GOMC

GPU Optimized Monte Carlo

Jason Mick[‡], Eyad Hailat[†], Kamel Rushaidat[†], Yuanzhe Li[†],
Loren Schwiebert[†], and Jeffrey J. Potoff[‡]

[‡]Department of Chemical Engineering & Materials Science, and
[†]Department of Computer Science, College of Engineering,
Wayne State University, Detroit, MI

Outline

I. Overview:

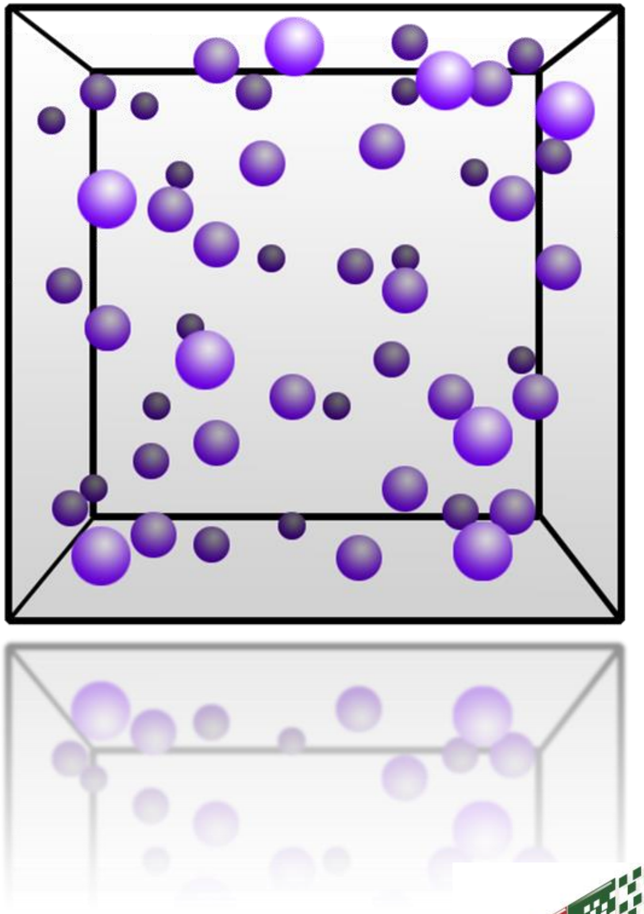
Jason R. Mick : Potoff Group

II. GPU Implementation

Kamel Rushaidat : Schwiebert Group

III. Kepler Enhancements

Yuanzhe Li : Schwiebert Group



GOMC

Jason R. Mick

- Ph.D Candidate
- B.S. – Computer Eng. Oakland University

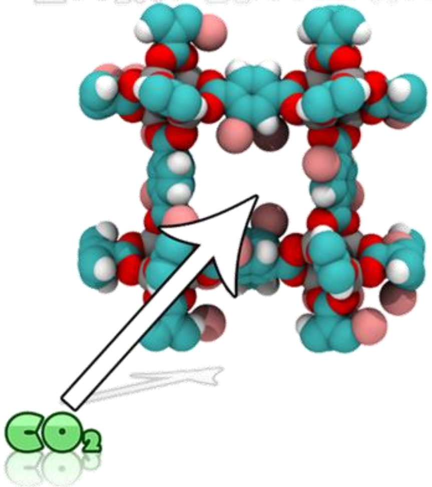
Potoff Group PI: Prof. Jeffrey J. Potoff
Dept. of Chem. Eng. Wayne State Univ.

GOMC

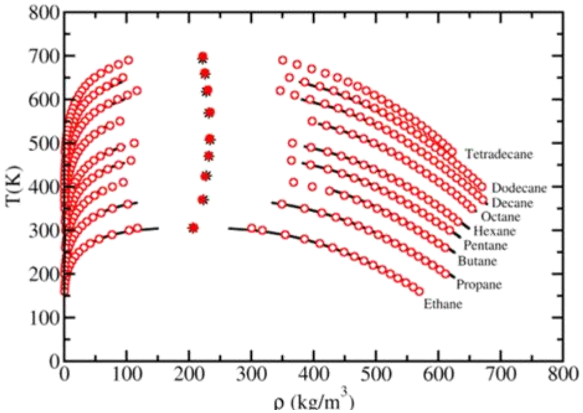
Goals/Motivation/Vision

- Create a multi-ensemble, open source Monte Carlo (MC) molecular simulation engine comparable to existing open source GPU-accelerated molecular dynamics (MD) engines [1,2,3].

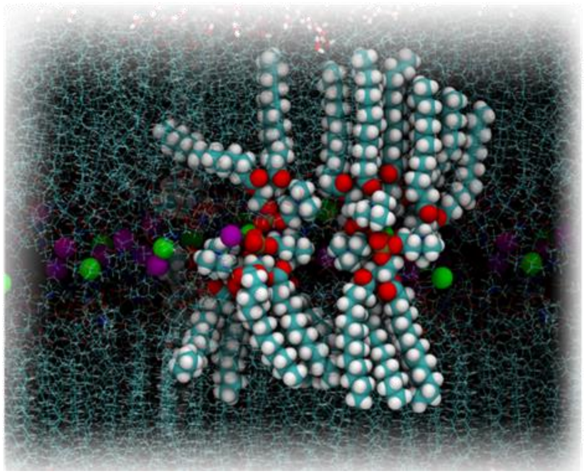
Zeolite Absorption

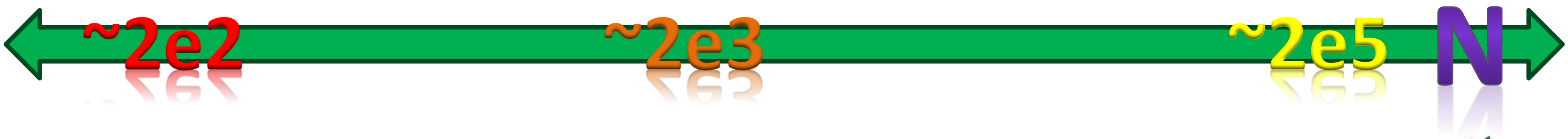


Phase Coexistence



Biomolecule Simulations





[1] Phillips, C.L., *et al.*, *J. Comput. Phys.*, 230(19):p7191-7201 (2011)
[2] Anderson, J.A., *et al.*, *J. Comput. Phys.*, 227(10):5342-5359 (2008).
[3] Brown, W.M., *et al.*, *Comp. Phys. Comm.*, 182(4):p898-911 (2011)

Features

ENSEMBLES

NVT

GEMC[4]

GCMC

Open Source

Obj. Oriented (C++)

CODES

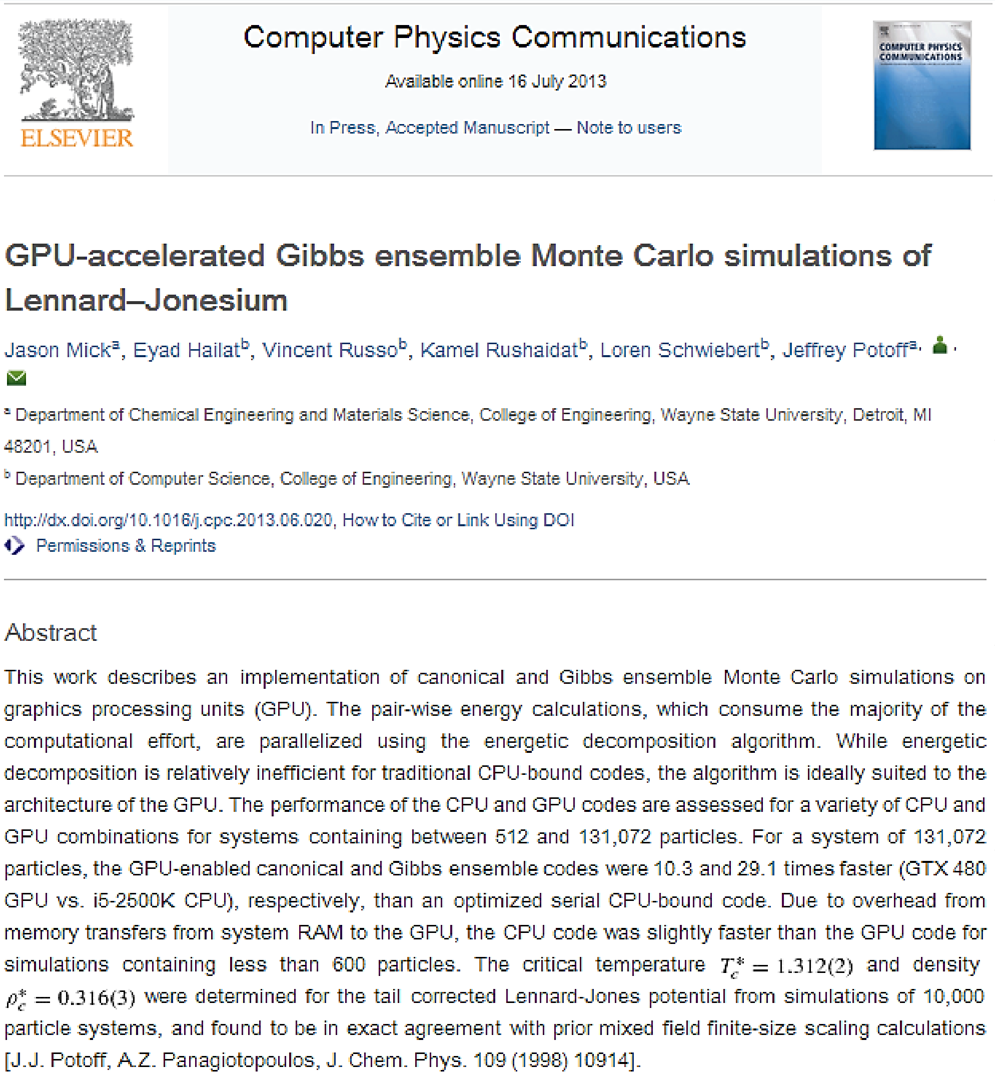
Serial Source

(Side by Side)

CUDA Source

[4] Panagiotopoulos, A.Z., Mol. Phy., **1987**, *61(4)*, 813.

Publications



ELSEVIER

Computer Physics Communications

Available online 16 July 2013

In Press, Accepted Manuscript — Note to users

GPU-accelerated Gibbs ensemble Monte Carlo simulations of Lennard–Jonesium

Jason Mick[a], Eyad Hailat[b], Vincent Russo[b], Kamel Rushaidat[b], Loren Schwiebert[b], Jeffrey Potoff[a],

[a] Department of Chemical Engineering and Materials Science, College of Engineering, Wayne State University, Detroit, MI 48201, USA

[b] Department of Computer Science, College of Engineering, Wayne State University, USA

http://dx.doi.org/10.1016/j.cpc.2013.06.020, How to Cite or Link Using DOI

Permissions & Reprints

Abstract

This work describes an implementation of canonical and Gibbs ensemble Monte Carlo simulations on graphics processing units (GPU). The pair-wise energy calculations, which consume the majority of the computational effort, are parallelized using the energetic decomposition algorithm. While energetic decomposition is relatively inefficient for traditional CPU-bound codes, the algorithm is ideally suited to the architecture of the GPU. The performance of the CPU and GPU codes are assessed for a variety of CPU and GPU combinations for systems containing between 512 and 131,072 particles. For a system of 131,072 particles, the GPU-enabled canonical and Gibbs ensemble codes were 10.3 and 29.1 times faster (GTX 480 GPU vs. i5-2500K CPU), respectively, than an optimized serial CPU-bound code. Due to overhead from memory transfers from system RAM to the GPU, the CPU code was slightly faster than the GPU code for simulations containing less than 600 particles. The critical temperature $T_c^* = 1.312(2)$ and density $\rho_c^* = 0.316(3)$ were determined for the tail corrected Lennard-Jones potential from simulations of 10,000 particle systems, and found to be in exact agreement with prior mixed field finite-size scaling calculations [J.J. Potoff, A.Z. Panagiotopoulos, J. Chem. Phys. 109 (1998) 10914].

Other Journal Papers:

- Parallel Monte Carlo Simulation for the Canonical Ensemble on the GPU (Intl'l J. Parallel, Emerg., & Dist. Sys.)

Conference Papers:

- Jason R. Mick, Jeffrey J. Potoff, Eyad Hailat, Vincent Russo, Loren Schwiebert
  "GPU Accelerated Monte Carlo Simulations In the Gibbs and Canonical Ensembles,"
  AIChE Annual Meeting, 72d, Minneapolis, MN, Oct. 17, 2011.
  http://www3.aiche.org/Proceedings/Abstract.aspx?PaperID=235324

- Jason R. Mick, Eyad Hailat, Yuanzhe Li, Kamel Rushaidat, Loren Schwiebert and Jeffrey J. Potoff,
  "Optimization of a Lennard-Jones Particle Monte Carlo GPU Code,"
  AIChE Annual Meeting, 405d, Pittsburgh, PN, Oct. 31, 2012.
  https://aiche.confex.com/aiche/2012/webprogramadapt/Paper283934.html

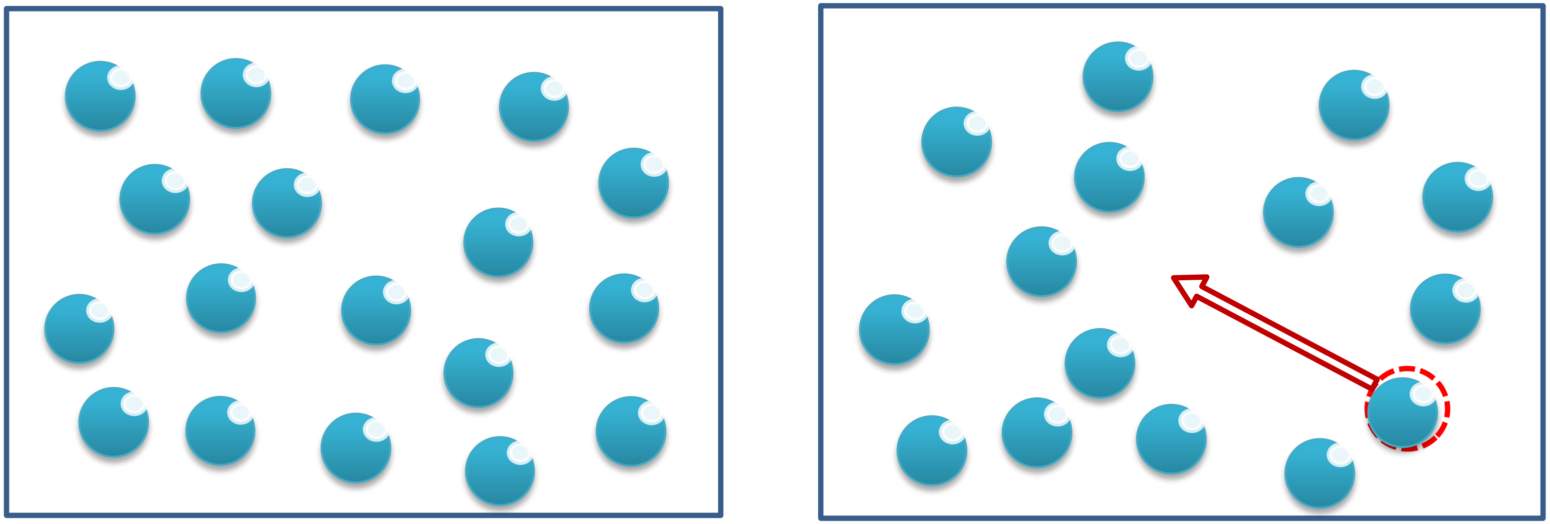
- Jason R. Mick, Kamel Ibrahem, Eyad Hailat, Vincent Russo, Loren Schwiebert and Jeffrey J. Potoff, "GPU Accelerated Configurational Bias Monte Carlo Simulations of Linear Alkanes," AIChE Annual Meeting, 51e, Pittsburgh, PN, October 29, 2012.
  https://aiche.confex.com/aiche/2012/webprogramadapt/Paper283711.html

- Eyad Hailat, Yuanzhe Li, Kamel Rushaidat, Jason R. Mick, Jeffrey J. Potoff, and Loren Schwiebert HPC 2013
  "Fast GPU Monte Carlo Simulation for the Gibbs Ensemble"
  HPC 2013, San Diego, Calif.

GOMC

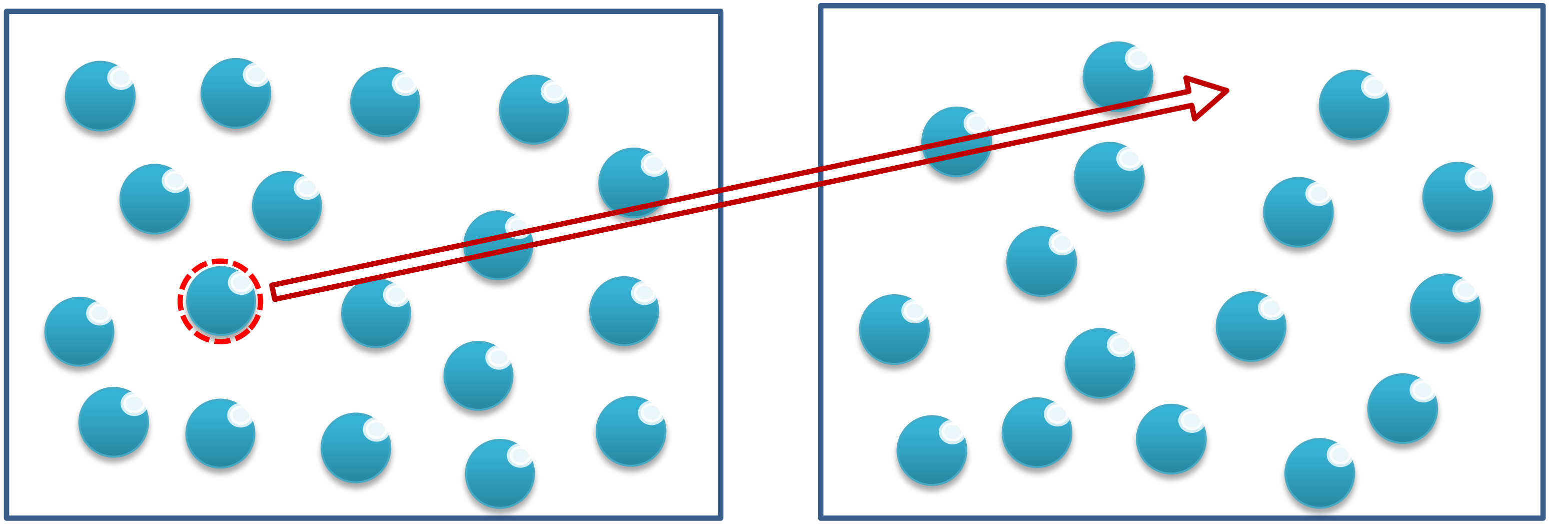
Gibbs Ensemble Monte Carlo (GEMC)

Molecule Displacement



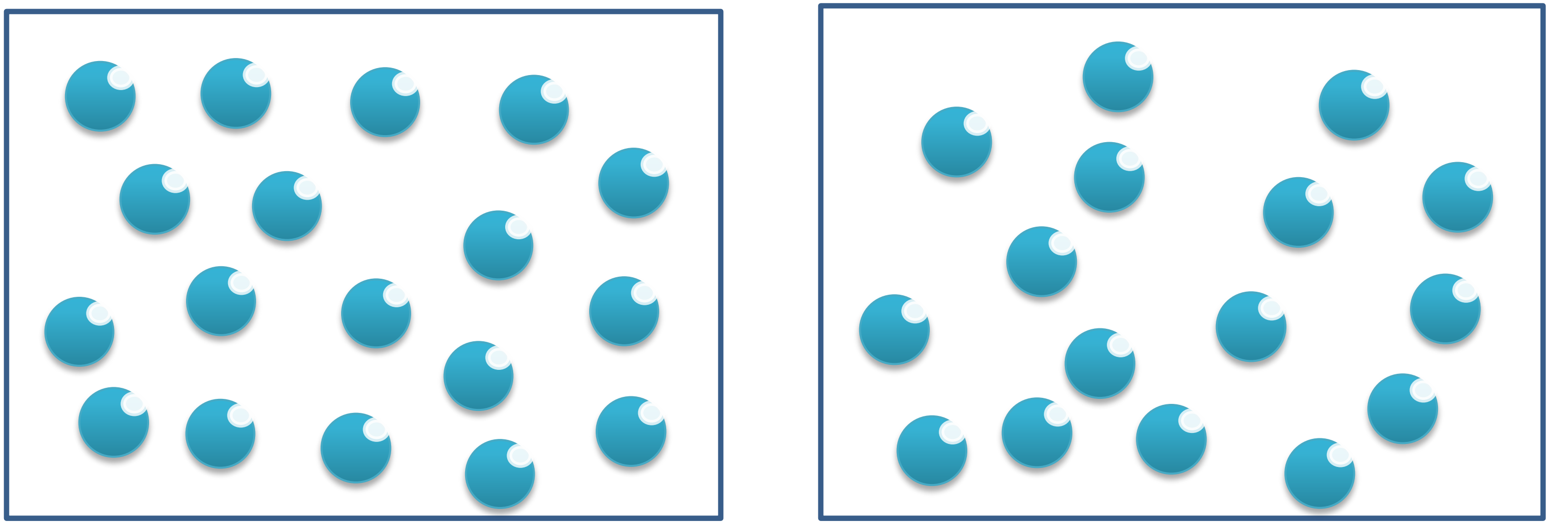
Gibbs Ensemble Monte Carlo (GEMC)

Molecule Transfer



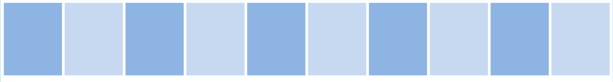
Gibbs Ensemble Monte Carlo (GEMC)

Volume Transfer



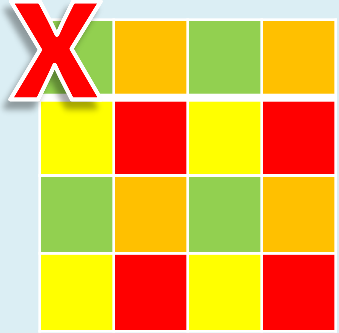
Optimizations

✓



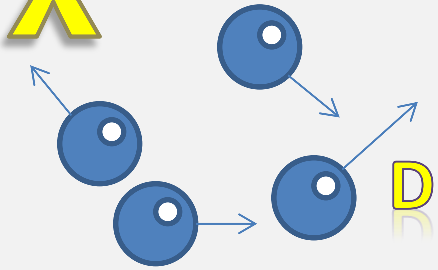
LOOKUP TABLE (ENERGY)

X



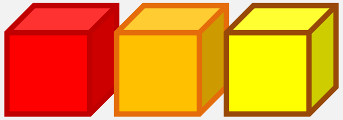
DOMAIN DECOMPOSITION

X



MD-STYLE DISPLACEMENT

✓



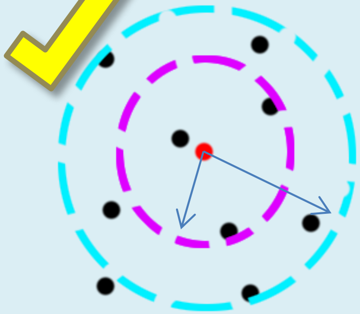
HYBRID – LOAD ONE SIM PER BLOCK [6-7]

✓

$e_i = -1.2156...e-3$

FLOAT v. DOUBLE

✓

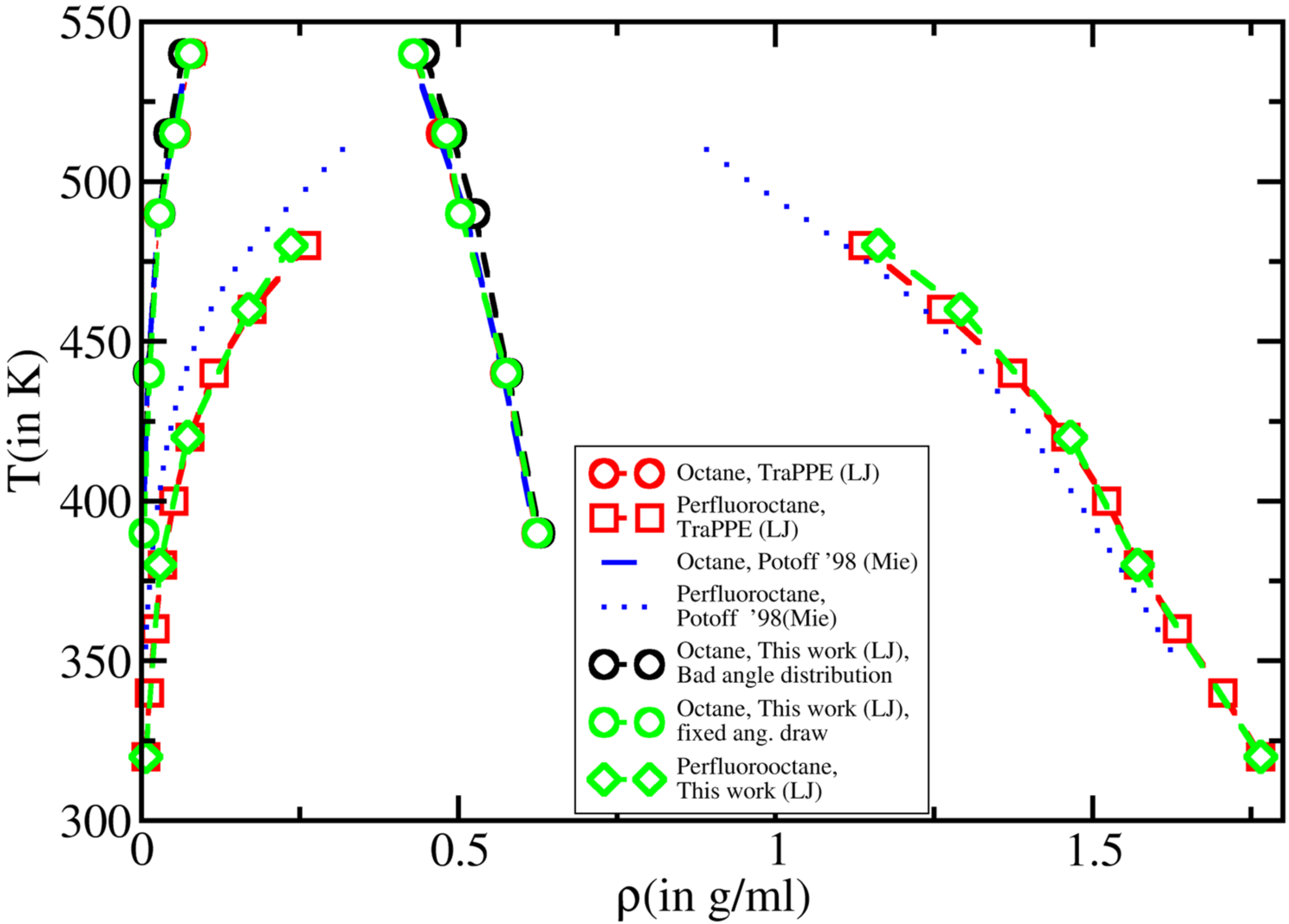


CELL LIST

[6] Kim, J., et al., J. Chem. Theory & Comp., 7(10): p3208-3222 (2011)

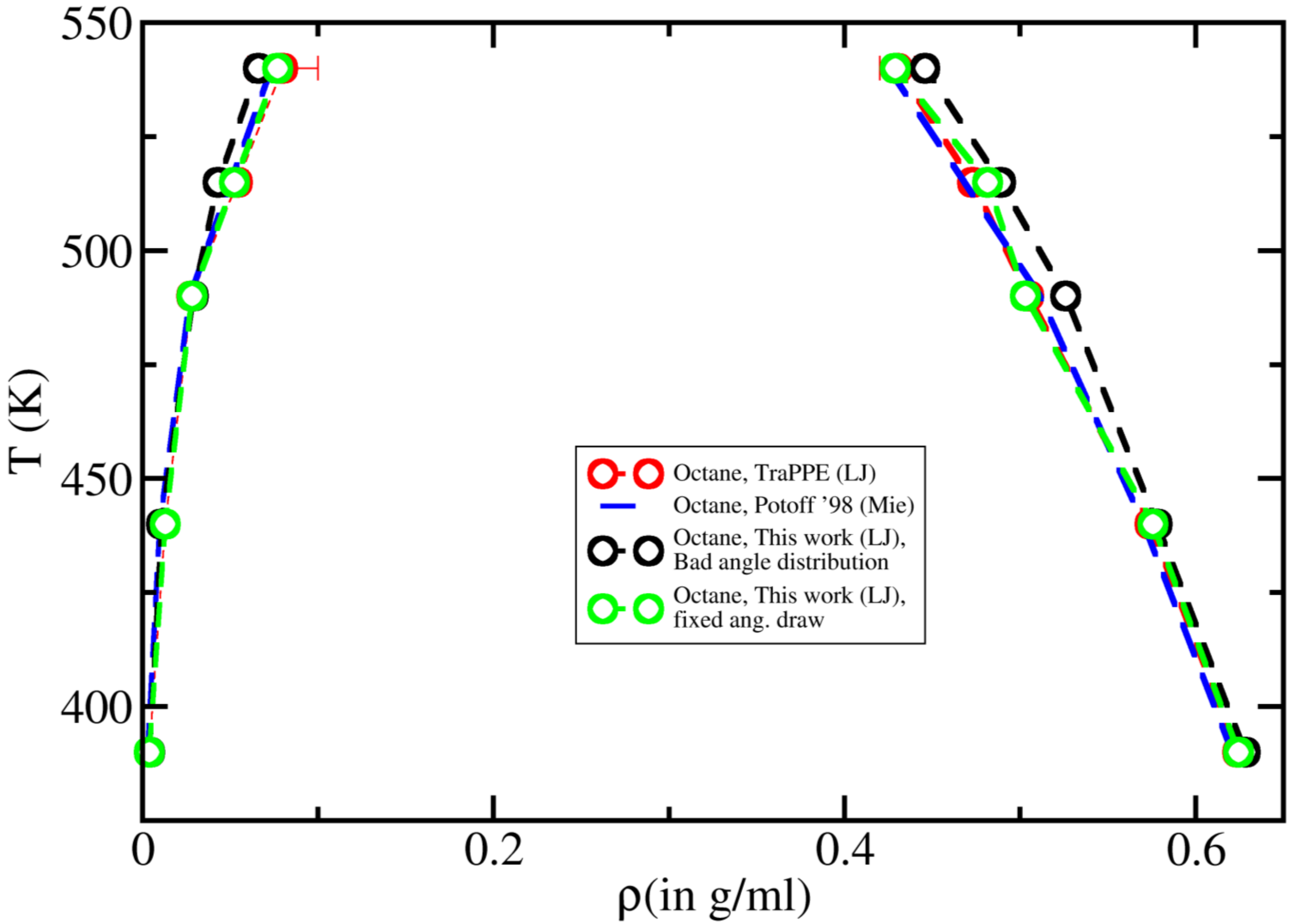
[7] Kim, J., et al., J. Chem. Theory & Comp., 8(5): p1684-1693 (2012)

Validation: n-Fluoroalkanes, n-Alkanes

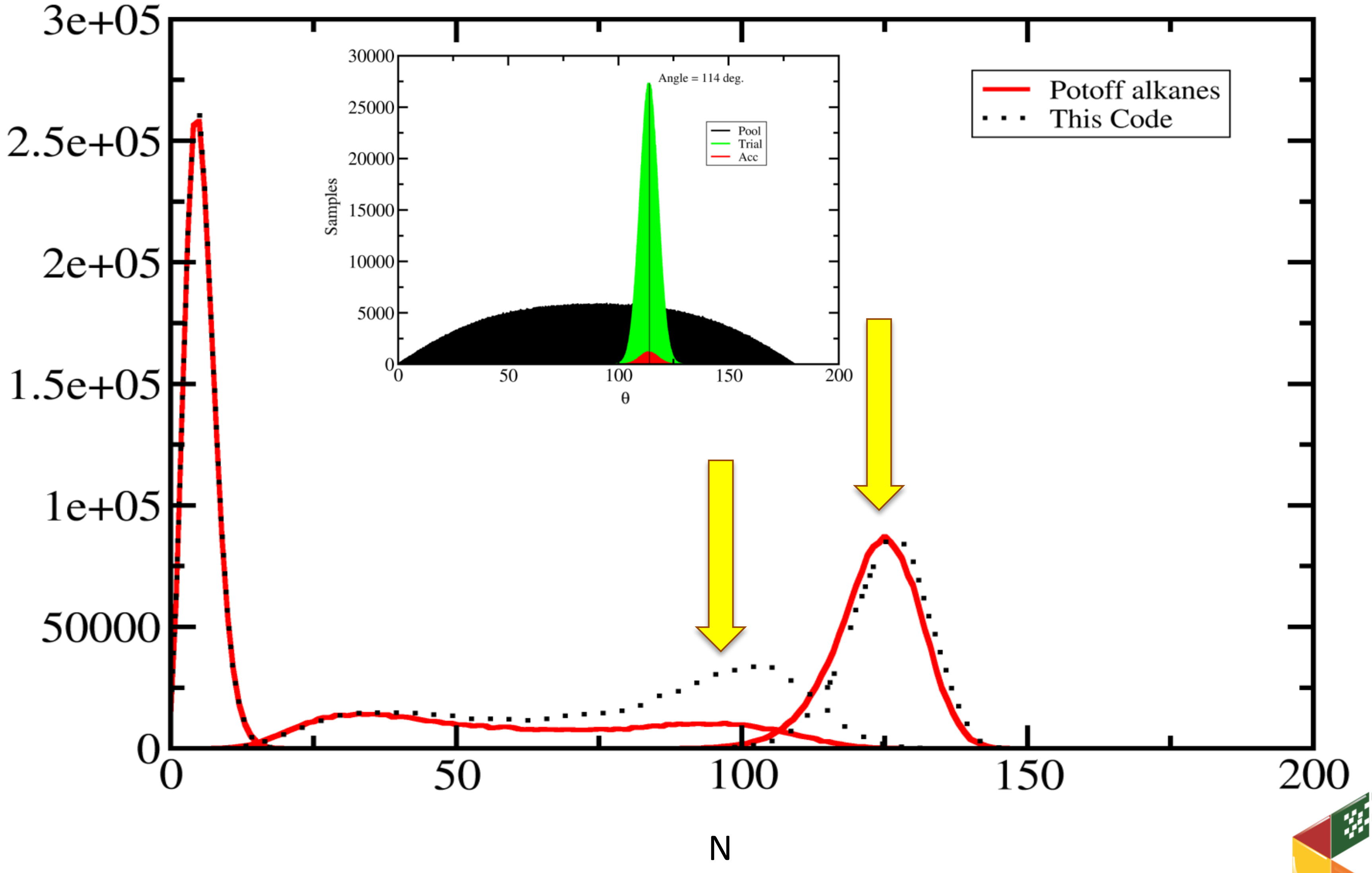


GOMC

Angle Picking: A Cautionary Tale

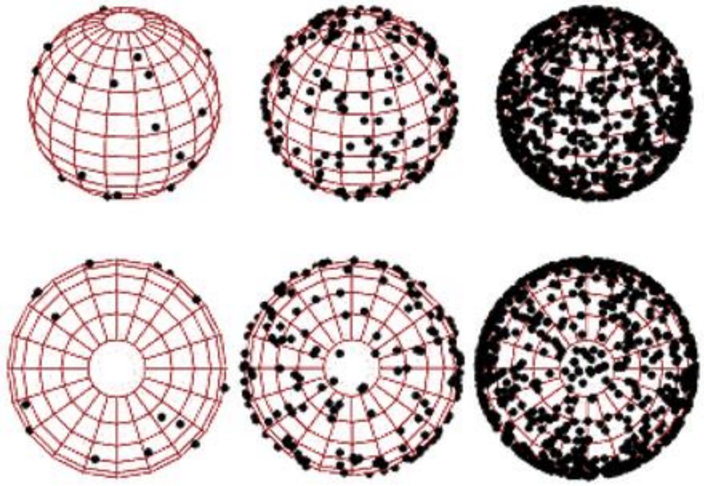


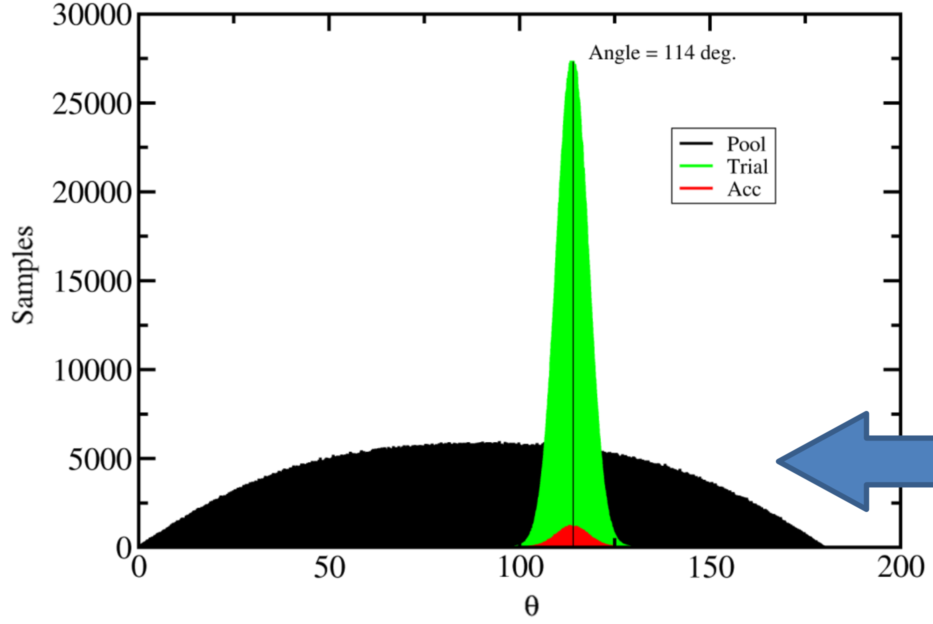
Angle Picking: A Cautionary Tale

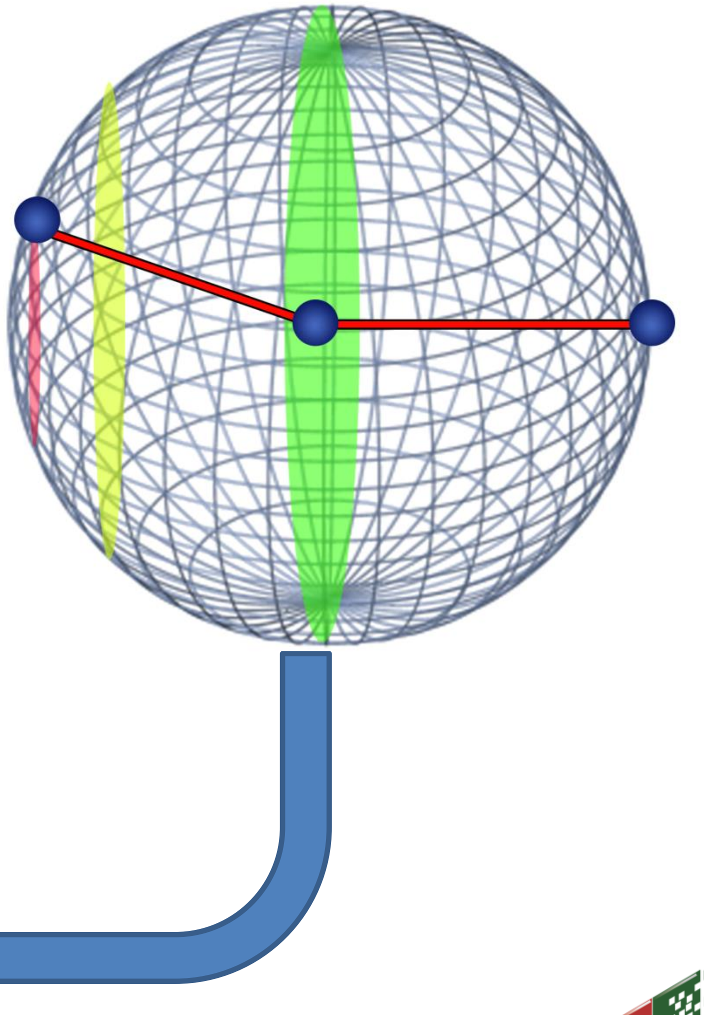


GOMC

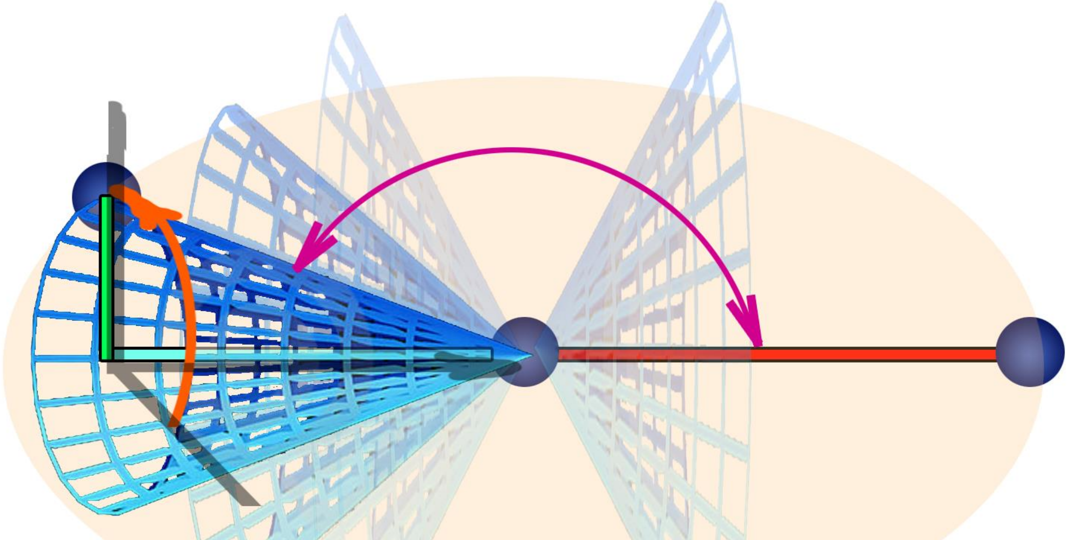
Angle Picking: A Cautionary Tale

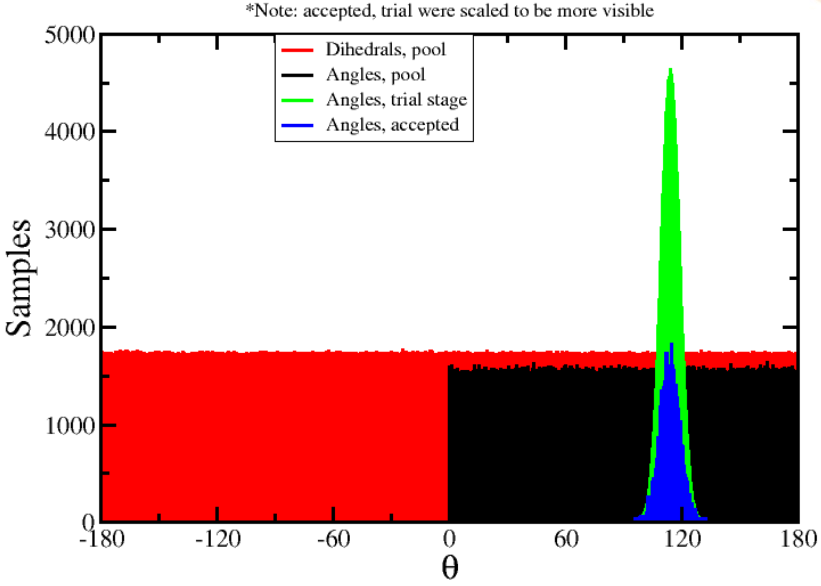






Angle Picking: A Cautionary Tale



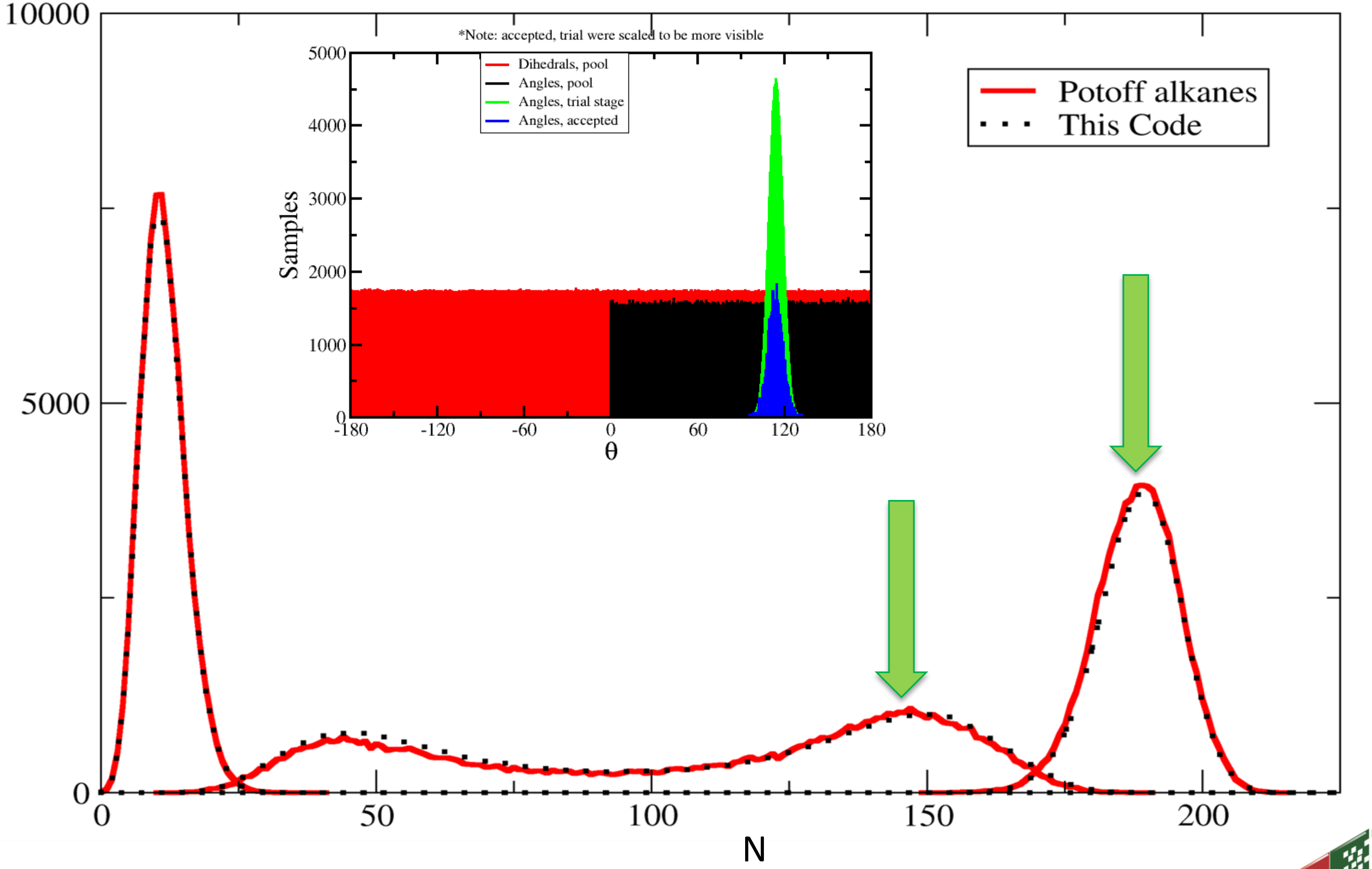


*Note: accepted, trial were scaled to be more visible

N

GOMC

Angle Picking: A Cautionary Tale



Kamel Rushaidat

- Ph.D Candidate
- B.S., M.S. Computer Sci.— Jordan Univ. of Science and Tech.

Schwiebert Group
Dept. of Comp. Sci.

PI: Prof. Loren Schwiebert
Wayne State Univ.

GOMC

GPU: Computing Resource Hierarchy

THREADS

WARPS

Auto-synchronous

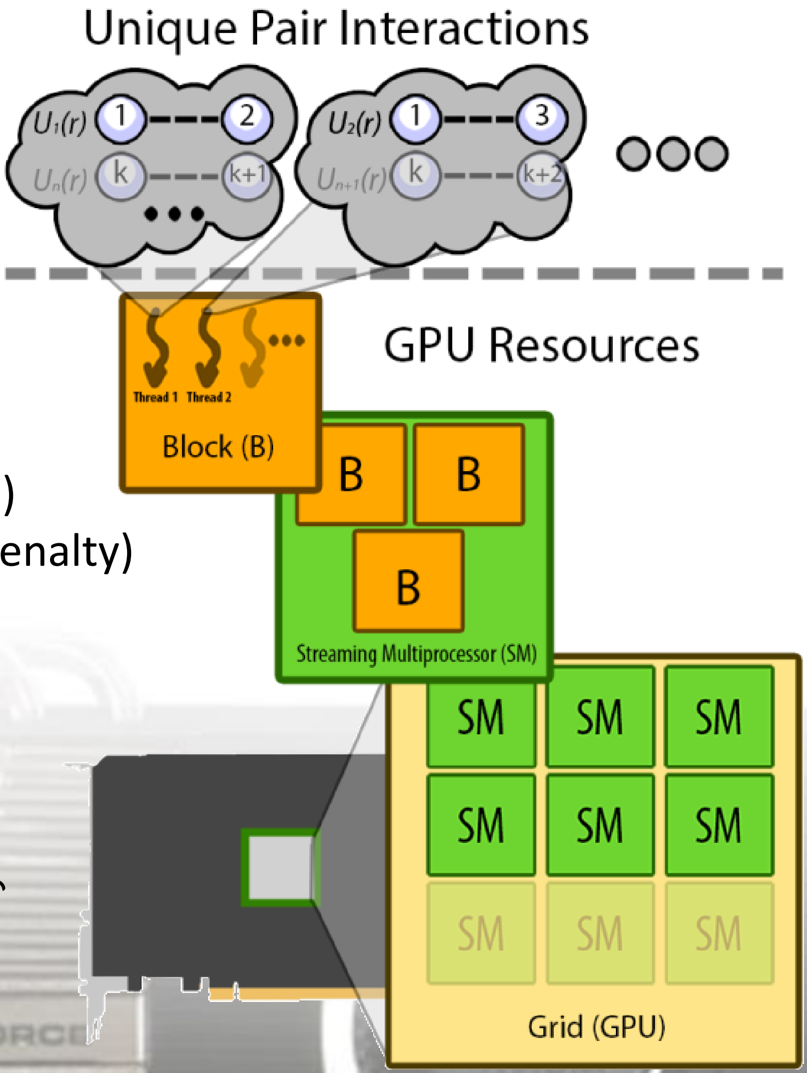
BLOCKS

CUDA __syncthreads()
(small performance penalty)

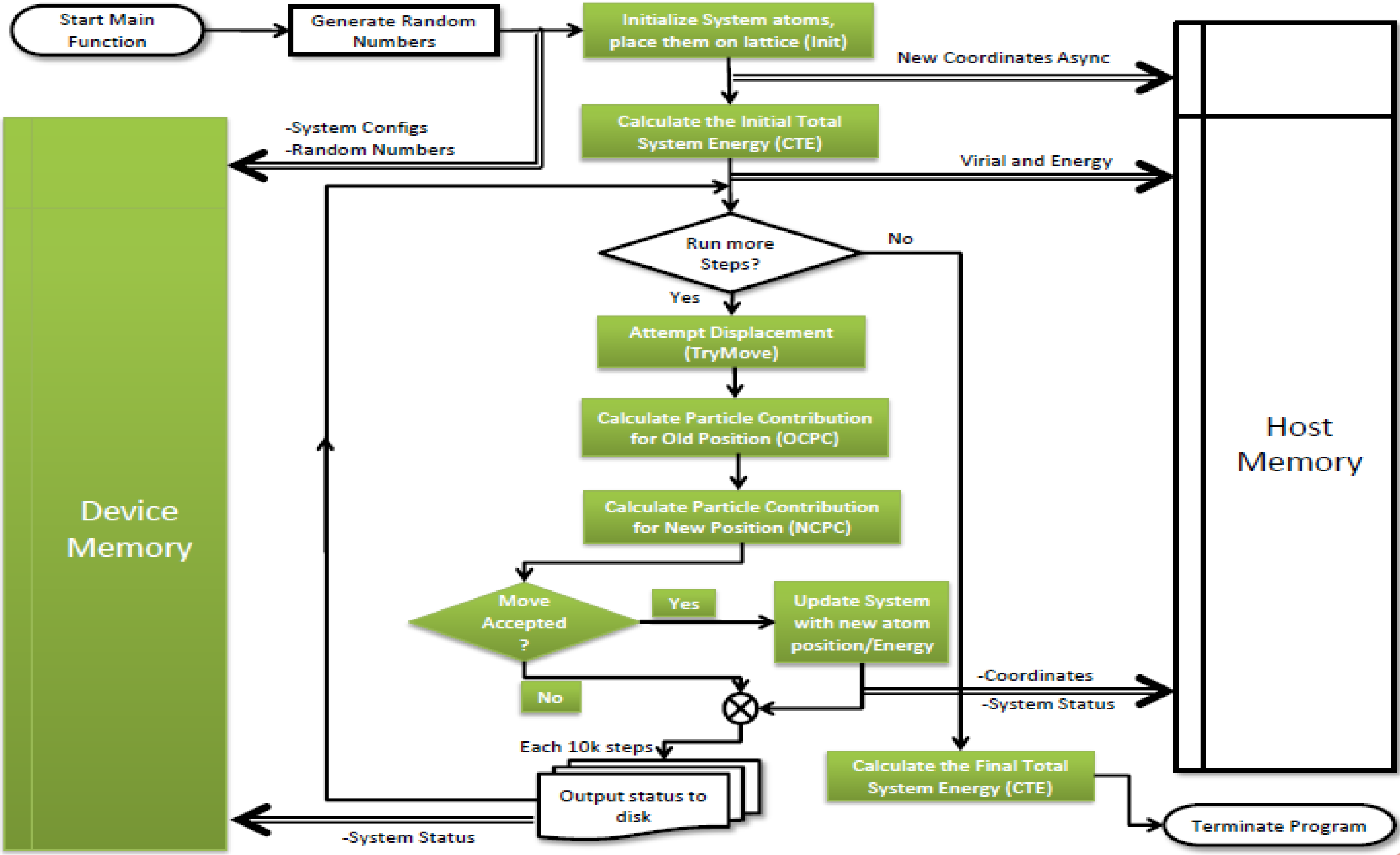
SMPs

DEVICE

Atomic counter-sync
(large performance
penalty)



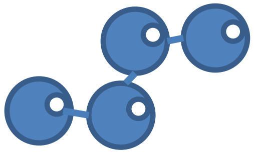
GPU Code Flow


Start Main Function
Generate Random Numbers
Initialize System atoms, place them on lattice (Init)
New Coordinates Async
-System Configs
-Random Numbers
Calculate the Initial Total System Energy (CTE)
Virial and Energy
Run more Steps?
No
Yes
Attempt Displacement (TryMove)
Calculate Particle Contribution for Old Position (OCPC)
Calculate Particle Contribution for New Position (NCPC)
Device Memory
Host Memory
Move Accepted ?
Yes
Update System with new atom position/Energy
No
-Coordinates
-System Status
Each 10k steps
Output status to disk
Calculate the Final Total System Energy (CTE)
Terminate Program
-System Status

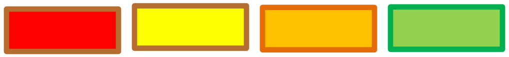
GOMC

Basic Memory Approach

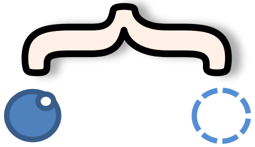
Things we write once per move and save between moves, or things that never change…



Particle Coordinates:
Global Memory

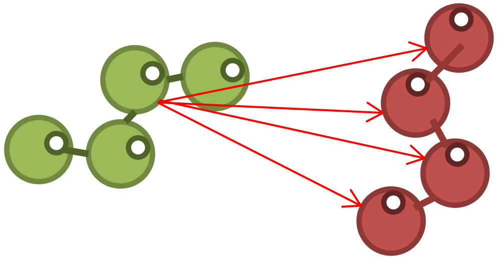


Block statistics:
Global Memory



Adjustable move constants:
Global Memory

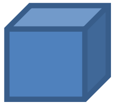
Things we only use inside the move and that need to be accessed fast…



Energetics:
Shared memory (thread calcs.)

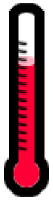
Global memory (block totals)

Volume



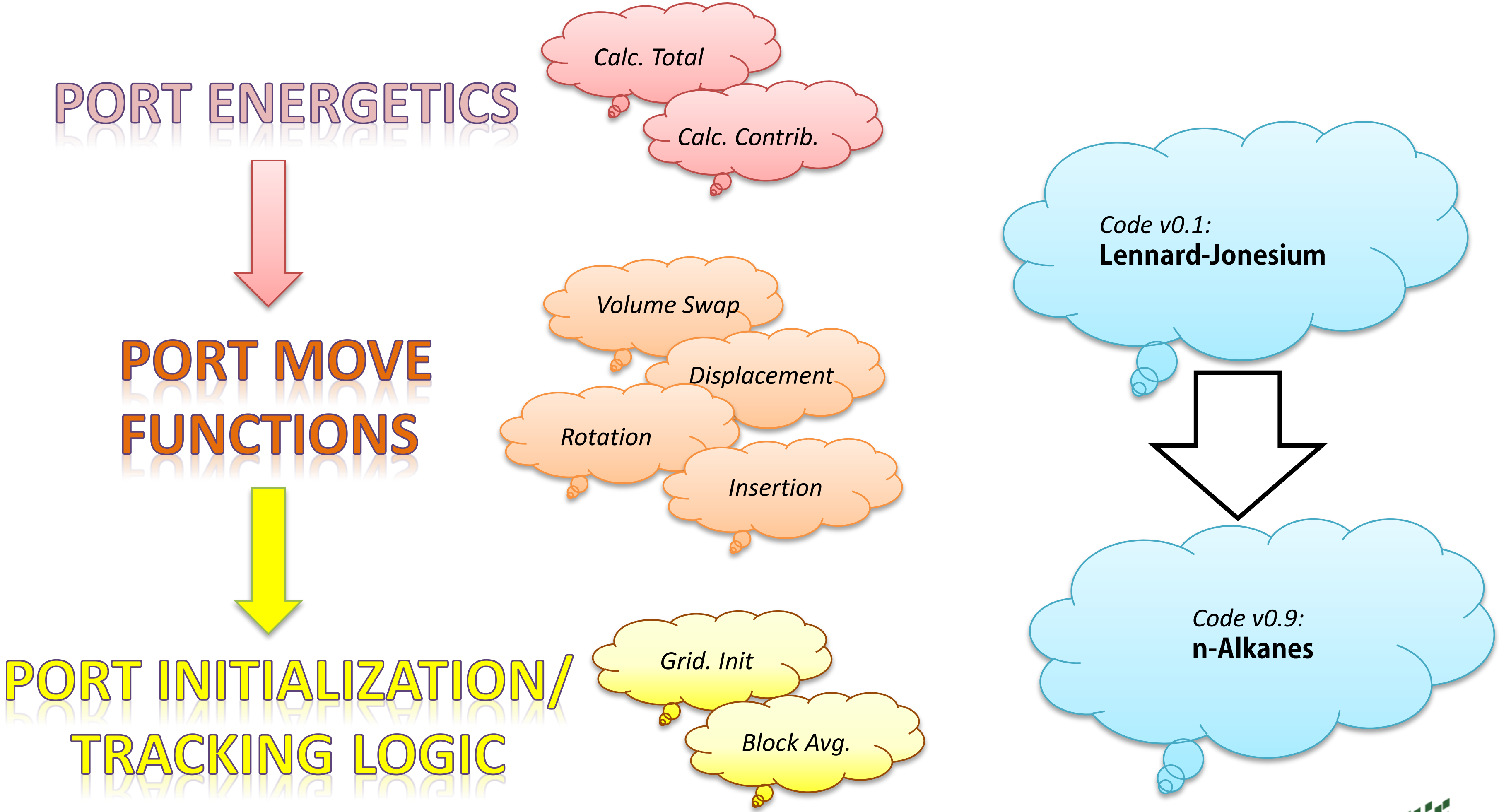
Index

Trial variables:
Register variables



Temperature:
Constant Memory

Basic Logic Porting Approach



PORT ENERGETICS

Calc. Total

Calc. Contrib.

PORT MOVE FUNCTIONS

Volume Swap

Displacement

Rotation

Insertion

PORT INITIALIZATION/ TRACKING LOGIC

Grid. Init

Block Avg.

Code v0.1:
Lennard-Jonesium

Code v0.9:
n-Alkanes

GPU: Computing Resource Hierarchy

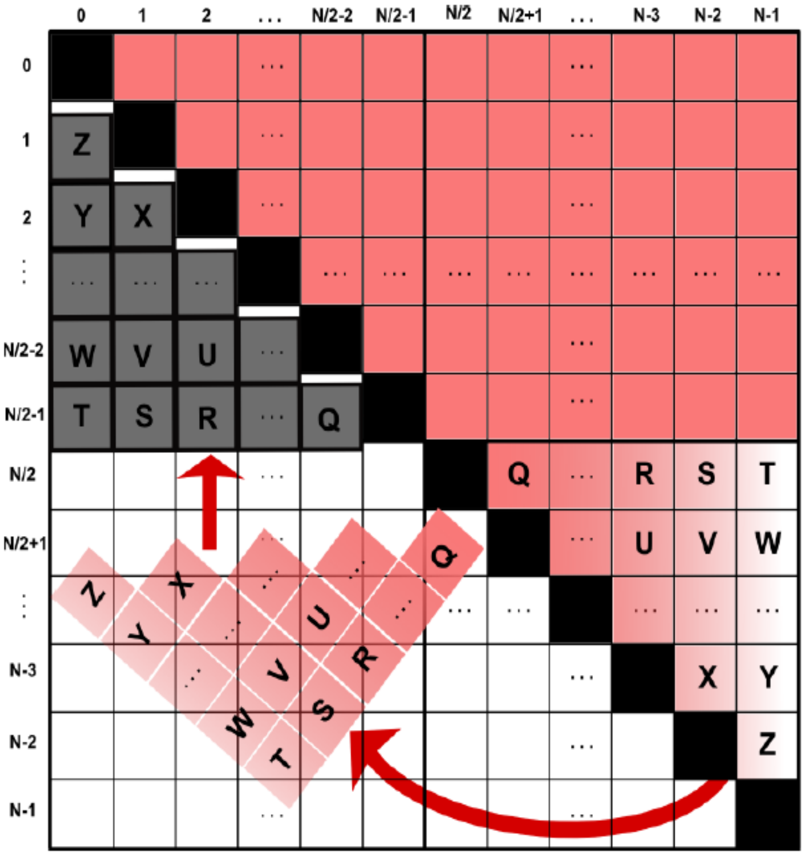
- Memory coalescing to fetch particle positions.
- Loop unrolling.
- Thread load balancing.
- GPU hardware functions.
- Texture lookup table.

Distributing $O(N^2)$ Operations on Unique Index Pairs

Traditional indexing in serial code:
Double 'for' loop

```
for i in 1 to N
  for j in i to N
```

HOW TO PORT?



- 🟥 Unique pair
- ⬛ Same particle
- ▨ Mapped pair
- □ Duplicate pair

Need to calculate index

Naïve thought:
Use for loop to get thread to correct index

Problem:
Thread divergence

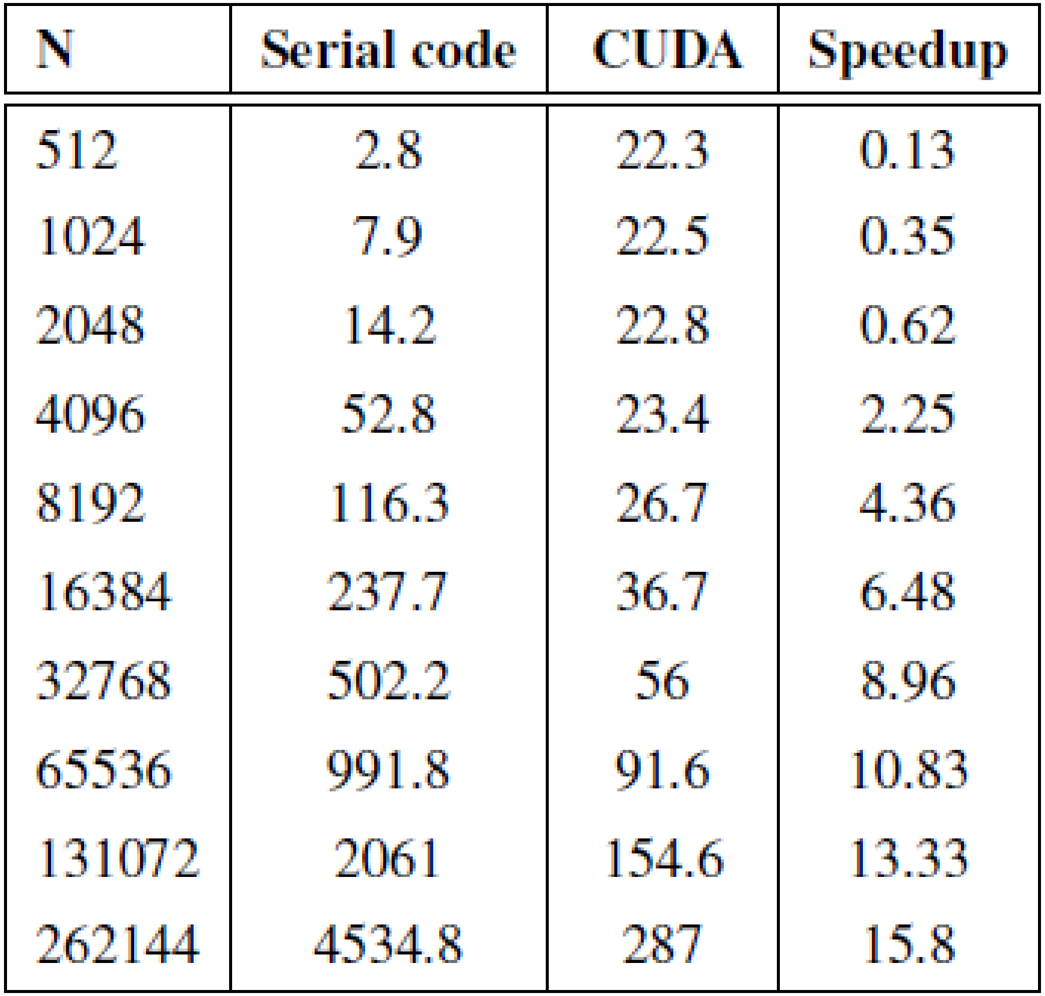
Naïve thought #2:
Just count all pairs, then divide by two

Problem:
Thread waste – one in two threads does a redundant calc.

Solution:
Remap indexes!

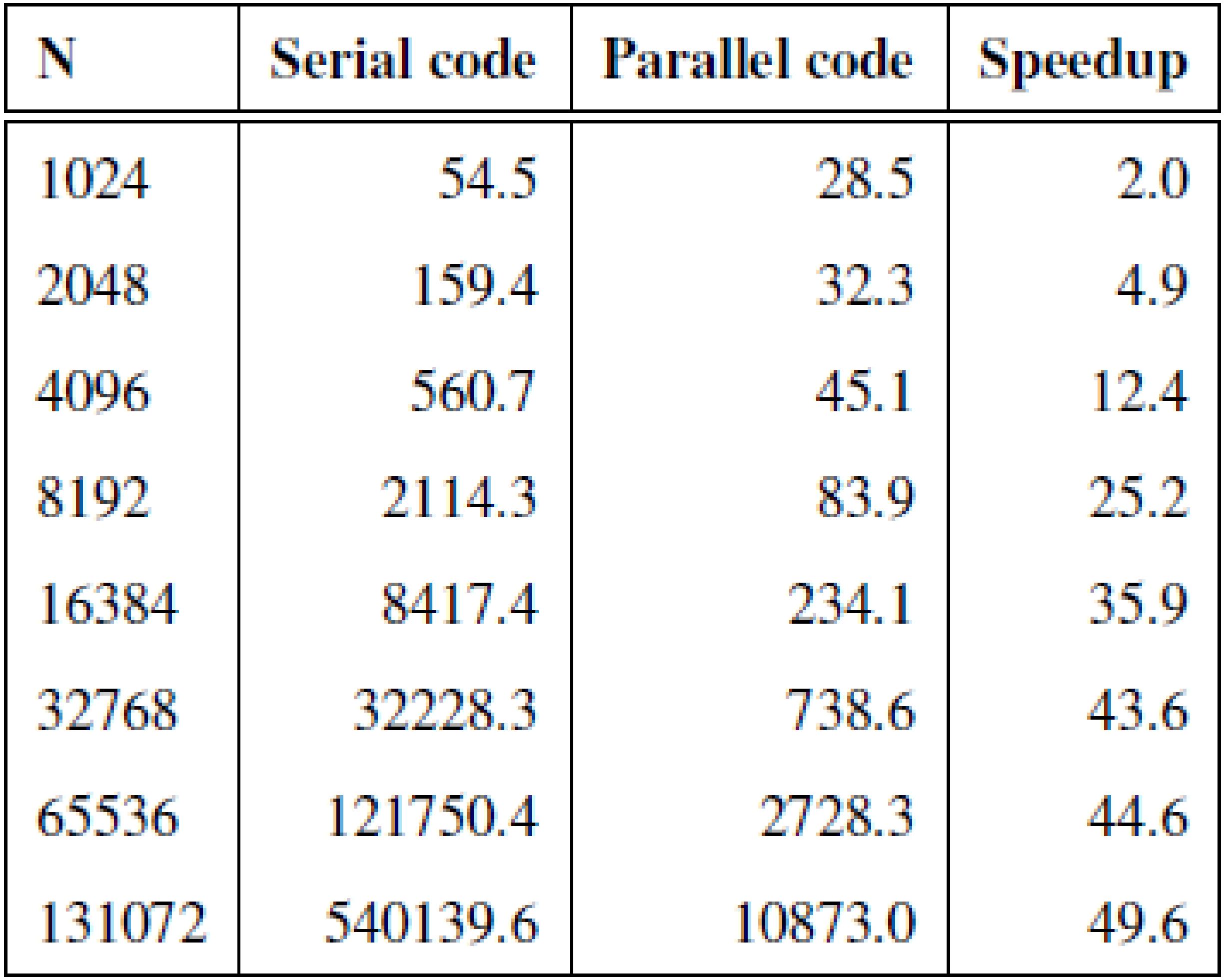
Often, not N^2 threads available, plus application is operation limited not bandwidth limited:
Thus 4*N threads is performance sweet spot

GCMC Results



N	Serial code	CUDA	Speedup
512	2.8	22.3	0.13
1024	7.9	22.5	0.35
2048	14.2	22.8	0.62
4096	52.8	23.4	2.25
8192	116.3	26.7	4.36
16384	237.7	36.7	6.48
32768	502.2	56	8.96
65536	991.8	91.6	10.83
131072	2061	154.6	13.33
262144	4534.8	287	15.8

GEMC Results



N	Serial code	Parallel code	Speedup
1024	54.5	28.5	2.0
2048	159.4	32.3	4.9
4096	560.7	45.1	12.4
8192	2114.3	83.9	25.2
16384	8417.4	234.1	35.9
32768	32228.3	738.6	43.6
65536	121750.4	2728.3	44.6
131072	540139.6	10873.0	49.6

Current Work

- Current development path focuses on simulating molecules of arbitrary geometry with established methods.
- Structure the code to enable easy data movement to and from the GPU.
- Refactor the code to have structures of arrays, as there are many arrays involved with Molecules.
- Float vs Double.

Yuanzhe Li

- Master’s student
- B.S. Computer Sci.— XiDian University, China

Schwiebert Group
Dept. of Comp. Sci.

PI: Prof. Loren Schwiebert
Wayne State Univ.

GOMC

Meet Kepler

Hardware Improvements:

- More resource:
  13 SMX, 192 cores per SMX,
  2048 threads per SMX
- Two new features:
  Dynamic Parallelism,
  WarpShuffle

Why Use Kepler?

Pre-Kepler -- Kernel Issues:

- So much expense on transporting data
- So many synchronizations
- High latency caused by the shared memory

Kepler -- Potential Gains:

- Using Dynamic Parallelism to move the original CPU workload to GPU
- Using WarpShuffle to exchange the data between two threads

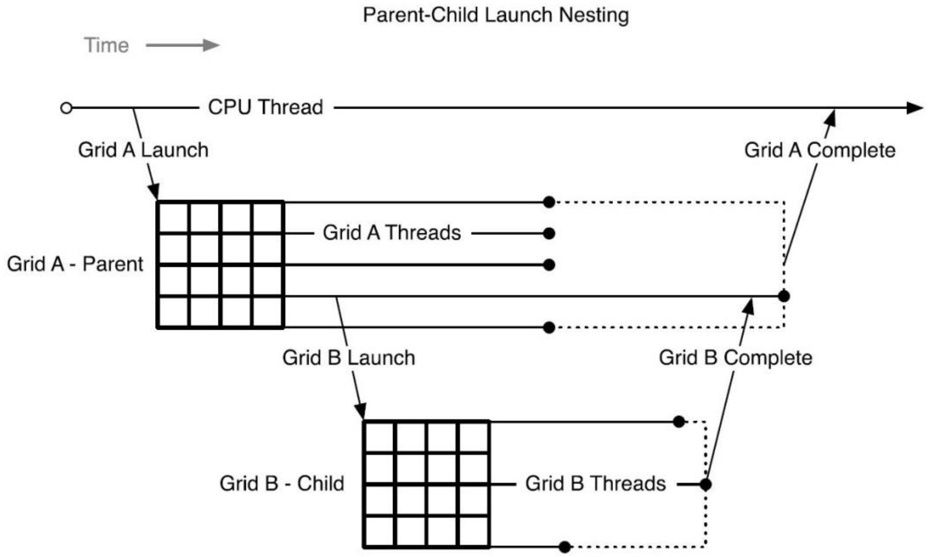
GEMC – Dynamic Parallelism

Dynamic Parallelism

- A parent kernel
- Three move kernels are set to child kernels
- All data kept on chip

Challenge

- How to reduce the idle time from synchronization
- Ensuring all data is synchronized
- Tuning block size accordingly



Warp Shuffle – Register Optimization

WarpShuffle

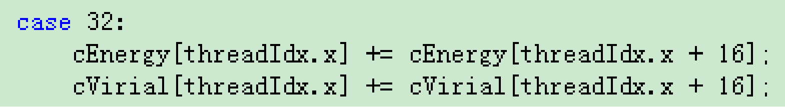
- Between threads within a warp.
- Making use of registers

Challenge

- Only 4 bytes of data can be moved per thread
- The parallelism sometimes is not good

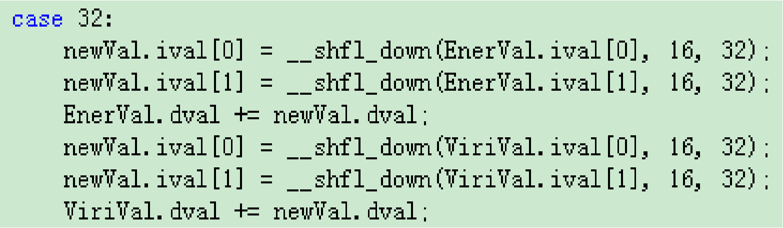
Comparison

- Code with shared memory



```
case 32:
    cEnergy[threadIdx.x] += cEnergy[threadIdx.x + 16];
    cVirial[threadIdx.x] += cVirial[threadIdx.x + 16];
```

- Code with WarpShuffle

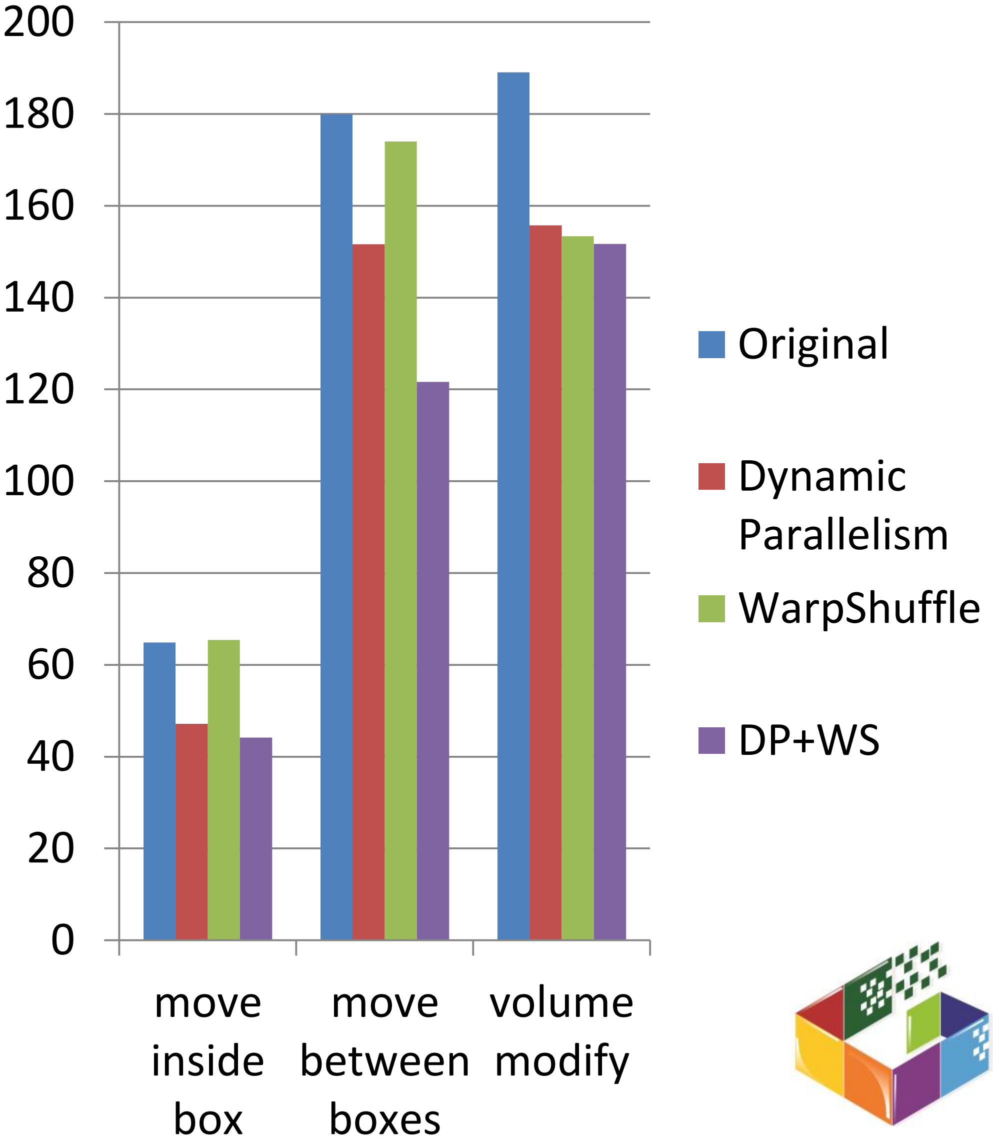


```
case 32:
    newVal.ival[0] = __shfl_down(EnerVal.ival[0], 16, 32);
    newVal.ival[1] = __shfl_down(EnerVal.ival[1], 16, 32);
    EnerVal.dval += newVal.dval;
    newVal.ival[0] = __shfl_down(ViriVal.ival[0], 16, 32);
    newVal.ival[1] = __shfl_down(ViriVal.ival[1], 16, 32);
    ViriVal.dval += newVal.dval;
```

Kepler – Preliminary Results

- The histogram illustrates the running time for each function
- The table shows the running time for all three functions

System size	32K particles
Original code on GTX 480	481.47s
Original code on K20c	343.536s
Dynamic Parallelism	285.746s
WarpShuffle	280.312s
DP+WS	270.506s



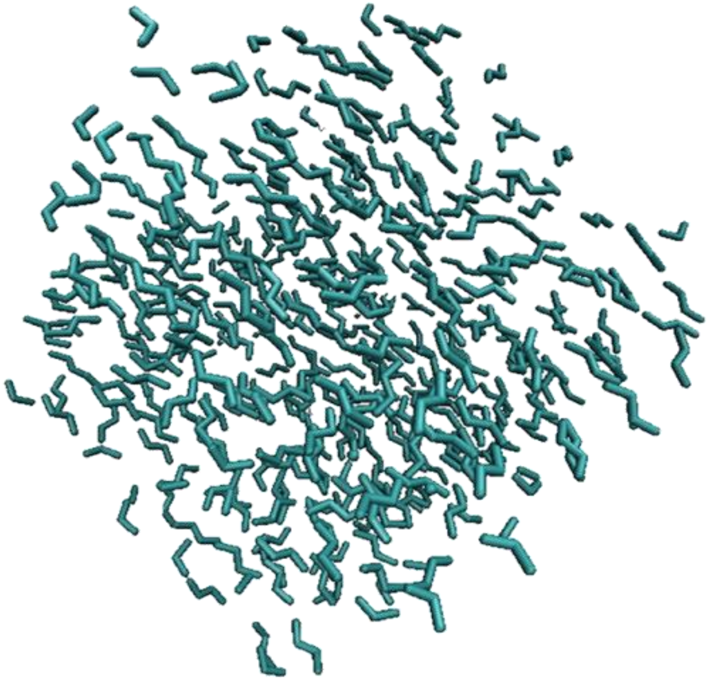
Concluding Remarks

- Coding a robust, open source code is a lot of work, but vital for community research progress

- GPU is an ideal accelerator for MC simulations, particularly compex ones

- Optimization tactics (lookup tables, streams, dynamic parallelism, multiple simultaneous simulations) are necessary to beat optimized serial neighbor list code

- *Kepler* provides the ability to cut memory transfers

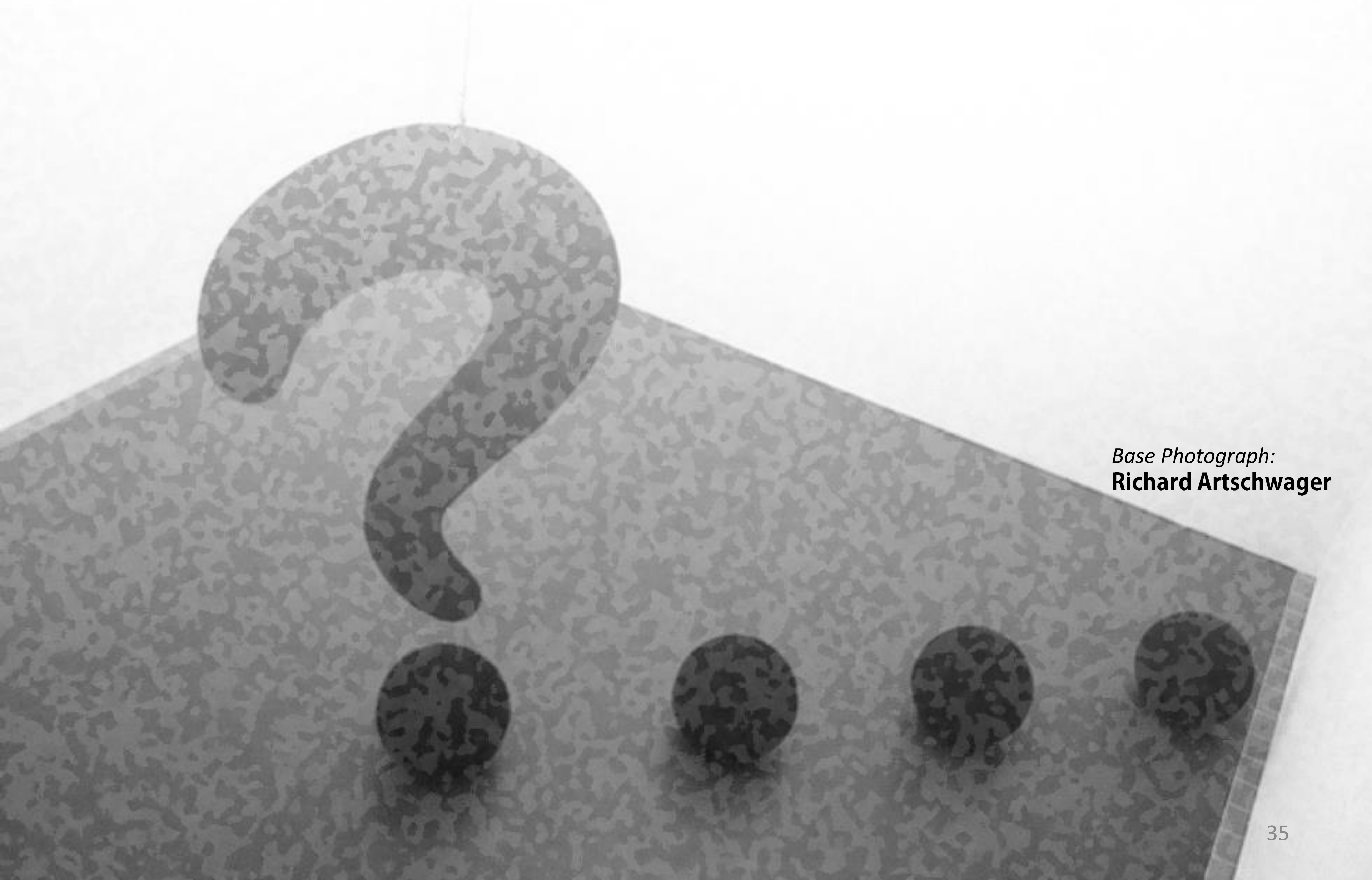


GOMC

Acknowledgements

- National Science Foundation NSF OCI-1148168
- Wayne State University Research Enhancement Program (REP)
- Wayne State University Graduate Research Fellowship.

An Invitation for Questions



Base Photograph:
Richard Artschwager