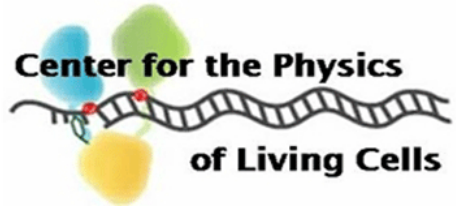
Lattice Microbes: CUDA Algorithms for Stochastic Simulation of Biochemical Reactions

Michael J. Hallock
Joseph R. Peterson
Luthey-Schulten Laboratory

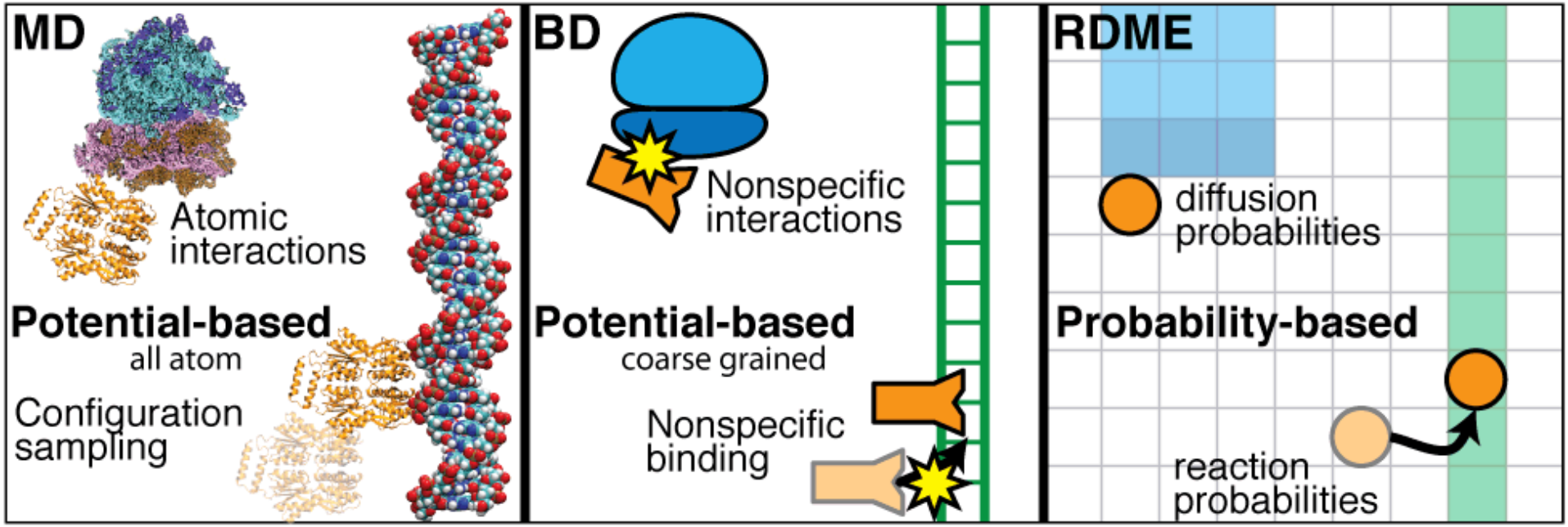


ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Modeling of Molecules to Cells

Biomolecular interactions span many orders of magnitude in space and time

Molecular Dynamics **Brownian Dynamics** **Reactions and Diffusion**



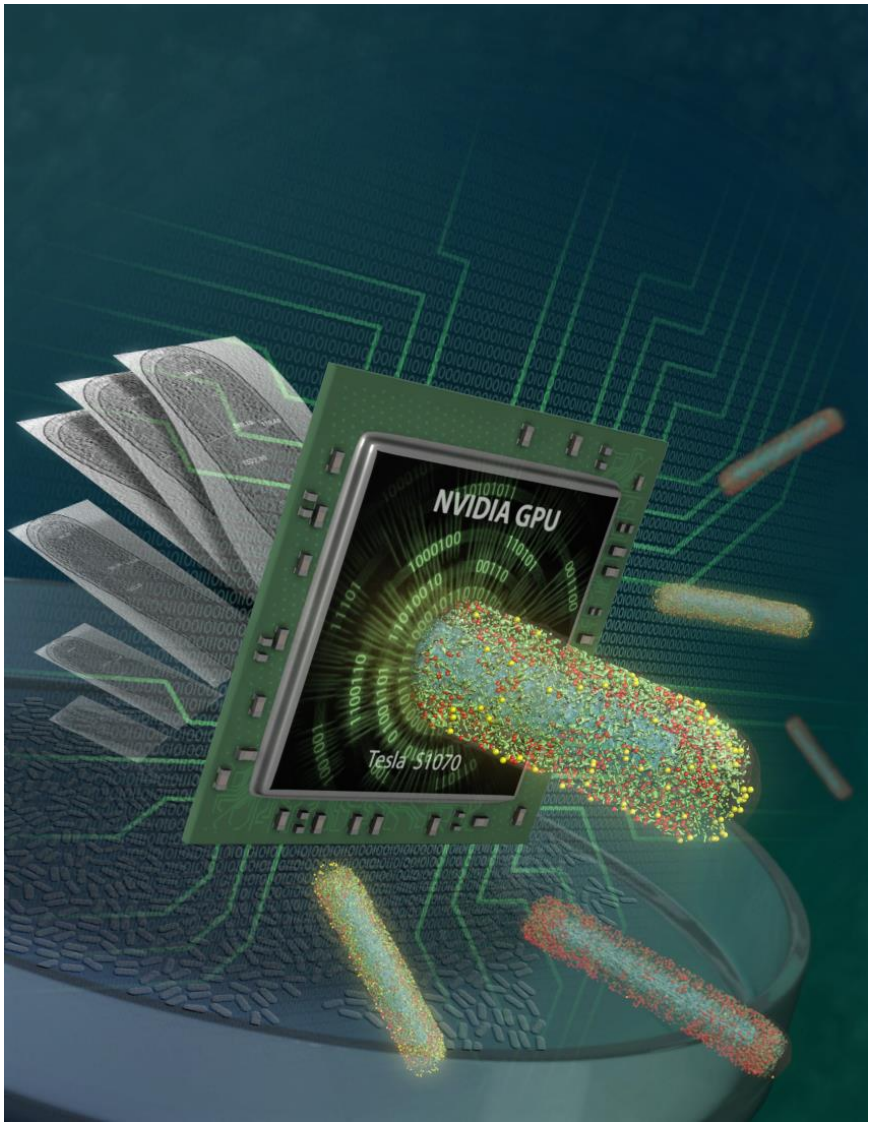
Femtoseconds **Picoseconds** **Microseconds**

Molecules to Macromolecular assemblies

Whole Cells

Lattice Microbes

High Performance Stochastic Simulator



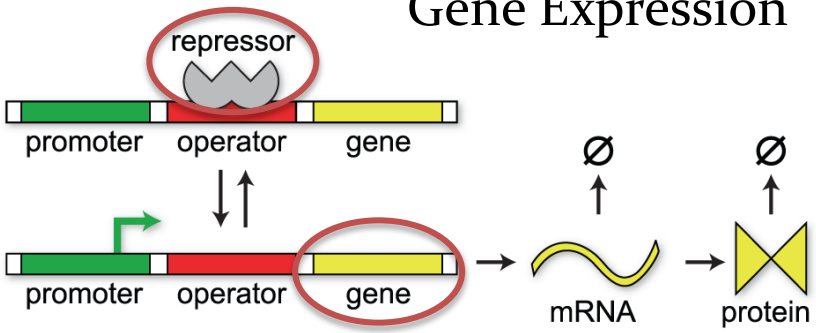
- Whole Cell Models
  - Based on experimentally determined geometry
  - Realistic *in vivo* environment
- Python interface
- Employs new algorithm to sample the Reaction-Diffusion Master Equation : **Muti-particle Diffusion** (MPD-RDME)
  - Lattice based method
  - Enables acceleration by GPU hardware
  - Up to 356x faster than similar codes

Roberts, Stone, Luthey-Schulten (2013) *JCC*

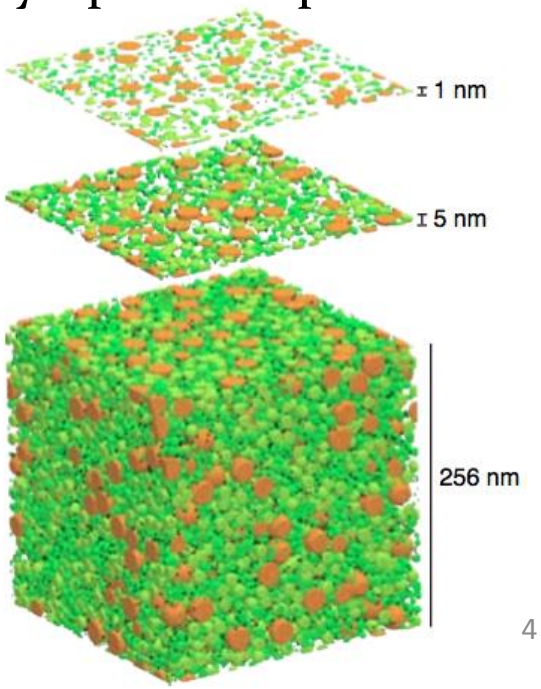
Biochemical Processes Can be Stochastic

- Small copy numbers involved
  - 1 or 2 copies of gene
  - Few copies of transcription factors
- Intracellular space is very crowded
  - Not homogenous

Gene Expression



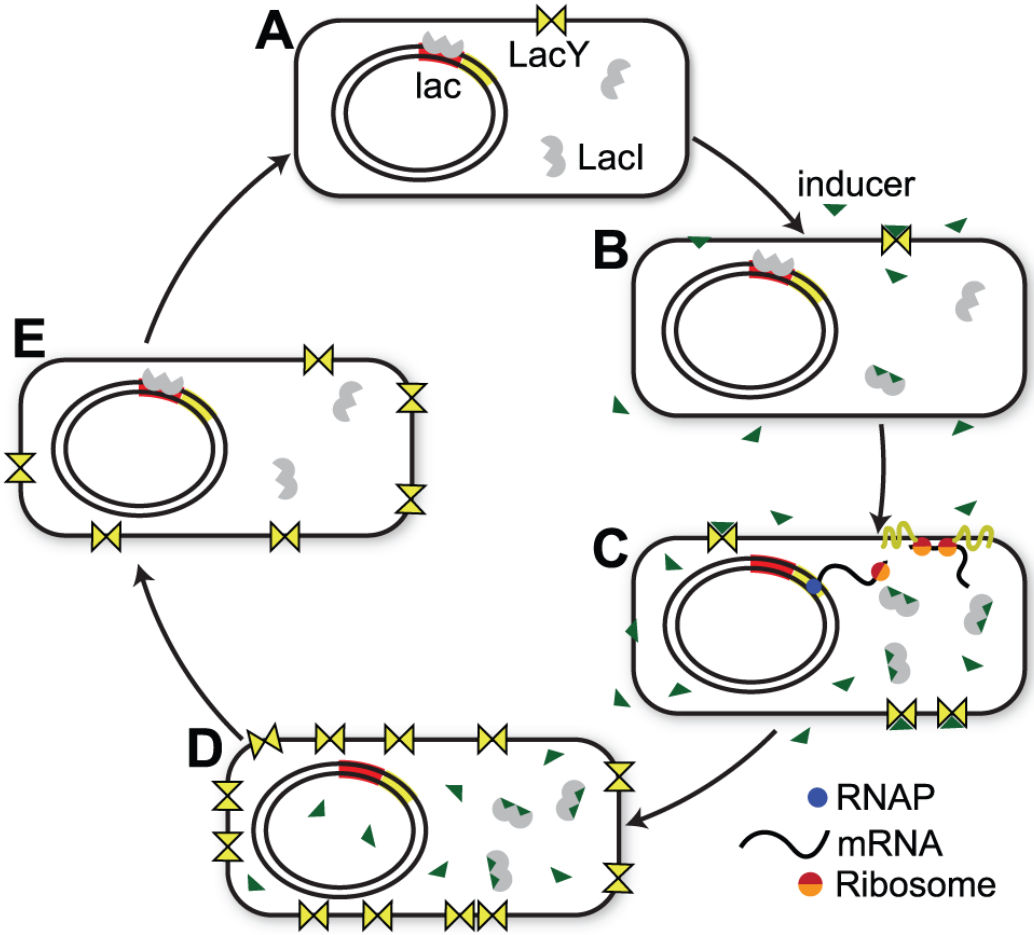
Cytoplasmic space

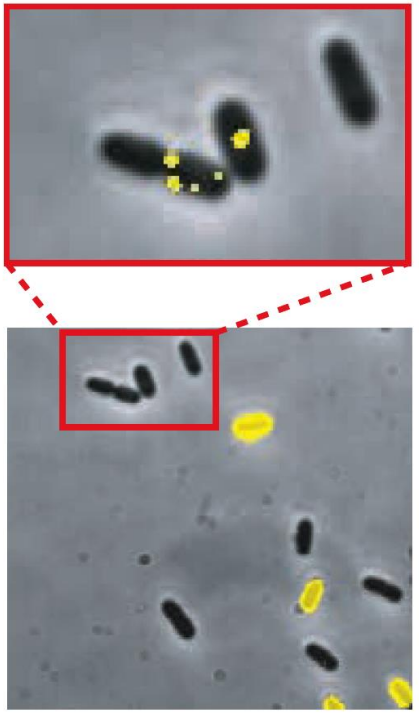


Roberts.....Luthey-Schulten, 2009, IEEE

Stochastic Example: Two State Lac Genetic Switch

- Inducible switch for utilization of lactose by *E. coli.*





Bimodality in Population

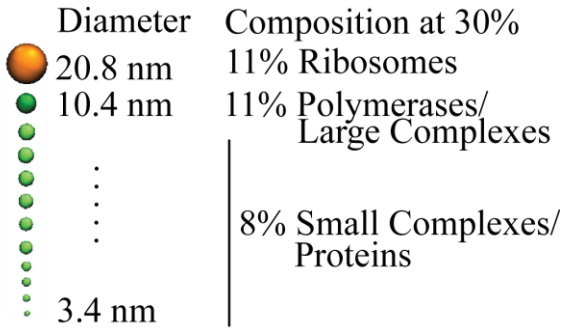
Choi, Cai, Frieda, Xie (2008) *Science*
Roberts.....Luthey-Schulten (2011) PLoS Comput Biol

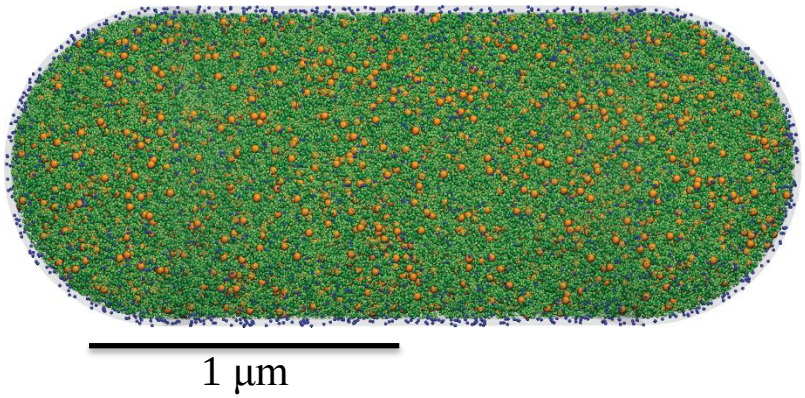
Switching due to mRNA Bursting

Lattice Microbes Workflow

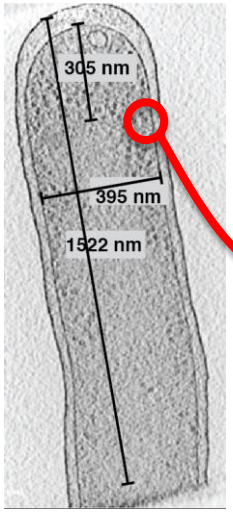
Step 1: Making Whole Cell Models

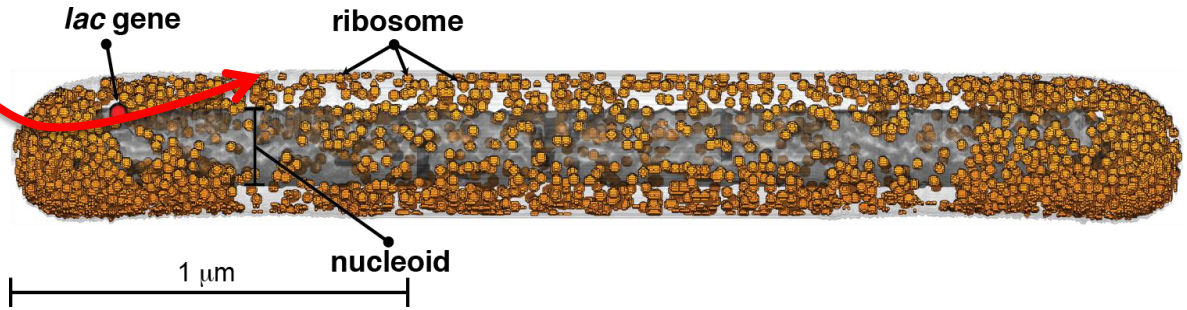
Proteomics Data





Cryo-electron Tomography

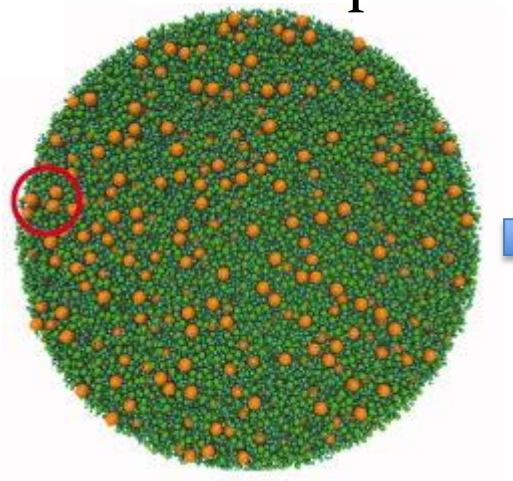




Roberts.....Luthey-Schulten (2009) IEEE IPDPS
Roberts..... Luthey-Schulten (2011) PLoS Comput Bio

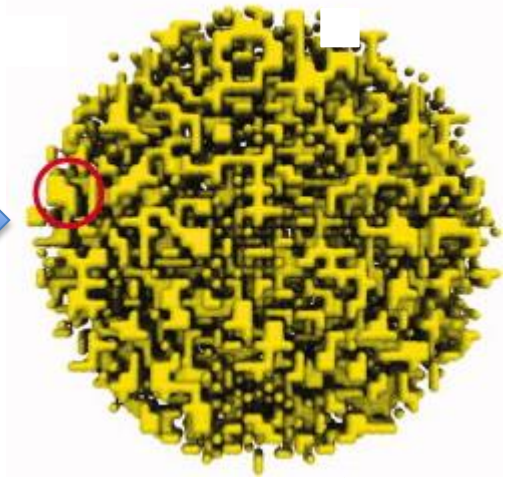
Lattice Microbes Workflow

Multi-particle Diffusion RDME (MPD-RDME)



Step 2

Discretization



Step 3

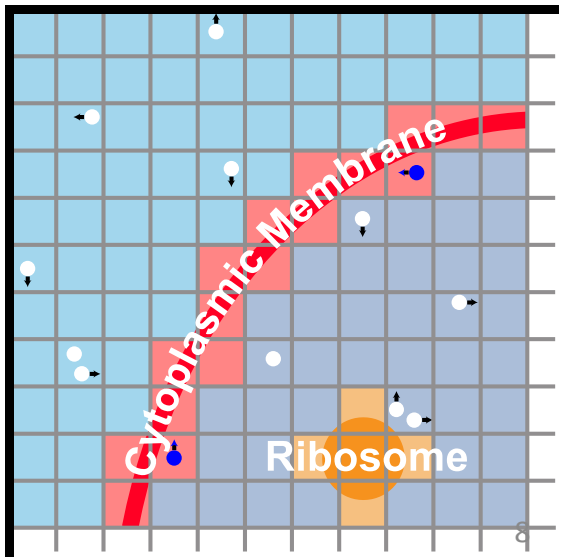
Initialization

Step 4

Sample RDME

MPD-RDME

- Constant (small) time step iterations:
  - One diffusion event per particle maximum
  - One reaction per time step
- At each time step
  - Particles diffuse between neighboring lattice sites
  - Particles react within lattice sites



Roberts, Stone, Luthey-Schulten (2013) *JCC*

Modeling Biochemical Reaction Networks

Deterministic		Stochastic (Probabilistic)	
Flux Balance Analysis (FBA)	**Ordinary Differential Equations (ODE)**	**Chemical Master Equation (CME)**	**Reaction-Diffusion Master Equation (RDME)**
$S \times v = 0$	$S \times v = \frac{d[C]}{dt}$	$\frac{dP(x,t)}{dt} = RP(x,t)$	$\frac{dP(x,t)}{dt} = RP(x,t) + DP(x,t)$

Lattice Microbes

Full RDME:

$$\frac{\partial P(x,t)}{\partial t} = \sum_{v \in V} \sum_{r=1}^{R} -a_r(x_v)\, P(x_v,t) + a_r(x_v - S_r)\, P(x_v - S_r,t)$$

$$+ \sum_{v}^{V} \sum_{x}^{\pm\hat{i},\hat{j},\hat{k}} \sum_{a}^{N} -d^a x_v^a\, P(x,t) + d^a (x_{v+x}^a + 1)\, P(x + 1_{v+x}^a - 1_v^a, t)$$

Chemical Master Equation

$$\frac{\partial P(x,t)}{\partial t} = \sum_{r=1}^{R} \left[-a_r(x)P(x,t) + a_r(x - S_r)P(x - S_r, t) \right]$$

Reverse reaction

Forward reaction

Bimolecular Reaction: $A + B \underset{k_r}{\overset{k_f}{\leftrightarrow}} C$

- Probability based reaction equation
  - Applicable to discrete/stochastic (Markovian) scenarios
  - a_r linked directly to deterministic rate constant
- Applicable in well-stirred systems

$$a_r(x) = k_r x_a$$

$$a_r(x - S_r) = \frac{k_f x_a x_b}{N_A V}$$

Reaction-Diffusion Master Equation

$$\frac{\partial P(x,t)}{\partial t} = \sum_{v \in V} \sum_{r=1}^{R} -a_r(x_v)\cdot P(x_v,t) + a_r(x_v - S_r)\cdot P(x_v - S_r,t)$$

$$+ \sum_{v}^{V} \sum_{x}^{\pm\hat{i},\hat{j},\hat{k}} \sum_{a}^{N} -d^a x_v^a \cdot P(x,t) + d^a (x_{v+x}^a + 1)\cdot P(x + 1_{v+x}^a - 1_v^a, t)$$

Diffusion into subvolume

Diffusion out of subvolume

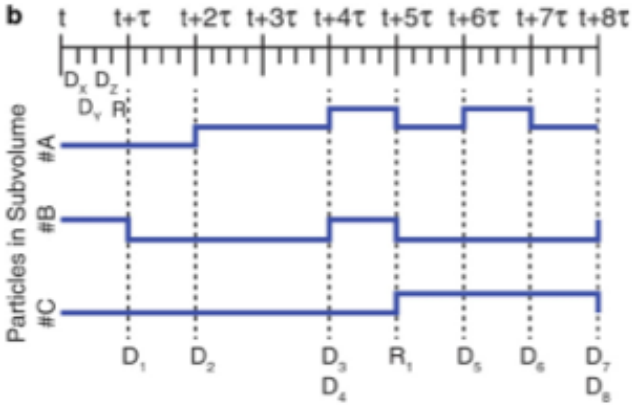
Diffusive propensity: $d^a = \frac{D^a}{l^2}$

- MPD-RDME
  - Cubic subvolumes of dimension: λ
  - Each subvolume assumed to be well-stirred
  - Diffusion operations to and from nearest neighbors
  - Fine-grained parallelism

MPD-RDME Algorithm

Algorithm 1: the MPD-RDME algorithm

```
1  while t < t_end do
2      # Diffusion operator.
3      for dim ∈ {x, y, z} do
4          parfor ν ∈ V do
5              for particle ∈ x_ν do
6                  n = uniformRand()
7                  if n ≤ P(particle, D^{+dim}) then
8                      move particle from subvolume x_ν to neighboring subvolume in the
                       +dim direction
9                  else if (n − P(particle, D^{+dim})) ≤ P(particle, D^{−dim}) then
10                     move particle from subvolume x_ν to neighboring subvolume in the
                       −dim direction
11                 end
12             end
13         end
14     end
15     # Reaction operator.
16     parfor ν ∈ V do
17         for r ∈ R do
18             a_tot = a_tot + a_r(x_ν)
19         end
20         n_1 = uniformRand()
21         if n_1 ≤ ∫_0^τ a_tot e^{−a_tot t} dt then
22             n_2 = uniformRand()
23             for r ∈ R do
24                 if a_{r−1}(x_ν) < n_2 · a_tot ≤ a_r(x_ν) then
25                     perform reaction r in subvolume x_ν
26                 end
27             end
28         end
29     end
30     t = t + τ
31 end
```



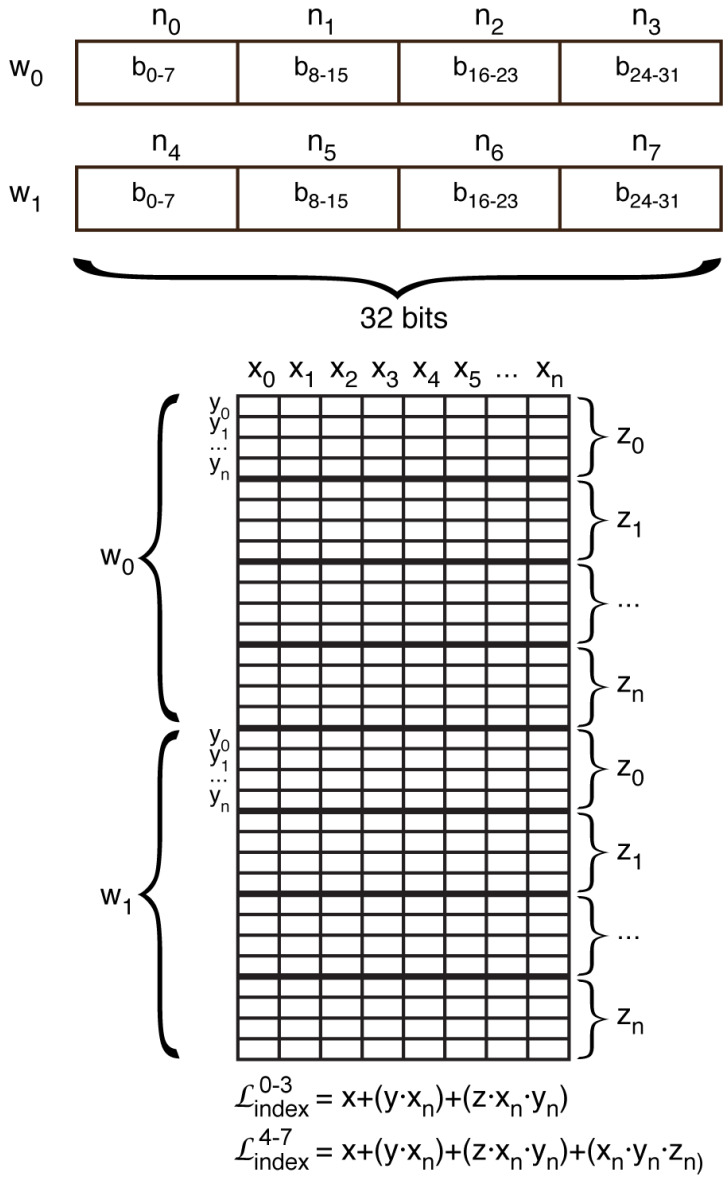
- Many short time steps:

$$t \pounds \frac{l^2}{6D_{\max}}$$

- GPU Implementation
  - Three sequential diffusion kernels: x, y, z
  - One reaction kernel

Roberts, Stone, Luthey-Schulten (2013) *JCC*

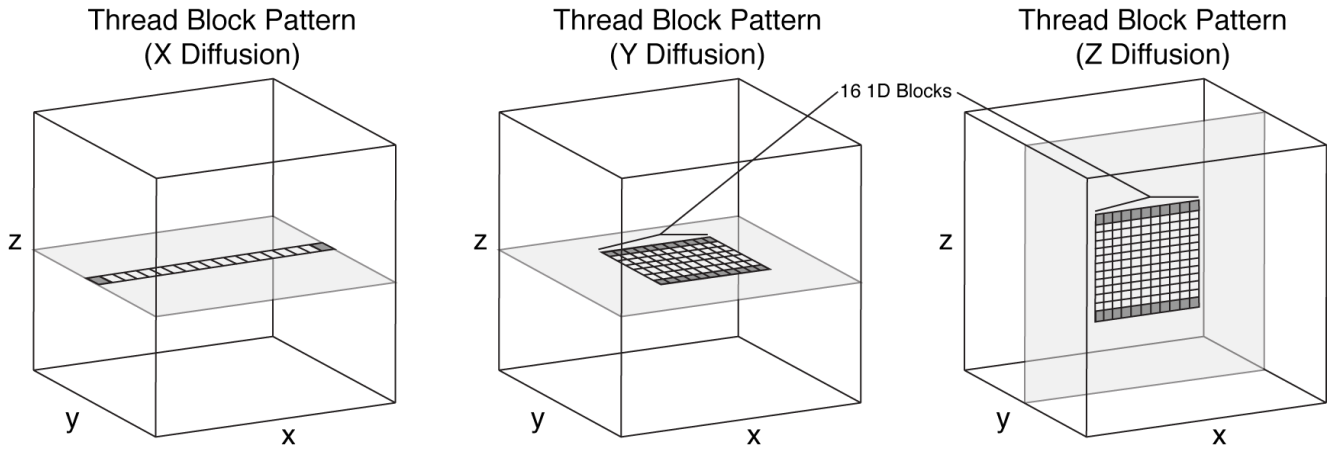
Lattice Data Structure



- 2 Lattices mapped as a 3D array in linear memory
  - Site type lattice
  - Particle lattice
    - 8 bit particles →256 particle types
    - 8 particles per site →320 µM at 10% full for 16 nm spacing

Roberts.....Luthey-Schulten, 2009, IEEE

Diffusion Kernels



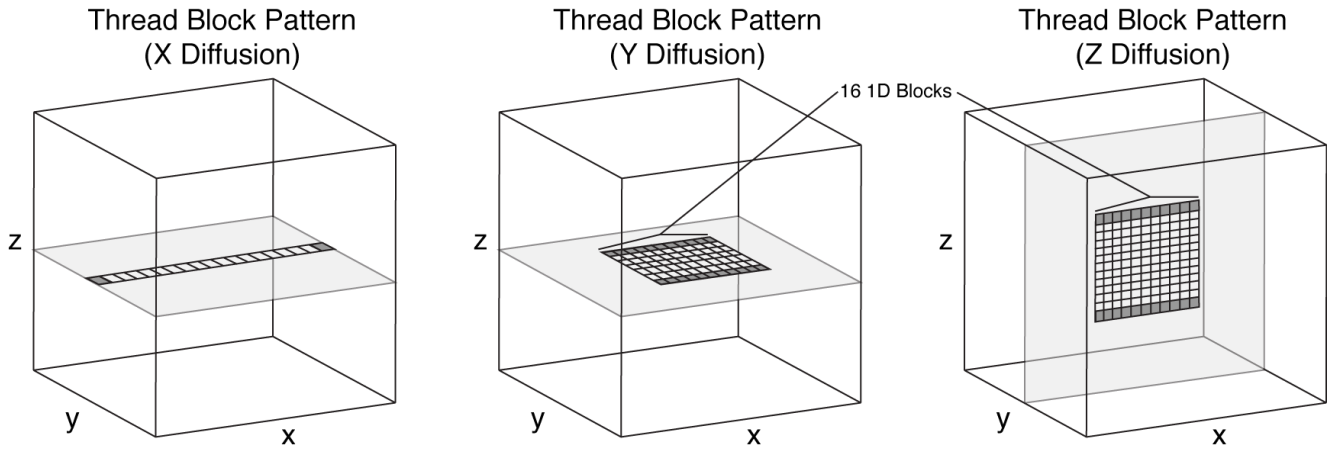
Coalesced memory access:

- 1D x axis
- 2D x-y and x-z blocks

Diffusion Operation

```
__global__ void MpdRdmeSolver_x_kernel(...)
{
   // Figure out the offset of this thread and block in the lattice and the lattice segment.
   ...
   // Shared memory to store the lattice segment.
   __shared__ unsigned int window[MPD_X_WINDOW_SIZE*MPD_WORDS_PER_SITE];
   __shared__ uint8_t sitesWindow[MPD_X_WINDOW_SIZE];

   // Copy the x window from device memory into shared memory.
   copyXWindowFromLattice(bx, inLattice, window, latticeIndex, latticeXIndex, windowIndex);
   copyXWindowFromSites(bx, inSites, sitesWindow, latticeIndex, latticeXIndex, windowIndex);
   __syncthreads();

   // Make the choices.
   __shared__ unsigned int choices[MPD_X_WINDOW_SIZE*MPD_WORDS_PER_SITE];
   makeXDiffusionChoices(window, sitesWindow, choices...);
   __syncthreads();

   // Propagate the choices to the new lattice segment.
   performPropagation(...);
}
```

Diffusion Kernels



Coalesced memory access:

- 1D x axis
- 2D x-y and x-z blocks

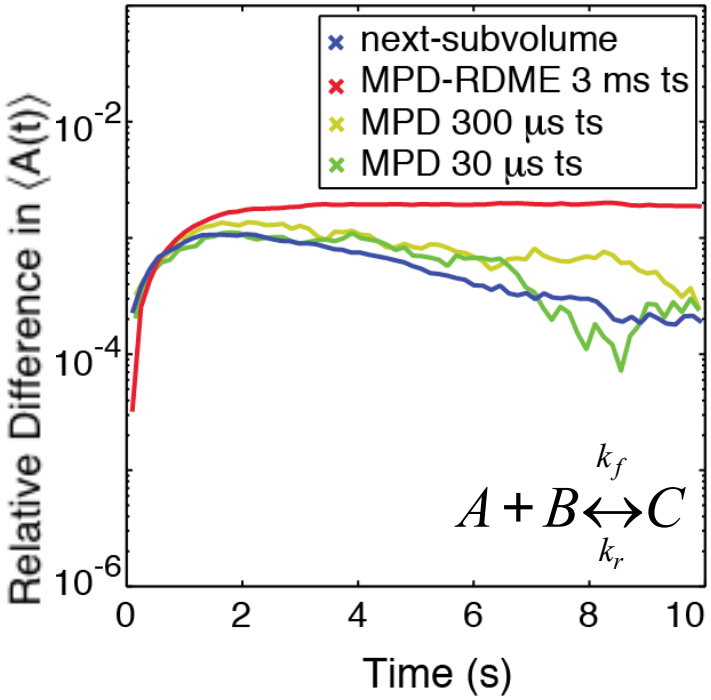
X-diffusion copy lattice window:

```
inline __device__ void copyXWindowFromLattice(...)
{
   if (latticeXIndex < latticeXSizeC)
   {
      // Load the block.
      for (uint w=0, latticeOffset=0, windowOffset=0;
           w < MPD_WORDS_PER_SITE;
           w++, latticeOffset+=latticeXYZSizeC, windowOffset+=MPD_X_WINDOW_SIZE)
         window[windowIndex+windowOffset] = lattice[latticeIndex+latticeOffset];

      // Calculate the effective thread block width,
      //  accounting for the fact that the block might not be aligned with the end of the lattice.
      int threadBlockWidth = ((bx+1)*blockDim.x <= latticeXSizeC)?(blockDim.x):(latticeXSizeC-(bx*blockDim.x));

      // Load leading apron
      ...
      // Load trailing apron
      ...
   }
}
```

Code Performance

Single GPU version 2.0 available since January 2012

Suitable error from MPD-RDME approximation:

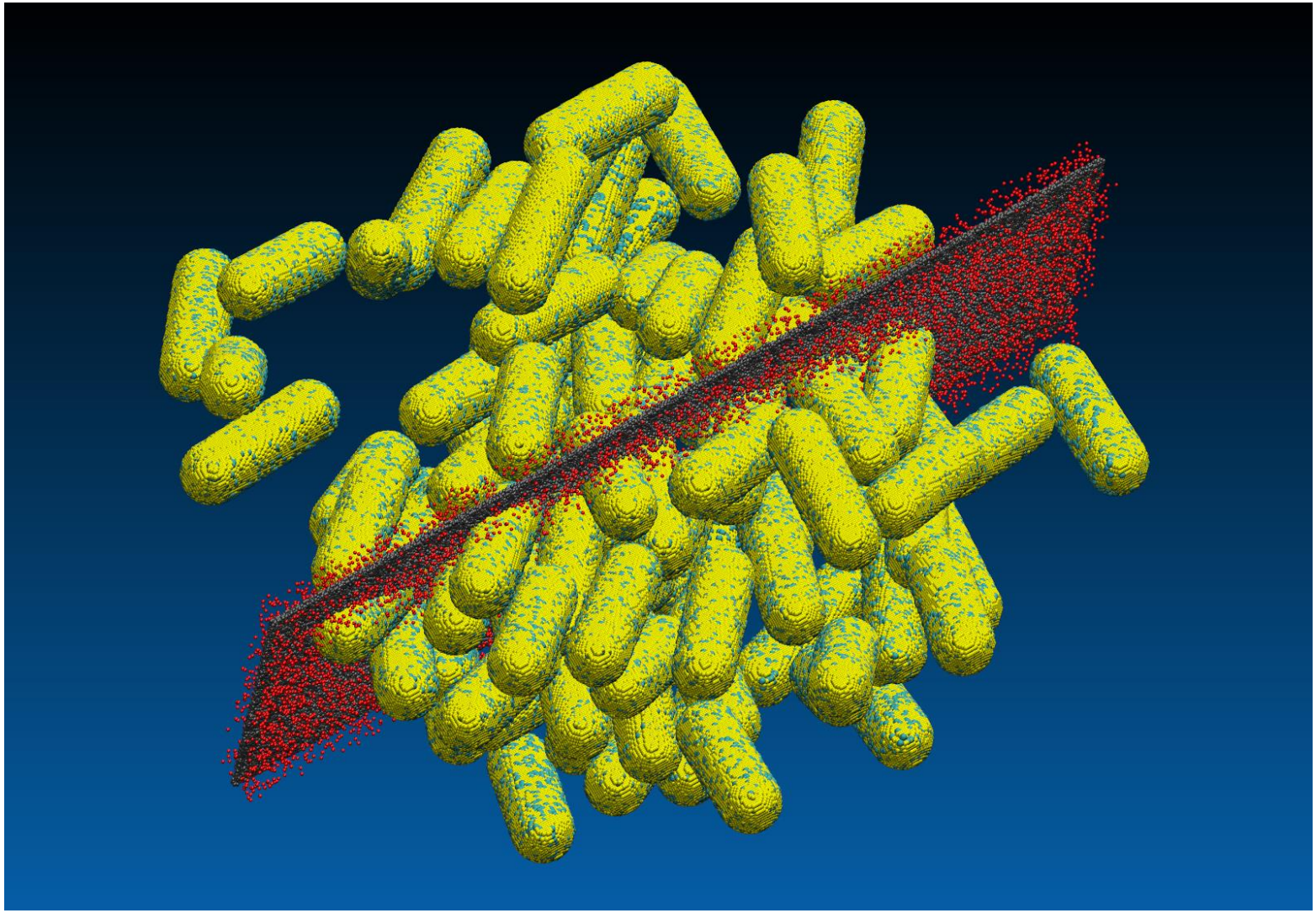


RDME: as much as *356 times faster*
CME: as much as *2.7 times faster*

Method	Lac Switch, [I] = 5µM rxns/sec	Lac Switch, [I] = 5µM wall clock/sim hr	Lac Switch, [I] = 40µM rxns/sec	Lac Switch, [I] = 40µM wall clock/sim hr
CME				
Direct GPU	7.3×10^6	0.018 sec	7.4×10^6	4.0 sec
Direct No-GPU	5.0×10^6	0.026 sec	5.0×10^6	5.9 sec
Next React GPU	5.5×10^6	0.024 sec	6.2×10^6	4.8 sec
Next React No-GPU	3.9×10^6	0.032 sec	4.5×10^6	6.5 sec
Stochkit SSA	4.2×10^6	0.031 sec	4.2×10^6	7.2 sec
COPASI	2.7×10^6	0.048 sec	2.8×10^6	11 sec
RDME				
MPD-RDME GPU	0.44	1.89 days	0.1200	0.191 yrs
Next Subvol GPU	0.10	8.33 days	0.0004	59 yrs
Next Subvol No-GPU	0.10	8.67 days	0.0004	59 yrs
MesoRD	0.01	83 days	0.0003	68 yrs

Roberts, Stone, Luthey-Schulten (2013) *JCC*

Larger and More Complex Models: Motivating Multiple GPUs



Multiple GPU Communication

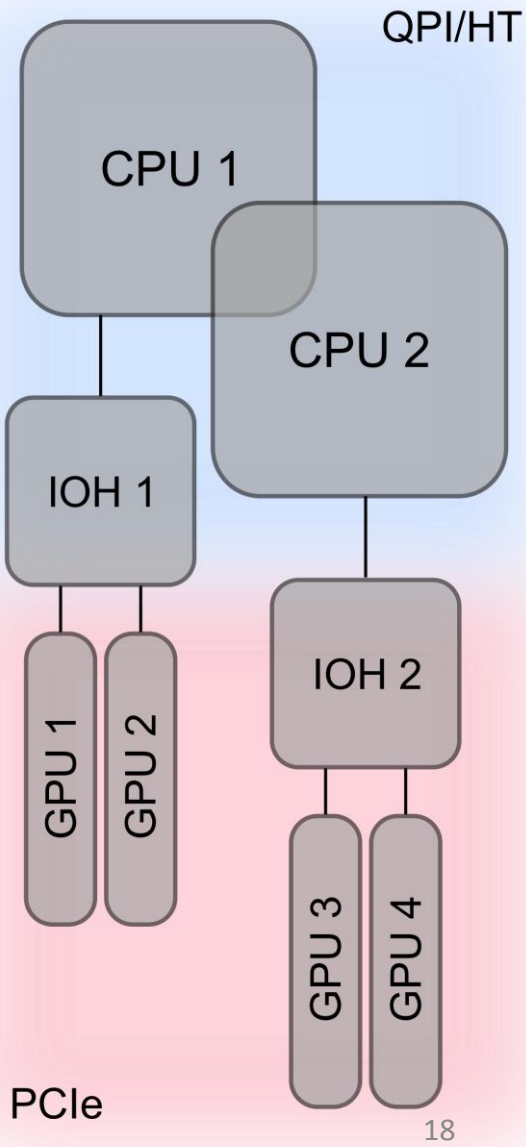
Data from one GPU to another can be moved:

- From GPU 1 to host memory, and then from host memory to GPU 2
- DMA directly from GPU 1 to GPU 2

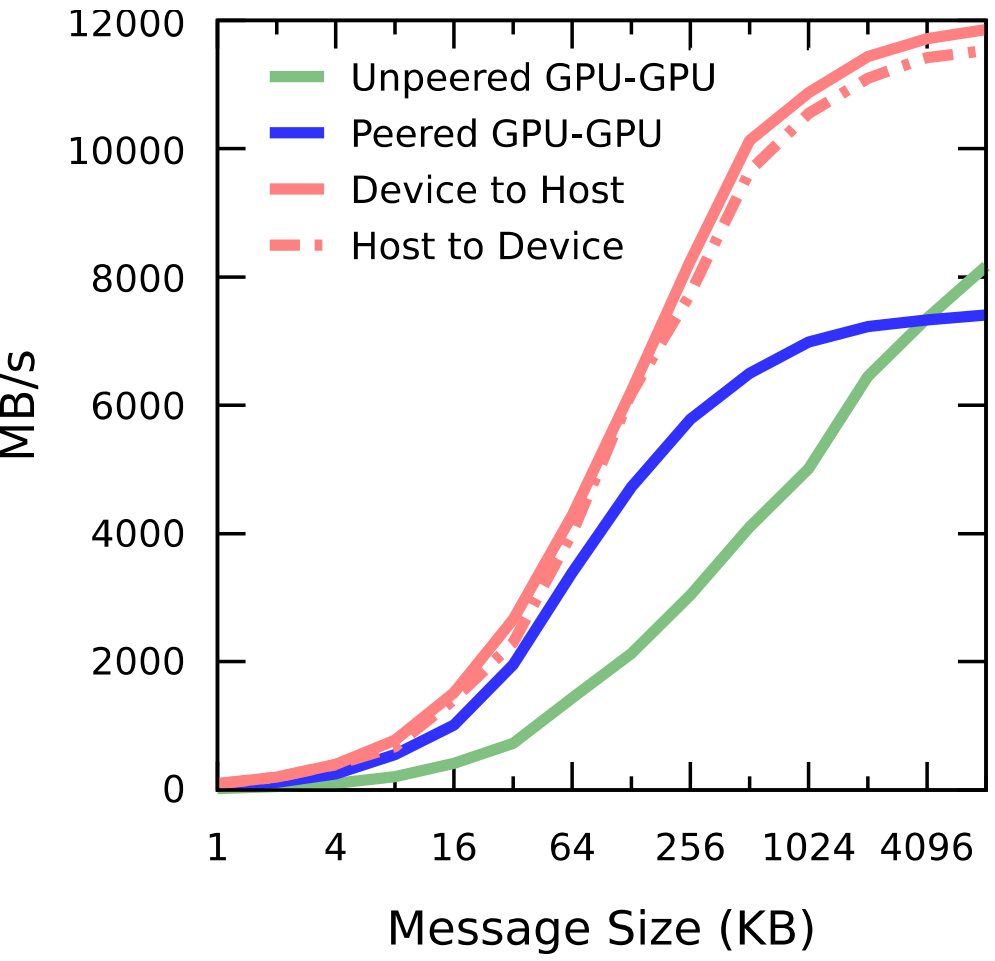
Device-to-Device copies are accelerated if the GPUs are *peered* and can directly address each other's memory

Ability to peer devices depends on machine topology, multi-IOH machines can result in GPU in non-peering GPUs

Check using cudaDeviceCanAccessPeer()



Bandwidth of Peered GPUs



Peer copies alleviate pressure on host memory by only occupying PCIe bus

Can allow for greater aggregate bandwidth

Test shown using two GTX680 cards on a Sandy Bridge system running at PCIe 3.0 speeds, which is not officially supported.

Hallock, Stone, Roberts, Fry, Luthey-Schulten *Submitted*

Peering GPUs

Peering is one-way, so two calls to cudaDeviceEnablePeerAccess are required to have bi-directional communication between devices

Thread 1:

```
cudaSetDevice(1);
cudaDeviceEnablePeerAccess(2, 0); // 0 = flags (reserved)
```

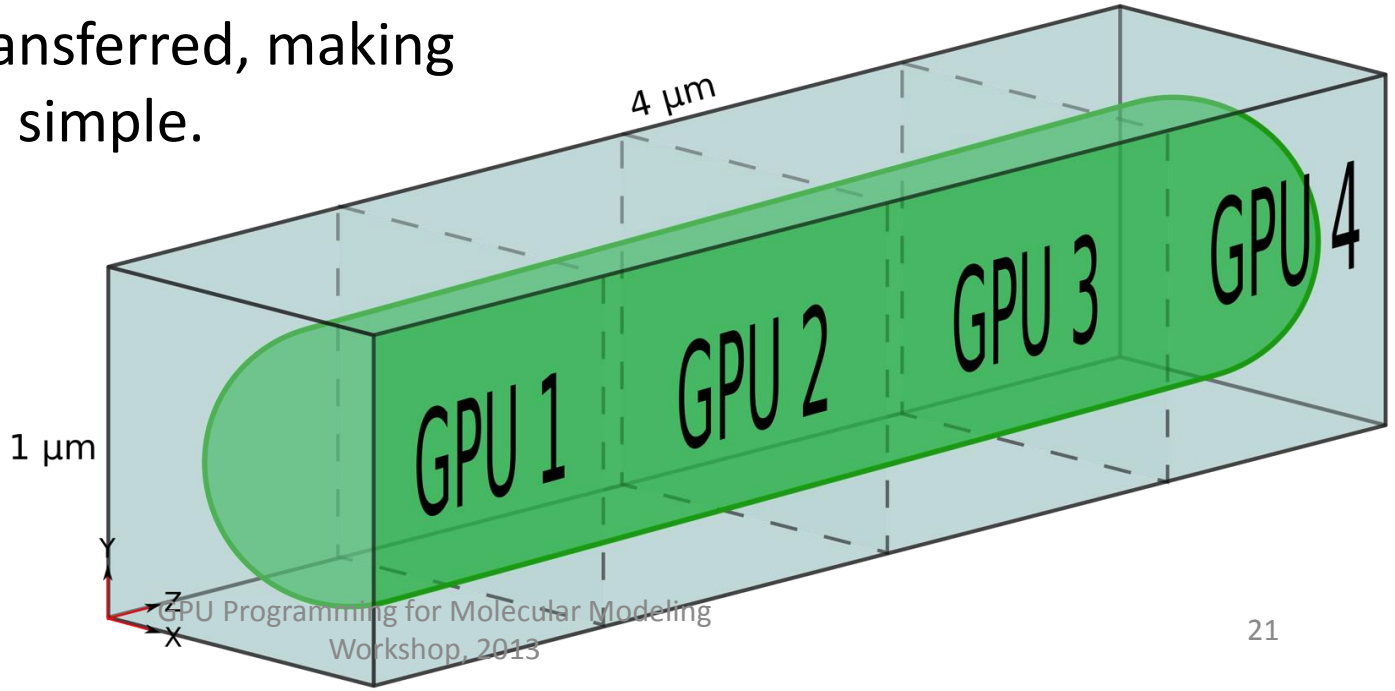
Thread 2:

```
cudaSetDevice(2);
cudaDeviceEnablePeerAccess(1, 0);
```

MPD-RDME over Multiple GPUs

To use multiple GPUs, we must divide up the work.

Apply a spatial decomposition to slice the lattice into sub-lattices to assign to each available GPU. A one-dimensional decomposition along the z-axis allows for contiguous data ranges that must be transferred, making communication simple.



Inter-GPU Data Dependencies

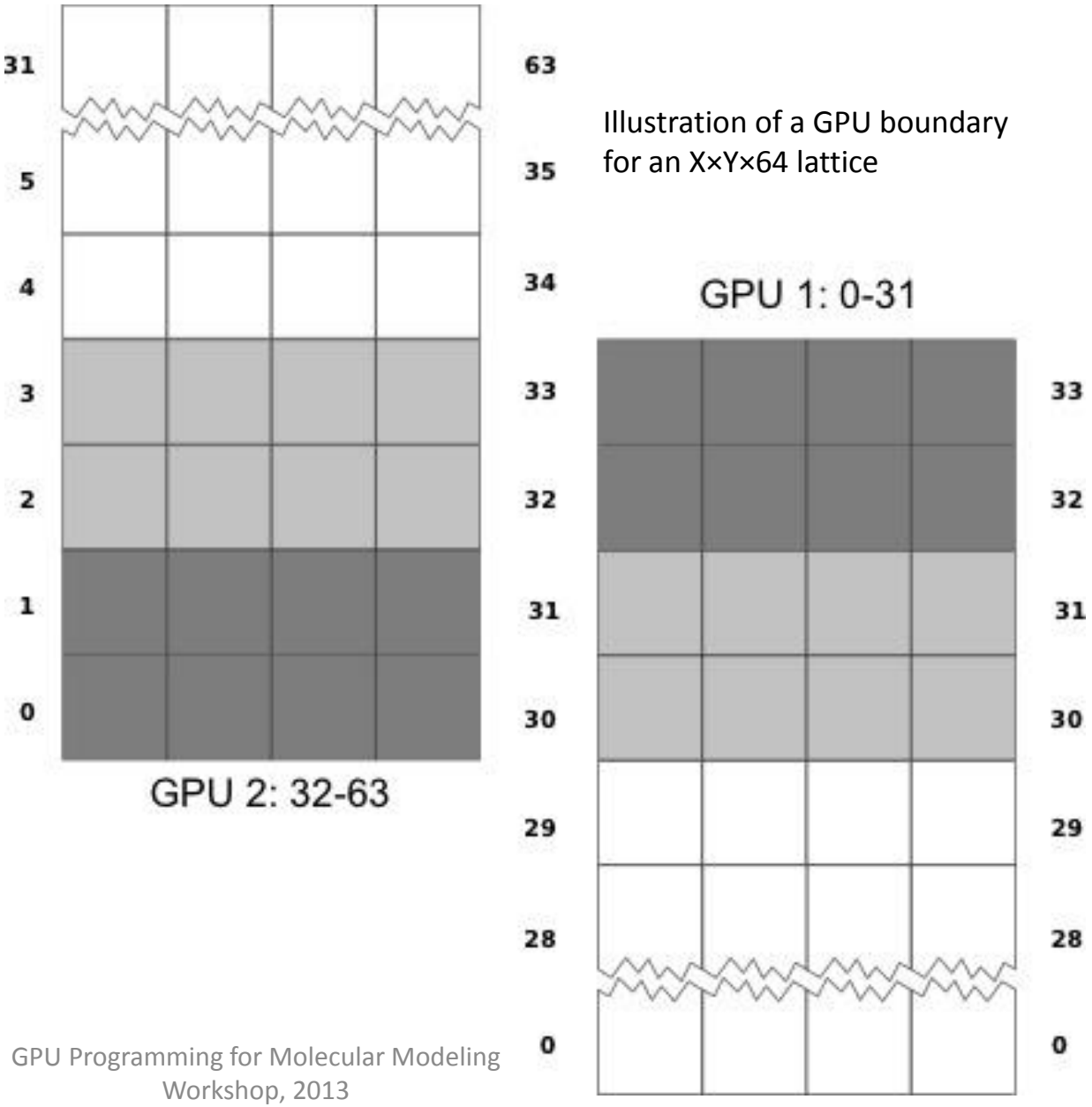


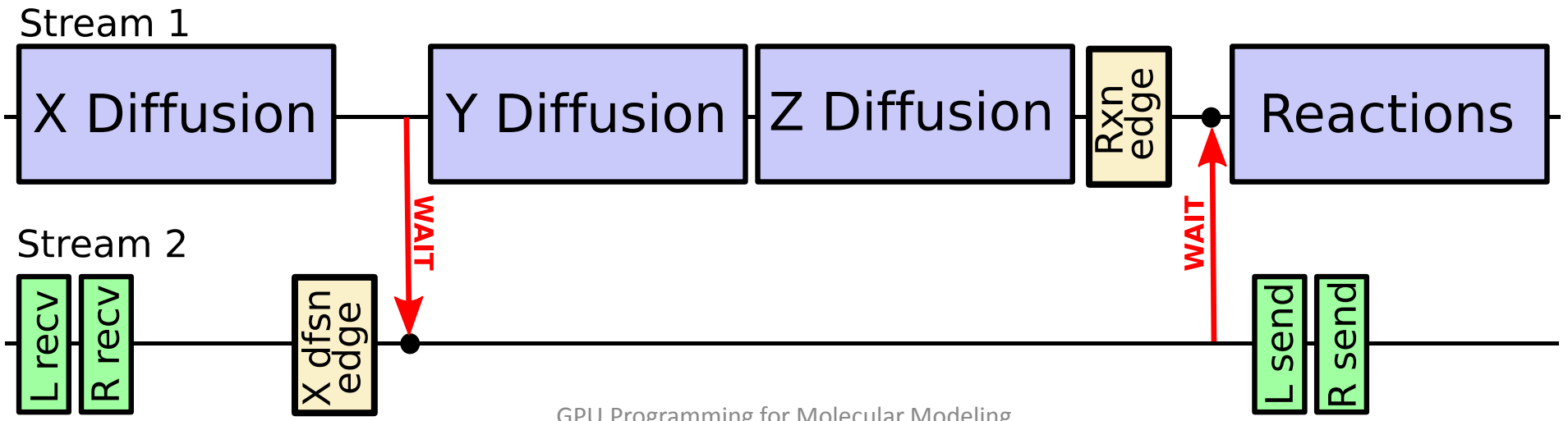
Illustration of a GPU boundary for an X×Y×64 lattice

Dark gray regions are the extensions to cover the neighboring lattice, and must be updated from the neighbor every timestep.

Light gray regions are data that needs a neighbor depends upon.

Hiding Copy Latency

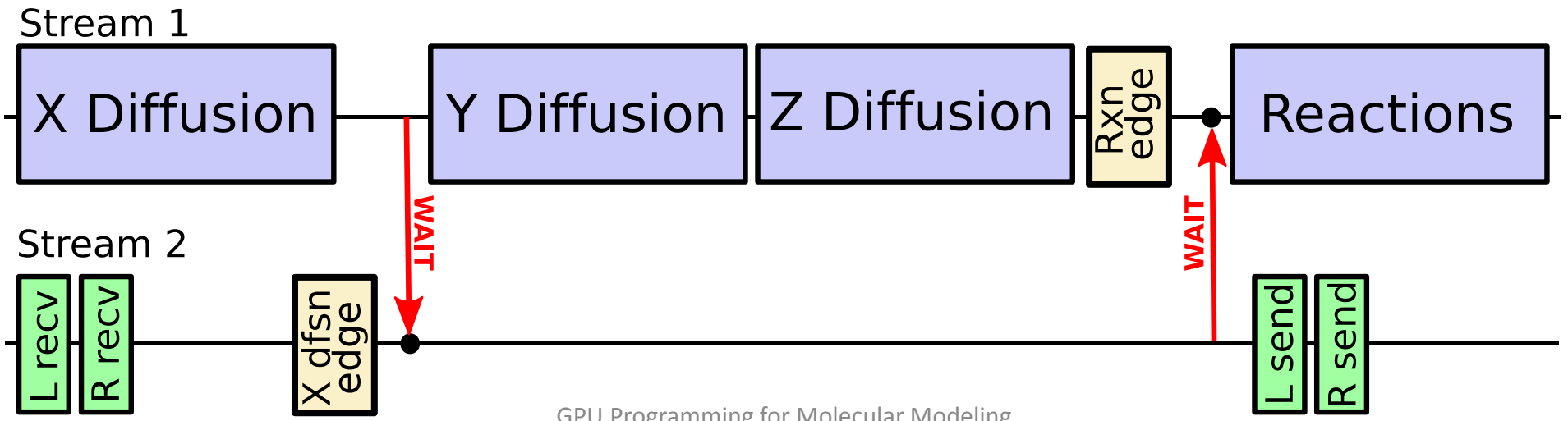
- Overlap non-dependent computation with communication using multiple CUDA streams
- Synchronize streams using events



Hallock, Stone, Roberts, Fry, Luthey-Schulten *Submitted*

Hiding Copy Latency

```
x_kernel<<<inner_grid,threads,0,stream1>>>( … );
cudaMemcpyAsync( … , stream2 );
x_kernel_edge<<<edge_grid, threads, 0, stream2>>>( … );
cudaEventRecord(x_event, stream2);
cudaStreamWaitEvent(stream1, x_event, 0);

y_kernel<<<grid, threads, 0, stream1>>>( … );
```



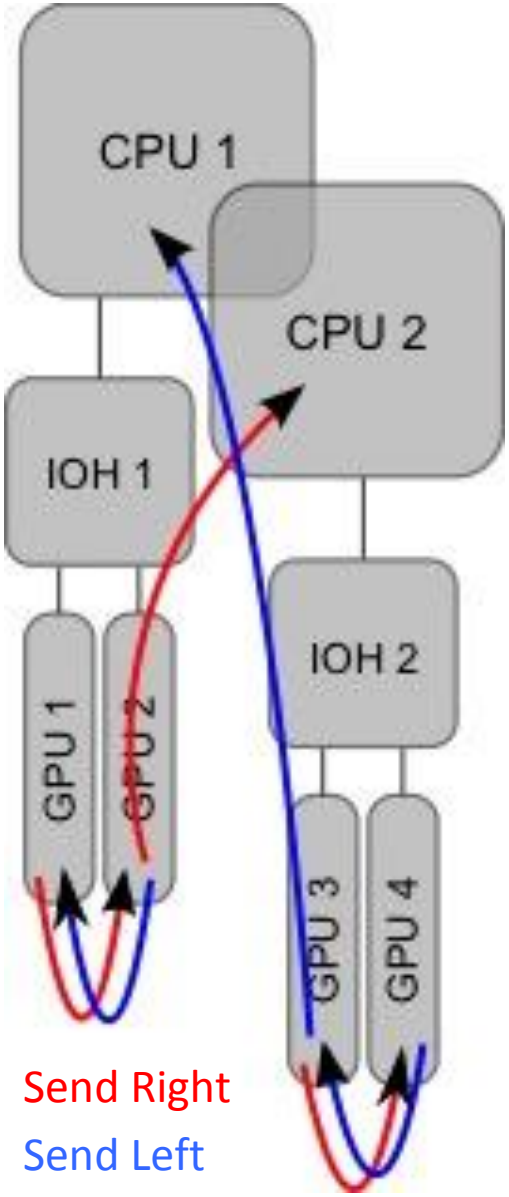
Hallock, Stone, Roberts, Fry, Luthey-Schulten *Submitted*

Decoupled Send and Receive

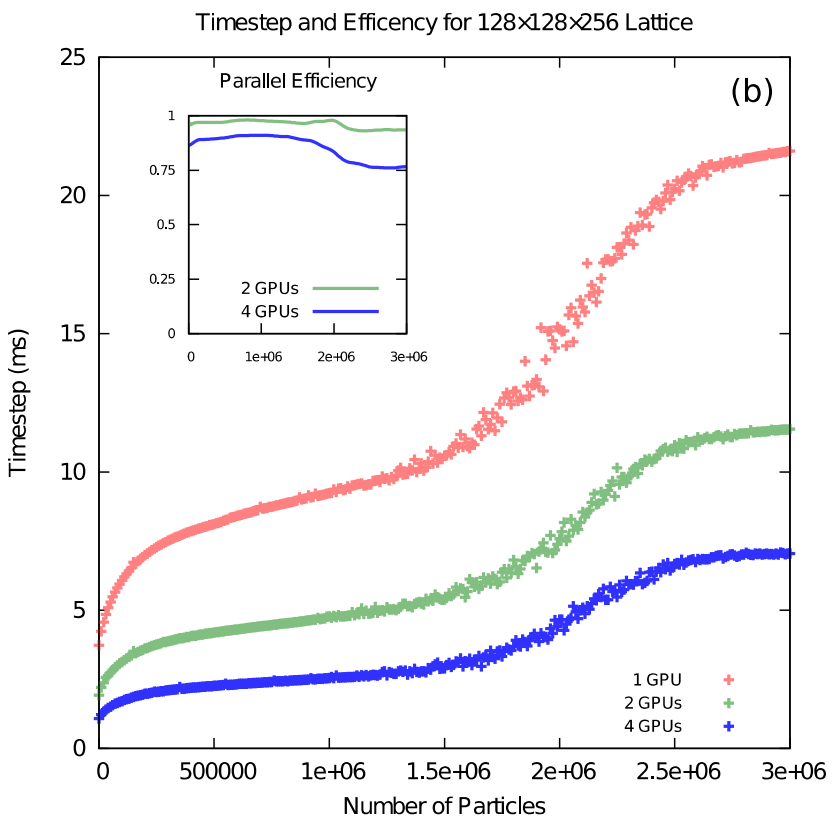
Data sent is not used until the beginning of the next timestep, so we do not need to place the updated lattice information onto the neighboring GPU if not peered.

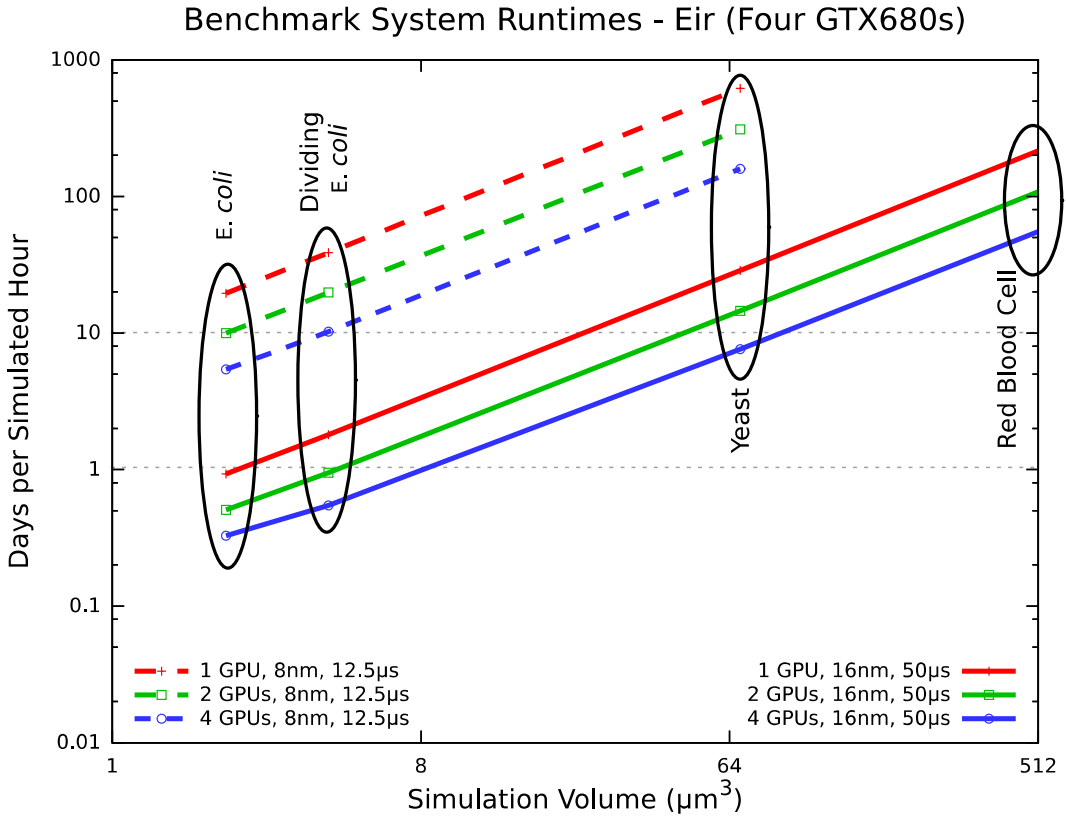
The receive operation for peered GPUs is a local memory copy, unpeered GPUs read from (NUMA local) host memory.

UVA makes it easy to implement both copies by only varying where the destination pointer was allocated.



Multi-GPU Performance



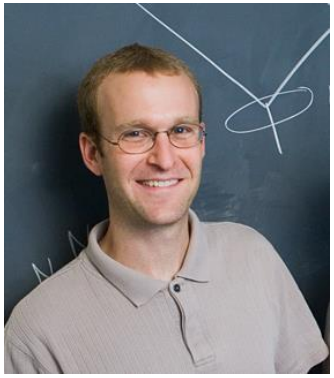


Hallock, Stone, Roberts, Fry, Luthey-Schulten *Submitted*

Acknowledgements

P. Labhsetwar

Elijah Roberts



Zan Schulten

John

John Cole

OLCF
Oak Ridge Leadership Computing Facility

NSF

Center for the Physics of Living Cells

XSEDE
Extreme Science and Engineering Discovery Environment

National Institutes of Health

ILLINOIS
University of Illinois at Urbana-Champaign