Carnegie Mellon

CUDA Force-directed Graph Drawing

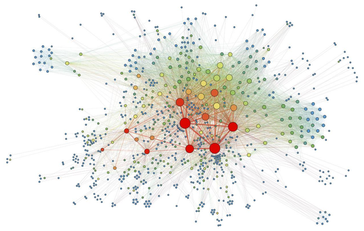
Qidu He (qiduh), Di Jin (djin2)

Overview

We parallelize a force-directed algorithm that considers force between any two nodes to draw an aesthetically-pleasing graph on NVIDIA GPUs.

Background

- We implement and optimize a specific version of force-directed graph drawing algorithm called *Fruchterman-Reingold*.
- Regard vertexes as steel rings and edges as springs between them. The attractive force is analogous to the spring force and the repulsive force is analogous to the electrical force.
- Minimizes the energy of the system by moving the vertexes and changing the forces between them.



Goal

- Parallelize the algorithm on GPU and CPU (extra).
- Compare and analyze the performance of several versions.

Challenge

- The time complexity of computation is O(mn^2) for *m* iterations and *n* vertexes, which is unacceptable for large datasets.
- Next position of one vertex depends on every other vertex, thus synchronization needs to be done after each iteration.
- Time locality and space locality of memory access is poor.

Method

Sequential Version

```
for i := 1 -> iterations
    for v in V
        v.disp = 0;
        for u in V //repulsive force
            d := v.pos - u.pos
            v.disp += (d/|d|) * fr(|d|)
        for e in E //attractive force
            d := e.v.pos - e.u.pos
            e.v.disp -= (d/|d|) * fa(|d|)

    for v in V //update
        v.pos += (v.disp/|v.disp|) * min(v.disp, t)

    t = cool(t)
```

Parallel Version on CPU (OpenMP)

Parallelize the sequential version by adding OpenMP annotation for outer vertexes loop.

Parallel Version on GPU (Naïve)

Each iteration contains 2 steps, force calculation kernel and update kernel. Nodes are evenly assigned to every thread.

Parallel Version on GPU (Barnes-Hut Tree)

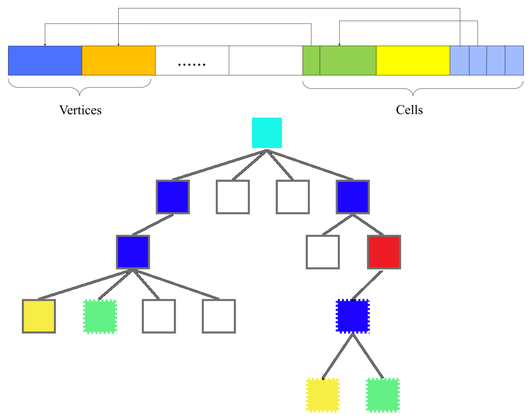
* *An efficient CUDA implementation of the tree-based barnes hut n-body algorithm.*

Each iteration contains 6 steps:

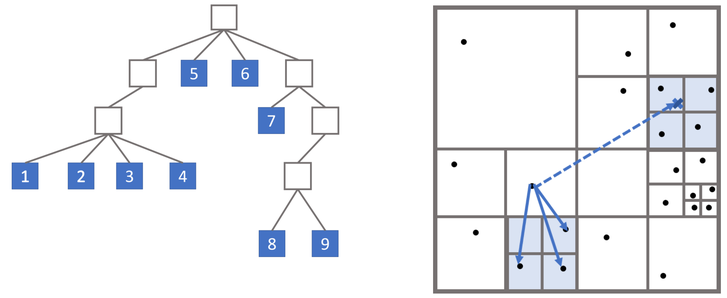
Kernel 1. Compute Bounding Box
Compute bounding box around all vertexes by calling min_element and max_element functions in Thrust

Kernel 2. Build Tree
Build hierarchical decomposition by inserting each vertex into quadtree in parallel. Consistency of the tree is maintained by adding fine-grained locks (CAS). To minimize space usage, the lock is represented by a special value stored in the child pointer array.
Data Representation: The quadtree is represented with a child-pointer array where each node has four elements in this array, each element stores the index of its child. The vertices are located at the beginning of the array and the cells are allocated from the other side.

Kernel 3. Summarize Internal Node Information
After the tree is built, the center and the mass of each cell will be computed for further calculation. Each cell wait until all its child finishes computation. Thread fences are added to ensure the order.



Kernel 4. Sort Vertexes by Spatial Distance
Concurrently place vertexes into an array such that they appear in the same order as in-order traversal of the quadtree. Group spatially close (in tree) vertexes together.



Kernel 5. Compute Forces
Assign vertexes to threads in round-robin fashion. Notice that every thread has to traverse the union of the tree prefixes of all threads in the warp. Reduce thread divergence with sorting and further eliminate it by expanding tree prefix to encompass the entire union. Moreover, allow only one thread per warp to read the pertinent data and cache it in shared memory.

Kernel 6. Update Positions

Results

* Seconds per iteration, #OpenMP threads = 236, # GPU threads = 32*512

Dataset	#Vertex	#Edge	Sequential	OpenMP	Naïve Parallel	Barnes-Hut
Test	200	199	0.0032	0.0024	0.00019	0.00083
Physicians	241	1098	0.0048	0.0047	0.00024	0.00084
WikiVote	8340	103722	1.97	0.012	0.0061	0.0026
Gowalla	196591	950327	1096.71	3.54	1.73	0.078
Texas	1379917	1921660	/	170.3	87.87	2.72
Hyves	1402673	2777419	/	176.14	91.91	2.8