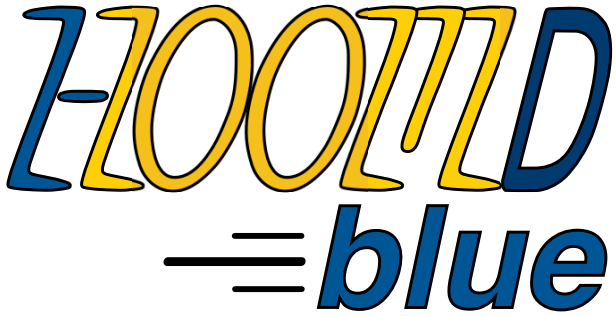
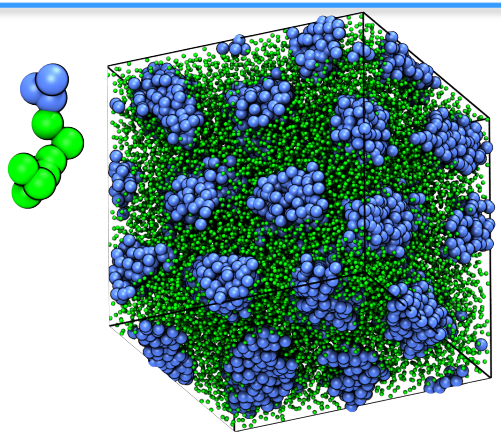


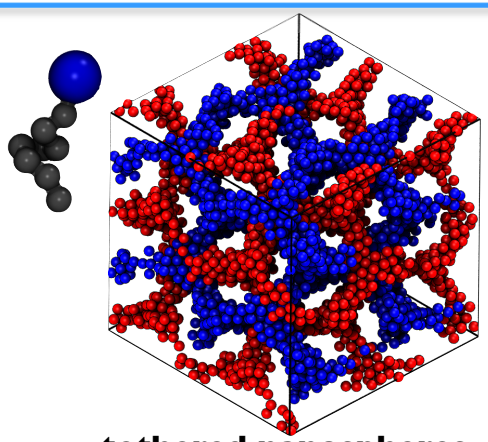
- Fast and flexible many-particle dynamics on the GPU

Joshua A. Anderson

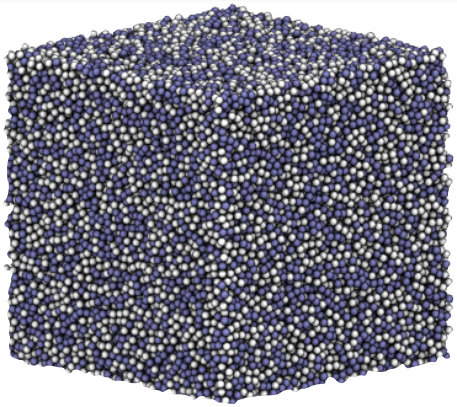
Many-particle dynamics - examples



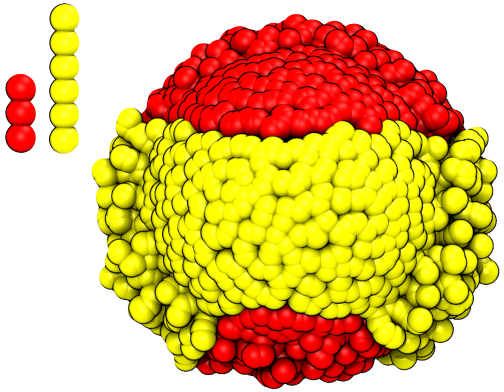
polymer systems
Dissipative Particle Dynamics (DPD)



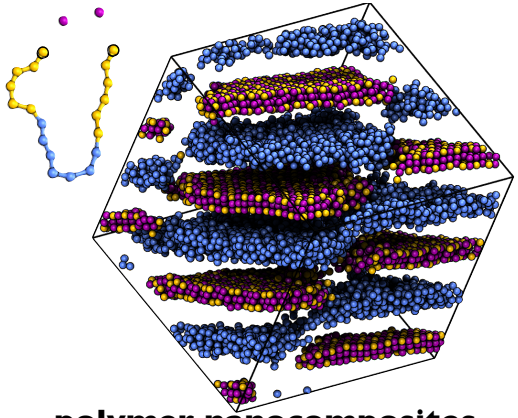
tethered nanospheres
Brownian Dynamics



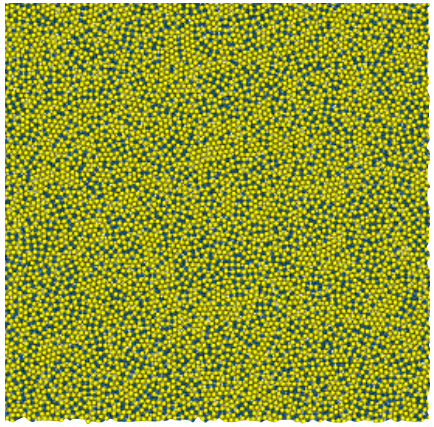
supercooled liquids
Molecular Dynamics



surfactant coated surfaces
DPD with constraints



polymer nanocomposites
coarse-grained Molecular Dynamics



supercooled liquids
2D Molecular Dynamics

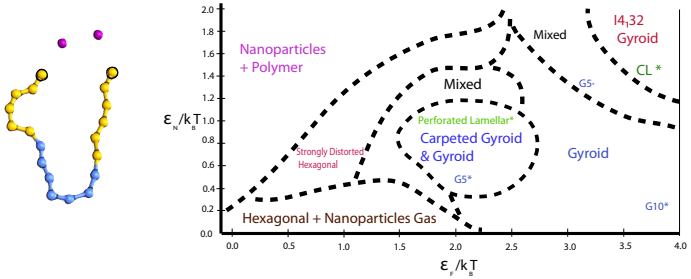
Live demo

Demo of HOOMD-blue outside of presentation

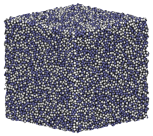
Benefits of GPU computing

Papers with CPU jobs...

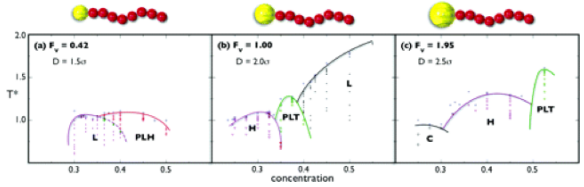
run thousands of serial jobs - often **one month** of CPU time for each



compute a phase diagram for one polymer architecture



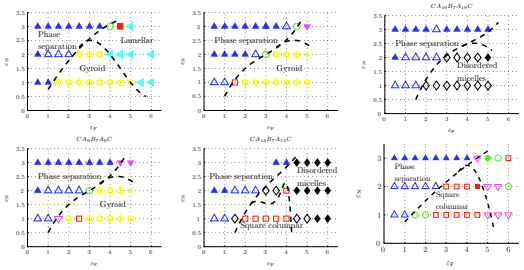
study one supercooled liquid model



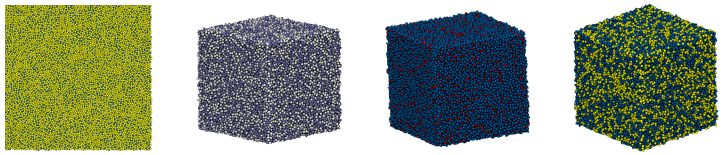
study three monodisperse tethered nanospheres

Papers with GPU jobs...

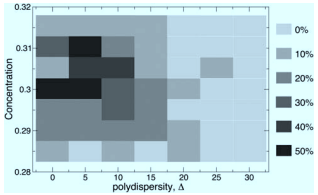
run thousands of single GPU jobs - **one day** of GPU time for each



compute phase diagrams for six polymer architectures



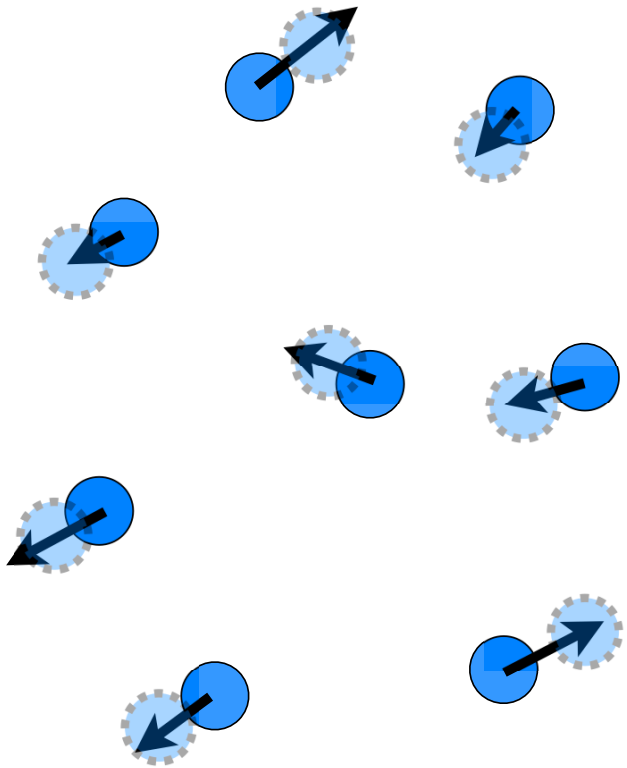
provide an in-depth comparison of four different models



study the effects of of varying polydispersity

490 jobs just for one figure!

Method overview



$$\vec{r}_i(t)$$
$$\vec{v}_i(t)$$

↓

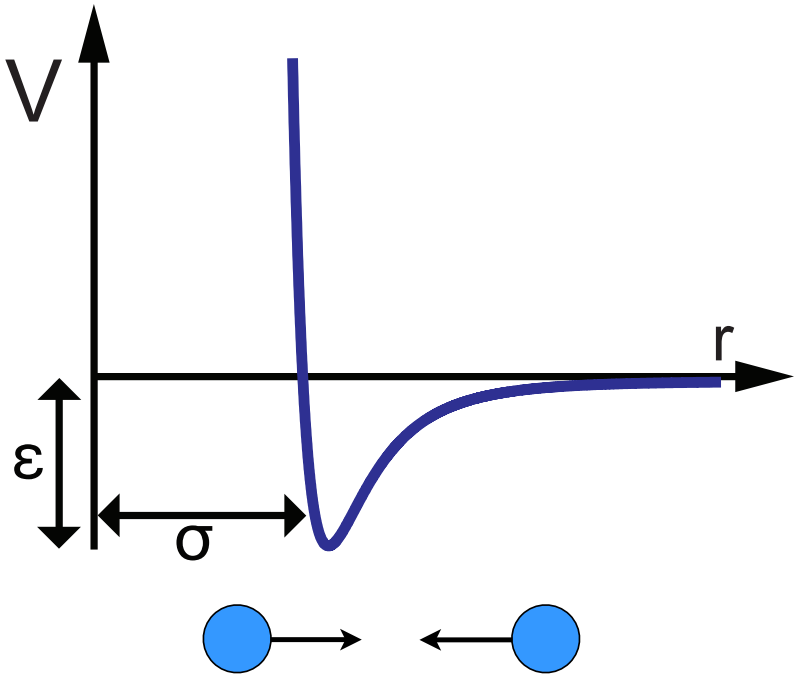
Calculate accelerations

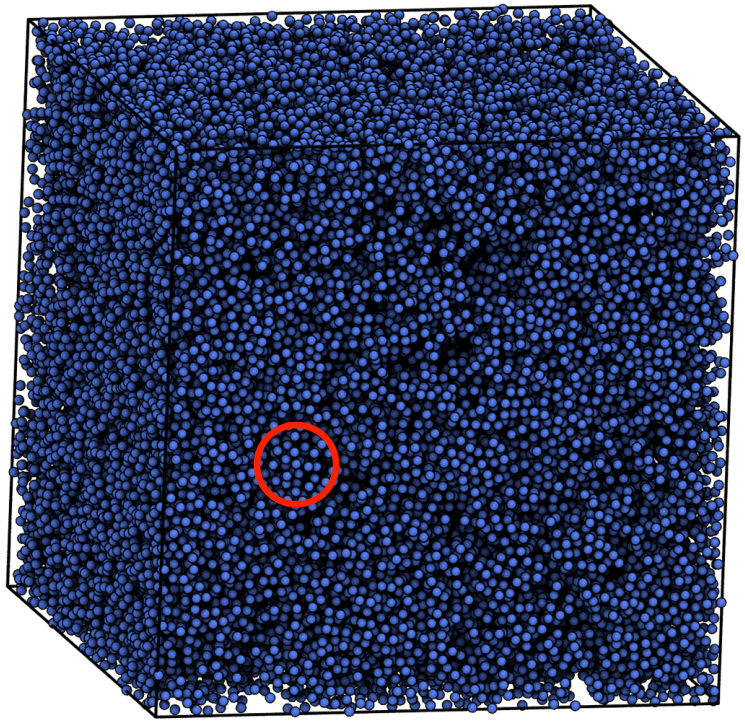
↓

$$\vec{r}_i(t+\delta t)$$
$$\vec{v}_i(t+\delta t)$$

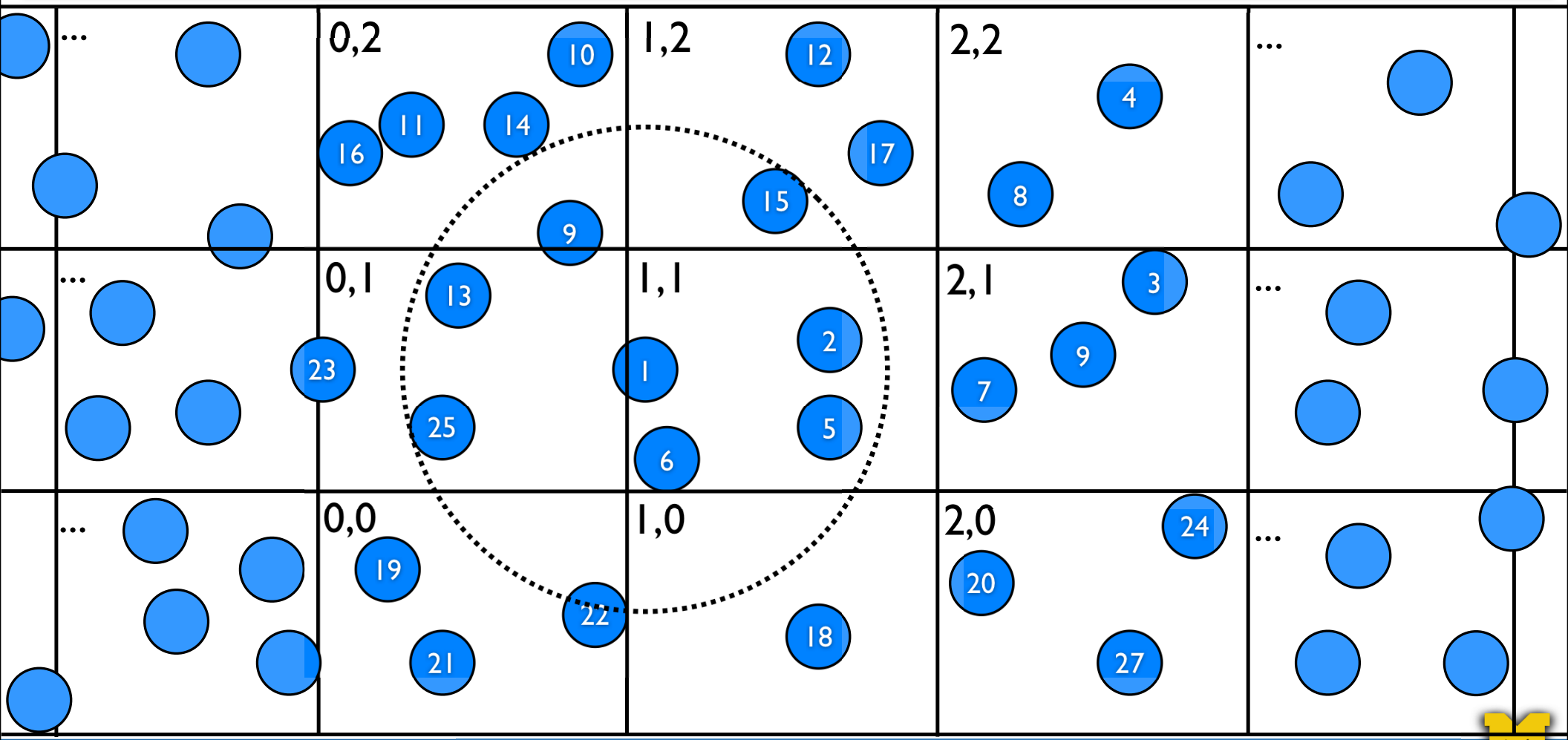
Pair potential

Example - Lennard-Jones

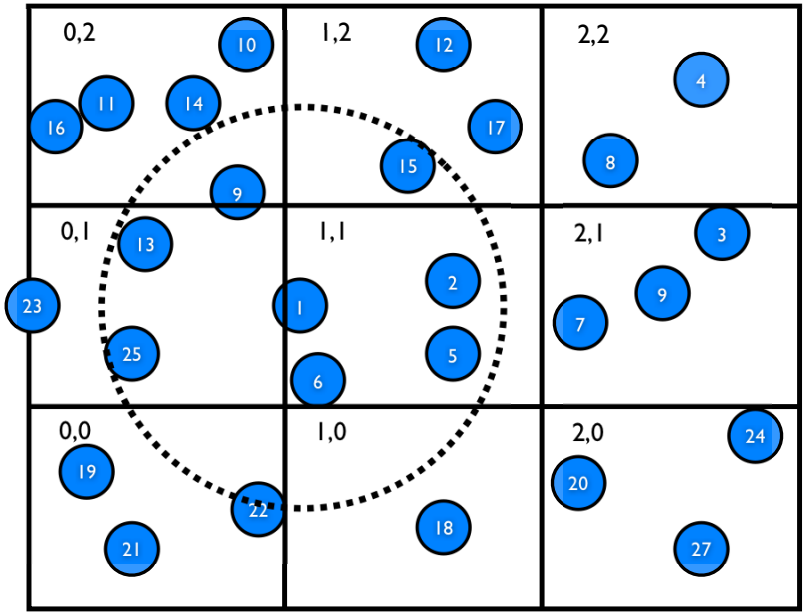




Cell list


0,2
1,2
2,2
0,1
1,1
2,1
0,0
1,0
2,0
...
10
12
4
11
14
16
17
15
8
9
13
2
3
1
9
23
7
25
5
6
24
19
20
22
18
21
27

Generating the cell list



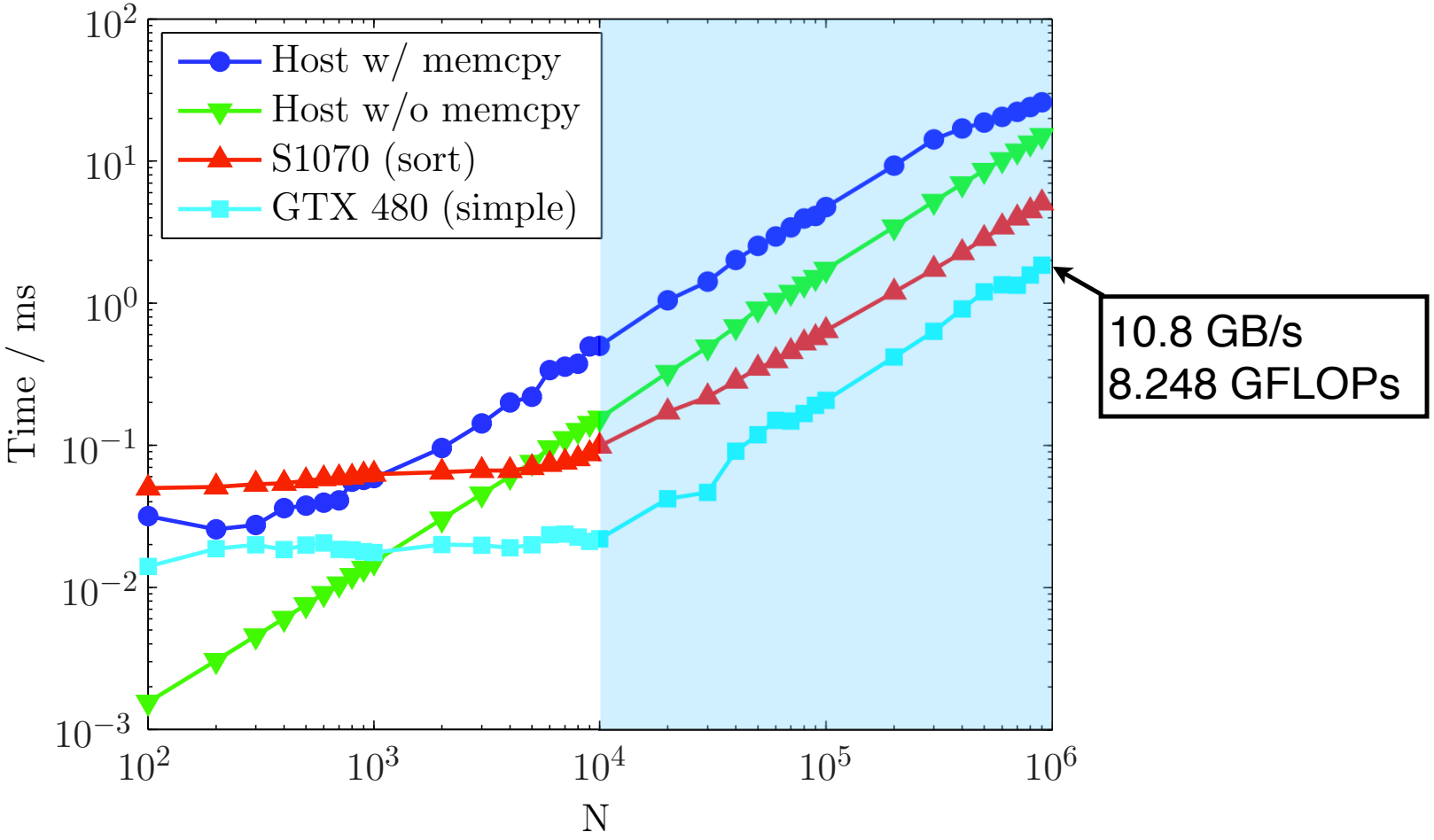

```
for each particle i in parallel
  load position pos[i]
  compute cell index ci
  cur_size = atomicInc length[ci]
  write (ci, pos[i]) to
    cell_list[ci][cur_size]
```

	1	2	3	4	5	6	7	8	9	10

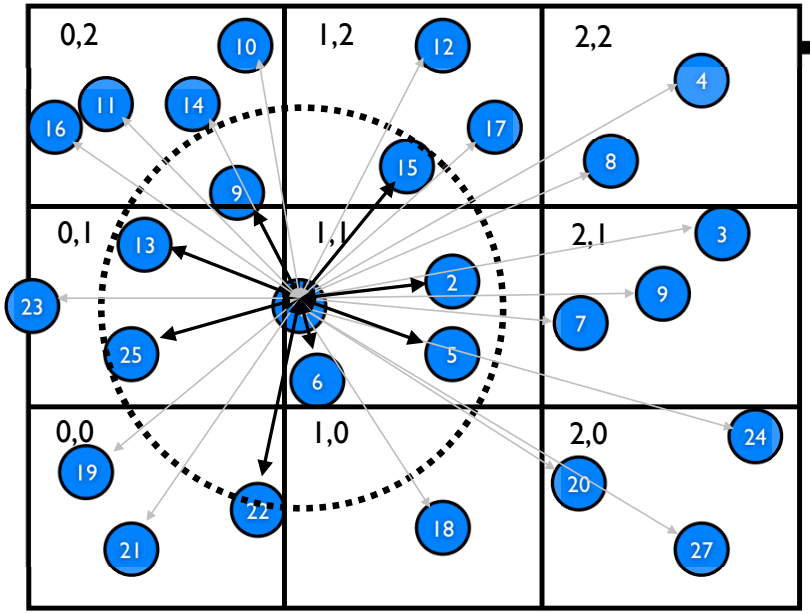
	Length	Cell list				
0,0	3	19	21	22		
1,0	1	18				
2,0	3	20	24	27		
0,1	3	23	25	13		
1,1	4	**1**	6	**2**	**5**	
2,1	3	7	9	**3**		
0,2	5	16	11	14	10	9
1,2	3	12	15	17		
2,2	2	8	**4**			

• • •

Cell list performance



Creating the neighbor list




```
for each particle i in parallel
  load position pos[i]
  compute cell index ci
  for each nearby cell cn
     for each particle p in cn
        load cell_list[cn][p]
        if distance < rcut
          append to n_list[][i]
```

1	2	3	4	5
25	1	7	3	18
13	6	9	8	20
6	5	8	...	1
2	7	4		2
5	15	...		6
9	17			7
15				15

Neighbor list

7	6	...	...	7

Num Neighbors

Neighbor list on GF100 and G200

GF100

```
for each particle i in parallel
  load position pos[i]
  compute cell index ci
  for each nid from 0 to 27
     cn=ld.global adj_list[ci][nid]
     for each particle p in cn
        ld.global cell_list[cn][p]
         if distance < rcut
           append to n_list[][i]
```

Notes

- Semi-random memory reads performed from L1
- Activate 48k L1 for best perf.
- Spatial sorting (later) increases cache hit ratio
- Bottleneck becomes the incoherent *n_list* append (only 1 in 8 writes pass the distance test)

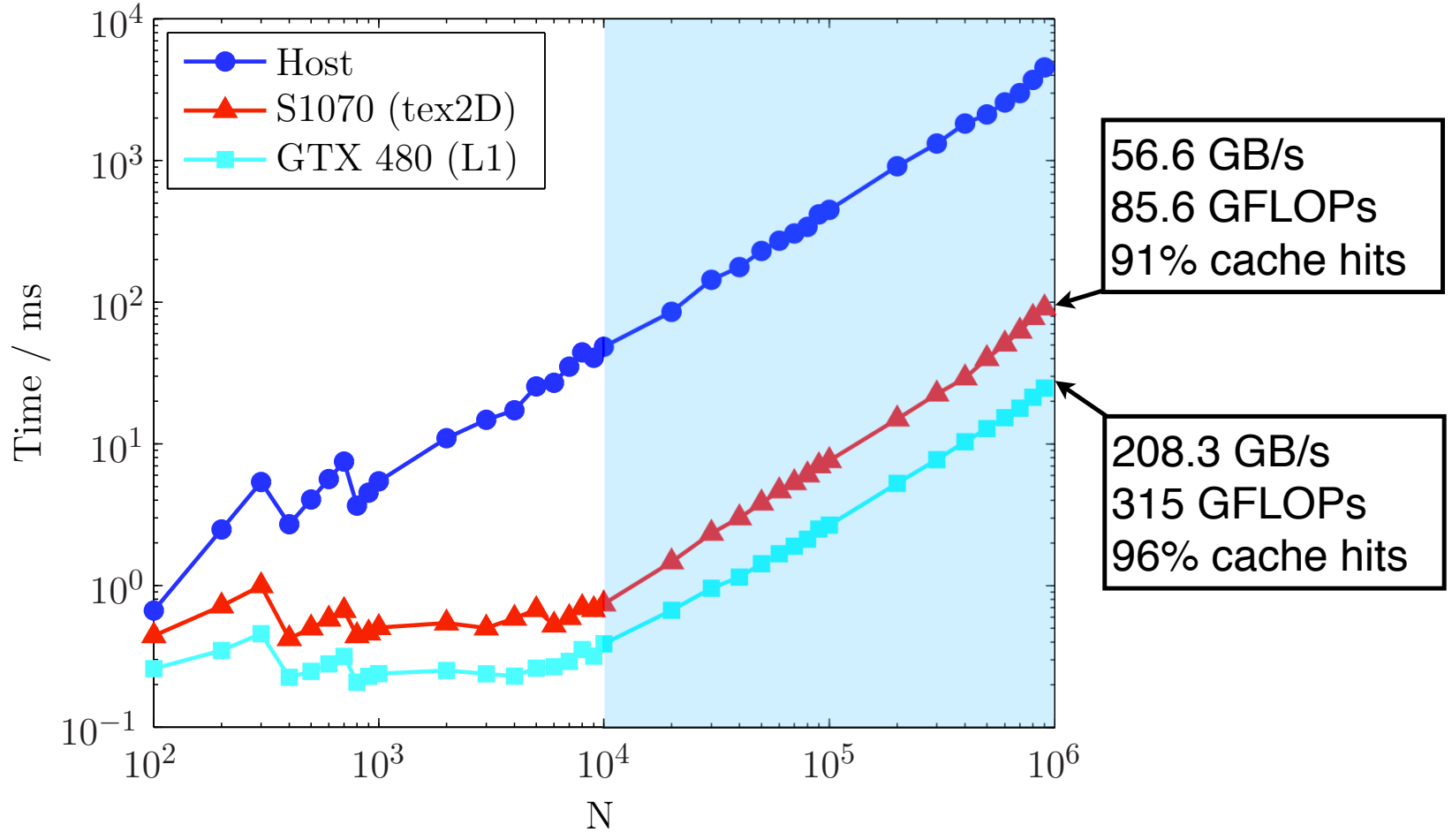
G200

```
for each particle i in parallel
  load position pos[i]
  compute cell index ci
  for each nid from 0 to 27
     cn=tex2D adj_list[ci][nid]
     for each particle p in cn
        tex2D cell_list[cn][p]
         if distance < rcut
           append to n_list[][i]
```

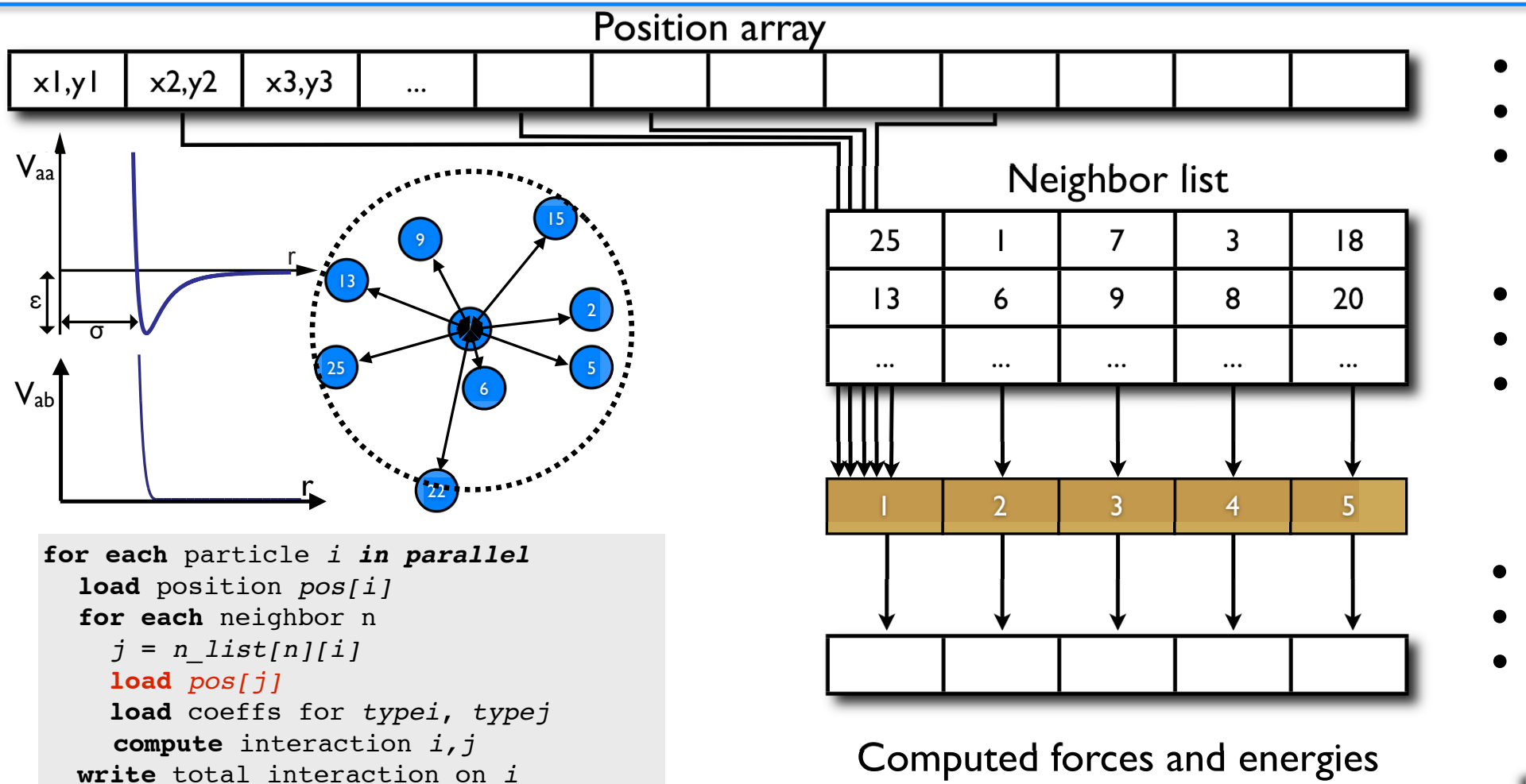
Notes

- Semi-random memory reads performed from 2D tex cache

Neighbor list performance



Computing pair forces




```
for each particle i in parallel
  load position pos[i]
  for each neighbor n
    j = n_list[n][i]
    load pos[j]
    load coeffs for typei, typej
    compute interaction i,j
  write total interaction on i
```

Pair forces on Fermi vs. Tesla

GF100

identical →

G200

```
for each particle i in parallel
  load position pos[i]
  nextj = ld.global n_list[n][i]
  for each neighbor n
    curj = nextj
    nextj=ld.global n_list[n+1][i]
    tex1Dfetch pos[curj]
    ld.shared coeffs[typei][typej]
    compute interaction i,j
  write total interaction on i
```

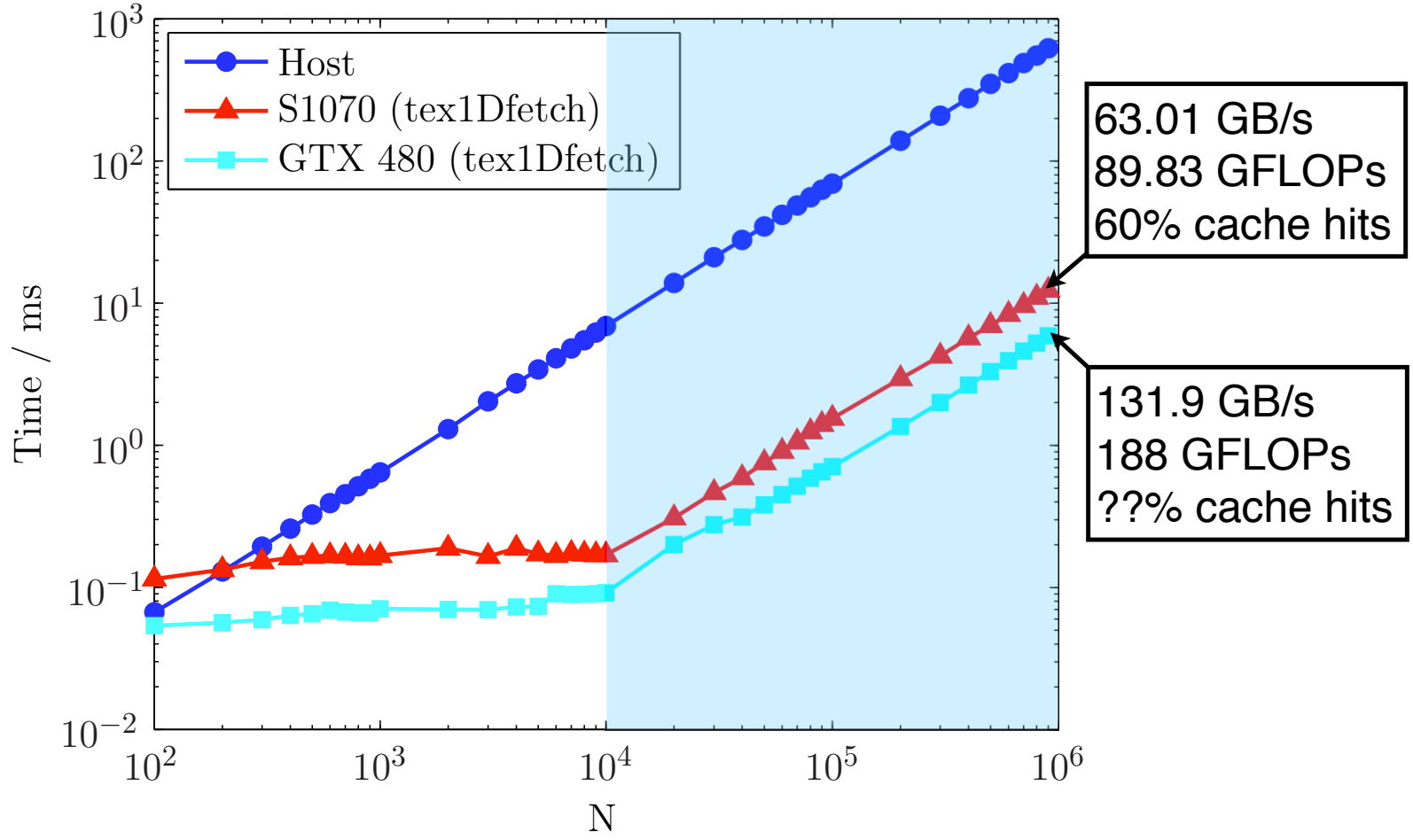
Notes

- Switching the `pos[curj]` read to use L1 reduces performance
- I'm not sure why..... 48k L1 >>> 8k tex cache
- Have tried a number of transformations without success
- The coefficient **`ld.shared`** can be converted to **`ld.global`** with no performance hit

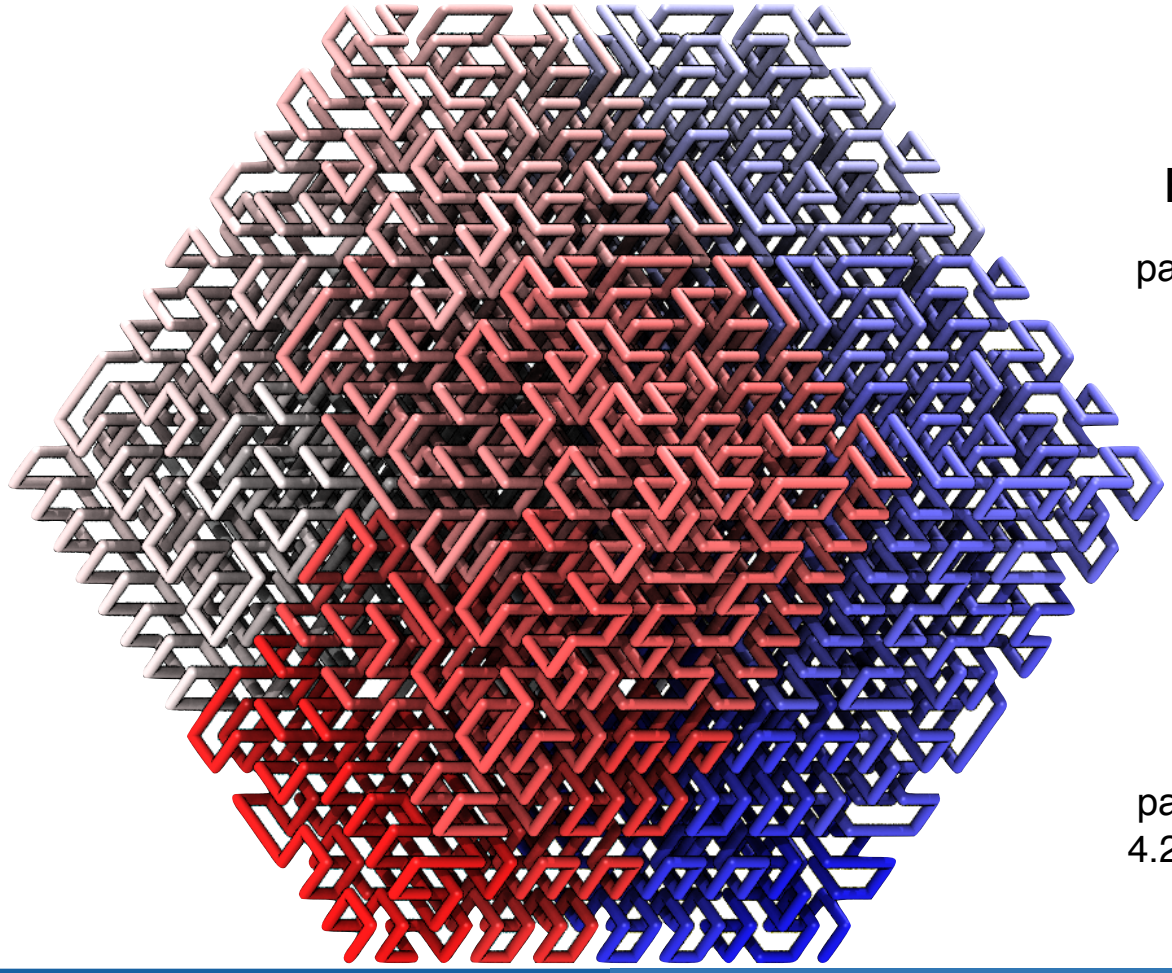
Notes

- Semi-random memory reads performed via `tex1Dfetch`

Pair force performance

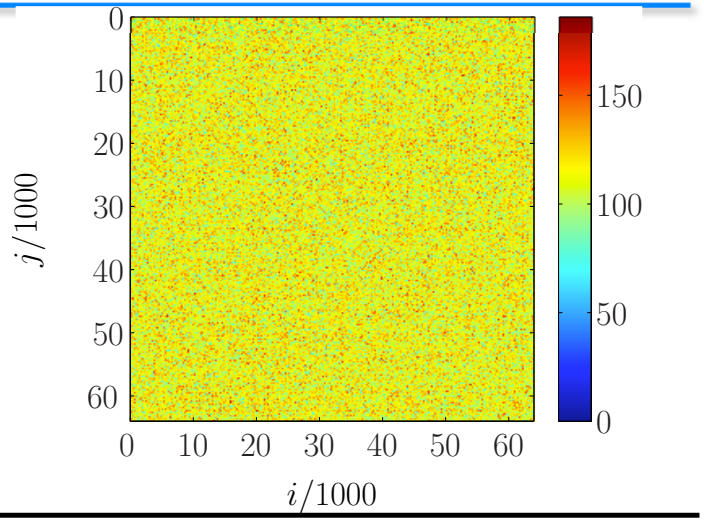


Spatial sorting reorders particles



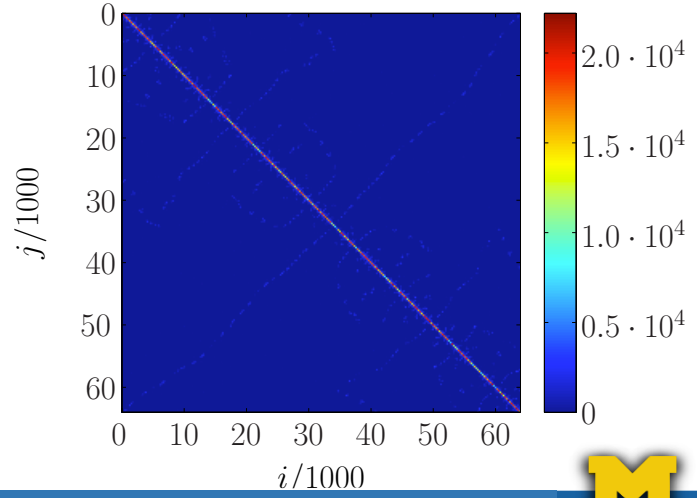
Random

pair: 50.4 ms



Sorted

pair: 12.3 ms
4.2x speedup!



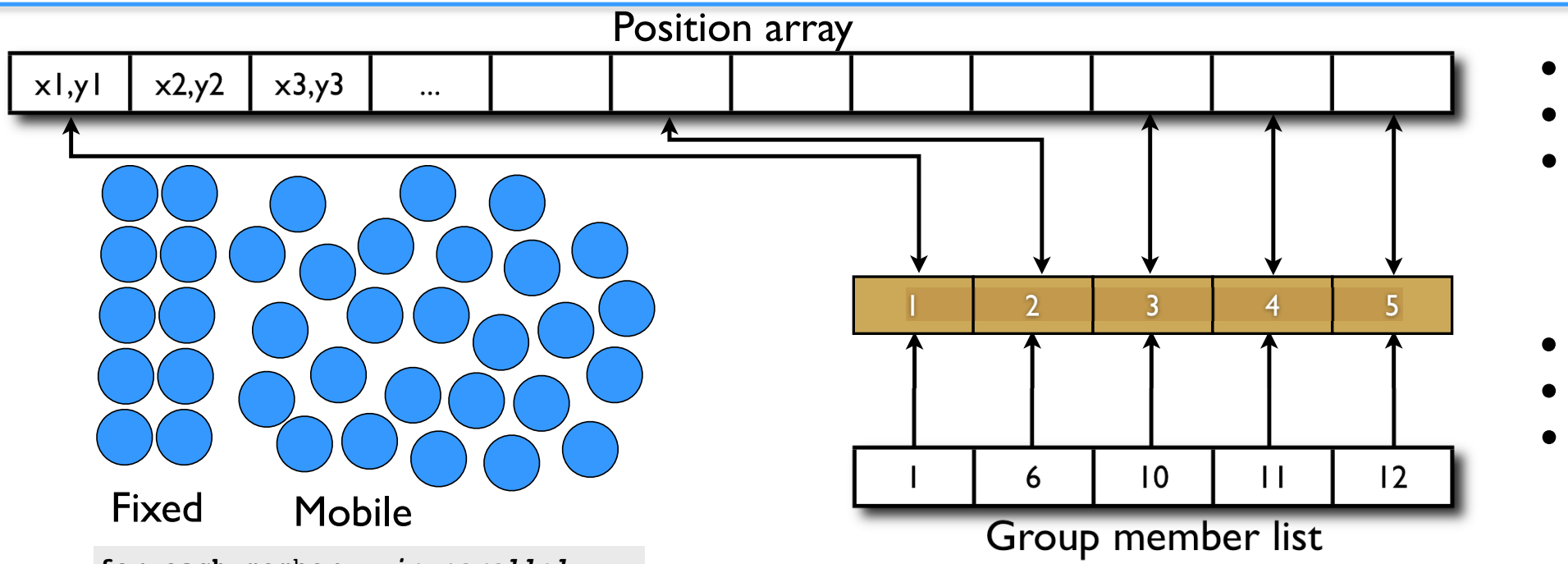
Flexible pair potentials

```
template< class evaluator> __global__ void
gpu_compute_pair_forces_kernel(float4 *d_force, float4 *d_pos, gpu_nlist_array nlist,
                                                typename evaluator::param_type *d_params,
                                                ...)
    {
    extern __shared__ typename evaluator::param_type s_params[]
    // load data from d_params into s_params ...
    __syncthreads();
    unsigned int idx = blockIdx.x * blockDim.x + threadIdx.x;
    // load in data for particle idx ...
    for (int neigh_idx = 0; neigh_idx < n_neigh; neigh_idx++)
        {
        // access current neighbor ...
        // calculate dr^2 (with periodic boundary conditions) ...
        float rsq = dx*dx + dy*dy + dz*dz;
        unsigned int typpair = typpair_idx(__float_as_int(posi.w), __float_as_int(posj.w));
        typename evaluator::param_type param = s_params[typpair];

        evaluator eval(rsq, rcutsq, param);
        eval.evalForceAndEnergy(force_divr, pair_eng, energy_shift);

        // tally results into force ...
        }
    d_force[idx] = force;
    }
```


Evaluator functor

```
class EvaluatorPairLJ
    {
    public:
        typedef Scalar2 param_type;

        DEVICE EvaluatorPairLJ(Scalar _rsq, Scalar _rcutsq, const param_type& _params)
            : rsq(_rsq), rcutsq(_rcutsq), lj1(_params.x), lj2(_params.y) { }

        DEVICE void evalForceAndEnergy(Scalar& force_divr, Scalar& pair_eng)
            {
            if (rsq < rcutsq && lj1 != 0)
                {
                Scalar r2inv = Scalar(1.0)/rsq;
                Scalar r6inv = r2inv * r2inv * r2inv;
                force_divr= r2inv * r6inv * (Scalar(12.0)*lj1*r6inv - Scalar(6.0)*lj2);

                pair_eng = r6inv * (lj1*r6inv - lj2);
                }
            }

    protected:
        Scalar rsq, rcutsq, lj1, lj2;
    };

class EvaluatorPairGauss
    {
    //...
```


Flexible integration




```
for each member g in parallel
  i = load group_idx[g]
  load pos[j]
  load vel[j]
  load force[j]
  compute updated quantities
  write pos[j]
  write vel[j]
```

Notes

- Member list is maintained in a sorted order
- This reduces the number of wasted memory transactions

Feature sheet

Integration
- NVT (Nosé-Hoover)
- NPT
- Brownian Dynamics
- Dissipative Particle Dynamics
- NVE
- FIRE energy minimization

Bond forces
- harmonic
- FENE

Angle forces
- harmonic
- CGCMM

Dihedral/Improper forces
- harmonic

Simulation types
- 2D and 3D
- Replica exchange (via script)

Snapshot formats
- MOL2
- DCD
- PDB
- XML

Pair forces
- Lennard Jones
- Gaussian
- CGCMM
- Morse
- Table (arbitrary)
- Yukawa
- PME (*in developpment*)

Many-body forces
- EAM (*in development*)

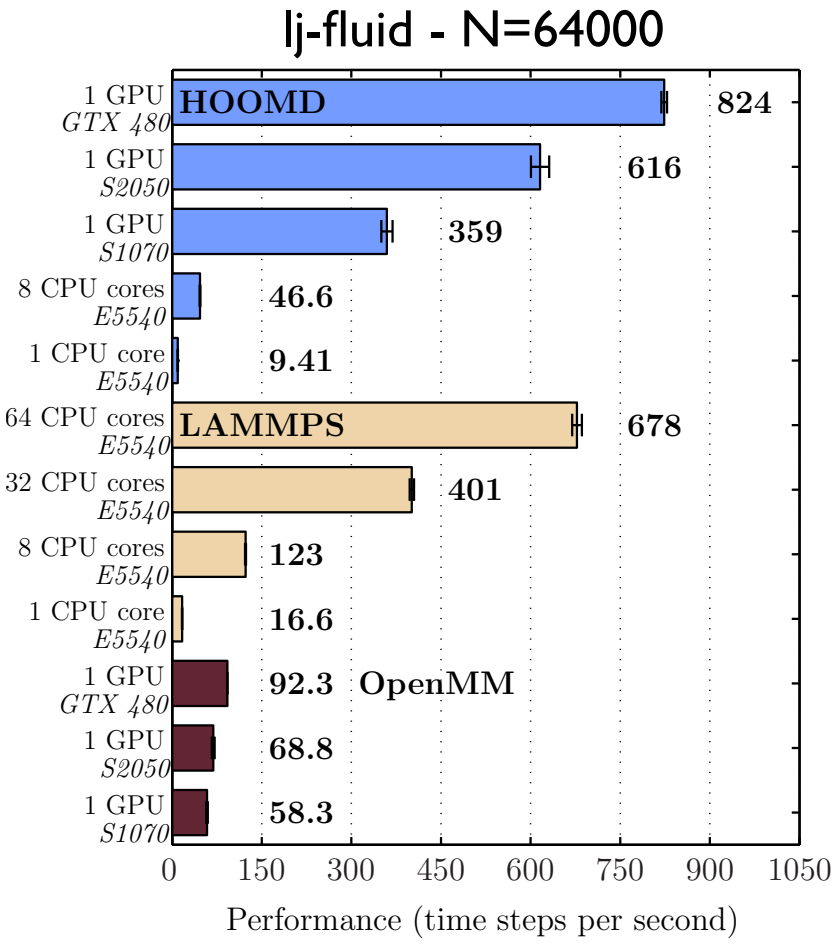
Hardware support
- All recent NVIDIA GPUs
- Multi-core CPUs via OpenMP

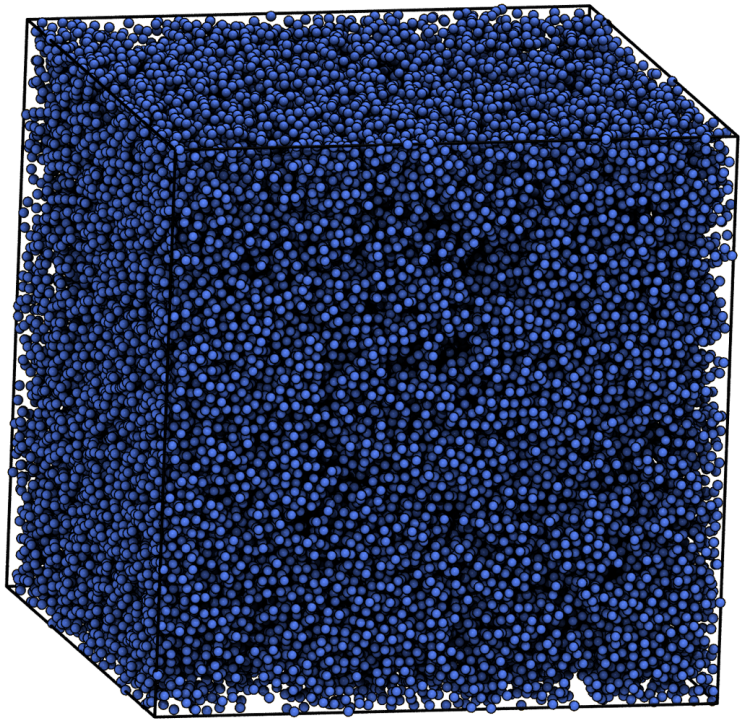
Script example

Documentation online: http://codeblue.umich.edu/hoomd-blue/doc

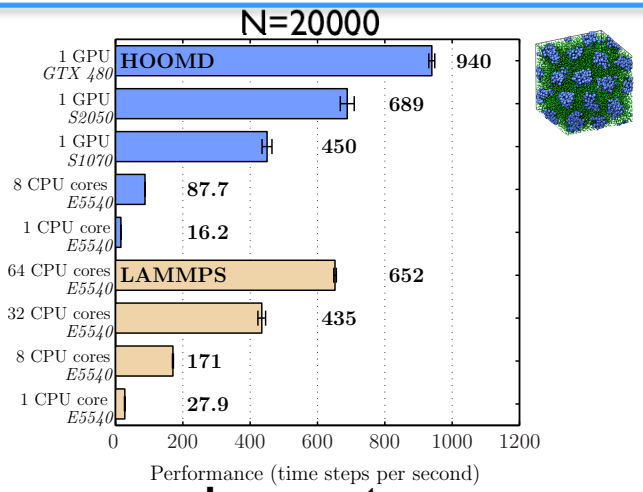
```
from hoomd_script import *

init.read_xml(filename='init.xml')

lj = pair.lj(r_cut=2.5)
lj.pair_coeff.set('A', 'A', epsilon=1.0, sigma=1.0)

integrate.mode_standard(dt=0.005)
integrate.nvt(T=1.2, tau=0.5)

run(10e3)
```

Overall performance

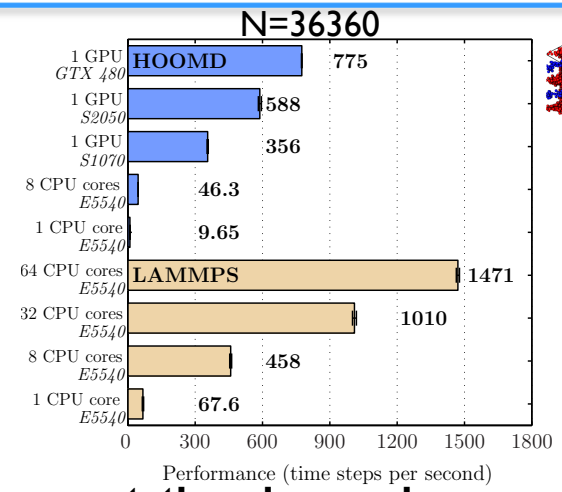




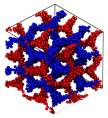
... more performance

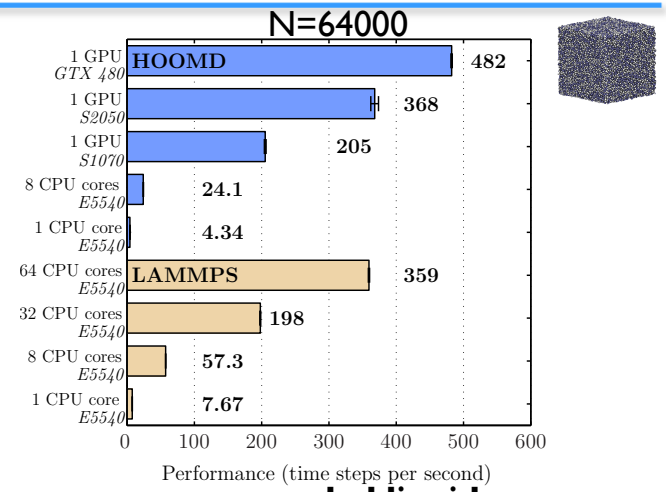


polymer systems

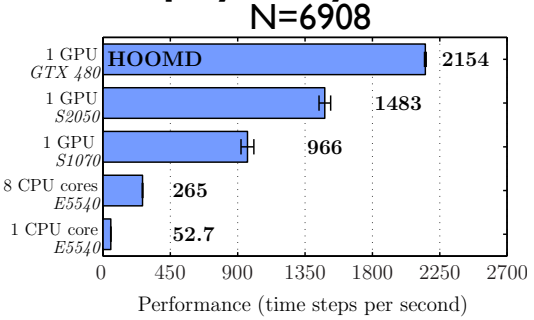


tethered nanospheres

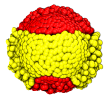


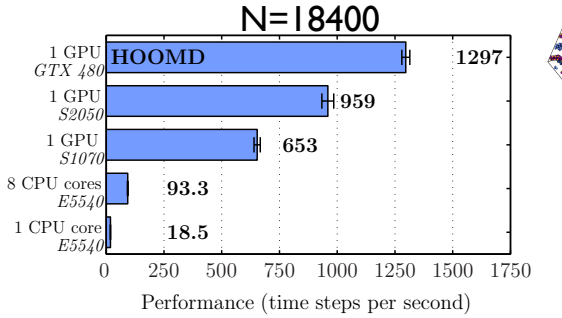


supercooled liquids

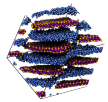


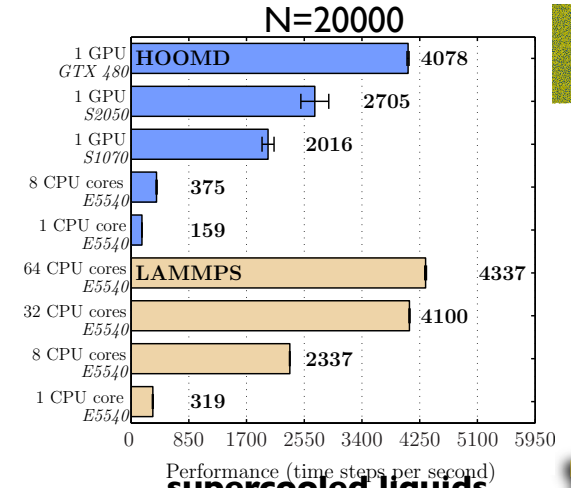
surfactant coated surfaces



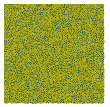


polymer nanocomposites





supercooled liquids



Acknowledgements

HOOMD-blue is open source!

- Contributions from around the world
  - Joshua Anderson, Aaron Keys, Trung Dac Nguyen, Carolyn Phillips (University of Michigan)
  - Rastko Sknepnek (Northwestern)
  - Alex Travesset (Iowa State University)
  - Axel Kohlmeyer, David Lebard, and Ben Levine (Temple)
  - Igor Morozov, Kazennov Andrey, Bystriy Roman (Russian Academy of Sciences)

Cell list on GF100 and G200

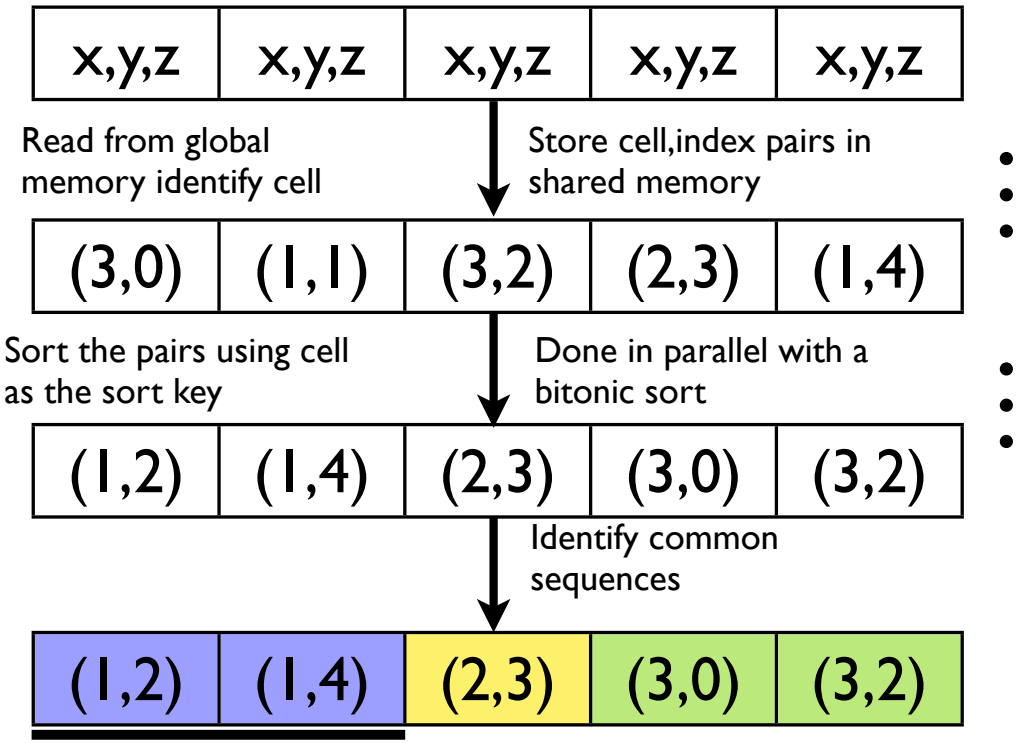
GF100

```
for each particle i in parallel
  load position pos[i]
  compute cell index ci
  cur_size = atomicInc length[ci]
  write (ci, pos[i]) to
    cell_list[ci][cur_size]
```

Notes

- `atomicInc` in L2 cache - *fast!*
- Typically ~30 possible collisions
- Spatial sorting (later) *increases* chances that collisions occur within the same block/warp

G200



- Only *one* `atomicAdd` per unique sequence is needed