Force Directed Placement: GPU Implementation

Bernice Chan
Ivan So
Mark Teper
Force Directed Placement

- Nodes represented by masses
- Edges represented by springs
- Motions of elements are simulated iteratively until steady state is reached
Basic Implementation

• The Force Directed Placement algorithm is implemented in both CPU and GPU
• CPU psuedo-code:
  
  do {
    calculate_forces()
    calculate_velocity()
    update_positions()
    compute_kinetic_energy()
  }while(kinetic_energy > threshold)

• Initial GPU algorithm achieved results similar to CPU performance
  – Parallelized each iteration at a node level
  – Used two kernels; one for computing new velocities and positions, second for computing kinetic energy.
Additional Optimizations

1. Increasing parallelism
   - Compute force between each pair of nodes in parallel (NxN threads)
   - New kernel to update velocities and positions per thread

2. Reducing Functional Units
   - Reordered floating point operations to reduce total number required

3. Reducing Sync Overhead
   - Compute the kinetic energy while updating velocity and positions to reduce increase work in each thread

4. Improve Memory Coalescing
   - Combine float x and y positions into float2 data
   - Change graph edge weights from char to ints
Additional Optimizations (2)

4. Using local memory
   – Cached all memory values locally before performing operations
5. Reducing bank conflicts
   • Transpose the force calculate kernel so that data can be coalesced in the position update kernel
6. Reducing memory accesses
   • Perform the force calculations on a block of the NxN
7. Using float4 instead of float
   • Similar to using float2 data structures use float4 to store both attractive and repulsive forces (in x and y direction)
GPU Speed-up vs. CPU

Benchmark Graphs

- actors.dot (n=100, e=260)
- actors2.dot (n=232, e=698)
- rand32.dot (n=32, e=31)
- rand64.dot (n=64, e=93)
- rand128.dot (n=128, e=388)
- rand256.dot (n=256, e=1599)
- rand512.dot (n=512, e=6481)

Speed-up (relative to CPU)

- GPU Best
- GPU Basic
GPU Optimizations
(Time vs. Benchmark Graph Size)

Size of Benchmark Graphs
(Number of Nodes + Number of Edges)

Time (ms)

0 1000 2000 3000 4000

Thousands

CPU

GPU Basic

GPU float2 (Memory Coalescing)

GPU (Increased Parallelism)

GPU (Reducing Functional Units & Sync Overhead)

GPU (Local Memory)

GPU data size (Memory Coalescing)

GPU Transpose (Reduce Memory Bank Conflict)

GPU Grid Block (Reducing Memory Accesses)

GPU Force Reduction

GPU float4 (Memory Access Coalescing)
GPU Optimizations (Speed-up vs. CPU)

Benchmark Graphs

- GPU float4 (Memory Access Coalescing)
- GPU Force Reduction
- GPU Grid Block (Reducing Memory Accesses)
- GPU Transpose (Reduce Memory Bank Conflict)
- GPU data size (Memory Coalescing)
- GPU (Local Memory)
- GPU (Reducing Functional Units & Sync Overhead)
- GPU (Increased Parallelism)
- GPU Basic
- CPU