Force Directed Placement: GPU Implementation

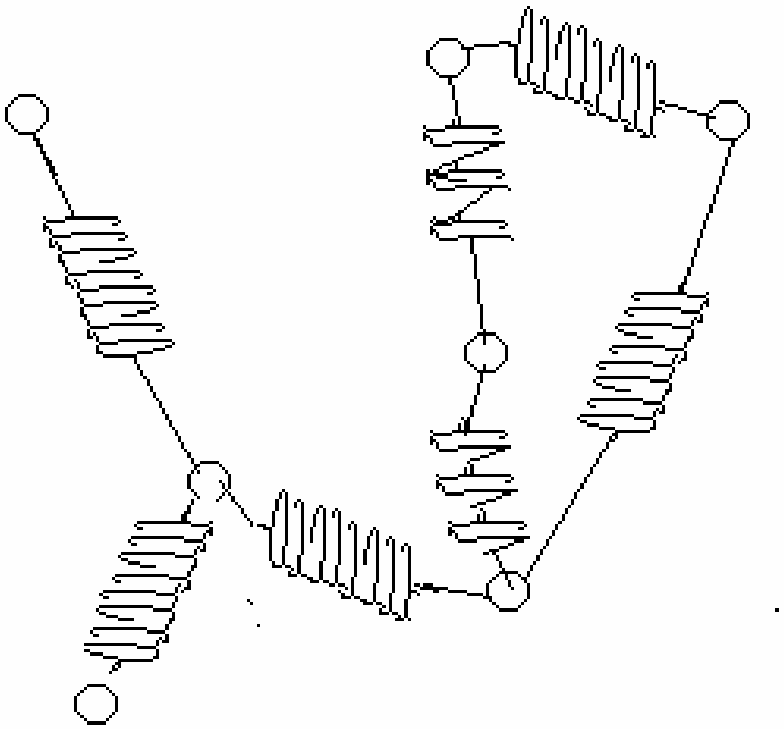
Bernice Chan

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

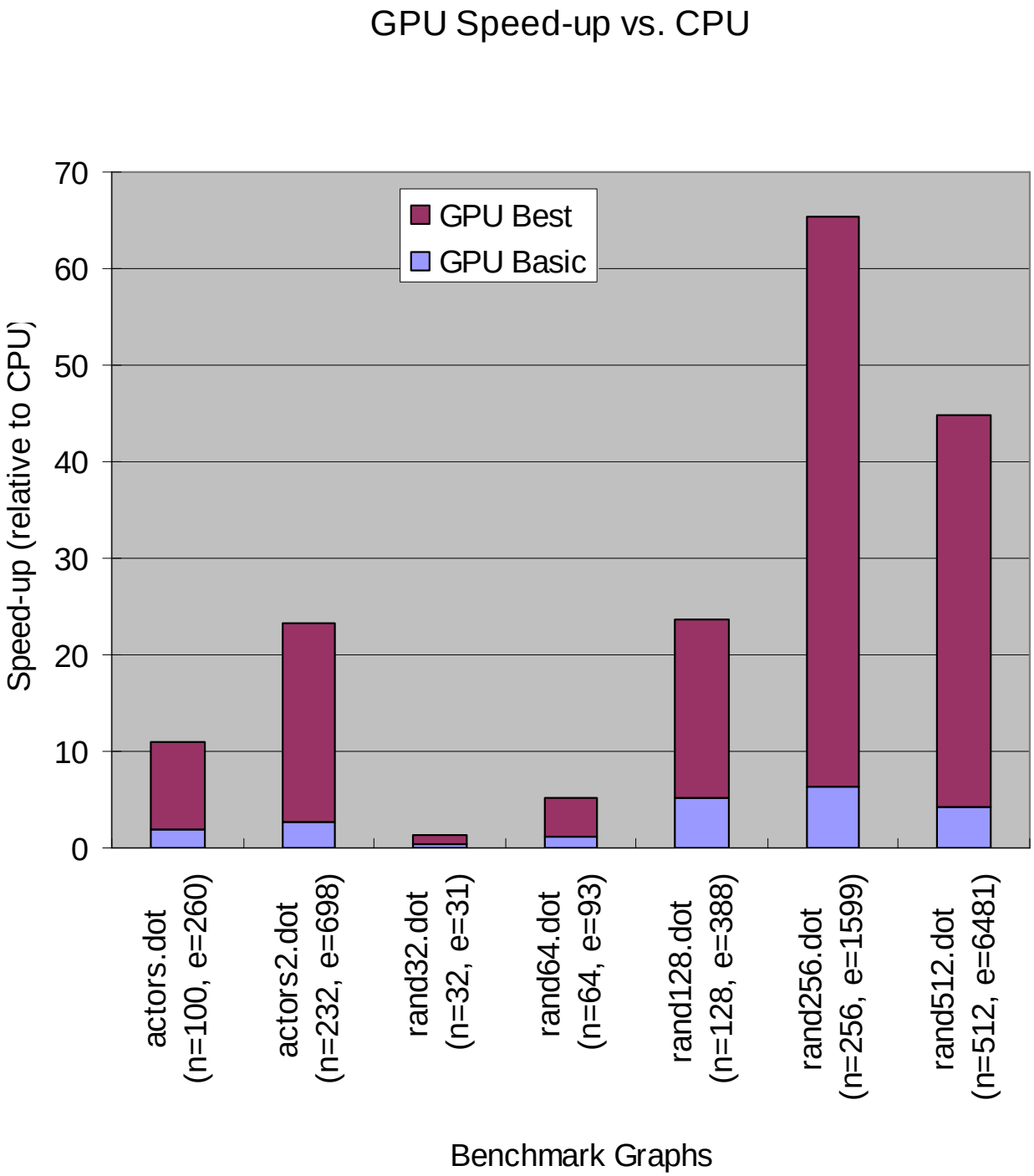
1. Increasing parallelism
   - Compute force between each pair of nodes in parallel (NxN threads)
   - New kernel to update velocities and positions per thread
2. Reducing Functional Units
   - Reordered floating point operations to reduce total number required
3. Reducing Sync Overhead
   - Compute 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 fro char to ints

	Calculate Forces					Calculate Velocity X and Velocity Y	Calculate Position X and Position Y
	Node0	Node1	Node2	Node ...	NodeN		
Node0		T1	T1	T1	T1	T1	T1
Node1	T2		T2	T2	T2	T2	T2
Node2	T3	T3		T3	T3	T3	T3
Node...	T...	T...	T...		T...	T...	T...
NodeN	TN	TN	TN	TN		TN	TN

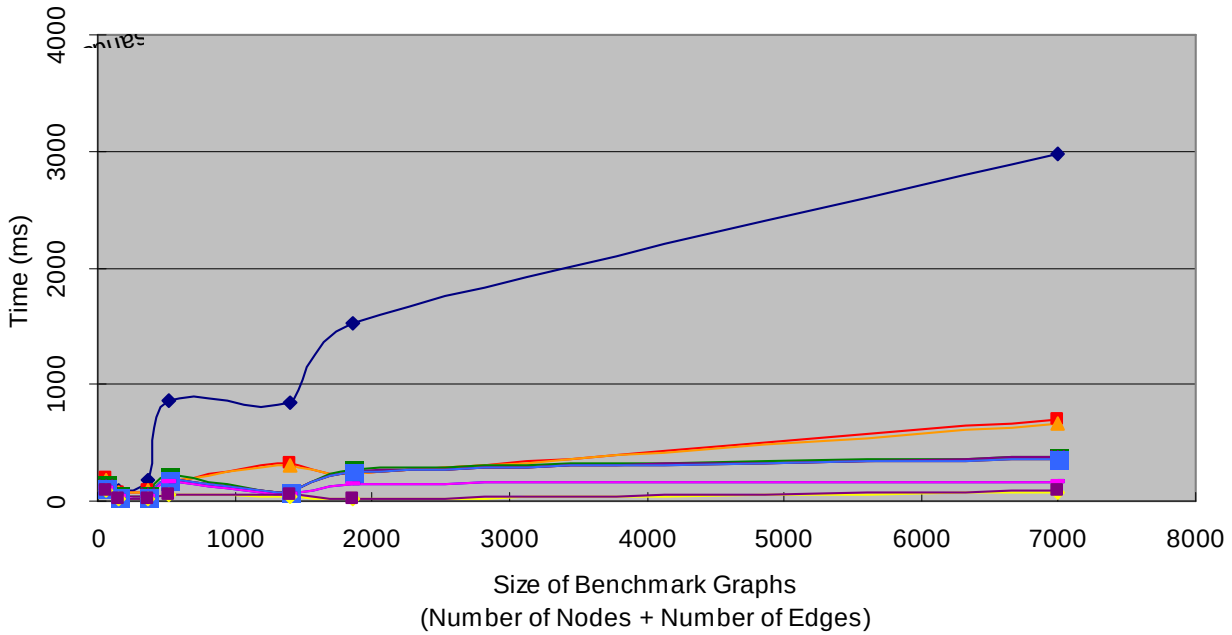
	Calculate Forces	Calculate Velocity X and Velocity Y	Calculate Position X and Position Y
Node0,1	T1	T1	T1
Node0,2	T2		
Node0,3	T3		
Node0,...	T4		
Node0,N	T5		
Node1,0	T6	T2	T2
Node1,2	T7		
Node1,3	T8		
Node1,...	T9		
Node1,N	T10		
Node...,0	T11	T3	T3
Node...,1	T12		
Node...,2	T13		
Node...,N	T14		
....	T...	T...	T...
NodeN,0	TN-4	TN	TN
NodeN,1	TN-3		
NodeN,2	TN-2		
NodeN,...	TN-1		
NodeN,N-1	TN		

Additional Optimizations (2)

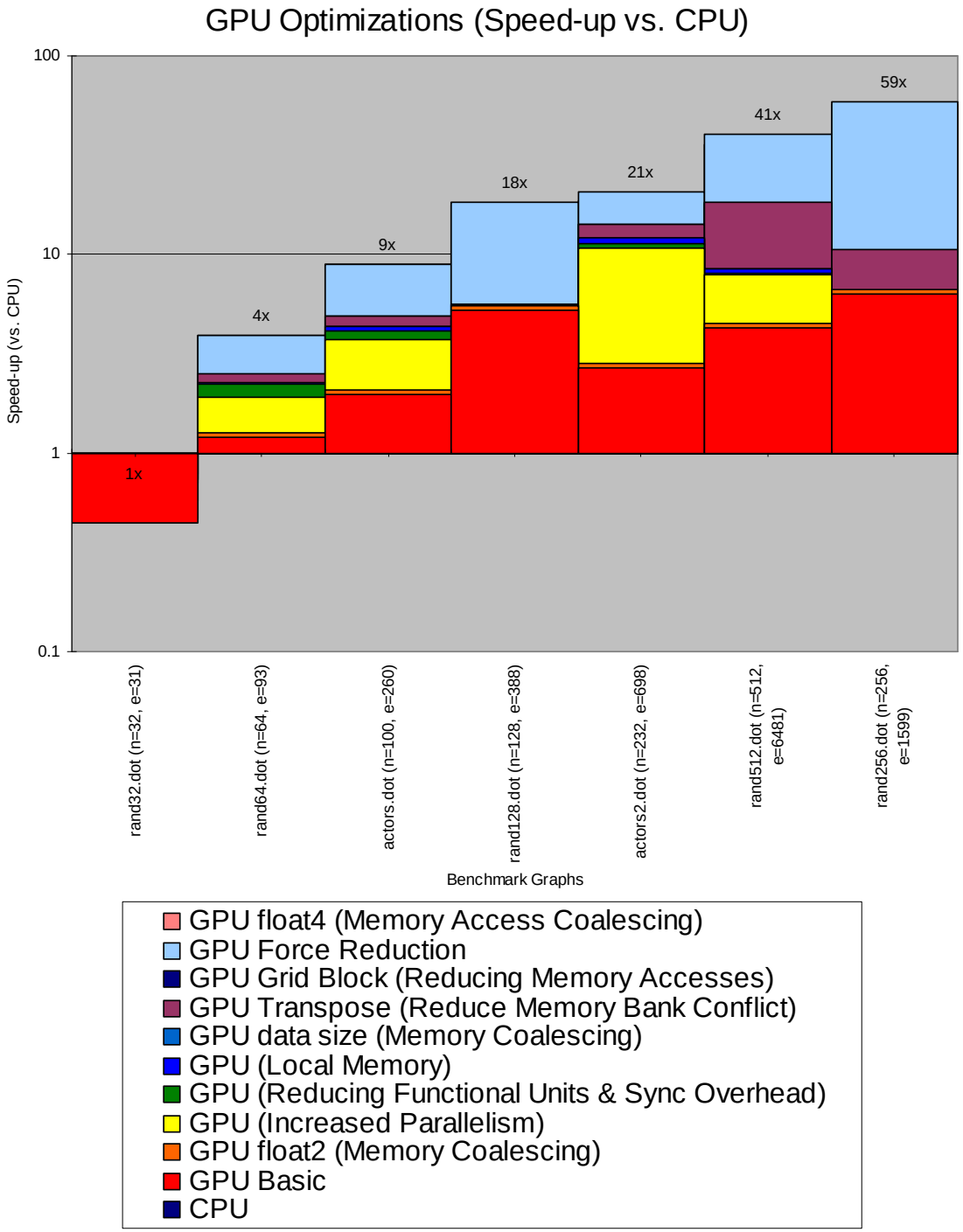
4. Using local memory
   - Cached all memory values locally before performing operations
5. Reducing bank conflicts
   - Transpose the force calculate kernel so that data can be coalesced in the position update kernel
6. Reducing memory accesses
   - Perform the force calculations on a block of the NxN
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
GPU Best
GPU Basic
Speed-up (relative to CPU)
70
60
50
40
30
20
10
0
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)
Benchmark Graphs

GPU Optimizations
(Time vs. Benchmark Graph Size)



- 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)
Speed-up (vs. CPU)
100
10
1
0.1
1x
4x
9x
18x
21x
41x
59x
rand32.dot (n=32, e=31)
rand64.dot (n=64, e=93)
actors.dot (n=100, e=260)
rand128.dot (n=128, e=388)
actors2.dot (n=232, e=698)
rand512.dot (n=512, e=6481)
rand256.dot (n=256, e=1599)
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 float2 (Memory Coalescing)
GPU Basic
CPU