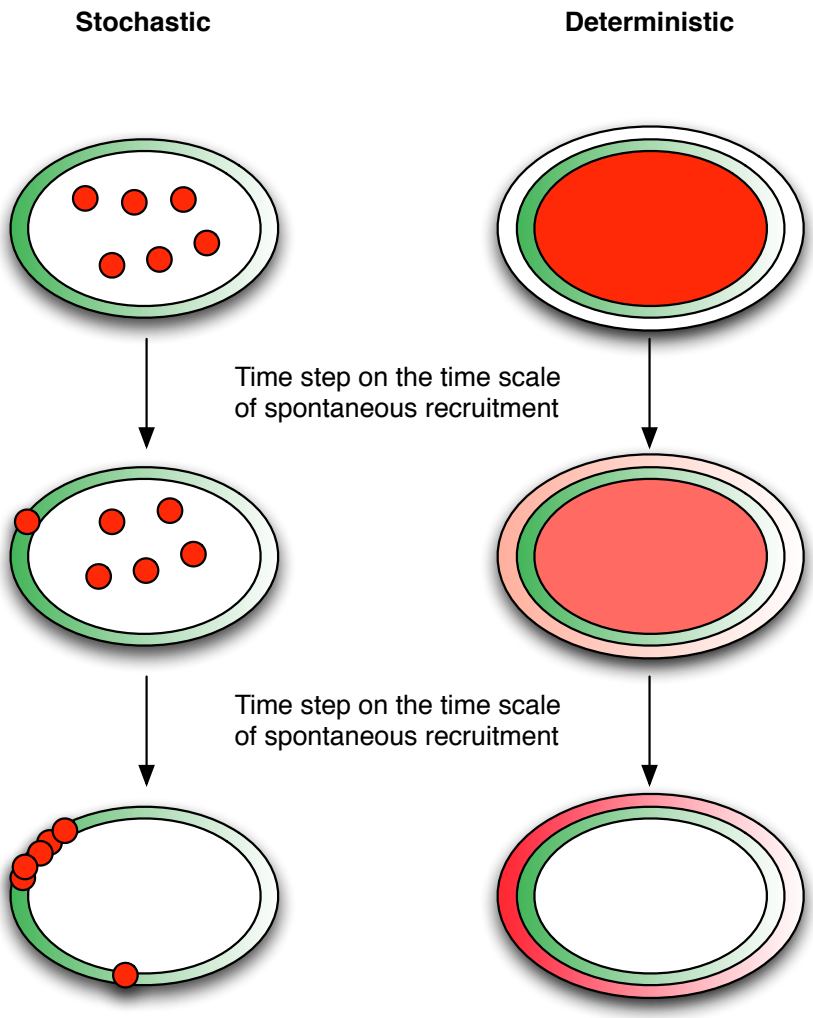

Figure 6.8: Stochastic versus deterministic polarization schematic time course: Green is the input Cdc42a profile, Red is the output (Spa2 or Bni1). **Top:** Initially, in both the stochastic and deterministic simulations, all of the output protein is in the cytoplasm. **Middle:** After a short time period one molecule has been recruited to the membrane. In the stochastic simulation this addition takes place in one discrete location, whereas in the deterministic simulation the addition is in a continuous concentration gradient along the membrane. **Bottom:** This difference in allocation of molecules results in differing final profiles. In the stochastic case, feedback has recruited most of the output protein to the location of the first addition, whereas in the deterministic simulation output protein has been added smoothly along the membrane, resulting in a smooth final distribution.

addition of a molecules is a unit step in population. If molecules were continuous in the sense of concentration, then there would be a smooth addition of molecules to the membrane. Figure 6.8 explains the results of these essential differences. In both the stochastic and deterministic cases we begin at the top of the diagram with all of the Spa2 (or Bni1) in the cytoplasm. After some period of time the first Spa2 molecule in the stochastic simulation has moved to the membrane. By contrast, in the deterministic simulation, a concentration of Spa2 has been added to the membrane in a smooth distribution, in which the total membrane population is equivalent to one molecule of Spa2. Finally, the presence of Spa2 creates positive feedback. In the stochastic simulation the Spa2 molecule is in a discrete location, however in the deterministic simulation, Spa2 exists in a spatially varying continuum across the membrane. In this way it becomes clear how stochastic simulation of the same parameter set, resulting in roughly the same membrane fraction of protein, produces tighter polarization than deterministic simulation. In the parameter regime where this difference is most notable, the output and input shapes are decoupled. That is, the output polarization is the same regardless of input shape (e.g. gradient steepness), but its location is biased by the input profile. As noted above, this process is related to the positive feedback induced symmetry breaking without a directional cue observed in [2].

The polarisome engages in a dynamic search process during mating projection formation and tracking. There are two aspects to this behavior. First, in a single projection, the polarisome scans within the polarized region of Cdc42a as the projection attempts to align with the gradient. Second, during the formation of the second projection, the polarisome explores the new region before it becomes more tightly localized. Both of these behaviors can be observed in the stochastic simulations.

As has been noted in [18, 94, 69] and is clear from the phase planes in Figure 6.6, there is a tradeoff between tight polarization and tracking. Given tighter localization it might seem intuitive that the stochastic simulations would be less likely to track. Tight polarization generally requires stability and strong feedback. Reliable tracking requires instability and high input sensing. The noise in the system lends additional intrinsic instability to the system independent of feedback/sensing tradeoffs, allowing for reliable tracking for relatively tight polarization.

An important feature of robust biological models is that they do not require careful selection of parameters. When modeling bacterial chemotaxis, Barkai and Leibler [8, 136] demonstrated that the experimentally observed perfect adaptation was a structural property of their model, while alternative models required fine-tuning of the parameters to achieve similar performance. We have shown

that for the same model structure and parameters, stochastic dynamics were robust to parameter variation whereas deterministic dynamics required fine-tuning to produce the experimentally observed phenotype. These results add to a growing body of evidence that stochastic noise can play a beneficial role through the introduction of novel and/or robust functionality, which in turn endows cells with a performance advantage [106, 130, 5, 134].

6.5 Materials and Methods

Computational strategy

We modeled polarization of a yeast cell on a one-dimensional periodic domain (i.e. a circle) representing the membrane, which surrounds a well-mixed cytoplasmic region. On the membrane, the spatial location of the biochemical species was critical to understanding the polarization process. Thus, we tracked the location of populations of proteins on the membrane and allowed them to move via a diffusive random walk.

Our stochastic model was formalized via the RDME and simulated with the ISSA for accuracy (see Chapter 3). The deterministic model was described by a set of partial differential equations and solved using standard methods.

Tracking

To determine the effectiveness of our model in replicating this phenotype, we performed the following *in silico* experiment in which the initial Cdc42 signal served as an input in one direction (see first panel of Figure 6.5B). After 1020 seconds, the input was switched by 180 degrees (see second panel of Figure 6.5B). 816 seconds later, we measured how well the polarisome tracked the input signal (see third panel of Figure 6.5B, where the polarisome in the simulation has successfully tracked the input signal). Our criterion for successful tracking was that the polarisome be within 90 degrees of the final input (using average location during the final 204 seconds of the simulation).

Full Width at Half-Maximum (FWHM)

We characterized the polarization tightness with the FWHM of a normal distribution fit to the intensity profile data (see Figure 6.3C for an example fit to experimental data). The width was measured in the same trajectory as above. The polarisome was allowed to form and stabilize for the first 510 seconds, then the FWHM was sampled at each point in time for the next 510 seconds and the time average was taken to be the width value for that trajectory. Final values presented in the phenotype space plots represent the ensemble mean value.

Cell culture and pheromone treatment

All yeast strains were derivatives of RJD415 (W303). Genetic techniques were performed according to standard methods. Complete strain details are presented in [80]. Cells were cultured in YPD (yeast extract-peptone-dextrose) media.

Cells were treated with α-factor for 2 hours before imaging. A low concentration of pheromone (10-20 nM) was used for imaging a dynamic single projection, and a high concentration (100 nM) was used for imaging the second projection formation in the tracking experiments.

Microscopy

Live yeast cells were immobilized on glass slides with concanavalin A (ConA) in the presence of YPD media containing α-factor. They were then imaged on an Olympus Fluoview confocal microscope with a 60x objective using 488 nm (GFP) and 568 nm (mCherry) excitation wavelengths. Time-lapse images were taken at 30s intervals over a 30 min to 1 hour period. A relatively long dwell time and scan averaging removed much of the imaging noise.

Chapter 7

Conclusions

To conclude, we summarize the research contributions of this dissertation and provide some further discussion.

In Chapter 3 we introduced the Diffusive Finite State Projection (DFSP) algorithm for spatial stochastic simulation. The DFSP algorithm is a computational framework for accurate and efficient simulation of stochastic spatially inhomogeneous biochemical systems. This new computational method employs a fractional step hybrid strategy. A novel formulation of the Finite State Projection (FSP) method [98], called the Diffusive FSP (DFSP) method, is used for the efficient and accurate simulation of diffusive transport. Reactions are handled by the Stochastic Simulation Algorithm (SSA).

In Chapter 4 we presented the software package URDME for simulation of Reaction Diffusion Master Equation models on Unstructured tetrahedron based spatial grids. URDME is designed for both investigators studying spatial stochastic models in the field of molecular systems biology, and for developers of spatial stochastic algorithms. For investigators, URDME provides an interactive Matlab interface for model building, script computing, data management, and post-processing. In addition, it provides an interactive environment for 3D model construction through a connection to the Comsol Multiphysics geometry and mesh handling software. For algorithm developers, URDME is designed to be easily modified and extended with newly developed simulation routines. The complexity of model building, and management of the geometry and meshes are logically abstracted from the simulation routines. This facilitates the testing of newly developed methods in realistic settings at an early stage of development.

In Chapter 5 we presented an extension to DFSP that includes automatic error control, and explored the benefits of parallel execution on NVIDIA graphics processing units. The biggest challenge faced by scientists utilizing the DFSP algorithm is a selection of the operator splitting timestep that produces optimum speed without violating the specified error tolerance. This requires *a priori* knowledge of the system under study. To address this challenge, we introduced the adaptive DFSP algorithm (ADFSP) that automatically and adaptively se-

lects the appropriate timestep for performance and error control. In addition, we demonstrated the parallel efficiency inherent in the ADFSP algorithm with an implementation on the NVIDIA Graphics Processing Unit.

In Chapter 6 we explored the spatial stochastic dynamics of cellular polarization in yeast mating. Polarization is an essential behavior of living cells, yet the dynamics of this symmetry-breaking process are not fully understood. In this work, we focused on the ability of yeast cells to sense a spatial gradient of mating pheromone and respond by forming a projection in the direction of the mating partner. Using experiments and modeling, we examined the formation of the yeast polarisome, a punctate structure localized to the tip of the mating projection, and we elucidated the intricate role of spatial stochastic effects in its formation. We found that only the stochastic models were able reproduce observed wild-type characteristics of polarization. Additionally, we found that higher levels of stochastic noise levels result in increased robustness of polarization to parameter variation.

There are several promising directions for future work. Potential enhancements to the ADFSP method include an advanced error control algorithm that is more efficient to compute and that would allow for larger operator splitting timesteps to be taken. Another enhancement would be a formulation of the algorithm that would be analogous to an implicit method for differential equations. This would

address the limitations of explicit timestep selection and allow for greater efficiency by using larger timesteps. A third is to utilize asynchronous time stepping, allowing different spatial subdomains to take steps of different sizes for greater efficiency.

The ADFSP_GPU method could be enhanced with a hybrid multi-core, multi-GPU implementation. This would allow larger models to be simulated, and could allow some components to be simultaneously executed on the CPU and GPU. One example would be to execute the error estimate on the CPU while the GPU computes the next reaction-diffusion step. In addition, the new Kepler series GPU from NVIDIA promises several features that would significantly speed up execution. The dynamic parallelism feature, the ability for GPU kernels to launch other GPU kernels, enables global synchronization without returning control to the CPU and would allow the entire algorithm to be implemented in the GPU. The hyper-q feature enables multiple CPU threads to launch GPU kernels. This enables greater GPU utilization by reducing serialization and idle computational units.

Advances in spatial stochastic algorithms allow scientists to study larger, more complex systems. And, as computational resources increase in their size and complexity, the size of the models that are able to be simulated also increases. As a result of this progress, computational models of a whole cell have become

feasible [74], and have begun to play an important role in molecular systems biology. Software and algorithms that enable modeling of a whole cell, where all components are handled using spatial stochastic methods, would be of great utility of the scientific community.

Bibliography

[1] B. Alberts, D. Bray, J. Lewis, M. Raff, K. Roberts, and J. Watson, *Molecular Biology of the Cell*, Garland Publishing, New York, 1994.

[2] S. J. Altschuler, S. B. Angenent, Y. Wang, and L. F. Wu, *On the spontaneous emergence of cell polarity*, Nature, 454 (2008), pp. 886–889.

[3] M. Ander, P. Beltrao, B. D. Ventura, J. Ferkinghoff-Borg, M. Foglierini, C. Lemerle, I. Tomas-Oliveira, and L. Serrano, *Smartcell, a framework to simulate cellular processes that combines stochastic approximation with diffusion and localisation: analysis of simple networks*, Syst. Biol., 1 (2004), pp. 129–138.

[4] S. S. Andrews, N. J. Addy, R. Brent, and A. P. Arkin, *Detailed simulations of cell biology with Smoldyn 2.1*, PLoS Comput. Biol., 6 (2010), p. e1000705.

[5] A. Arkin, J. Ross, and H. H. McAdams, *Stochastic kinetic analysis of developmental pathway bifurcation in phage λ-infected Escherichia coli cells*, Genetics, 149 (1998), pp. 1633–1648.

[6] R. Arkowitz and N. Lowe, *A small conserved domain in the yeast Spa2p is necessary and sufficient for its polarized location*, J. Cell Biol., 138 (1997), pp. 17–36.

[7] R. A. Arkowitz, *Cell polarity: Connecting to the cortex*, Curr Biol, 11 (2001), pp. R610–R612.

[8] N. Barkai and S. Leibler, *Robustness in simple biochemical networks*, Nature, 387 (1997), pp. 913–7.

[9] N. Barkai and S. Leibler, *Circadian clocks limited by noise*, Nature, 403 (2000), pp. 267–268.

[10] B. Bayati, P. Chatelin, and P. Koumoutsakos, *Adaptive mesh refinement for stochastic reaction-diffusion processes*, J. Comput. Phys., 230 (2011), pp. 13–26.

[11] D. Bernstein, *Simulating mesoscopic reaction-diffusion systems using the Gillespie algorithm*, Phys. Rev. E, 71 (2005), p. 041103.

[12] S. Bidlingmaier and M. Snyder, *Regulation of polarized growth initiation and termination cycles by the polarisome and Cdc42 regulation*, J. Cell Biol., 164 (2004), pp. 207–218.

[13] K. Burrage, M. Hegland, S. MacNamara, and R. Sidje, *A krylov-based finite state projection algorithm for solving the chemical master equation arising in the discrete modeling of biological systems*, in Proceedings of the 150th Markov Anniversary Meeting, A. Langville and W. Stewart, eds., Raleigh, NC, 2006, Boson Books, p. 2138.

[14] S. M. Buttery, S. Yoshida, and D. Pellman, *Yeast formins Bni1 and Bnr1 utilize different modes of cortical interaction during the assembly of actin cables*, Mol. Biol. Cell, 18 (2007), pp. 1826–1838.

[15] Y. Cao, H. Li, and L. Petzold, *Efficient formulation of the stochastic simulation algorithm for chemically reacting systems*, The Journal of Chemical Physics, 121 (2004), pp. 4059–4067.

[16] M.-F. Carlier, V. Laurent, J. Santolini, R. Melki, D. Didry, G.-X. Xia, Y. Hong, N.-H. Chua, and D. Pantaloni, *Actin Depolymerizing Factor (ADF/Cofilin) Enhances the Rate of Filament Turnover: Implication in Actin-based Motility*, The Journal of Cell Biology, 136 (1997), pp. 1307–1322.

[17] J. Chant, *Cell polarity in yeast*, Annual Review of Cell and Developmental Biology, 15 (1999), pp. 365–391.

[18] C.-S. Chou, Q. Nie, and T.-M. Yi, *Modeling robustness tradeoffs in yeast cell polarization induced by spatial gradients*, PLoS ONE, 3 (2008), p. e3103.

[19] R. Courant, K. Friedrichs, and H. Lewy, *Über die partiellen Differenzengleichungen der mathematischen Physik*, Mathematische Annalen, 100 (1928), pp. 32–74.

[20] A. Dawes and L. Edelsteinkeshet, *Phosphoinositides and Rho proteins spatially regulate actin polymerization to initiate and maintain directed movement in a one-dimensional model of a motile cell*, Biophysical Journal, 92 (2007), pp. 744–768.

[21] G. Deco, E. Rolls, and R. Romo, *Stochastic dynamics as a principle of brain function*, Progress in Neurobiology, 88 (2009), pp. 1–16.

[22] R. Dixit, J. L. Ross, Y. E. Goldman, and E. F. Holzbaur, *Differential regulation of dynein and kinesin motor proteins by Tau*, Science, 319 (2008), pp. 1086–1089.

[23] H. G. Dohlman and J. Thorner, *Regulation of G protein-initiated signal transduction in yeast: Paradigms and principles*, Annual Review of Biochemistry, 70 (2001), pp. 703–754.

[24] R. Dorer, C. Boone, T. Kimbrough, J. Kim, and L. H. Hartwell, *Genetic analysis of default mating behavior in Saccharomyces cerevisiae*, Genetics, 146 (1997), pp. 39–55.

[25] B. Drawert, S. Engblom, and A. Hellander, *Urdme 1.1: User's manual*, Tech. Rep. 2011-003, Department of Information Technology, Division of Scientific Computing, Uppsala University, 2011.

[26] B. Drawert, S. Engblom, and A. Hellander, *Urdme: a modular framework for stochastic simulation of reaction-transport processes in complex geometries*, BMC Systems Biology, 6 (2012), p. 76.

[27] B. Drawert, A. Hellander, M. Lawson, and L. Petzold, *Accelerated Spatial Stochastic Simulation on Multicore and GPU Systems*, Manuscript in preparation, (2013).

[28] B. Drawert, M. J. Lawson, L. Petzold, and M. Khammash, *The diffusive finite state projection algorithm for efficient simulation of the stochastic reaction-diffusion master equation*, J. Chem. Phys., 132 (2010), p. 074101.

[29] D. G. Drubin and W. J. Nelson, *Origins of cell polarity.*, Cell, 84 (1996), pp. 335–344.

[30] H. El-Samad and M. Khammash, *Coherence resonance: A mechanism for noise induced stable oscillations in gene regulatory networks*, in Conf. on Decision and Control, San Diego, CA, 2006, IEEE.

[31] J. Elf and M. Ehrenberg, *Spontaneous separation of bi-stable biochemical systems into spatial domains of opposite phases*, Systems Biology, IEE Proceedings, 1 (2004), pp. 230–236.

[32] J. Elf, G.-W. Li, and X. S. Xie, *Probing transcription factor dynamics at the single-molecule level in a living cell*, Science, 316 (2007), pp. 1191–1194.

[33] M. B. Elowitz, A. J. Levine, E. D. Siggia, and P. S. Swain, *Stochastic gene expression in a single cell*, Science, 297 (2002), pp. 1183–1186.

[34] D. Elversson and E. Kieri, *Spatial stochastic simulation of spatial stochastic simulation of spatial stochastic simulation of cellular reaction networks: A comparison of discretizations of the laplace operator for mesoscopic diffusion*, Master's thesis, Department of Infomation Technology, Uppsala University, 2010.

[35] S. E. Encalada, L. Szpankowski, C.-h. Xia, and L. S. B. Goldstein, *Stable kinesin and dynein assemblies drive the axonal transport of mammalian prion protein vesicles*, Cell, 144 (2011), pp. 551–565.

[36] S. Engblom, L. Ferm, A. Hellander, and P. Lötstedt, *Simulation of stochastic reaction–diffusion processes on unstructured meshes*, SIAM J. Sci. Comput., 31 (2009), pp. 1774–1797.

[37] R. Erban and J. Chapman, *Stochastic modelling of reaction-diffusion processes: Algorithms for bimolecular reactions*, Phys. Biol., 6 (2009), p. 046001.

[38] M. Evangelista, K. Blundell, M. S. Longtine, C. J. Chow, N. Adames, J. R. Pringle, M. Peter, and C. Boone, *Bni1p, a yeast formin linking Cdc42p and the actin cytoskeleton during polarized morphogenesis*, Science, 276 (1997), pp. 118–122.

[39] M. Evengelista, D. Pruyne, D. C. Amberg, C. Boone, and A. Bretscher, *Formins direct Arp2/3-independent actin filament assembly to polarize cell growth in yeast*, Nat Cell Biol, 4 (2002), pp. 32–41.

[40] D. Fange, O. G. Berg, P. Sjöberg, and J. Elf, *Stochastic reaction-diffusion kinetics in the microscopic limit*, Proc. Natl. Acad. Sci. USA, 107 (2010), pp. 19820–19825.

[41] D. Fange and J. Elf, *Noise-induced min phenotypes in E. coli*, PLoS Computational Biology, 2 (2006), p. e80.

[42] L. Ferm, A. Hellander, and P. Lötstedt, *An adaptive algorithm for simulation of stochastic reaction-diffusion processes*, J. Comput. Phys., 229 (2010), pp. 343–360.

[43] T. Fujiwara, K. Tanaka, A. Mino, M. Kikyo, K. Takahashi, K. Shimizu, and Y. Takai, *Rho1p-Bni1p-Spa2p interactions: implication in localization of Bni1p at the bud site and regulation of the actin cytoskeleton in Saccharomyces cerevisiae*, Mol. Biol. Cell, 9 (1998), p. 12211233.

[44] C. W. Gardiner, *Handbook of Stochastic Methods for Physics, Chemistry and the Natural Sciences*, Springer-Verlag, 3rd ed., 2004.

[45] C. W. Gardiner, K. J. McNeil, D. F. Walls, and I. S. Matheson, *Correlations in stochastic theories of chemical reactions*, Journal of Statistical Physics, 14 (1976), pp. 307–331.

[46] T. Gardner, C. Cantor, and J. Collins, *Construction of a genetic toggle switch in Escherichia coli*, Nature, 403 (2000), pp. 339–342.

[47] C. Geuzaine and J.-F. Remacle, *Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities*. http://www.geuz.org/gmsh/.

[48] M. Gibson and J. Bruck, *Efficient exact stochastic simulation of chemical systems with many species and many channels*, The Journal of Physical Chemistry A, 104 (2000), pp. 1876–1889.

[49] D. Gillespie, *Exact stochastic simulation of coupled chemical reactions*, The Journal of Physical Chemistry, 81 (1977), pp. 2340–2361.

[50] ——, *A rigorous derivation of the chemical master equation*, Physica A: Statistical Mechanics and its Applications, 188 (1992), pp. 404 – 425.

[51] ——, *Approximate accelerated stochastic simulation of chemically reacting systems*, The Journal of Chemical Physics, 115 (2001), pp. 1716–1733.

[52] D. T. Gillespie, *A general method for numerically simulating the stochastic time evolution of coupled chemical reacting systems*, J. Comput. Phys., 22 (1976), pp. 403–434.

[53] D. T. Gillespie, *The chemical Langevin equation*, The Journal of Chemical Physics, 113 (2000), pp. 297–306.

[54] L. S. B. Goldstein and Z. Yang, *Microtubule-based transport systems in neurons: The roles of kinesins and dyneins*, Ann. Rev. Neurosci., 23 (2000), pp. 39–71.

[55] B. L. Goode and M. J. Eck, *Mechanism and function of formins in the control of actin assembly*, Annual Review of Biochemistry, 76 (2007), pp. 593–627.

[56] S. P. Gross, M. Vershinin, and G. T. Shubeita, *Cargo transport: Two motors are sometimes better than one*, Curr. Biol., 17 (2007), pp. R478–R486.

[57] J. Hasty, D. McMillen, and J. Collins, *Engineered gene circuits*, Nature, 420 (2002), pp. 224–230.

[58] J. Hattne, D. Fange, and J. Elf, *Stochastic reaction-diffusion simulation with MesoRD*, Bioinformatics, 21 (2005), pp. 2923–2924.

[59] A. Hellander, S. Hellander, and P. Lötstedt, *Coupled mesoscopic and microscopic simulation of reaction-diffusion processes in mixed dimensions*, Multiscale. Model. Simul., 10 (2012).

[60] A. Hellander and P. Lötstedt, *Incorporating active transport in mesoscopic reaction-transport models inside living cells*, Tech. Rep. 2010-003, Department of Information Technology, Uppsala University, 2010.

[61] S. Hellander, A. Hellander, and L. Petzold, *On the reaction-diffusion master equation in the microscopic limit*, Phys. Rev. E, 85 (2012).

[62] I. Hepburn, W. Chen, S. Wils, and E. D. Schutter, *STEPS: efficient simulation of stochastic reaction-diffusion models in realistic morphologies*, BMC Systems Biology, 6 (2012).

[63] N. Hirokawa and R. Takemura, *Molecular motors and mechanisms of directional transport in neurons*, Nat. Rev. Neurosci., 6 (2005), pp. 201–214.

[64] B. Houchmandzadeh, E. Wieschaus, and S. Leibler, *Establishment of developmental precision and proportions in the early drosophila embryo*, Nature, 415 (2002), pp. 798–802.

[65] J. Howard, *The movement of kinesin along microtubules*, Annu. Rev. Physiol., 58 (1996), pp. 703–729.

[66] M. Howard and A. Rutenberg, *Pattern formation inside bacteria: Fluctuations due to the low copy number of proteins*, Phys. Rev. Lett., 90 (2003), p. 128102.

[67] K. C. Huang, Y. Meir, and N. S. Wingreen, *Dynamic structures in* escherichia coli*: Spontaneous formation of MinE and MinD polar zones*, Proc. Natl. Acad. Sci. USA, 100 (2003), pp. 12724–12728.

[68] M. Hucka, A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, K. H., , the rest of the SBML Forum:, A. P. Arkin, B. J. Bornstein, D. Bray, A. Cornish-Bowden, A. A. Cuellar, S. Dronov, E. D. Gilles, M. Ginkel, V. Gor, I. I. Goryanin, W. J. Hedley, T. C. Hodgman, J.-H. Hofmeyr, P. J. Hunter, N. S. Juty, J. L. Kasberger, A. Kremling, U. Kummer, N. Le Novere, L. M. Loew, D. Lucio, P. Mendes, E. Minch, E. D. Mjolsness, Y. Nakayama, M. R. Nelson, P. F. Nielsen, T. Sakurada, J. C. Schaff, B. E. Shapiro, T. S. Shimizu, H. D. Spence, J. Stelling, K. Takahashi, M. Tomita, J. Wagner, and J. Wang, *The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models*, Bioinformatics, 19 (2003), pp. 524–531.

[69] P. A. Iglesias and A. Levchenko, *Modeling the cell's guidance system*, Sci. STKE, 2002 (2002), p. re12.

[70] S. Isaacson, *The reaction-diffusion master equation as an asymptotic approximation of diffusion to a small target*, SIAM Journal on Applied Mathematics, 70 (2009), pp. 77–111.

[71] S. A. Isaacson and C. S. Peskin, *Incorporating diffusion in complex geometries into stochastic chemical kinetics simulations*, SIAM J. Sci. Comput., 28 (2006), pp. 47–74.

[72] T. Jahnke and C. Lubich, *Error bounds for exponential operator splittings*, BIT Numerical Mathematics, 40 (2000), pp. 735–744. 10.1023/A:1022396519656.

[73] A. Jensen, *Markoff chains as an aid in the study of Markoff processes*, Skand. Aktuarietidskr., 36 (1953), pp. 87–91.

[74] J. R. Karr, J. C. Sanghvi, D. N. Macklin, M. V. Gutschow, J. M. Jacobs, B. Bolival, N. Assad-Garcia, J. I. Glass, and M. W. Covert, *A Whole-Cell Computational Model Predicts Phenotype from Genotype*, Cell, 150 (2012), pp. 389–401.

[75] R. A. Kerr, T. M. Bartol, B. Kaminsky, M. Dittrich, J.-C. J. Chang, S. B. Baden, T. J. Sejnowski, and J. R. Stiles, *Fast Monte Carlo simulation methods for biological reaction-diffusion systems in solution and on surfaces*, SIAM J. Sci. Comput., 30 (2008), pp. 3126–3149.

[76] E. Kieri, *Accuracy aspects of the reaction-diffusion master equation on unstructured meshes*, Master's thesis, Department of Information Technology, Uppsala University, 2010.

[77] A. Kolmogorov, *Logical basis for information theory and probability theory*, IEEE Transactions on Information Theory, IT-14 (1968), pp. 662–664.

[78] J. Krishnan and P. Iglesias, *Uncovering directional sensing: where are we headed?*, Systems Biology, 1 (2004), pp. 54–61.

[79] S. Lampoudi, D. Gillespie, and L. Petzold, *The multinomial simulation algorithm for discrete stochastic simulation of reaction-diffusion systems*, The Journal of Chemical Physics, 130 (2009), p. 094104.

[80] M. J. Lawson, B. Drawert, M. Khammash, L. Petzold, and T.-M. Yi, *Spatial stochastic dynamics enable robust cell polarization*, PLoS Computational Biology, (2012 *submitted*).

[81] A. T. Layton, N. S. Savage, A. S. Howell, S. Y. Carroll, D. G. Drubin, and D. J. Lew, *Modeling vesicle traffic reveals unexpected consequences for Cdc42p-mediated polarity establishment*, Current Biology, 21 (2011), pp. 184 – 194.

[82] H. Li and L. Petzold, *Efficient parallelization of stochastic simulation algorithm for chemically reacting systems on the graphics processing unit*, International Journal of High Performance Computing, 00 (2009).

[83] H. Liu and A. Bretscher, *Characterization of TPM1 disrupted yeast cells indicates an involvement of tropomyosin in directed vesicular transport*, J. Cell Biol., 118 (1992), pp. 285–299.

[84] K. Madden and M. Snyder, *Cell polarity and morphogenesis in budding yeast*, Annu. Rev. Microbiol., 52 (1998), pp. 687–744.

[85] N. Maheshri and E. O'Shea, *Living with noisy genes: How cells function reliably with inherent variability in gene expression*, Annu. Rev. Biophys. Biomolec. Struct., 36 (2007), pp. 413–434.

[86] R. Mallik and S. P. Gross, *Molecular motors: Strategies to get along*, Curr. Biol., 14 (2004), pp. 971–982.

[87] E. Marco, R. Wedlich-Soldner, R. Li, S. Altschuler, and L. Wu, *Endocytosis optimizes the dynamic localization of membrane proteins that regulate cortical polarity*, Cell, 129 (2007), pp. 411 – 422.

[88] T. T. Marquez-Lago and K. Burrage, *Binomial tau-leap spatial stochastic simulation algorithm for applications in chemical kinetics*, The Journal of Chemical Physics, 127 (2007), p. 104101.

[89] K. C. Martin and R. S. Zukin, *Rna trafficking and local protein synthesis in dendrites: An overview*, J. Neurosci., 26 (2006), pp. 7131–7134.

[90] H. McAdams and A. Arkin, *Stochastic mechanisms in gene expression*, Proc. National Academy Sciences USA, 94 (1997), pp. 814–819.

[91] ——, *It's a noisy business! Genetic regulation at the nanomolar scale*, Trends Genetics, 15 (1999), pp. 65–69.

[92] S. McNamara, K. Burrage, and R. Sidje, *Application of the Strang splitting to the chemical master equation for simulating biochemical kinetics*, International Journal of Computational Science, 2 (2008), pp. 402–421.

[93] H. Meinhardt, *Models of biological pattern formation*, Academic Press, London and New York, 1982.

[94] ——, *Orientation of chemotactic cells and growth cones: models and mechanisms*, J Cell Sci, 112 (1999), pp. 2867–2874.

[95] A. Mogilner, J. Allard, and R. Wollman, *Cell polarity: Quantitative modeling as a tool in cell biology*, Science, 336 (2012), pp. 175–179.

[96] C. Moler and C. V. Loan, *Nineteen dubious ways to compute the exponential of a matrix*, SIAM Review, 20 (1978), pp. 801–836.

[97] T. Moore, C.-S. Chou, Q. Nie, N. Jeon, and T.-M. Yi, *Robust spatial sensing of mating pheromone gradients by yeast cells*, PLoS One, 3 (2008), p. e3865.

[98] B. Munsky and M. Khammash, *The finite state projection algorithm for the solution of the chemical master equation*, The Journal of Chemical Physics, 124 (2006), p. 044104.

[99] B. Munsky and M. Khammash, *The finite state projection algorithm for the solution of the chemical master equation*, J. Chem. Phys., 124 (2006), p. 044104.

[100] B. Munsky and M. Khammash, *A multiple time interval finite state projection algorithm for the solution to the chemical master equation*, Journal of Computational Physics, 226 (2007), pp. 818 – 835.

[101] Y. Nakayama, J. Shivas, D. Poole, J. Squirrell, J. Kulkoski, J. Schleede, and A. Skop, *Dynamin participates in the maintenance of anterior polarity in the caenorhabditis elegans embryo*, Developmental Cell, 16 (2009), pp. 889 – 900.

[102] P.-O. Östberg, A. Hellander, B. Drawert, E. Elmroth, S. Holmgren, and L. Petzold, *Abstractions For Scaling eScience Applications to Distributed Computing Environments*, in BIOINFORMATICS, 2012.

[103] P.-O. Östberg, A. Hellander, B. Drawert, E. Elmroth, S. Holmgren, and L. Petzold, *Reducing complexity in management of scientific computations*, in Proceedings of CCGrid 2012 - The 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 2012.

[104] E. M. Ozbudak, A. Becskei, and A. van Oudenaarden, *A system of counteracting feedback loops regulates Cdc42p activity during spontaneous cell polarization*, Developmental Cell, 9 (2005), pp. 565 – 571.

[105] S. Paliwal, P. Iglesias, K. Campbell, Z. Hilioti, A. Groisman, and A. Levchenko, *MAPK-mediated bimodal gene expression and adaptive gradient sensing in yeast*, Nature, 446 (2007), pp. 46–51.

[106] J. Paulsson, O. G. Berg, and M. Ehrenberg, *Stochastic focusing: Fluctuation-enhanced sensitivity of intracellular regulation*, Proc Natl Acad Sci USA, 97 (2000), pp. 7148–7153.

[107] S. Peleš, B. Munsky, and M. Khammash, *Reduction and solution of the chemical master equation using time scale separation and finite state projection*, The Journal of Chemical Physics, 125 (2006), p. 204104.

[108] D. Pruyne and A. Bretscher, *Polarization of cell growth in yeast: I. Establishment and maintenance of polarity states*, J. Cell Sci., 113 (2000), pp. 365–375.

[109] ——, *Polarization of cell growth in yeast: II. The role of the cortical actin cytoskeleton.*, J. Cell Sci., 113 (2000), pp. 571–585.

[110] A. Raj, P. van den Bogaard, S. A. Rifkin, and A. van Oudenaarden, *Imaging individual mRNA molecules using multiple singly labeled probes*, Nat. Meth., 5 (2008), pp. 877–879.

[111] C. Rao, D. Wolf, and A. Arkin, *Control, exploitation, and tolerance of intracellular noise*, Nature, 420 (2002), pp. 231–237.

[112] J. M. Raser and E. K. O'Shea, *Noise in gene expression: Origins, consequences and control*, Science, 309 (2005), pp. 2010–2013.

[113] J. Rodriguez, J. Kaandorp, M. Dobrzynski, and J. Blom, *Spatial stochastic modelling of the phosphoenolpyruvate-dependent phosphotransferase (PTS) pathway in Escherichia Coli*, Bioinformatics, 22 (2006), pp. 1895–1901.

[114] D. Rossinelli, B. Bayati, and P. Koumoutsakos, *Accelerated stochastic and hybrid method for spatial simulations of reaction-diffusion systems*, Chem. Phys. Lett., 451 (2008), pp. 136–140.

[115] N. S. Savage, A. T. Layton, and D. J. Lew, *Mechanistic mathematical model of polarity in yeast*, Molecular Biology of the Cell, 23 (2012), pp. 1998–2013.

[116] M. A. Schlager and C. C. Hoogenraad, *Basic mechanisms for recognition and transport of synaptic cargos*, Molecular Brain, 2 (2009).

[117] J. L. Shih, S. L. Reck-Peterson, R. Newitt, M. S. Mooseker, R. Aebersold, and I. Herskowitz, *Cell polarity protein Spa2p associates with proteins involved in actin function in Saccharomyces cerevisiae*, Mol. Biol. Cell, 16 (2005), pp. 4595–4608.

[118] N. Shnerb, Y. Louzoun, E. Bettelheim, and S. Solomon, *The importance of being discrete: Life always wins on the surface*, Proceedings of the National Academy of Sciences, 97 (2000), pp. 10322–10324.

[119] A. Slepoy, A. Thompson, and S. Plimpton, *A constant-time kinetic Monte Carlo algorithm for simulation of large biochemical reaction networks*, The Journal of Chemical Physics, 128 (2008), p. 205101.

[120] B. L. Sprague, R. L. Pego, D. A. Stavreva, and J. G. McNally, *Analysis of binding reactions by fluorescence recovery after photobleaching*, Biophysical Journal, 86 (2004), pp. 3473 – 3495.

[121] G. Strang, *On the construction and comparison of difference schemes*, SIAM Journal on Numerical Analysis, 5 (1968), pp. 506–517.

[122] A. B. Stundzia and C. L. Lumsden, *Stochastic simulation of coupled reaction-diffusion processes*, J. Comput. Phys., 127 (1996), pp. 196–207.

[123] P. S. Swain, M. B. Elowitz, and E. D. Siggia, *Intrinsic and extrinsic contributions to stochasticity in gene expression*, Proc. Natl. Acad. Sci. USA., 99 (2002), pp. 12795–12800.

[124] K. Takahashi, S. Tănase-Nicola, and P. R. ten Wolde, *Spatio-temporal correlations can drastically change the response of a MAPK pathway*, Proc. Natl. Acad. Sci. USA., 107 (2010), pp. 2473–2478.

[125] H. Tanaka and T.-M. Yi, *Synthetic morphology using alternative inputs*, PLoS ONE, 4 (2009), p. e6946.

[126] M. Thattai and A. van Oudenaarden, *Intrinsic noise in gene regulatory networks*, Proc. Natl. Acad. Sci. USA., 98 (2001), pp. 8614–8619.

[127] J. Valdez-Taubas and H. R. Pelham, *Slow diffusion of proteins in the yeast plasma membrane allows polarity to be maintained by endocytic cycling*, Current Biology, 13 (2003), pp. 1636 – 1640.

[128] R. D. Vale, *The molecular motor toolbox for intracellular transport*, Cell, 112 (2003), pp. 467–480.

[129] N. G. van Kampen, *Stochastic Processes in Physics and Chemistry*, Elsevier, third ed., 2007.

[130] J. Vilar, H. Kueh, N. Barkai, and S. Leibler, *Mechanisms of noise-resistance in genetic oscillators*, Proc. National Academy Sciences USA, 99 (2002), pp. 5988–5992.

[131] X. Wang, N. Hao, H. G. Dohlman, and T. C. Elston, *Bistability, stochasticity, and oscillations in the mitogen-activated protein kinase cascade*, Biophysical Journal, 90 (2006), pp. 1961 – 1978.

[132] R. Wedlich-Soldner, S. Altschuler, L. Wu, and R. Li, *Spontaneous cell polarization through actomyosin-based delivery of the Cdc42 GTPase*, Science, 299 (2003), pp. 1231–1235.

[133] R. Wedlich-Soldner and R. Li, *Spontaneous cell polarization: undermining determinism*, Nat Cell Biol, 5 (2003), pp. 267–270.

[134] L. S. Weinberger, J. C. Burnett, J. E. Toettcher, A. P. Arkin, and D. V. Schaffer, *Stochastic gene expression in a lentiviral positive-feedback loop: HIV-1 Tat fluctuations drive phenotypic diversity*, Cell, 122 (2005), pp. 169 – 182.

[135] M. A. Welte, *Bidirectional transport along microtubules*, Curr. Biol., 14 (2004).

[136] T.-M. Yi, Y. Huang, M. I. Simon, and J. Doyle, *Robust perfect adaptation in bacterial chemotaxis through integral feedback control*, Proc Natl Acad Sci USA, 97 (2000), pp. 4649–4653.

[137] H. Yu and R. Wedlich-Soldner, *Cortical actin dynamics: Generating randomness by formin(g) and moving*, BioArchitecture, (2011), pp. 165–168.

[138] J. Yu, A. Crevenna, M. Bettenbuhl, T. Freisinger, and R. Wedlich-Soldner, *Cortical actin dynamics driven by formins and myosin V*, J. Cell Sci., 124 (2011), pp. 1533–1541.

[139] Z. Zheng, R. Stephens, R. Braatz, R. Alkire, and L. Petzold, *A hybrid multiscale kinetic Monte Carlo method for simulation of copper electrodeposition*, The Journal of Computational Physics, 227 (2008), pp. 5184–5199.

[140] J. S. van Zon and P. Rein ten Wolde, *Simulating biochemical networks at the particle level and in time and space: Green's function reaction dynamics*, Phys. Rev. Lett., 94 (2005), p. 128103.