Parallelization and Optimization of 4D Binary Mixture Monte Carlo Simulations using Open MPI and CUDA

Sergey Artemchuk and Paula A. Whitlock
Brooklyn College/CUNY
Many thanks to the High Performance Computing Center, College of Staten Island/CUNY for a generous grant of computer time

and

Marvin Bishop, Manhattan College
Why mixtures of hyperspheres in 4 dimensions?

- Hard sphere interactions govern most fluid and solid systems.

- Mathematically, the number of lattices increases as the dimensionality increases.

- Networks where each node has more than three connections are modeled by multidimensional representations.

- Networks that have different sized, non-interacting objects in them can be modeled as mixtures.
The Metropolis Monte Carlo Method applied to hard hypersphere systems

Must sample the Boltzmann distribution function

\[ f(R) = \frac{\exp\left[-\sum \phi(r_{ij})/k_bT\right]}{\int \exp\left[-\sum \phi(r_{ij})/k_bT\right]dR} \]  

(1)

\( R \) is the d-dimensional vector of coordinates of the centers of mass of the M hyperspheres and \( r_i = (x_{i1}, x_{i2}, \cdots, x_{id}), \ i = 1, \cdots, M \)

\( k_b \) is Boltzmann’s constant and \( T \) is the absolute temperature of the system.

The pair potential, \( \phi(r_{ij}) \) represents the interaction between two hyperspheres.
The sampling algorithm generates random walks

Propose a move from the current position of a hypersphere, \( \mathbf{X} \), to a new position, \( \mathbf{X}' \) chosen from a probability distribution function, \( H(\mathbf{X}' | \mathbf{X}) \).

The new position is accepted or rejected based on the probability \( p(\mathbf{X}' | \mathbf{X}) \).

A recursive relationship develops between the distribution functions, \( f_n(\mathbf{R}) \), represented by each step of the random walk.

As long as the system is ergodic and obeys detailed balance, \( f_n(\mathbf{R}) \rightarrow f(\mathbf{R}) \) is guaranteed to be true as \( n \), the number of passes, becomes large.
The network where the Monte Carlo simulations are executed

- 48 8 core SGI X3481U host machines with Intel Core 7i - Nehalem
- Each host has 2 Nvidia Fermi GPUs.
- Hosts are connected by a 40 Gbit/sec QDR infiniband.
Functional Unit Foot-Print of Data-Oriented PTX ISA
FERMI PTX ISA and micro-architecture, 512 (448)

Figure 5. NVIDIA's Fermi GPU architecture consists of multiple streaming multiprocessors (SMs), each consisting of 32 cores, each of which can execute one floating-point or integer instruction per clock. The SMs are supported by a second-level cache, host interface, GigaThread scheduler, and multiple DRAM interfaces. (Source: NVIDIA)
Virtual Topology

3D VIRTUAL TOPOLOGY WITH PERIODIC BOUNDARIES
Amount of data that needs to be transferred in 4D vs 3D
Data must be communicated to the GPUs as well as the CPUS on the network
Ghost Cells

1) kernel moves

GRID
- block 1
- block 2
- block 3
- block N

DATA size + initial system density
DATA size + RESERVED SPACE size = max system density

PREPARE GHOST BOUNDARIES

X
- DATA
- RESERVED SPACE
- LEFT
- RIGHT
- TOP
- BOTTOM
- FROT
- BACK

Y
- DATA
- RESERVED SPACE
- LEFT
- RIGHT
- TOP
- BOTTOM
- FROT
- BACK

Z
- DATA
- RESERVED SPACE
- LEFT
- RIGHT
- TOP
- BOTTOM
- FROT
- BACK

W
- DATA
- RESERVED SPACE
- LEFT
- RIGHT
- TOP
- BOTTOM
- FROT
- BACK

2) kernel prefixsum

GRID
- block 1
- block 2
- block 3
- block N

LEFT
- 011111222222233333...

RIGHT
- 011111222222233333...

TOP
- 011111222222233333...

BOTTOM
- 011111222222233333...

FROT
- 011111222222233333...

BACK
- 011111222222233333...

END

3) kernel reduction

MAX size = 6

4) COPY L,R,T,B,F,B

CPU

RAM

1) Perform random moves in parallel and check if particle's new position is within a boundary. For spheres that are on the ghost boundary write 1 in corresponding buffer with corresponding index.

2) Run prefix sum on data received from kernel 1

3) Using results from prefix sum & kernel 1 evaluation Copy particles that are on the boundary into corresponding indexes of LEFT, RIGHT, TOP, BOTTOM, FROT, BACK buffers

4) Host will copy received data & send to corresponding CPUs
Prefix Sum (stream compaction) implementation with CUDA

Example

<table>
<thead>
<tr>
<th>Stream Compaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>a b c d e f g h k l m n</td>
</tr>
<tr>
<td>1 0 0 1 0 1 0 0 0 1 0 1</td>
</tr>
</tbody>
</table>

prefix sum

<table>
<thead>
<tr>
<th>Fragmented Prefix Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>block</td>
</tr>
<tr>
<td>1 0 1 1 0 1 0 1 1 0 1 0 0 0 0 1</td>
</tr>
<tr>
<td>1 1 2 3 0 1 1 2 1 1 2 2 0 0 0 1</td>
</tr>
<tr>
<td>3 2 2 1</td>
</tr>
<tr>
<td>prefix sum</td>
</tr>
<tr>
<td>3 5 7 8</td>
</tr>
<tr>
<td>+ + +</td>
</tr>
<tr>
<td>1 1 2 3 0 1 1 2 1 1 2 2 0 0 0 1</td>
</tr>
<tr>
<td>1 1 2 3 3 4 4 5 6 6 7 7 7 7 7 8</td>
</tr>
</tbody>
</table>
Copy ghost cells data from GPU to CPU
Copy ghost cells data from CPU to GPU
4D space visualization (parallel mirrors)
**Initialization**

**DATA INITIALIZATION**

On CPU initialization $O(n^4)$

```plaintext
for x -> n
  for y -> n
    for z -> n
      for w -> n
        set_point(x, y, z, w)
```

In simple case on Fermi max dimension of the block cannot exceed 32x32. If this condition is satisfied 4D cube could be initialized in N steps as opposed to $N^4$ on CPU.

On GPU initialization $O(n)$

```plaintext
for blocks_idx 0 -> n in parallel do
  for w -> n
    block[block_idx].set_points(x[0..n], y[0..n], z[blocks_z], w);
```
Distributing the Mersenne Twister pseudorandom number generator over the network.

**Step 1**

1. Load random number generator configuration parameters from the file.
2. Broadcast copy of config parameters data.

**Step 2**

CPU 0 → GPU 0
CPU 1 → GPU 1
CPU 2 → GPU 2
CPU 3 → GPU 3
CPU 4 → GPU 4
CPU N → GPU N

**Step 3**

GPU 0 → Global Memory
GPU 1 → Global Memory
GPU 2 → Global Memory
GPU 3 → Global Memory
GPU 4 → Global Memory
GPU N → Global Memory

Mersenne Twister generates an array of random numbers
rng.cfg
Results from running multiple threads on a single CPU/GPU pair

<table>
<thead>
<tr>
<th>Number of hyperspheres</th>
<th>256</th>
<th>512</th>
<th>1024</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computer code</td>
<td># of threads</td>
<td>time(s)</td>
<td>time(s)</td>
</tr>
<tr>
<td>C program on host</td>
<td>1</td>
<td>6.5</td>
<td>26.73</td>
</tr>
<tr>
<td>Cuda code on GPU</td>
<td>1</td>
<td>75.56</td>
<td>296.81</td>
</tr>
<tr>
<td>Cuda code on GPU</td>
<td>32</td>
<td>5.37</td>
<td>17.38</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>3.09</td>
<td>9.54</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>1.96</td>
<td>5.78</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>1.32</td>
<td>3.77</td>
</tr>
<tr>
<td></td>
<td>512</td>
<td>--</td>
<td>2.8</td>
</tr>
</tbody>
</table>
Simulation distributed over a network of 8 cpu/gpu pairs. Each subdomain is 2X2X2.

<table>
<thead>
<tr>
<th>System Size</th>
<th># of subdomains per side</th>
<th>IO Time (seconds)</th>
<th>Total Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4X4X4X4</td>
<td>2</td>
<td>5.174</td>
<td>8.588</td>
</tr>
<tr>
<td>6X6X6X6</td>
<td>3</td>
<td>14.825</td>
<td>20.413</td>
</tr>
<tr>
<td>8X8X8X8</td>
<td>4</td>
<td>33.870</td>
<td>45.209</td>
</tr>
<tr>
<td>10X10X10X10</td>
<td>5</td>
<td>65.410</td>
<td>86.934</td>
</tr>
<tr>
<td>12X12X12X12</td>
<td>6</td>
<td>112.126</td>
<td>166.576</td>
</tr>
<tr>
<td>14X14X14X14</td>
<td>7</td>
<td>179.147</td>
<td>274.769</td>
</tr>
<tr>
<td>16X16X16X16</td>
<td>8</td>
<td>268.812</td>
<td>440.752</td>
</tr>
<tr>
<td>18X18X18X18</td>
<td>9</td>
<td>384.145</td>
<td>644.352</td>
</tr>
</tbody>
</table>
Conclusions

- Significant speedups in runtime were achieved by converting the code to run on the GPU
- Dividing configuration space into subdomains allows much larger systems, up to 104,976 hyperspheres, to be studied
Future Improvements

- Use streams to copy data between the CPU and GPU
- Use an optimized stream compaction algorithm:
  - http://gpgpu.org/developer/cudpp
- Using the warp vote functions would increase the rate of hypersphere overlap detection.