Hoomd blue

- 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

- **surfactant coated surfaces**
  - DPD with constraints

- **polymer nanocomposites**
  - coarse-grained Molecular Dynamics

- **supercooled liquids**
  - Molecular Dynamics

- **supercooled liquids**
  - 2D Molecular Dynamics
Live demo

Demo of HOOMD-blue outside of presentation
Benefits of GPU computing

<table>
<thead>
<tr>
<th>Papers with CPU jobs...</th>
<th>Papers with GPU jobs...</th>
</tr>
</thead>
<tbody>
<tr>
<td>run thousands of serial jobs - often <strong>one month</strong> of CPU time for each</td>
<td>run thousands of single GPU jobs - <strong>one day</strong> of GPU time for each</td>
</tr>
<tr>
<td>compute a phase diagram for one polymer architecture</td>
<td>compute phase diagrams for six polymer architectures</td>
</tr>
<tr>
<td>study one supercooled liquid model</td>
<td>provide an in-depth comparison of four different models</td>
</tr>
<tr>
<td>study three monodisperse tethered nanospheres</td>
<td>study the effects of varying polydispersity</td>
</tr>
</tbody>
</table>

The Glotzer Group @ University of Michigan
Calculate accelerations

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

\[
\begin{align*}
\vec{r}_i(t) & \\
\vec{v}_i(t) & \\
\text{Calculate accelerations} & \\
\vec{r}_i(t + \delta t) & \\
\vec{v}_i(t + \delta t) &
\end{align*}
\]
Pair potential

Example - Lennard-Jones
for each particle $i$ in parallel
load position $pos[i]$
compute cell index $ci$
$cur_size = \text{atomicInc} length[ci]$
write ($ci$, $pos[i]$) to $cell_list[ci][cur_size]$
Cell list performance

![Graph showing time vs. N for different configurations.]

- Host w/ memcpy
- Host w/o memcpy
- S1070 (sort)
- GTX 480 (simple)

10.8 GB/s
8.248 GFLOPs
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 $n\_list[][i]$
## Neighbor list on GF100 and G200

### GF100

```plaintext
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]
```

### G200

```plaintext
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 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)
Neighbor list performance

![Graph showing performance comparison between Host, S1070 (tex2D), and GTX 480 (L1).]

- **Host**: 56.6 GB/s, 85.6 GFLOPs, 91% cache hits
- **S1070 (tex2D)**: 208.3 GB/s, 315 GFLOPs, 96% cache hits

The Glotzer Group @ University of Michigan
for each particle \( i \) in parallel

1. load position \( \text{pos}[i] \)
2. for each neighbor \( n \)
   1. \( j = \text{n_list}[n][i] \)
   2. load \( \text{pos}[j] \)
   3. load coeff for \( \text{type}_i, \text{type}_j \)
   4. compute interaction \( i,j \)
3. write total interaction on \( i \)
Pair forces on Fermi vs. Tesla

GF100

for each particle \( i \) in parallel
load position \( \text{pos}[i] \)
\( \text{nextj} = \text{ld.global} \ n\text{\_list}[n][i] \)
for each neighbor \( n \)
curj = nextj
\( \text{nextj} = \text{ld.global} \ n\text{\_list}[n+1][i] \)
\( \text{tex1Dfetch} \ \text{pos}[\text{curj}] \)
\( \text{ld.shared} \ \text{coeffs[typei][typej]} \)
compute interaction \( i,j \)
write total interaction on \( i \)

G200

Notes
- Switching the \( \text{pos[\text{curj}]} \) read to use L1 reduces performance
- I’m not sure why...... 48k L1 \( \gg \gg \) 8k tex cache
- Have tried a number of transformations without success
- The coefficient \( \text{ld.shared} \) can be converted to \( \text{ld.global} \) with no performance hit

Notes
- Semi-random memory reads performed via \( \text{tex1Dfetch} \)
Pair force performance

- **Host**: 131.9 GB/s, 89.83 GFLOPs, 60% cache hits
- **S1070 (tex1Dfetch)**: 63.01 GB/s, 89.83 GFLOPs, 60% cache hits
- **GTX 480 (tex1Dfetch)**: 63.01 GB/s, 89.83 GFLOPs, 60% cache hits
Spatial sorting reorders particles

Random pair: 50.4 ms

Sorted pair: 12.3 ms 4.2x speedup!
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;
}

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
{
  //...
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 development)*

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

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

lj-fluid - N=64000

Performance (time steps per second)

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 GPU GTX 480</td>
<td>HOOMD: 824</td>
</tr>
<tr>
<td>1 GPU S2050</td>
<td></td>
</tr>
<tr>
<td>1 GPU S1070</td>
<td></td>
</tr>
<tr>
<td>8 CPU cores E5540</td>
<td>46.6</td>
</tr>
<tr>
<td>1 CPU core E5540</td>
<td>9.41</td>
</tr>
<tr>
<td>64 CPU cores E5540</td>
<td>LAMMPS: 678</td>
</tr>
<tr>
<td>32 CPU cores E5540</td>
<td></td>
</tr>
<tr>
<td>8 CPU cores E5540</td>
<td>123</td>
</tr>
<tr>
<td>1 CPU core E5540</td>
<td>16.6</td>
</tr>
<tr>
<td>1 GPU GTX 480</td>
<td>OpenMM: 92.3</td>
</tr>
<tr>
<td>1 GPU S2050</td>
<td>68.8</td>
</tr>
<tr>
<td>1 GPU S1070</td>
<td>58.3</td>
</tr>
</tbody>
</table>
... more performance

**polymer systems**

N=20000

- 1 GPU
  - GTX 480
  - HOOMD
  - Performance (time steps per second)
    - 319
- 1 CPU
  - E5540
  - LAMMPS
  - Performance (time steps per second)
    - 588

N=6908

- 1 GPU
  - GTX 480
  - HOOMD
    - Performance (time steps per second)
      - 265
- 1 CPU
  - E5540
  - LAMMPS
    - Performance (time steps per second)
      - 450

N=18400

- 1 GPU
  - GTX 480
  - HOOMD
    - Performance (time steps per second)
      - 93.3
- 1 CPU
  - E5540
  - LAMMPS
    - Performance (time steps per second)
      - 653

N=20000

- 1 GPU
  - GTX 480
  - HOOMD
    - Performance (time steps per second)
      - 319
- 1 CPU
  - E5540
  - LAMMPS
    - Performance (time steps per second)
      - 588

**tethered nanospheres**

N=36360

- 1 GPU
  - GTX 480
    - Performance (time steps per second)
      - 966
- 1 CPU
  - E5540
    - Performance (time steps per second)
      - 4078

N=18400

- 1 GPU
  - GTX 480
  - HOOMD
    - Performance (time steps per second)
      - 283
- 1 CPU
  - E5540
  - LAMMPS
    - Performance (time steps per second)
      - 4078

N=20000

- 1 GPU
  - GTX 480
  - HOOMD
    - Performance (time steps per second)
      - 966
- 1 CPU
  - E5540
  - LAMMPS
    - Performance (time steps per second)
      - 4078

**supercooled liquids**

N=64000

- 1 GPU
  - GTX 480
    - Performance (time steps per second)
      - 966
- 1 CPU
  - E5540
    - Performance (time steps per second)
      - 4078

N=20000

- 1 GPU
  - GTX 480
  - HOOMD
    - Performance (time steps per second)
      - 966
- 1 CPU
  - E5540
  - LAMMPS
    - Performance (time steps per second)
      - 4078

polymer systems

tethered nanospheres

supercooled liquids

surfactant coated surfaces

polymer nanocomposites

The Glotzer Group @ University of Michigan
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)
for each particle \( i \) in parallel
load position \( pos[i] \)
compute cell index \( ci \)
cur_size = \texttt{atomicInc} length[\( ci \)]
write \((ci, pos[i])\) to \( \text{cell\_list}[ci][\text{cur\_size}] \)

Notes
- \texttt{atomicInc} in L2 cache - fast!
- Typically \( \sim 30 \) possible collisions
- Spatial sorting (later) \( \text{increases} \) chances that collisions occur within the same block/warp

\textbf{GF100}

\begin{tabular}{|c|c|c|c|}
\hline
\( x,y,z \) & \( x,y,z \) & \( x,y,z \) & \( x,y,z \) & \( x,y,z \) \\
\hline
\end{tabular}

\textbf{G200}

\begin{tabular}{|c|c|c|c|c|}
\hline
\( x,y,z \) & \( x,y,z \) & \( x,y,z \) & \( x,y,z \) & \( x,y,z \) \\
\hline
\end{tabular}

Read from global memory identify cell
Store cell,index pairs in shared memory

\begin{tabular}{|c|c|c|c|}
\hline
(3,0) & (1,1) & (3,2) & (2,3) & (1,4) \\
\hline
\end{tabular}

Sort the pairs using cell as the sort key
Done in parallel with a bitonic sort

\begin{tabular}{|c|c|c|c|}
\hline
(1,2) & (1,4) & (2,3) & (3,0) & (3,2) \\
\hline
\end{tabular}

Identify common sequences

\begin{tabular}{|c|c|c|c|}
\hline
(1,2) & (1,4) & (2,3) & (3,0) & (3,2) \\
\hline
\end{tabular}

- Only one \texttt{atomicAdd} per unique sequence is needed