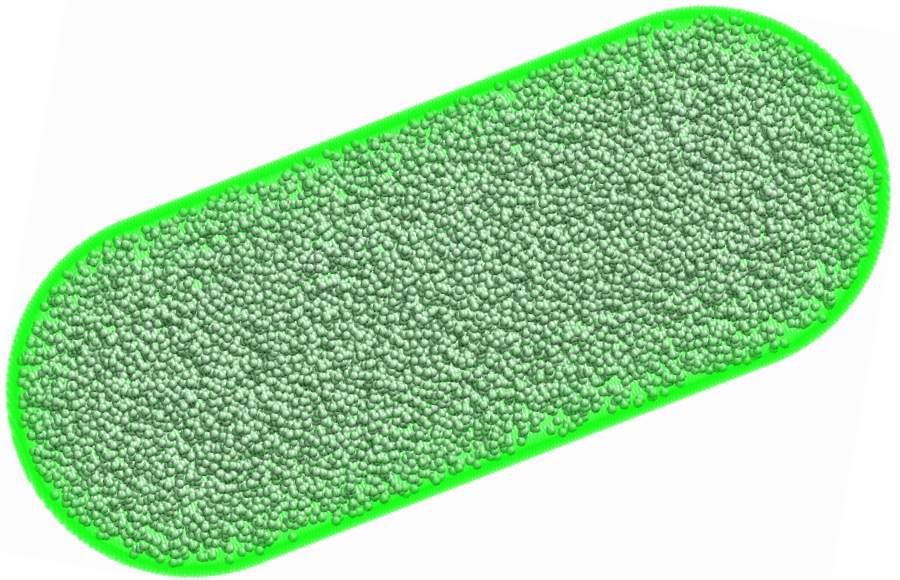
CUDA algorithms for stochastic simulation of biochemical reactions

Andrew Magis

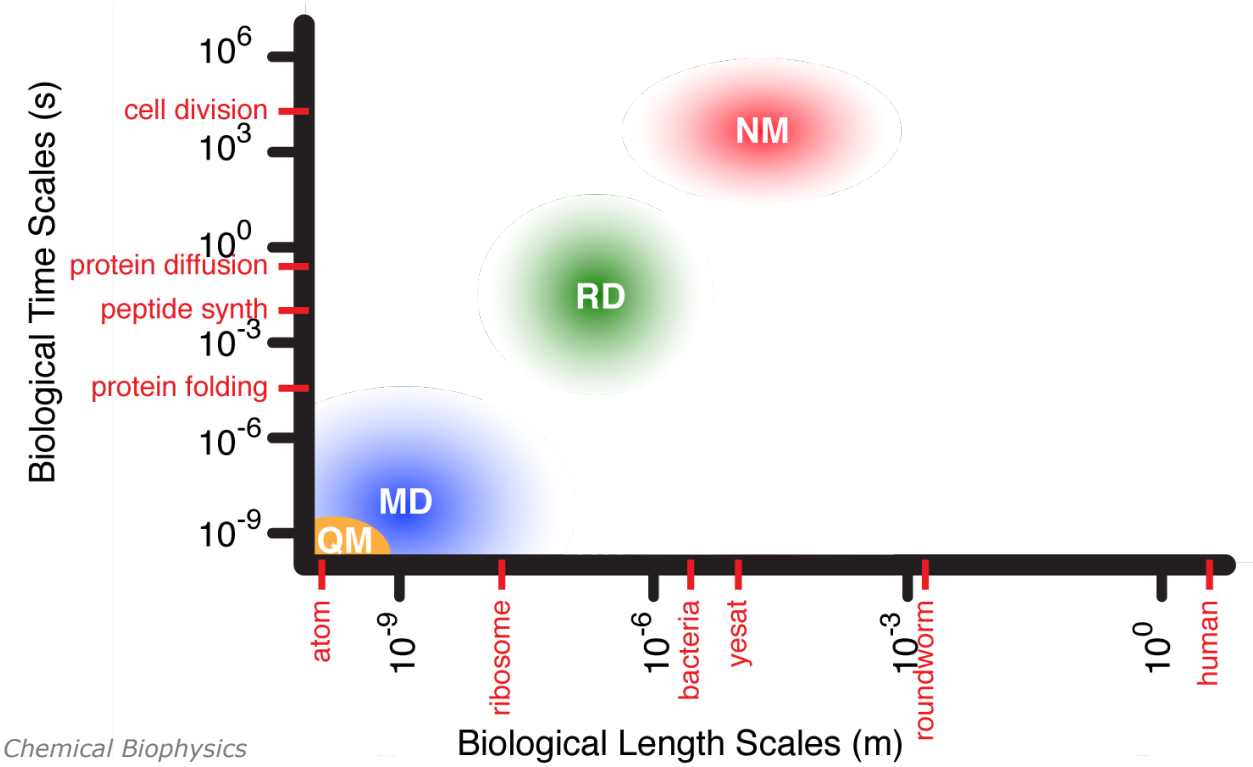
Workshop on GPU Programming for Molecular Modeling

August 7th, 2010



What is computational biology?

- "The use of mathematical and computational tools to simulate and analyze the biophysical processes underlying biological phenomena."[1]
- One long-term goal of computational biology is a dynamic model of an entire cell under natural conditions.
- Such a model would allow detailed studies of genetic regulation, cell differentiation, signaling, effects of drugs on cellular function, etc.
- The time and length scales involved make such a model currently infeasible with a single simulation methodology.



1) Beard & Qian (2008) *Chemical Biophysics*

Overview

- Part 1: “Well mixed” stochastic simulation
- Part 2: Spatially resolved stochastic simulation
- Part 3: Simulation example: the *E. coli. lac* operon.

Analysis of biochemical reactions

Law of mass action:

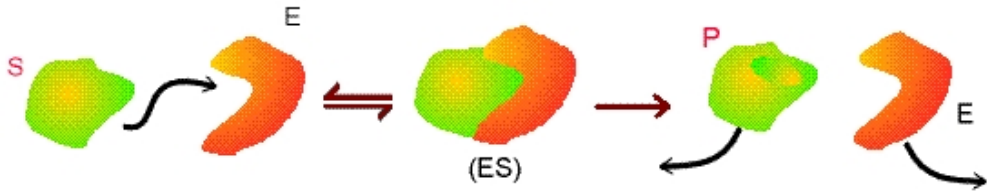
Reaction Format

$rate = k[A]$ First Order $A \rightarrow B$

$rate = k[A][B]$ Second Order $A + B \rightarrow C + D$

[A] = Concentration of A

Michaelis-Menten system:



$$\left.\begin{aligned} E + S &\xrightarrow{k_1} ES \\ ES &\xrightarrow{k_2} E + S \\ ES &\xrightarrow{k_3} E + P \end{aligned}\right\} \quad \frac{d[S]}{dt} = k_2[ES] - k_1[E][S]$$

Analysis of biochemical reactions

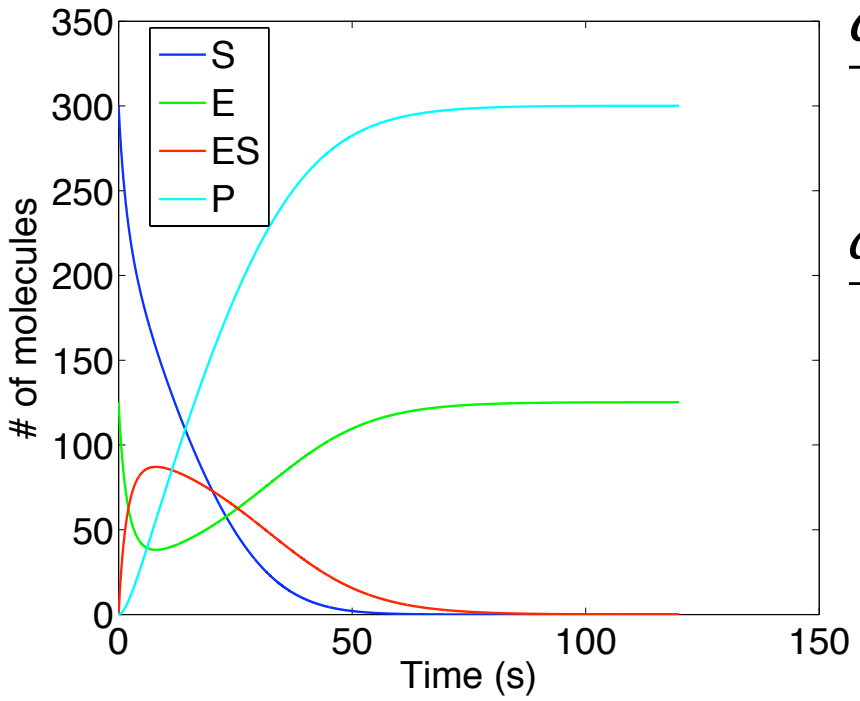
$$E + S \xrightarrow{k_1} ES$$
$$ES \xrightarrow{k_2} E + S$$
$$ES \xrightarrow{k_3} E + P$$

$$\frac{d[S]}{dt} = k_2[ES] - k_1[E][S]$$

$$\frac{d[E]}{dt} = k_2[ES] + k_3[ES] - k_1[E][S]$$

$$\frac{d[ES]}{dt} = k_1[E][S] - k_2[ES] - k_3[ES]$$

$$\frac{d[P]}{dt} = k_3[ES]$$



This is a sample system, whereas typical biochemical systems may consist of hundreds or thousands of such reactions.

Biology is not deterministic!

(top) fingerprints of identical twins

(middle) cloned kitten and her 'mother'

(bottom) genetically identical *E. coli* cells responding differently to an identical external environment.

1

Not all molecules in the cell exist in high copy numbers. Many transcription factors exist in only a few copies in each cell.

1

2

1) Raser & O'Shea (1995) *Science* 2) Choi *et al* (2008) *Science*

Deterministic vs. Stochastic Simulation

- Deterministic models
  - Every future state can be predicted from a set of initial conditions, rate constants, and differential equations.
  - All states can be determined once initial conditions are known; hence *deterministic.*
- Stochastic models
  - Elements of randomness are introduced.
  - Fluctuations in the system can be measured.
  - The low copy number of certain biological molecules (transcription factors) result in stochastic behavior.

So how do we model a system stochastically?

The Chemical Master Equation[1]

P(x,t) represents the probability that a system will be in a state *x* at time *t*, where

$$P(x,t) = P\left[X_1(t) = x_1, X_2(t) = x_2, .., X_n(t) = x_n\right]$$

The time evolution of *p*(*x,t*) is described by the chemical master equation:

$$\frac{dP_n(t)}{dt} = \sum_{j=1}^{M} [a_j(n - D_j)P_{n-D_j}(t) - a_j(n)P_n(t)]$$

Where $a_j(n)$ is the *propensity* of reaction R_j, defined such that $a_j(n)\,dt$ is the probability that the reaction will occur in the time interval [*t, t+dt*] given that *X(t) = n*. *D* represents the vector of step changes that will occur for each chemical species after completion of reaction R_j. *M* refers to the number of reactions.

1) McQuarry, D. A (1967) *J. Appl. Prob.*

See Gillespie, D.T. (1992) *Physica A* for a rigorous derivation of the chemical master equation and it's relation to the SSA.

Gillespie's Stochastic Simulation Algorithm (SSA) [1]

- Time evolution of system is not a continuous process.
- Molecule concentrations can only change by discrete amounts.
- Chemical reactions occur when two species collide.
- Probabilities of collision can be calculated for any number of molecules in a volume.
- Algorithm assumes a 'well mixed' environment: all reactions are equally probable in any region of the system.

1) Gillespie, D.T. (1977) *J. Phys. Chem*

Gillespie's SSA

The Gillespie algorithm provides a stochastic solution to the chemical master equation, which is frequently impossible to solve analytically. Inputs to the algorithm include a set of rates for each of *M* reactions:

$$k = [k_1, k_2, \ldots, k_M]$$

an initial state describing the initial numbers of *N* molecule types:

$$X = [X_1, X_2, \ldots, X_N]$$

and an *MxN* stoichiometry matrix:

$$\begin{array}{l} \\ E + S \xrightarrow{k_1} ES \\ ES \xrightarrow{k_2} E + S \\ ES \xrightarrow{k_3} E + P \end{array} \begin{array}{c} \begin{array}{cccc} E & S & ES & P \end{array} \\ \begin{bmatrix} -1 & -1 & 1 & 0 \\ 1 & 1 & -1 & 0 \\ 1 & 0 & -1 & 1 \end{bmatrix} \end{array}$$

Stochastic Rate Constants and Reaction Propensities:

Each deterministic rate constant k_i is translated into a stochastic rate constant c_i:

Zeroth Order: $c_i = k_i \cdot N_A \cdot V$

where N_A = Avogadro's Number
and V = system volume

First Order: $c_i = k_i$

Second Order: $c_i = \frac{k_i}{N_A \cdot V}$

The propensity a_i is calculated for each reaction, given the current state:

Zeroth Order: $a_i = c_i$

First Order: $a_i = c_i \cdot X$

Second Order: $a_i = c_i \cdot X \cdot Y$

Propensity for reaction

$$E + S \xrightarrow{k_1} ES$$

$$a = \frac{k_1}{N_A \cdot V} \cdot E \cdot S$$

Gillespie's SSA

The total reaction propensity for a given state is calculated:

$$a^* = \sum_{i=1}^{M} a_i$$

and the time to next reaction τ is calculated using random number r_1 *[0:1)*:

$$\tau = -\frac{1}{a^*} \ln r_1$$

The next reaction to occur is chosen based on reaction propensities and random number r_2 [0:1) such that:

$$0 \leq r_2 a^* < a^*$$

The state vector is updated based on the stoichiometry matrix and the time is updated. This is repeated while $t < t_{max}$.

$$E + S \xrightarrow{k_1} ES$$

$$a_1 = \frac{k_1}{N_A \cdot V} \cdot E \cdot S$$

$$ES \xrightarrow{k_2} E + S$$

$$a_2 = k_2 \cdot ES$$

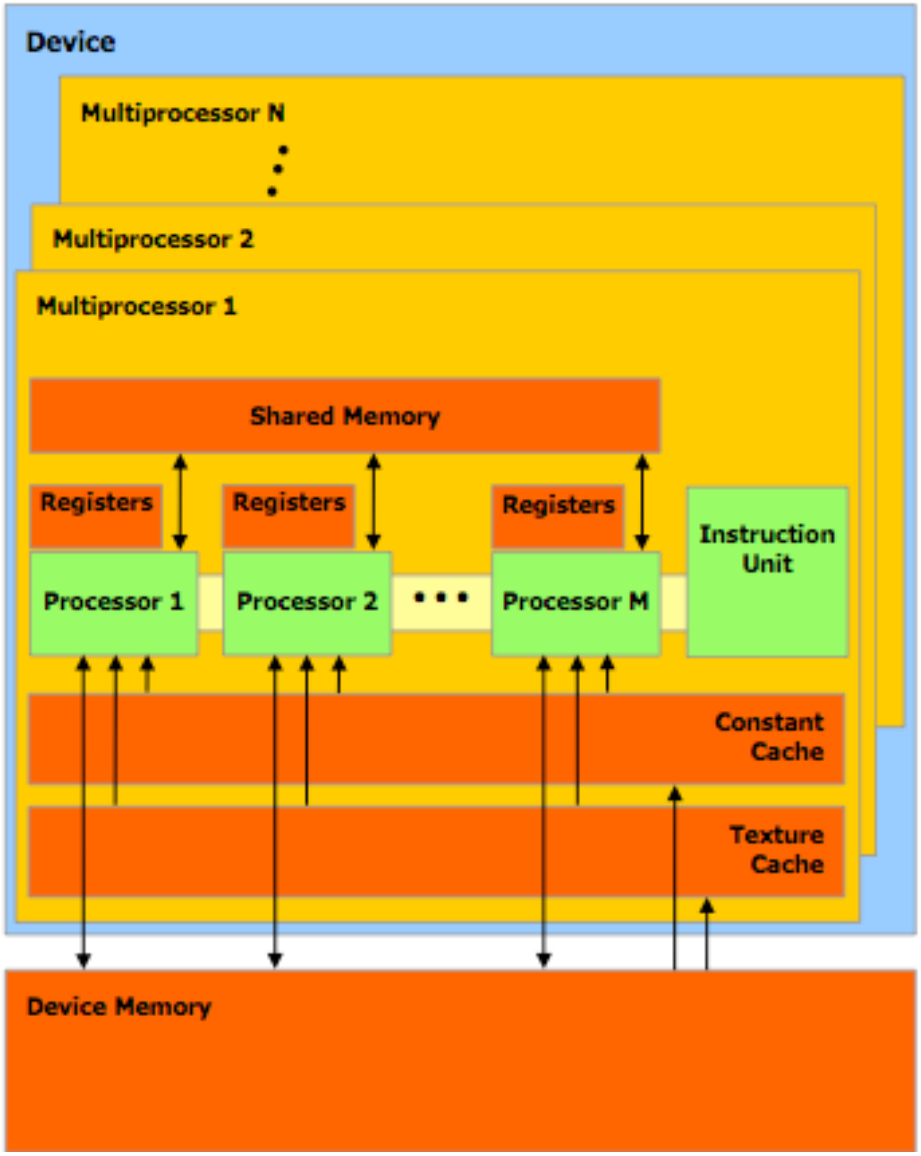
$$ES \xrightarrow{k_3} E + P$$

$$a_3 = k_3 \cdot ES$$

$$a^* = a_1 + a_2 + a_3$$

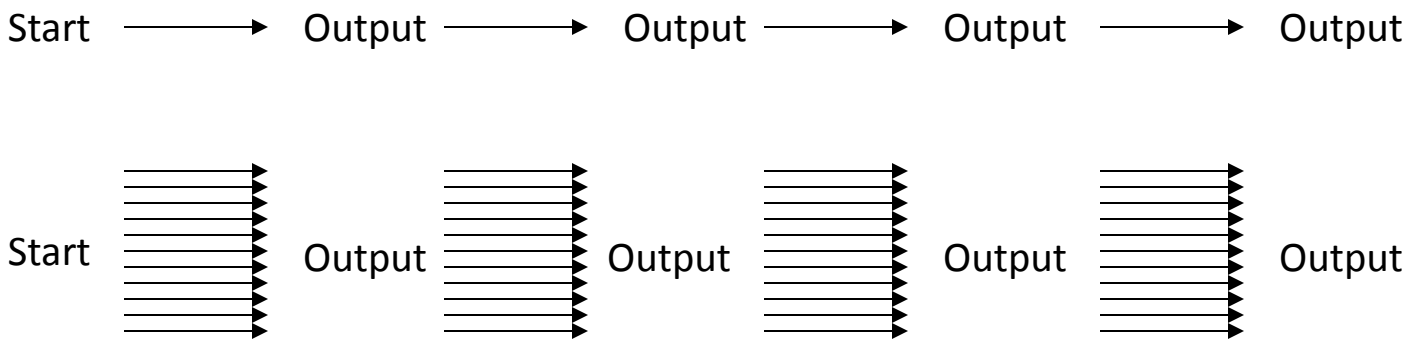
GPU Memory Model

- **Device Memory:**
  - Large (max 4 GB in Tesla, more in Fermi)
  - Slow (400 clock cycles for access)
  - Coalescing a concern
- **Shared Memory:**
  - Small (16 KB Tesla / 64 KB Fermi)
  - Fast (1 clock cycle)
  - Bank conflicts a concern (not so much in Fermi)
  - Shared across threads of the same block

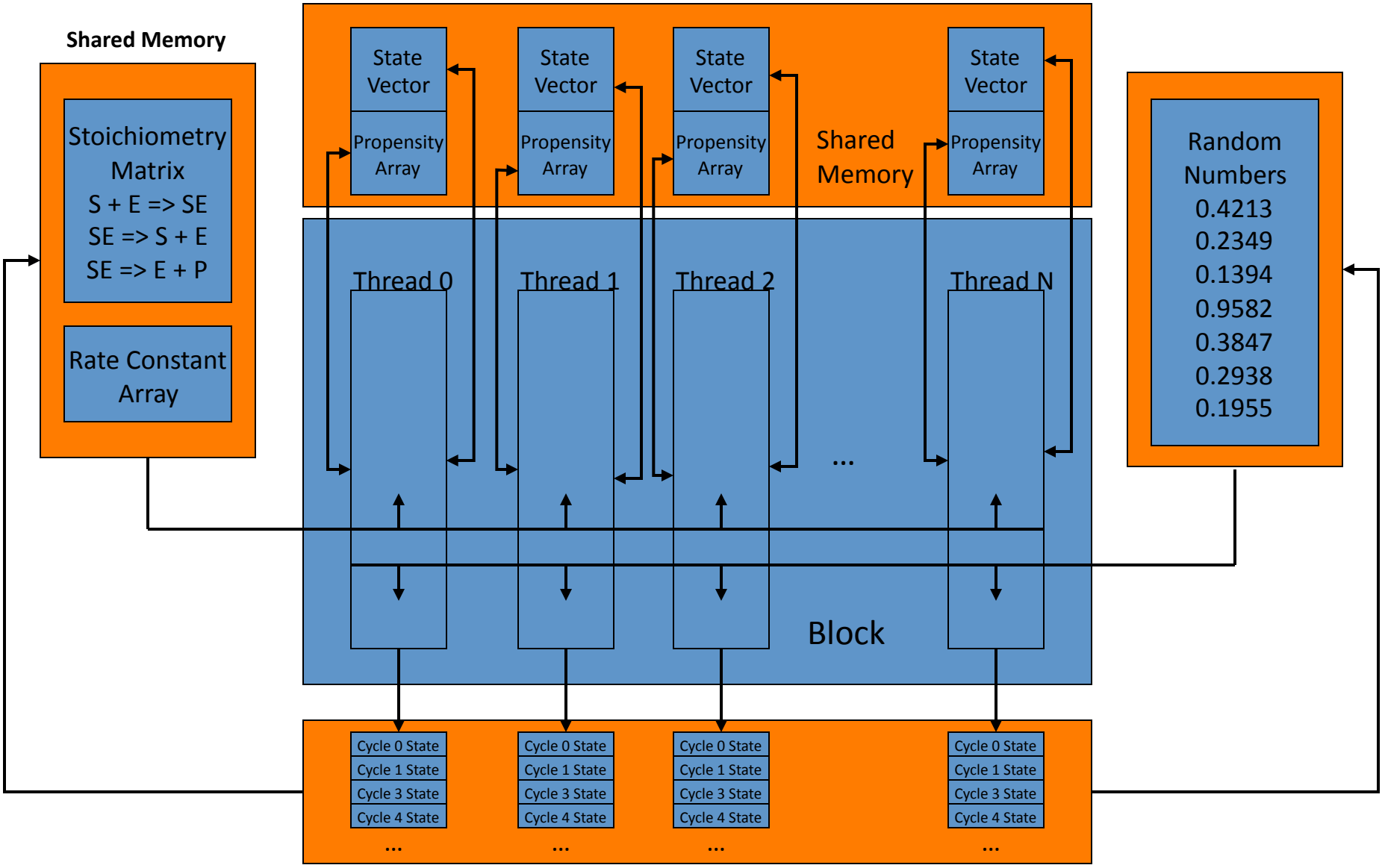


GPU Implementation

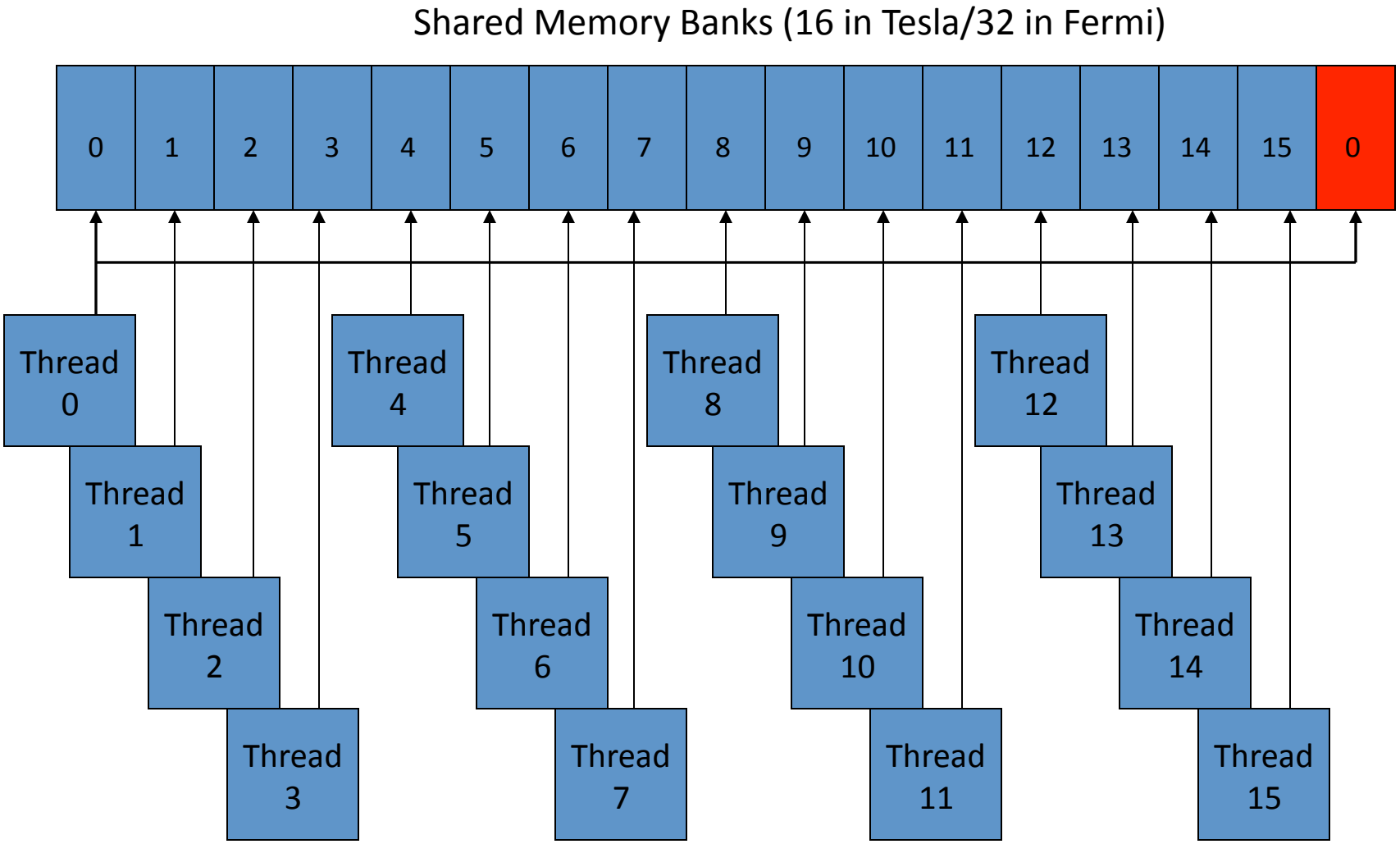
- Parallelization within algorithm not obvious.
- Individual Gillespie runs do not need to share data.
- Threads can share the same S-matrix and rate constant matrix within a block
- Multiple Gillespie runs can be executed simultaneously, one per thread.



Implementation



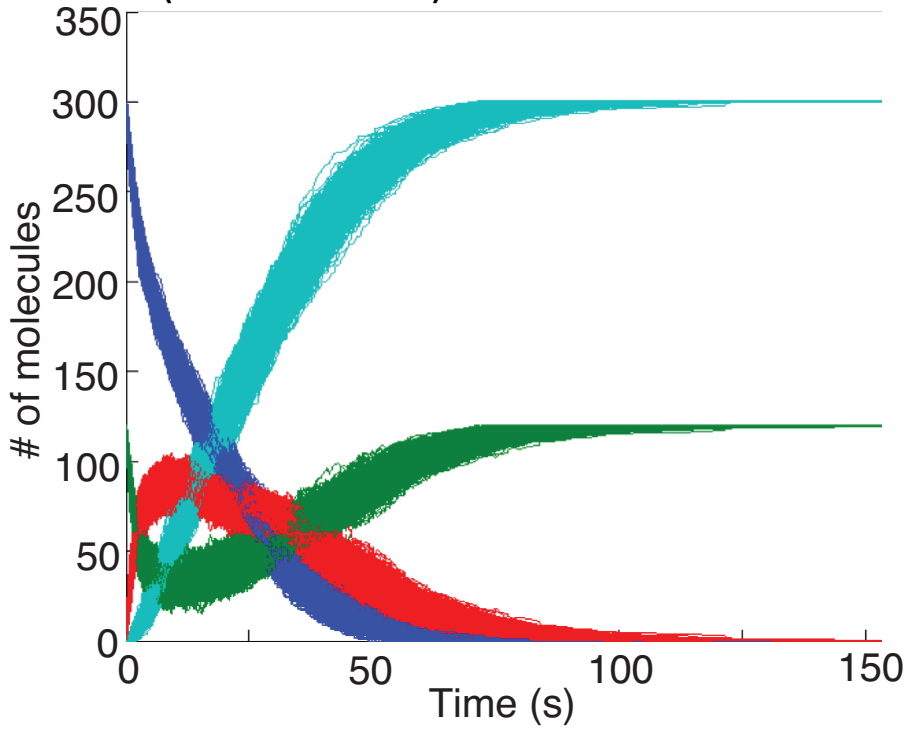
Avoiding Bank Conflicts
State vector storage per thread



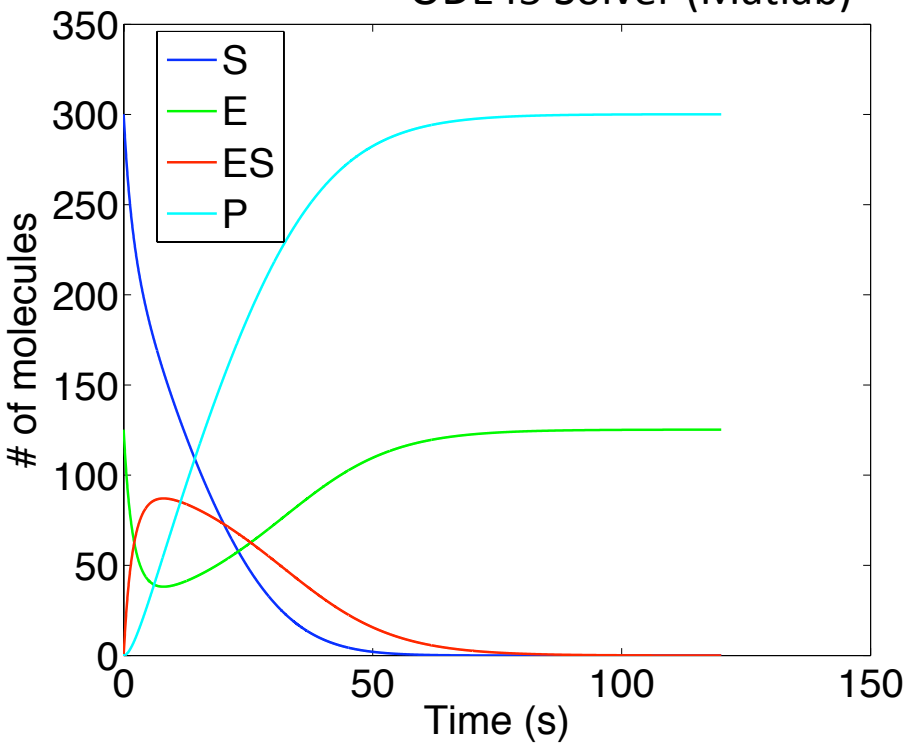
Stochastic *vs*. Deterministic Solutions to the Michaelis-Menten System

$$E + S \xrightarrow{k_1} ES$$
$$ES \xrightarrow{k_2} E + S$$
$$ES \xrightarrow{k_3} E + P$$

Gillespie's SSA on GPU
(2000 threads)



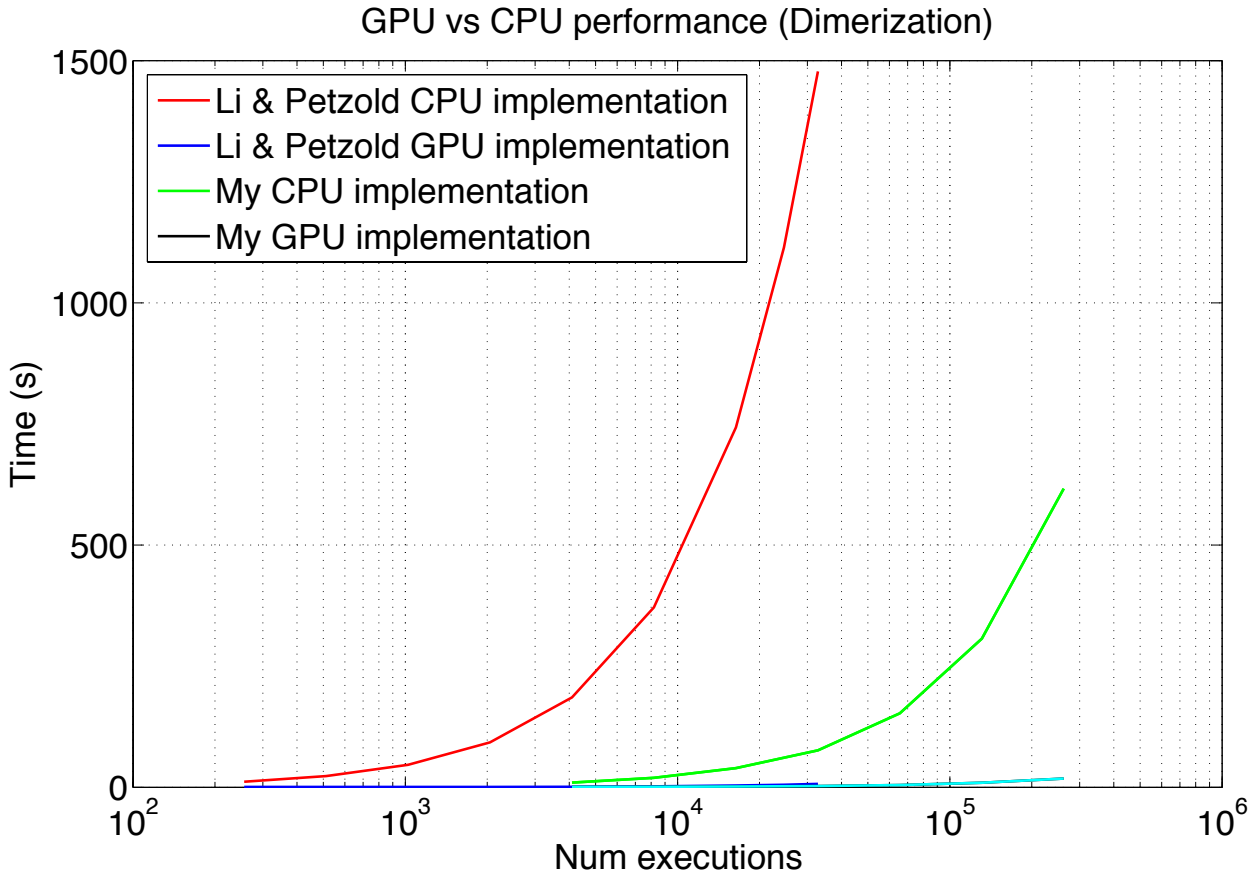
ODE45 Solver (Matlab)



GPU vs CPU performance

My implementation of the algorithm achieves a maximum 35X speedup over a CPU implementation on a simple molecule dimerization.

Li and Petzold[1] present a 200X speedup in their recent 2009 publication, although the differences may be more in the speed of the respective CPU implementations than in the GPU implementations.



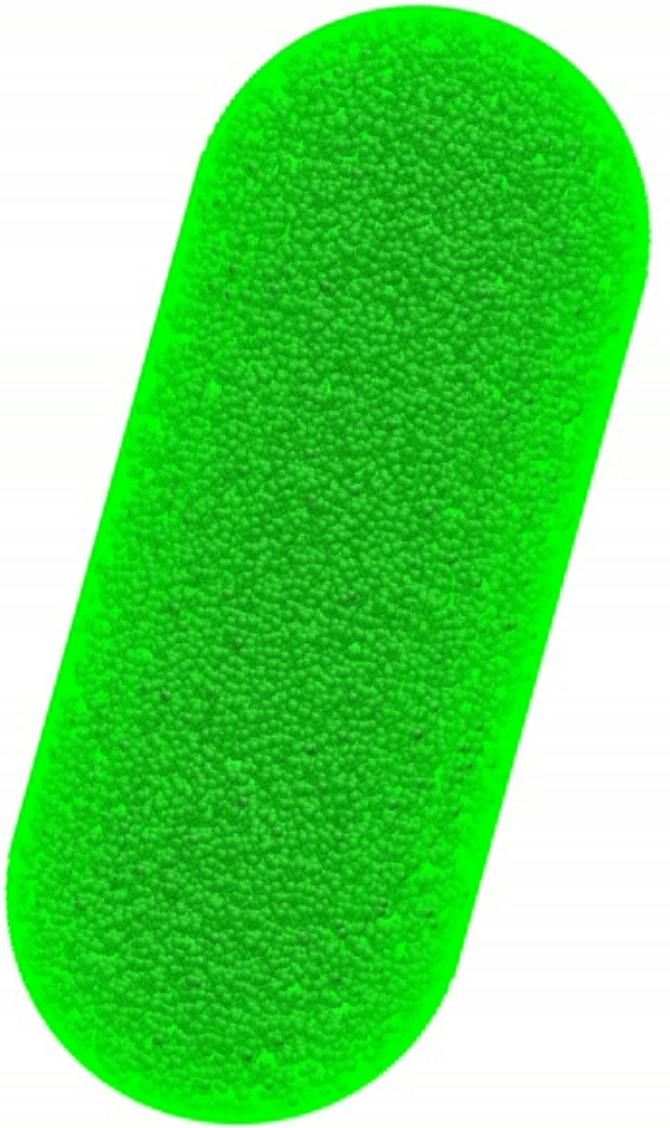
1) Li & Petzold (2009) *International Journal of High Performance Computing Applications*

Overview

- Part 1: “Well mixed” stochastic simulation
- Part 2: Spatially resolved stochastic simulation
- Part 3: Simulation example: the *E. coli. lac* operon.

The *in vivo* environment: The cell is not well-mixed!

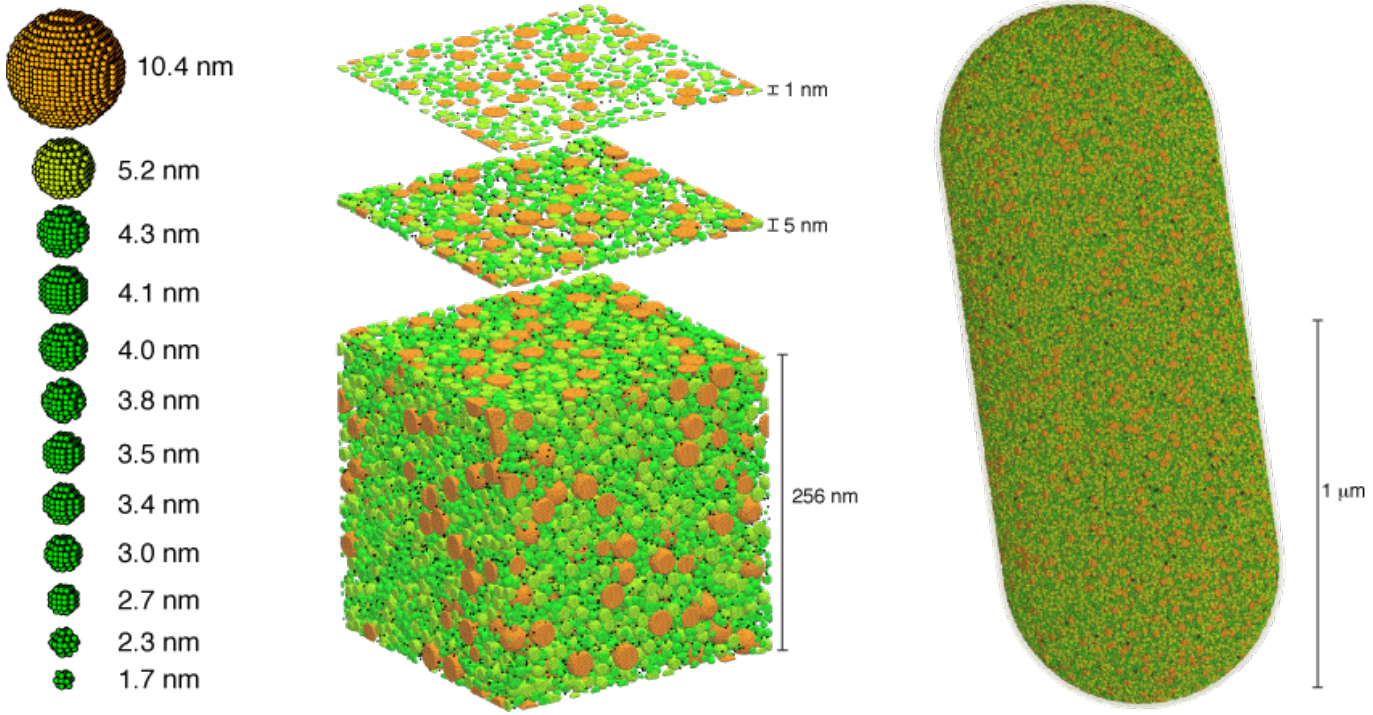
- The cellular environment is crowded!
- *Escherichia coli* is a model organism for many biological studies[1]:
  - 2 um long, 0.8 um diameter
  - 18,000 ribosomes (right)
  - 2 million proteins
  - 18 million small organic molecules
  - ~2 chromosomes 4-5 million base pairs (1.5 mm each, when fully extended)
  - 30-50% of the cellular volume is occupied by molecules other than water



1) Sundararaj *et al.* (2004) *Nucl Acids Res* 32:D293; CyberCell Database (http://redpoll.pharmacy.ualberta.ca/CCDB)

In vivo diffusion on a lattice

- To approximate an *in vivo* environment, we fill 50% of the volume with obstacles of varying sizes.
- We use the size classes and populations defined by Ellison and colleagues from analysis of proteomic data regarding relative cellular protein populations[1].



1) Ridgway et al., *Biophys J* (2008)

Traditional lattice diffusion

- Traditional lattice-based diffusion methods[1] invoke a particle randomly moving from site to site on a lattice (L) with spacing λ.
- Each lattice site is located on the lattice at position

$$\vec{r} = a\lambda\hat{i} + b\lambda\hat{j} + c\lambda\hat{k}.$$

- Particles move from site to in discrete jumps in a single dimension at each timestep ($\pm\lambda\hat{i}, \pm\lambda\hat{j}, or \pm\lambda\hat{k}$).

1) Karapiperis & Blankleider (1994) *Physica D*

3D diffusion coefficient

- It can be shown that, if the probability of moving in any direction is *p* and the probability of staying at the lattice site is p_0 ($p_0 + 2d \cdot p = 1$, where *d* is the dimensionality), then the model obeys the standard diffusion equation,

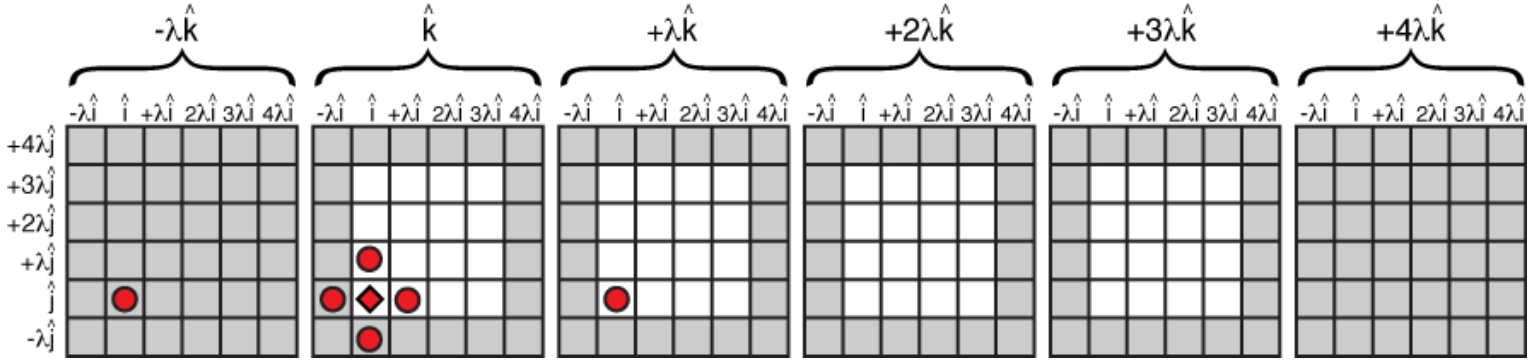
$$\frac{\partial}{\partial t} N_\alpha(\vec{r}, t) = D \nabla^2 N_\alpha(\vec{r}, t),$$

where D is the diffusion coefficient,

$$D = \frac{\lambda^2}{2d\tau}(1 - p_0).$$

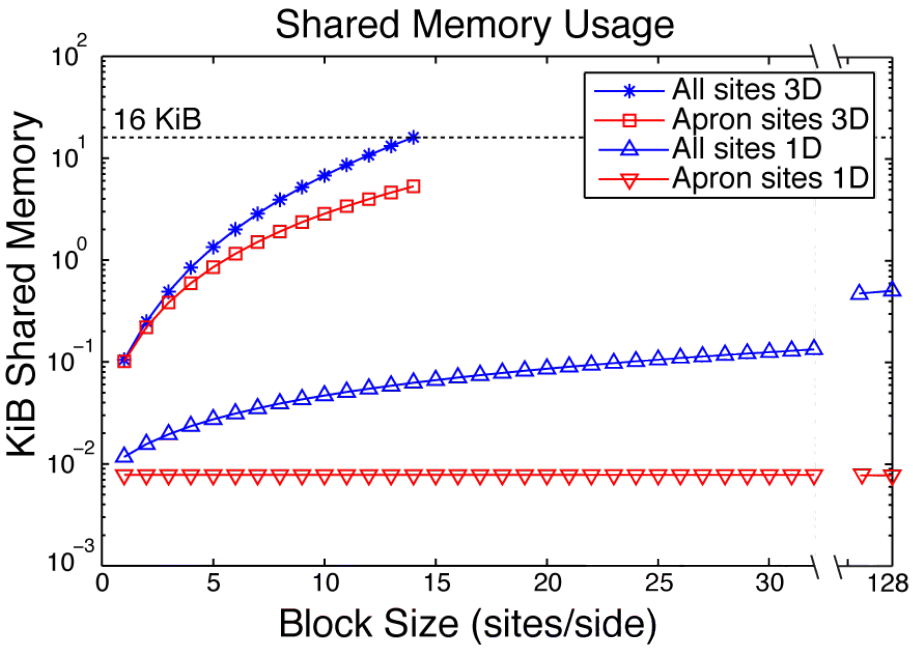
GPU implementation and 3D lattices

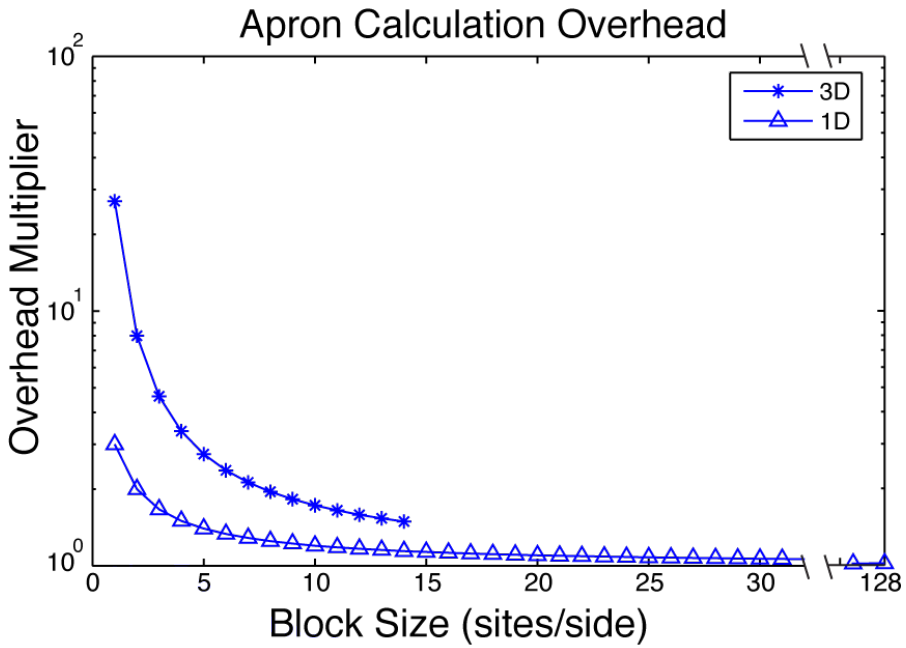
- Basic implementation of this model on a GPU is straightforward: load, choose, move.
- The GPU processes many calculations in parallel, but each calculation is limited in terms of the amount of memory it can access.
- Processing a 3D block of the lattice requires many apron sites.



Shared memory limitations for 3D lattice

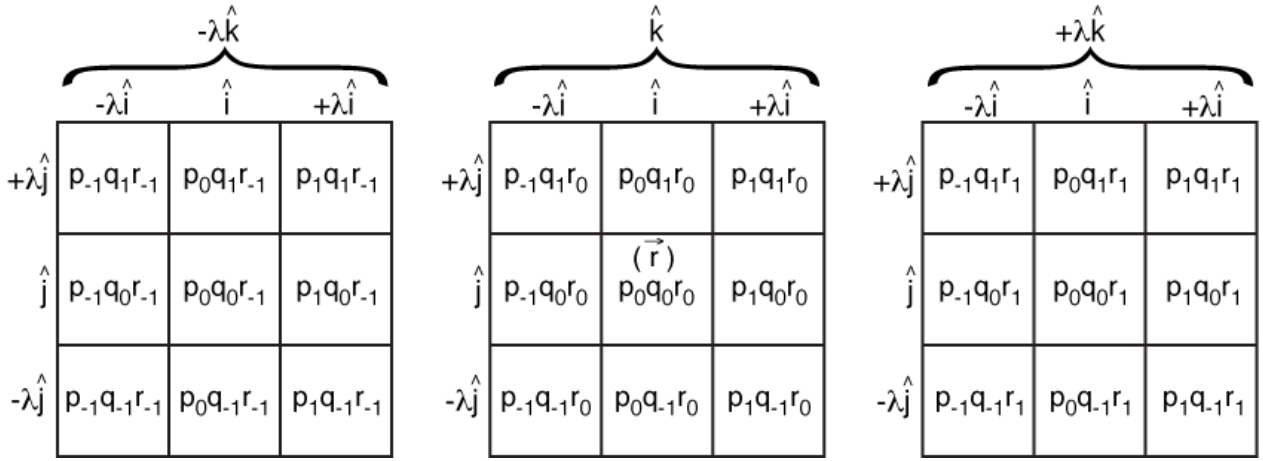
- Apron sites increase both the memory usage and the number of redundant calculation that need to be performed
- At 8 sites per side, half of the 4 KiB needed are for apron sites and half of the calculations are redundant.
- Switching to a 1D model eliminates these issues.





1D multiparticle diffusion model

- In a 1D diffusion model, each particle diffuses independently in each dimension[1].
- At each time step a particle has probabilities of moving in the minus direction, staying at the site, or moving in the plus direction for the x (p_{-1},p_0,p_1), y (q_{-1},q_0,q_1), and z (r_{-1},r_0,r_1) dimensions.
- The net probability of moving to one of the 26 neighboring sites is the the product of the probabilities of the specific x-y-z move.



1) Roberts, Stone, Sepulveda, Hwu, Luthey-Schulten (2009) *The Eighth IEEE International Workshop on High-Performance Computational Biology*

Free diffusion on a lattice

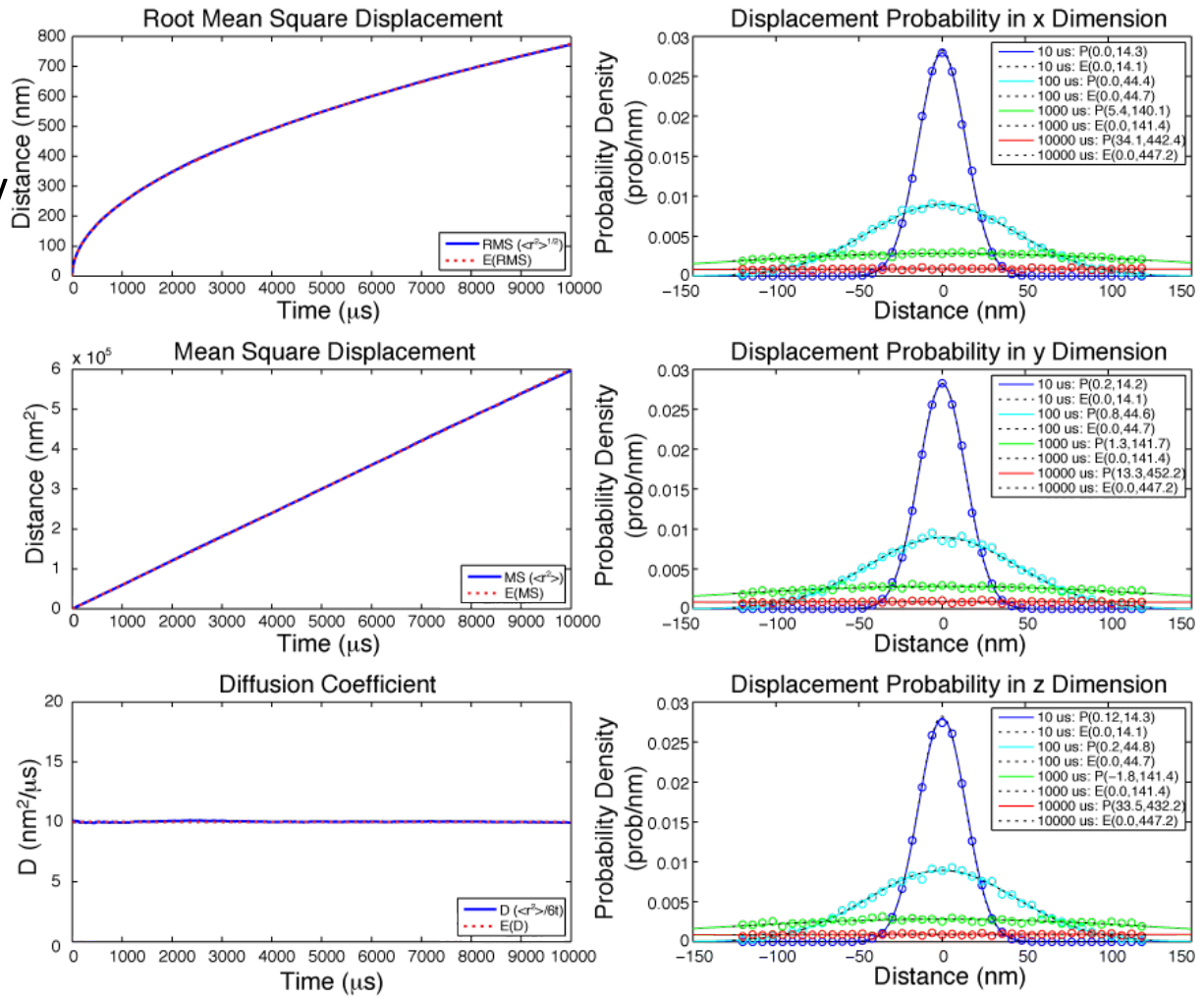
- To check that the GPU implementation agrees with the model, we ran free simulations of particles freely diffusing on a periodic lattice (256x256x256, 2 nm spacing, 10 ns time step, natural D of 200 nm^2/us).
- Particle distribution in each dimension should be Gaussian,

$$\frac{1}{2\sqrt{\pi Dt}} e^{-x^2/4Dt}$$

with mean 0 and variance

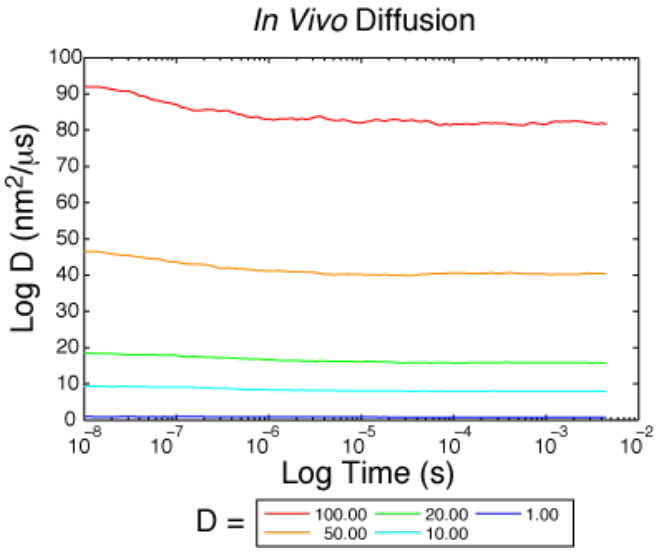
$$\sigma = \sqrt{2Dt}$$

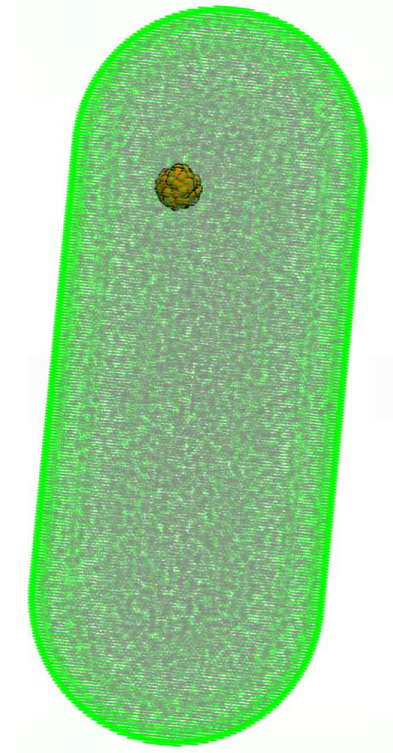
- Excellent agreement.



Long-time, whole-cell simulations

- *In vivo* diffusion results show the correct qualitative behavior, but a higher final D (10%) compared to Brownian dynamics simulations.
- Brownian dynamics must calculate each particle, can only scale *in vivo* simulations to ~100 us.
- Lattice based models sacrifice accuracy to reach very long times, into the seconds on a single GPU.





Lattice Size	Spacing (nm)	Time Step (μs)	Calculation Performance ($\times 10^6$ site updates/sec)		Simulation Performance (sec/GPU·day)		
			FX5600	GTX280	FX5600	GTX280	Speedup
64×64×128	20	8.00	219	533	290	700	2.4X
64×64×128	16	5.12	212	522	180	440	2.4X
128×128×256	10	2.00	310	781	13	32	2.5X
128×128×256	9	1.62	307	747	10	25	2.5X
128×128×256	8	1.28	302	776	8.0	20	2.5X
256×256×512	7	0.94	349	648	0.85	1.6	1.8X
256×256×512	6	0.72	348	647	0.65	1.2	1.8X
256×256×512	5	0.50	347	645	0.45	0.83	1.8X
256×256×512	4	0.32	346	642	0.29	0.52	1.8X

How to measure GPU performance?

- Since our technique is a native GPU algorithm, no optimized CPU version exists by which to measure its performance..

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

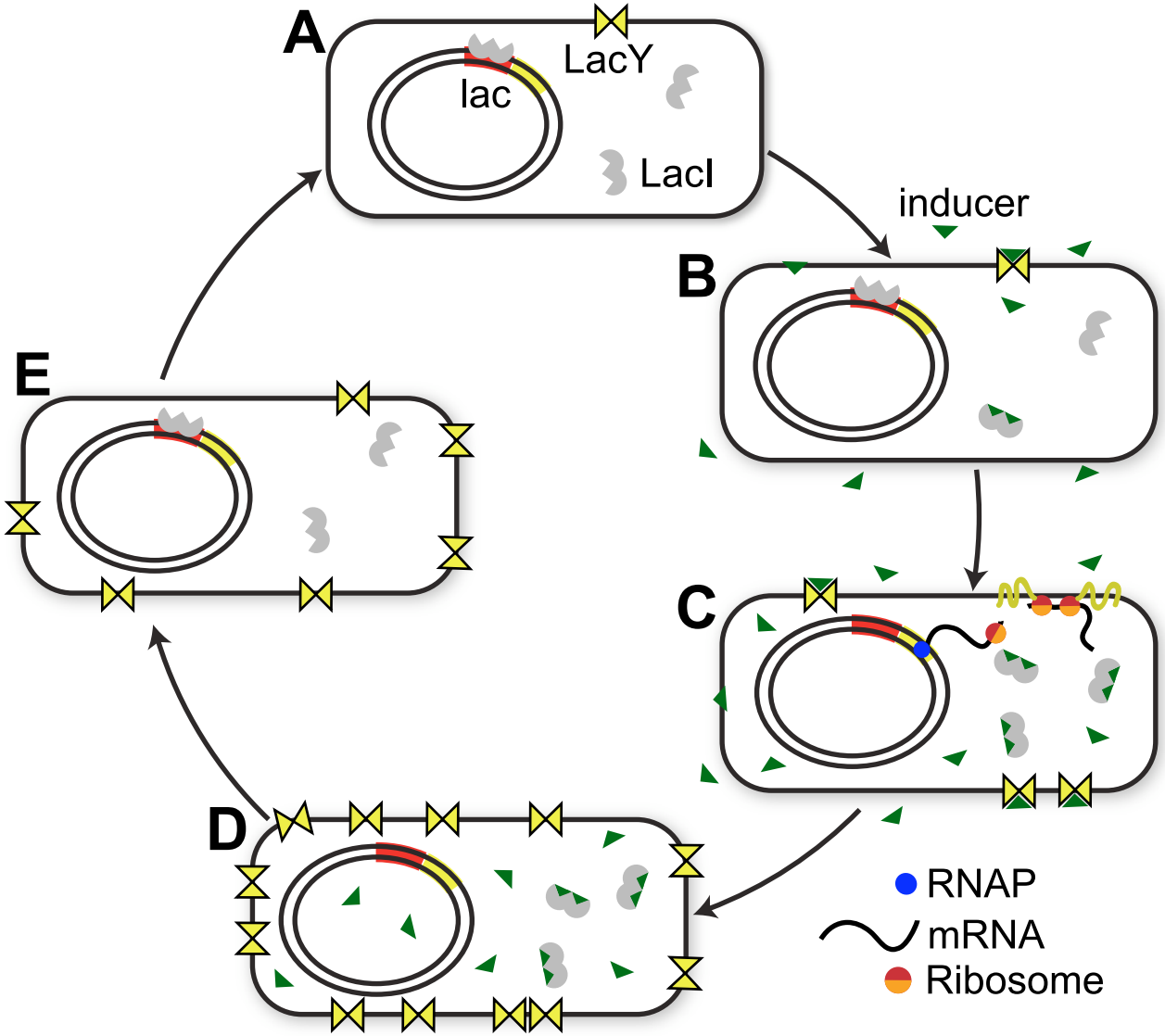
†Bandwidth rates were calculated as four bytes times the number of lattice sites being transferred divided by the runtime. Operation rates were calculated as the number of logical operations per site times the number of lattice sites divided by the runtime.

‡Operation count was calculated using 64-bit operations, but current hardware implements 64-bit operations using 32-bit instructions. Performance calculated using the 32-bit instruction count (88) yields 210 GIPS and 387 GIPS, respectively.

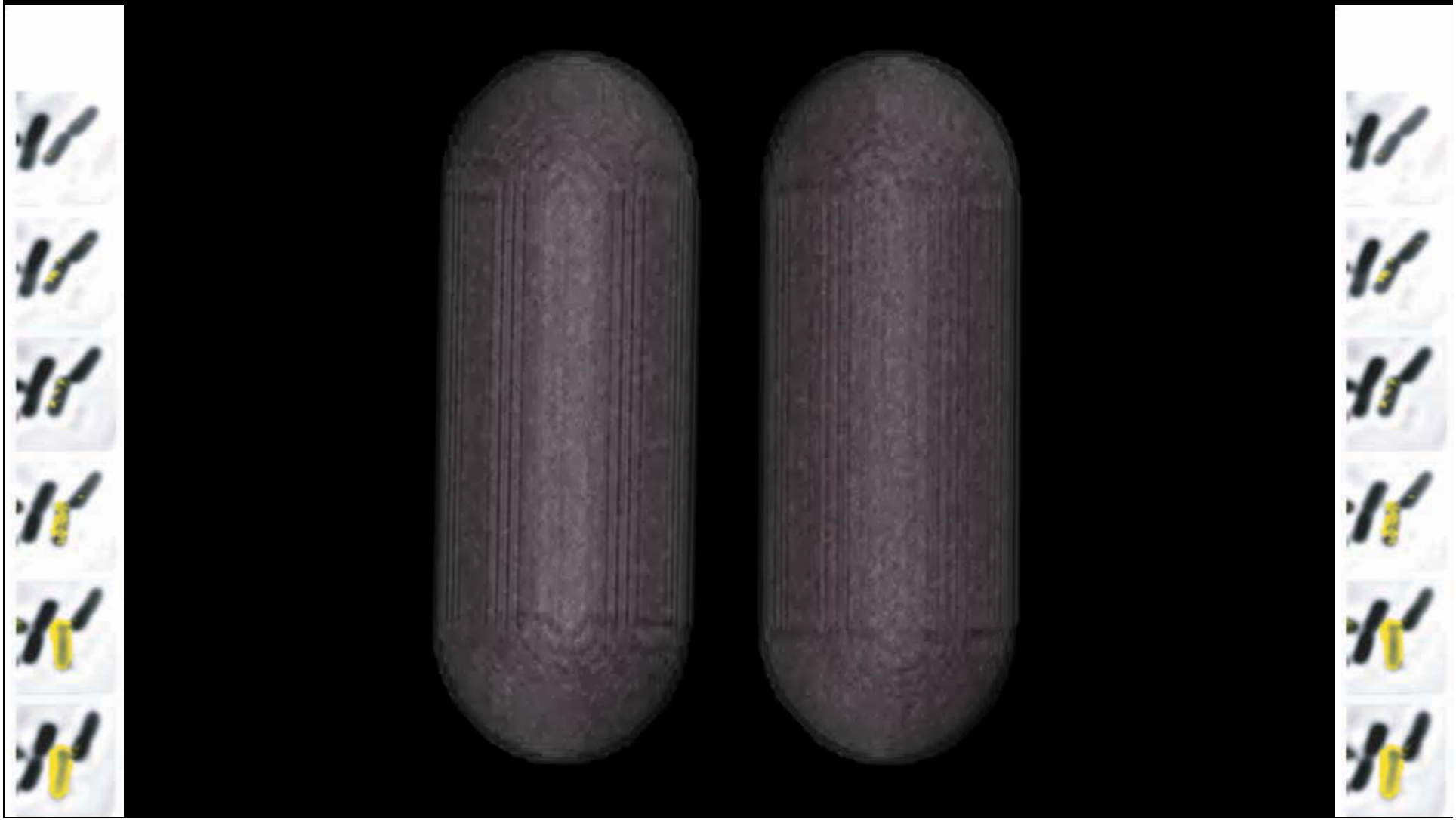
Overview

- Part 1: “Well mixed” stochastic simulation
- Part 2: Spatially resolved stochastic simulation
- Part 3: Simulation example: the *E. coli. lac* operon.

Lac Circuit Overview



Spatially-resolved stochastic simulation of the entire cell cycle of *E.coli* on the GPU



Acknowledgements

- Zan Luthey-Schulten (Chemistry)
  - Elijah Roberts (Chemistry)
  - Leonardo Sepulveda (Biophysics)
- John Stone (Beckman)
- Wen-mei Hwu (ECE)
- Jeremy Enos (NCSA)
- NCSA (Lincoln)