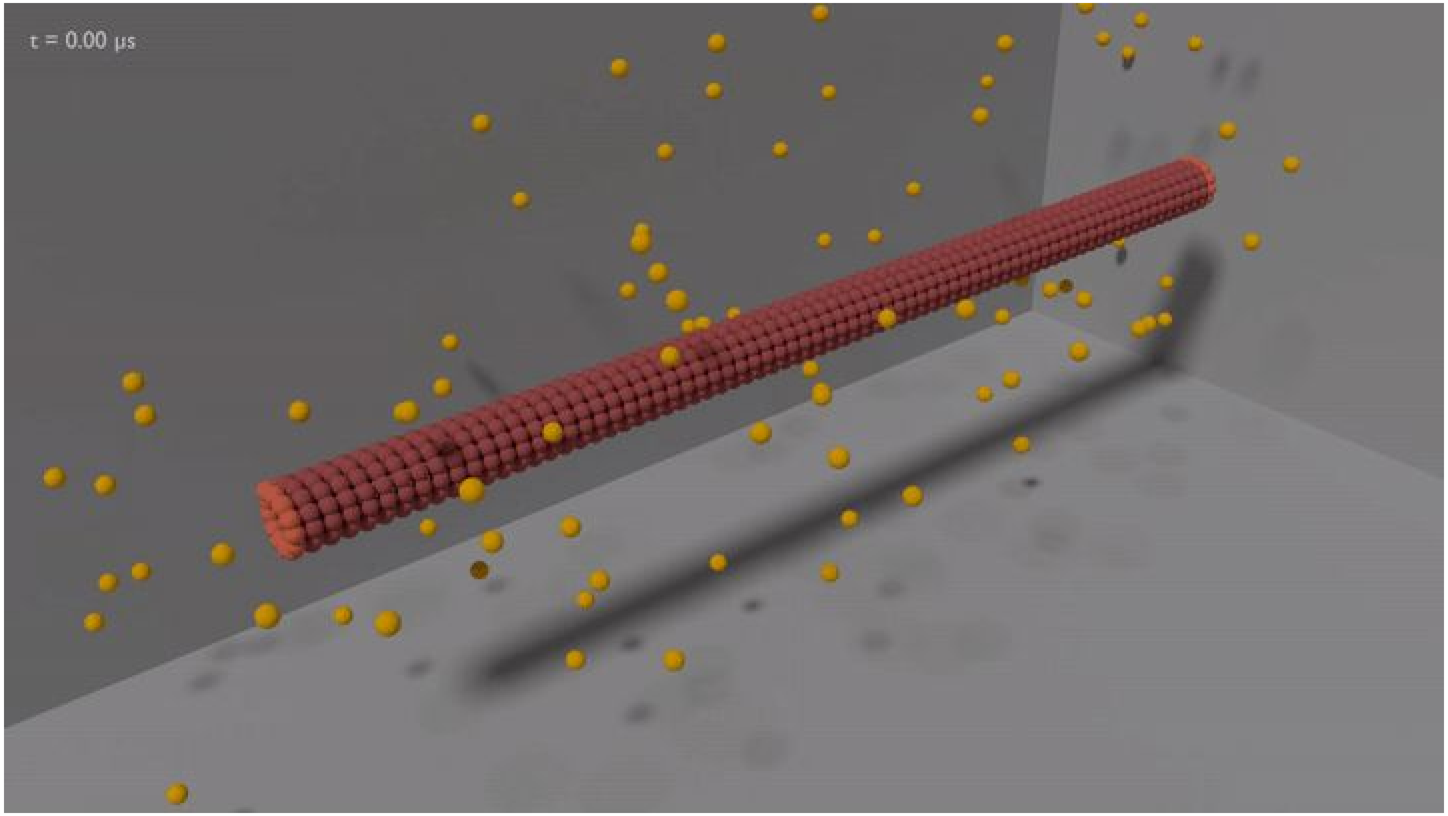
# Implementing the Spatiocyte method on a Graphics Processing Unit (GPU)

Satya Arjunan

Laboratory for Biochemical Simulation
RIKEN Quantitative Biology Center

3 Sept 2016

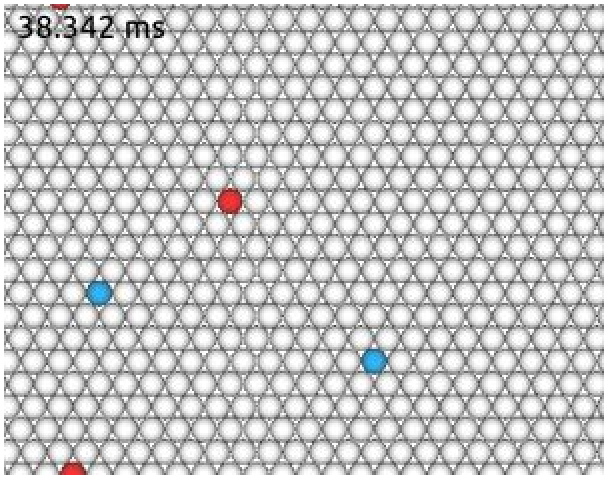
# Spatiocyte: a lattice-based particle simulator

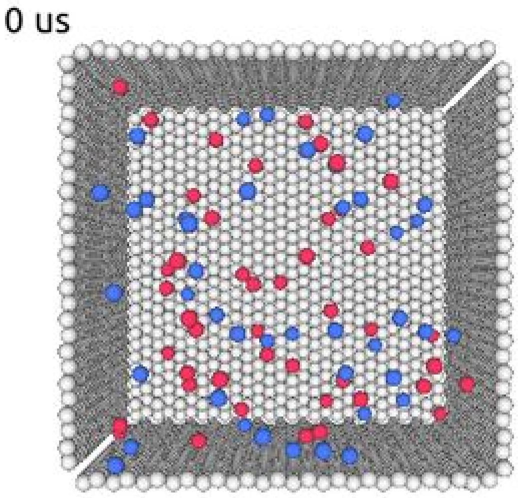



Actual Spatiocyte simulation snapshots of kinesins (yellow and blue) and a microtubule

- Development started during my PhD in 2005 at Keio University
- Open source software available at http://spatiocyte.org
- Now part of E-Cell ver. 4 (Credits to Kazunari Kaizu and Suguru Kato)

# Features of Spatiocyte




Arjunan & Tomita, Syst Synth Biol (2010)

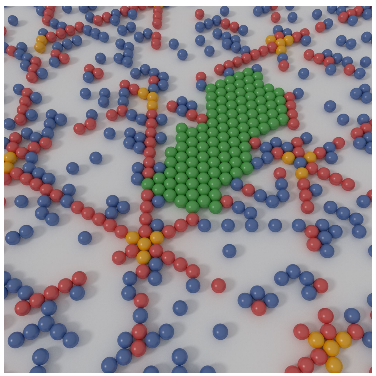
- Spatiocyte can simulate diffusion and reaction of each molecule individually
- Reduces the computational time to resolve molecular collisions (reactions) by discretizing the simulation space into fine lattice voxels (size of a protein, 8 nm diameter)
- 1D, 2D and 3D stochastic reaction-diffusion

# Examples of Spatiocyte application

E. coli MinE ring

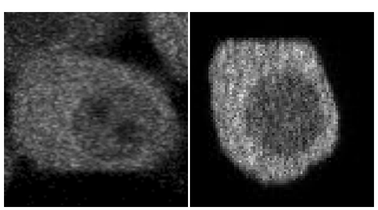

Arjunan & Tomita, Syst Synth Biol (2010)

Erythrocyte Band3 clustering



Shimo et al., PLoS Comp Biol (2015)

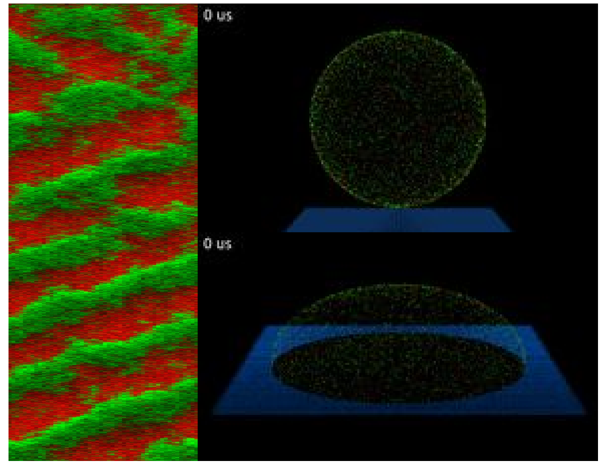
Microscopy bioimaging simulation



real simulated

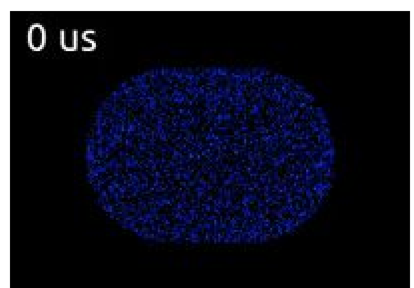
Watabe et al., PLoS ONE (2015)

Cell geometry directs PIP3/PTEN travelling waves in D. discoideum




Arjunan et al., In preparation

Spontaneous polar localization of PAR-1/2 in C. elegans

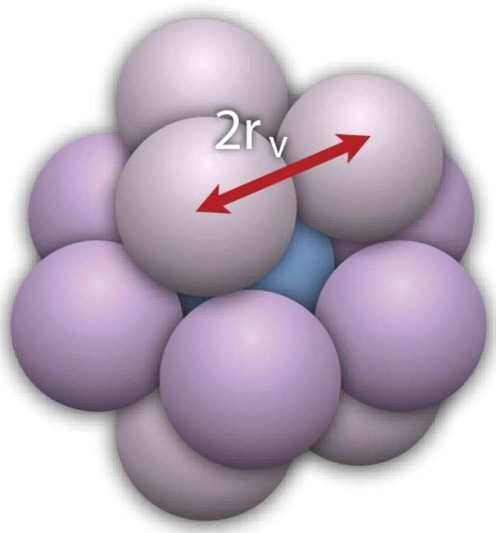



Arjunan et al., In progress

Anionic lipid mediated spontaneous protein clustering and cooperative recruitment

Arjunan et al., In preparation

# Computational issues of Spatiocyte




$$\langle r^2 \rangle = (2\, r_v)^2$$

$$\tau_i = \frac{2r_v^2}{3D_i}$$

- Diffusion is the most costly operation
- Parallelize diffusion with GPUs
- Use C++ STL-like library of CUDA API called Thrust
- Code is available at https://github.com/satya-arjunan/spatiocyte-cuda

This repository | Search | Pull requests | Issues | Gist

satya-arjunan / **spatiocyte-cuda**

Unwatch 1 | Star 0 | Fork 0

Code | Issues 0 | Pull requests 0 | Wiki | Pulse | Graphs | Settings

CUDA implementation of Spatiocyte — Edit

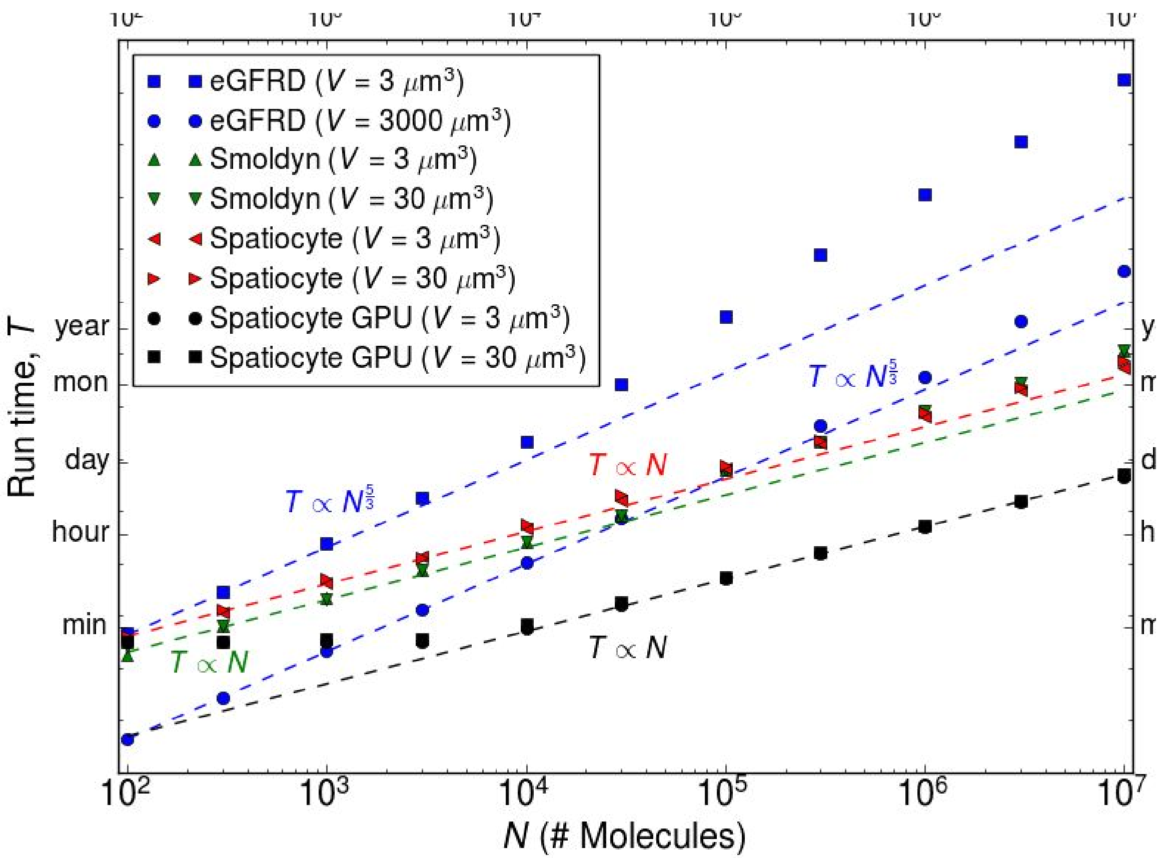
**28** commits | **1** branch | **0** releases | **1** contributor

Branch: **master** | New pull request | Create new file | Upload files | Find file | Clone or download

**satya-arjunan** Small change — Latest commit `b41a6e3` 4 days ago

| | | |
|---|---|---|
| Common.hpp | Removed boxes, simplified code | 5 days ago |
| Compartment.cu | Moved device_vector offsets_ to Compartment | 4 days ago |
| Compartment.hpp | Moved device_vector offsets_ to Compartment | 4 days ago |
| Diffuser.cu | Small change | 4 days ago |
| Diffuser.hpp | Moved device_vector offsets_ to Compartment | 4 days ago |
| Lattice.cu | Moved lattice voxels into device_vector, faster initialization | 4 days ago |
| Lattice.hpp | Moved lattice voxels into device_vector, faster initialization | 4 days ago |
| Makefile | Update README and simplified Makefile | 4 days ago |
| Model.cpp | Removed boxes, simplified code | 5 days ago |
| Model.hpp | Removed boxes, simplified code | 5 days ago |
| README.md | Update README and simplified Makefile | 4 days ago |
| Spatiocyte.cpp | Removed debug messages | 4 days ago |
| Spatiocyte.hpp | Sequential works, no bugs with large lattice | 13 days ago |

# spatiocyte-cuda performance




1. Successfully implemented diffusion with GPU using CUDA Thrust on nVidia GTX 980
2. Avoided race conditions using atomic operation CAS (compare and swap)
3. Can simulate diffusion of 10,000,000 molecules with 5 nm diameter at 1e-12 $m^2s^{-1}$ for 10 s, which takes 11 hours of total run time
4. Achieved 125x speedup compared to serial implementation on Intel Xeon X5680 3.33 GHz

Thrust (PDF) - v7.5 (older) - Last updated September 1, 2015 - Send Feedback -

# Thrust

## 1. Introduction

Thrust is a C++ template library for CUDA based on the Standard Template Library (STL). Thrust allows you to implement high performance parallel applications with minimal programming effort through a high-level interface that is fully interoperable with CUDA C.

Thrust provides a rich collection of data parallel primitives such as scan, sort, and reduce, which can be composed together to implement complex algorithms with concise, readable source code. By describing your computation in terms of these high-level abstractions you provide Thrust with the freedom to select the most efficient implementation automatically. As a result, Thrust can be utilized in rapid prototyping of CUDA applications, where programmer productivity matters most, as well as in production, where robustness and absolute performance are crucial.

This document describes how to develop CUDA applications with Thrust. The tutorial is intended to be accessible, even if you have limited C++ or CUDA experience.

### 1.1. Installation and Versioning

Installing the CUDA Toolkit will copy Thrust header files to the standard CUDA include directory for your system. Since Thrust is a template library of header files, no further installation is necessary to start using Thrust.

In addition, new versions of Thrust continue to be available online through the GitHub Thrust project page. The version of Thrust included in this version of the CUDA Toolkit corresponds to version 1.7.0 from the Thrust project page.

## 2. Vectors

Thrust provides two vector containers, `host_vector` and `device_vector`. As the names suggest, `host_vector` is stored in host memory while `device_vector` lives in GPU device memory. Thrust's vector containers are just like `std::vector` in the C++ STL. Like `std::vector`, `host_vector` and `device_vector` are generic containers (able to store any data type) that can be resized dynamically. The following source code illustrates the use of Thrust's vector containers.

```
#include <thrust/host_vector.h>
#include <thrust/device_vector.h>

#include <iostream>

int main(void)
{
    // H has storage for 4 integers
```

```
void Diffuser::walk() {
 const size_t size(mols_.size());
 reacteds_.resize(size);
 thrust::transform(thrust::device,
     thrust::counting_iterator<unsigned>(0),
     thrust::counting_iterator<unsigned>(size),
     mols_.begin(),
     mols_.begin(),
     generate(
       seed_,
       stride_,
       id_stride_,
       vac_id_,
       thrust::raw_pointer_cast(&is_reactive_[0]),
       thrust::raw_pointer_cast(&offsets_[0]),
       thrust::raw_pointer_cast(&reacteds_[0]),
       thrust::raw_pointer_cast(&voxels_[0])));
 seed_ += size;
}
```

```
struct generate {
 __host__ __device__ generate(
     const unsigned seed,
     const voxel_t stride,
     const voxel_t id_stride,
     const voxel_t vac_id,
     const bool* is_reactive,
     const mol_t* offsets,
     umol_t* reacteds,
     voxel_t* voxels):
   seed_(seed),
   stride_(stride),
   id_stride_(id_stride),
   vac_id_(vac_id),
   is_reactive_(is_reactive),
   offsets_(offsets),
   reacteds_(reacteds),
   voxels_(voxels) {}
 __device__ umol_t operator()(const unsigned index, const umol_t vdx) const {
   curandState s;
   curand_init(seed_+index, 0, 0, &s);
   float ranf(curand_uniform(&s)*11.999999);
   const unsigned rand((unsigned)truncf(ranf));
   const bool odd_lay((vdx/NUM_COLROW)&1);
   const bool odd_col((vdx%NUM_COLROW/NUM_ROW)&1);
   mol2_t val(mol2_t(vdx)+offsets_[rand+(24&(-odd_lay))+(12&(-odd_col))]);
   const voxel_t res(atomicCAS(voxels_+val, vac_id_, index+id_stride_));
   //If not occupied, walk:
   if(res == vac_id_) {
     voxels_[vdx] = vac_id_;
     reacteds_[index] = 0;
     return val;
   }
   return vdx;
 }
```