HPCSE II - Exercise Class 18th May 2020

Ivica Kičić

Computational Science and Engineering Lab
ETH Zürich
Agenda

- Reduction
- Atomics
- Cell lists
Reduction
1. Naive tree-like

2. Contiguous tree-like

3. warpReduce with volatile

4. __shfl_down_sync
Atomics
Atomics

- Read-modify-write operation without interference from other threads
- 32-bit or 64-bit
- Do NOT imply synchronization or guaranteed ordering!

```c
double atomicAdd(double *address, double val);
Atomically add val to *address and return the old value.

unsigned int atomicInc(unsigned int *address,
unsigned int val);
Read the old value, compute and store (old >= val ? 0 : old+1). Returns the old value.

int atomicCAS(int *address, int compare, int val);
Read the old value, compute and store (old == compare ? val : old). Returns the old value.
```

See here for available overloads.
https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#atomic-functions
Cell lists
Cell lists

$O(N^2)$

Force range or cutoff

$O(r_c^2 \rho N)$
Implementation

Variant #1: Particles may not be rearranged

```
# Stage 1: Build cells lists.
Reset cellSize[] to 0
For each particle pIdx
    Compute cellIndex from position[pIdx]
    Increment cellSize[cellIndex]
Compuet cellOffset from cellSize
Reset cellSize[] to 0
For each particle pIdx
    Compute cellIndex from position[pIdx]
    Set particleIdx[cellSize[cellIndex]] to pIdx
    Increase cellSize[cellIndex]

# Stage 2: Compute forces.
For each particle pIdx
    Reset fTotal to 0
    Compute(cix, ciy)
    For each ciy' in (ciy - 1, ciy + 1)
        For each cix' in (cix - 1, cix + 1)
            For each particle pIdx' belonging to cell (cix', ciy')
                If pIdx' is under the cutoff:
                    Compute force between pIdx and pIdx'
                    Add force to fTotal
Save fTotal to totalForce[pId]
```

Race condition
(Atomics)
Non-contiguous access
Race condition
(Atomics)
Non-contiguous access
Implementation

Variant #2: Particles may be rearranged

<table>
<thead>
<tr>
<th>position</th>
<th>xy xy xy xy xy xy xy xy xy xy xy xy xy xy xy xy ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>cellIndex</td>
<td>23 0 3 5 1 6 10 23 2 2 1 ...</td>
</tr>
<tr>
<td>cellSize</td>
<td>1 1 1 0 1 0 1 2 2 1 2 0 2 1 ...</td>
</tr>
<tr>
<td>cellOffset</td>
<td>0 1 2 3 3 4 4 6 7 9 11 12 14 14 16 ...</td>
</tr>
<tr>
<td>position'</td>
<td>xy' xy' xy' xy' xy' xy' xy' xy' xy' xy' xy' xy' xy'...</td>
</tr>
</tbody>
</table>

# Stage 1: Build cells lists.
Reset cellSize[] to 0
For each particle pIdx
  - Compute cellIndex from position[pIdx]
  - Increment cellSize[cellIndex]

Compute cellOffset from cellSize
Reset cellSize[] to 0
For each particle pIdx
  - Compute cellIndex from position[pIdx]
  - Set position'[cellSize[cellIndex]] to position[pIdx]
  - Increase cellSize[cellIndex]

# Stage 2: Compute forces.
For each particle pIdx
  - Reset fTotal to 0
  - Compute cix, ciy
  - For each ciy' in (ciy - 1, ciy + 1)
  - For each cix' in (cix - 1, cix + 1)
    - For each particle pIdx' belonging to cell (cix', ciy')
      - If pIdx' is under the cutoff:
        - Compute force between pIdx and pIdx'
        - Add force to fTotal
  - Save fTotal to totalForce[pid]

Swap position and position'
Before.

\[ f_{\text{Tmp}} = (0, 0) \]

For each particle in 9 neighboring cells

\[ F_{ij} = \ldots \]

\[ f_{\text{Tmp}} += F_{ij} \]

\[ f_{\text{Total}}[pIdx] = f_{\text{Tmp}} \]

After.

\[ f_{\text{Tmp}} = (0, 0) \]

For each particle in 4 neighboring cells

\[ F_{ij} = \ldots \]

\[ f_{\text{Tmp}} += F_{ij} \]

\[ f_{\text{Total}}[qIdx] -= F_{ij} \]

\[ \text{Atoms} \]

For each particle in current cell

If \( pIdx < qIdx \)

\[ f_{\text{Tmp}} += F_{ij}; \]

\[ f_{\text{Total}}[qIdx] -= F_{ij}; \]

\[ \text{Atoms} \]

\[ f_{\text{Total}}[pIdx] += f_{\text{Tmp}} \]

\[ \text{Atoms} \]

Optimization: Force symmetry

\[ F_{ji} = -F_{ij} \]
Only some of the considered pairs are useful (green vs red)

- 2D $\frac{\pi r_c^2}{(3r_c)^2} \approx 35\%$
- 3D $\frac{4}{3} \frac{\pi r_c^3}{(3r_c)^3} \approx 16\%$

Some threads of a warp want to skip the force computation, others don’t: leads to underutilization of warps.

Solution:
- First each thread of a warp determines which particles are within the cutoff (put in shared memory for performance).
- Then each thread compute forces.

Note (*):
- This bookkeeping adds extra overhead, and is useful only when interaction computation is very expensive.

For another approach, see Rossinelli et al (2015).