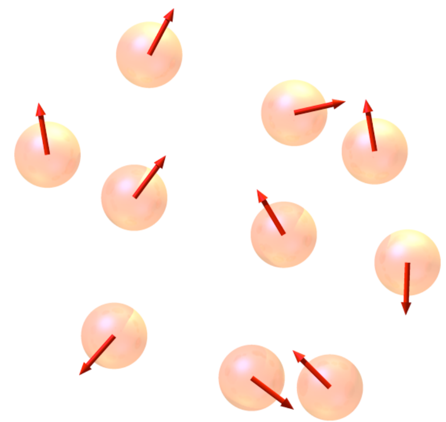
Parallelizing the Spot Model for Dense Granular Flow

18.337 Parallel Computing

Yee Lok Wong May 8, 2008
Department of Mathematics, MIT

Part 1: Background on Granular Flow and the Spot Model

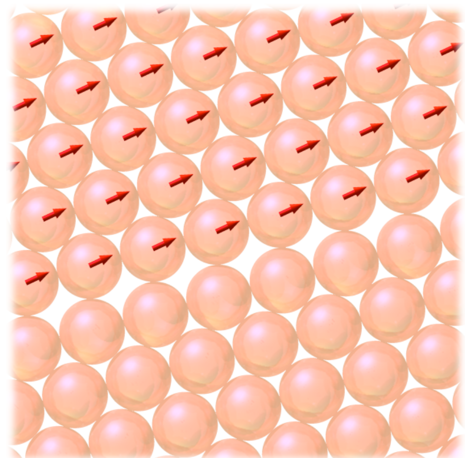
Microscopic Flow Mechanism of Granular Materials



Gas

Dilute, random “packing”

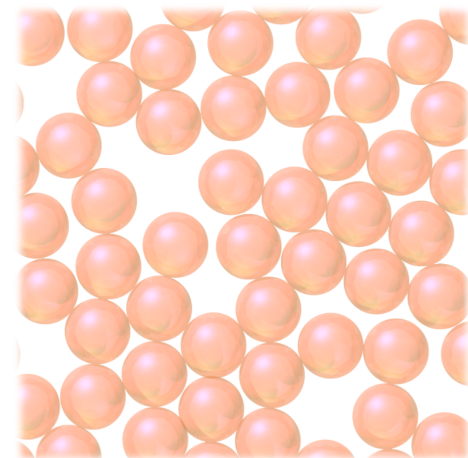
- Boltzman’s kinetic theory
- random collisions



Crystals

Dense, ordered packing

- Vacancy and Interstitial diffusion
- Dislocations and defects



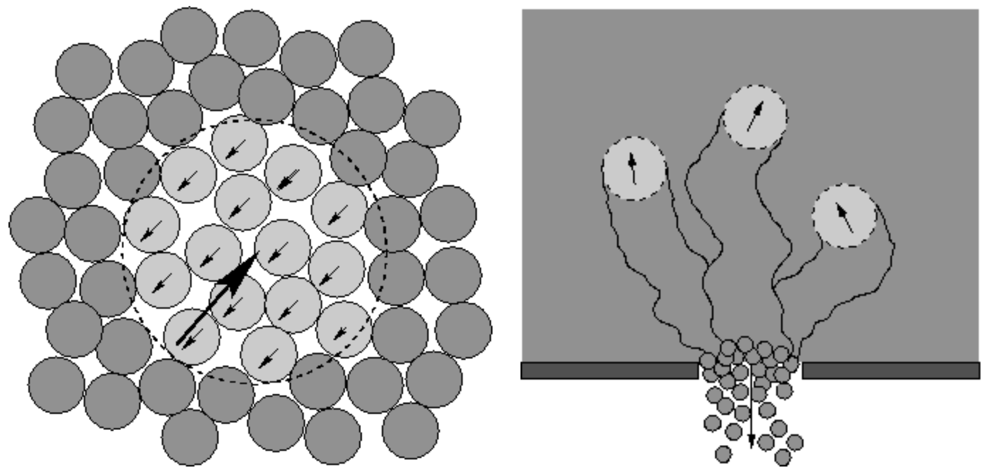
Granular

Dense, random packing

- Long lasting many-body contacts
- Lack of general microscopic model
- How to describe cooperative random motion?

Spot Model

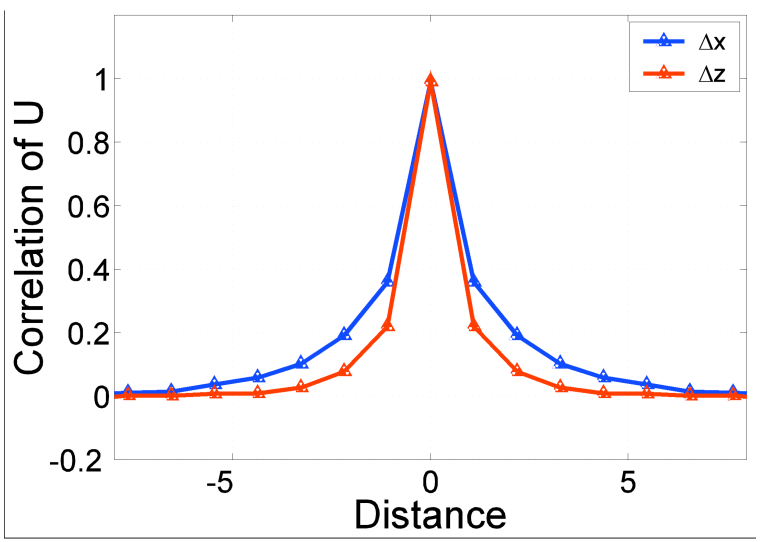
- "Spot" Model for random packing dynamics
  (Bazant et al., 2001)
- Developed for Silo Drainage
  - Spots - extended region of slightly enhanced interstitial
  - Spot move upwards from orifice, and also perform random walk at horizontal directions

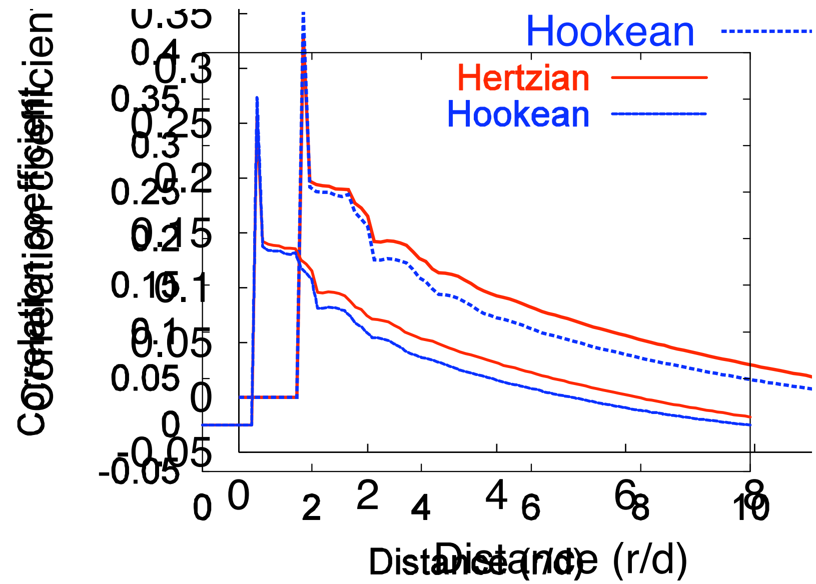


- When spots pass through particles, particles are displaced in the opposite direction

Velocity Correlation

- Motivation for Spot Model: Local velocity correlation suggests *correlated* motion

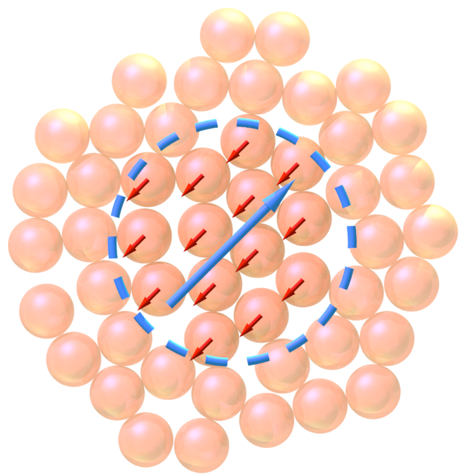




Experiments by MIT Dry Fluids Lab

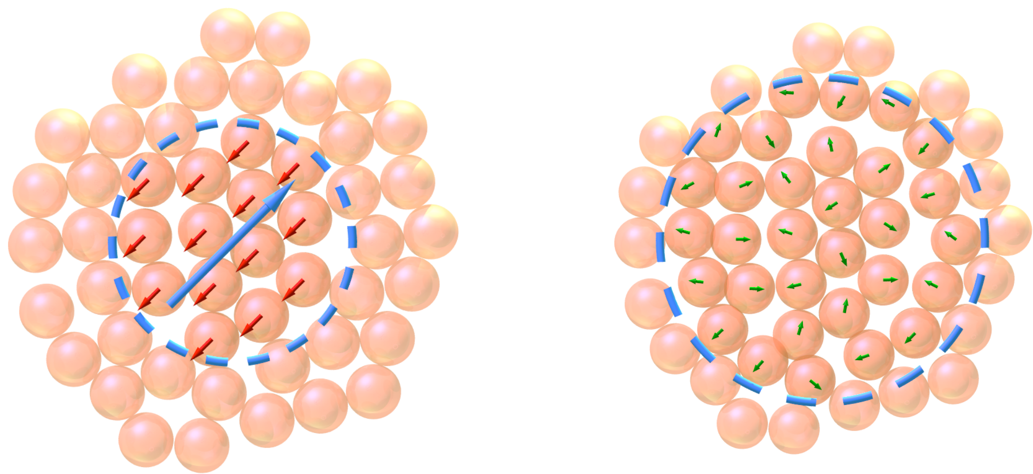
Simulation

Spot Model Microscopic Mechanism



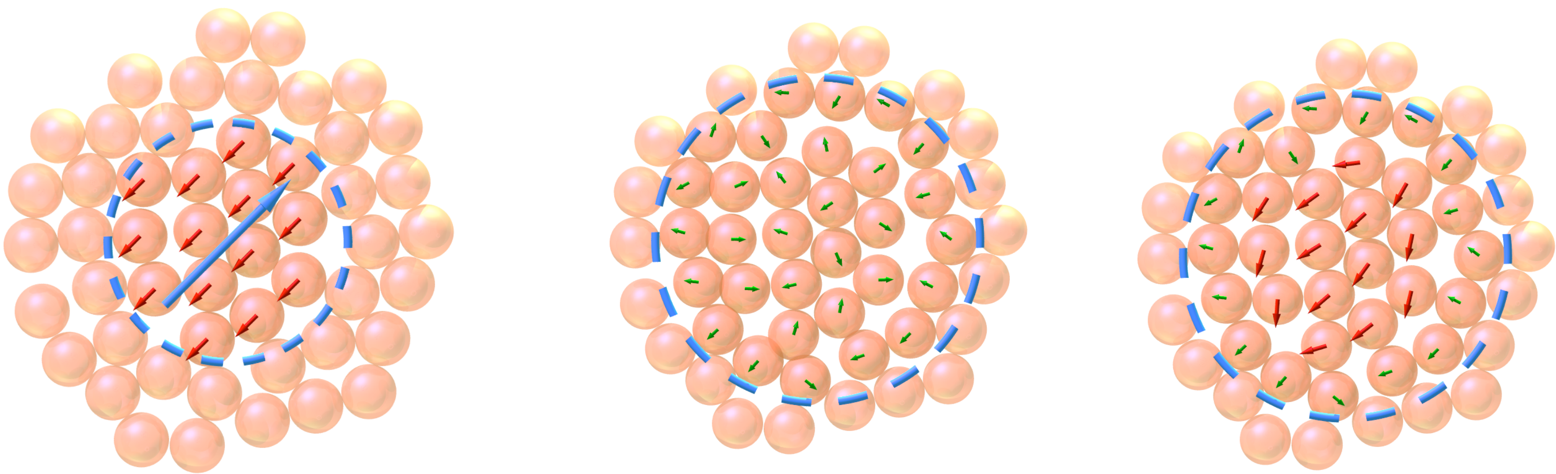
- Apply the spot displacement first to all particles within range
- Particles are displaced in the opposite direction

Spot Model Microscopic Mechanism



- Apply a relaxation step to all particles within a larger radius
- All overlapping pairs of particles experience a normal repulsive displacement (soft-core elastic repulsion)
- Very simple model - no “physical” parameters, only *geometry.*

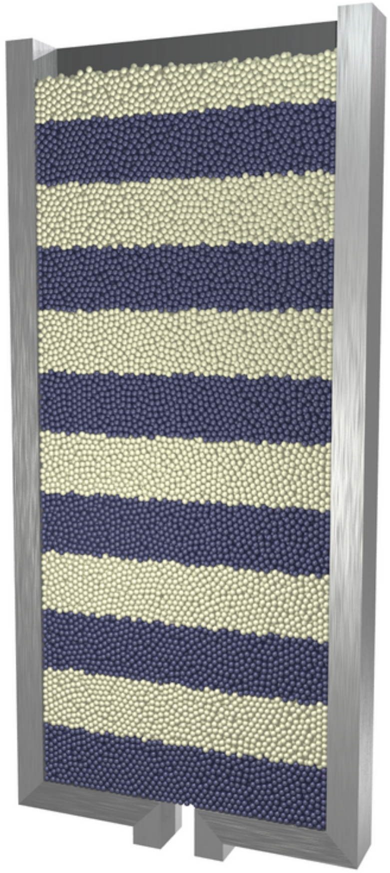
Spot Model Microscopic Mechanism



- Combined motion is bulk spot motion, while preserving packings
- Not clear *a priori* if this will produce realistic flowing random packings

DEM Simulations

- Discrete Element Method (DEM), codes developed by Sandia National Lab.
- Each particle is accurately modeled according to Newton's laws and a realistic friction model is employed to capture particle interactions
- Parallel code on 24 processors
- 50*d* x 8*d* x 110*d* container
- Drained from circular orifice 8*d* across



L. E. Silbert *et al.*, Phys Rev E, **64**, 051302 (2001)

J.W. Landry *et al.*, Phys Rev E, **67**, 041303 (2003)

Spot Simulations using C++

- Initial packing taken from DEM
- Spots introduced at orifice
- Spots move upwards and do random walk horizontally
- Systematically calibrate three parameters from DEM:
  - Spot radius *Rs* (from velocity correlations)
  - Spot volume *Vs* (from particle diffusion)
  - Spot diffusion rate *b* (from velocity profile width)

Comparison with DEM simulation

DEM

Spot Model

t = 1.05 s t = 2.10 s t = 3.15 s t = 4.20 s

Comparison with DEM simulation

- DEM: 3-7 days on 24 processors
- Spot Model Simulation: 8-12 hours on a single processor
- A factor of ~10^2 speedup

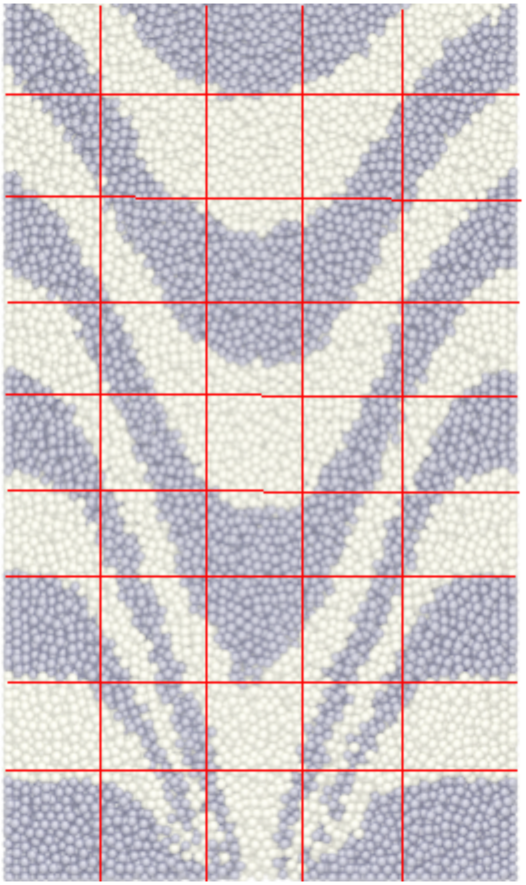
- Simulations run on AMCL

Part 2: Parallelizing the Spot Model

C++ codes

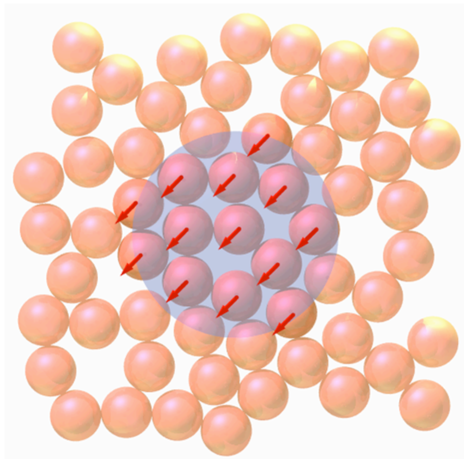
- Split into regions,each storing particles within it

```
class container {
    void import();
    void put(int n, vec &v);
    void dump();
    void regioncount();
    int count(vec &p, float r);
    ...
}
```



Important Routines

Spot Motion



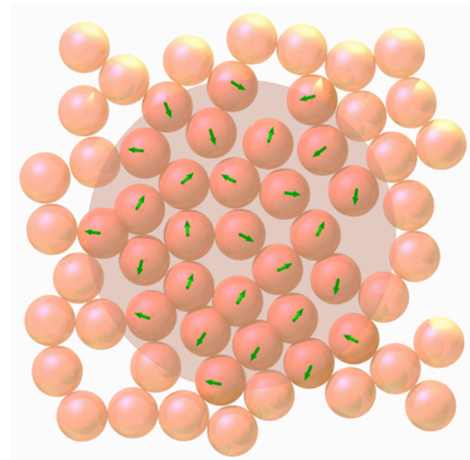
- void spot(vec &p,vec &v, float r);

  p: position

  v: displacement

  r: spot radius

Relaxation



- void relax(vec &p, float r, float s, float force, float damp, int steps);

  p: position

  r: inner relaxation radius

  s: outer relaxation

  force: particle repulsive force

  damp: particle velocity damping

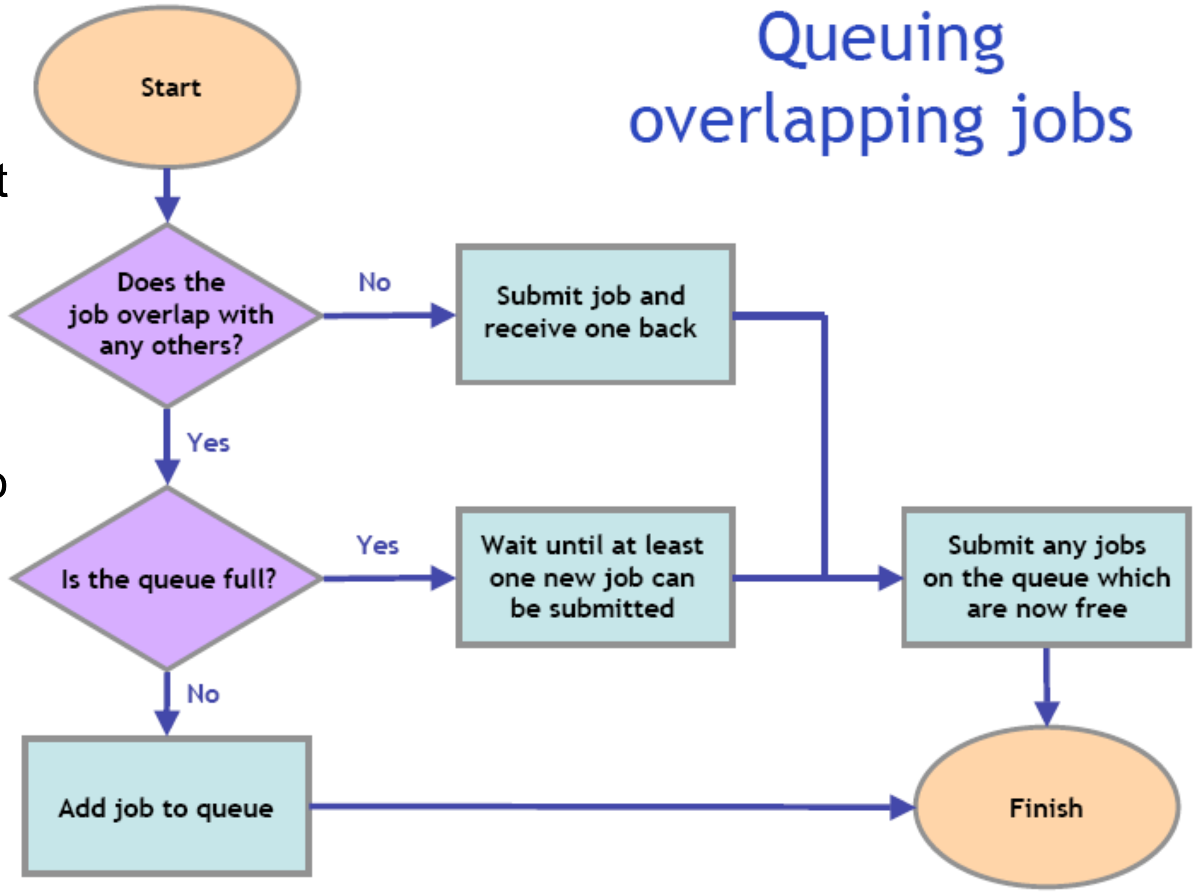
  steps: relaxation steps

Possible for parallel computing

- *Serial*: the elastic relaxation step is the computational bottleneck since it requires analyzing all pairs of neighboring particles within a small volume.
- In a *parallel* version, ideally we can distribute this computational load across many processors.
- Since each relaxation event occurs in a local area, we can pass out different relaxation jobs to different processors.
- Serial code written in C++ ---> Use MPI for parallel

Master/Slave

- entire state of the system (particle positions and spot positions) held on the master node
- The master node sequentially passes out jobs to the slave nodes for computation and receive them back.



Rycroft 2006

Master/Slave

- Timing results: computed 60 frames of snapshots and calculated the average time per frame.
- Run on AMCL

# of slaves	Time per frame (s)	Speedup	Efficiency
(Serial)	289	1	1
1	241	1.199	59.96%
3	414	0.698	17.45%
5	512	0.564	9.41%
7	551	0.524	6.56%

Master/Slave

- Problems:
  - too much stress is placed on the master node
  - very poor scalability with the number of nodes, as the slaves often stand idle waiting for the master node to pass jobs to them

Distributed Algorithm

- Container is divided up between the slaves, with each slave holding the particles in that section of the container.
- A master node holds the position of the spots and computes their motion. When a spot moves, the master node tells the corresponding slave node to carry out a spot displacement of the particles within it.
- Only the position and displacement carried by the spot need to be transmitted to the slave.
- Drawback:
  - A spot's region of influence may overlap with areas managed by other slaves.
  - Each slave must transmit particles to the slave carrying out the computation, and then receive back the displaced particles. (Communication between slaves is required)

Distributed Algorithm

- Timing results: (implemented and run on SiCortex)

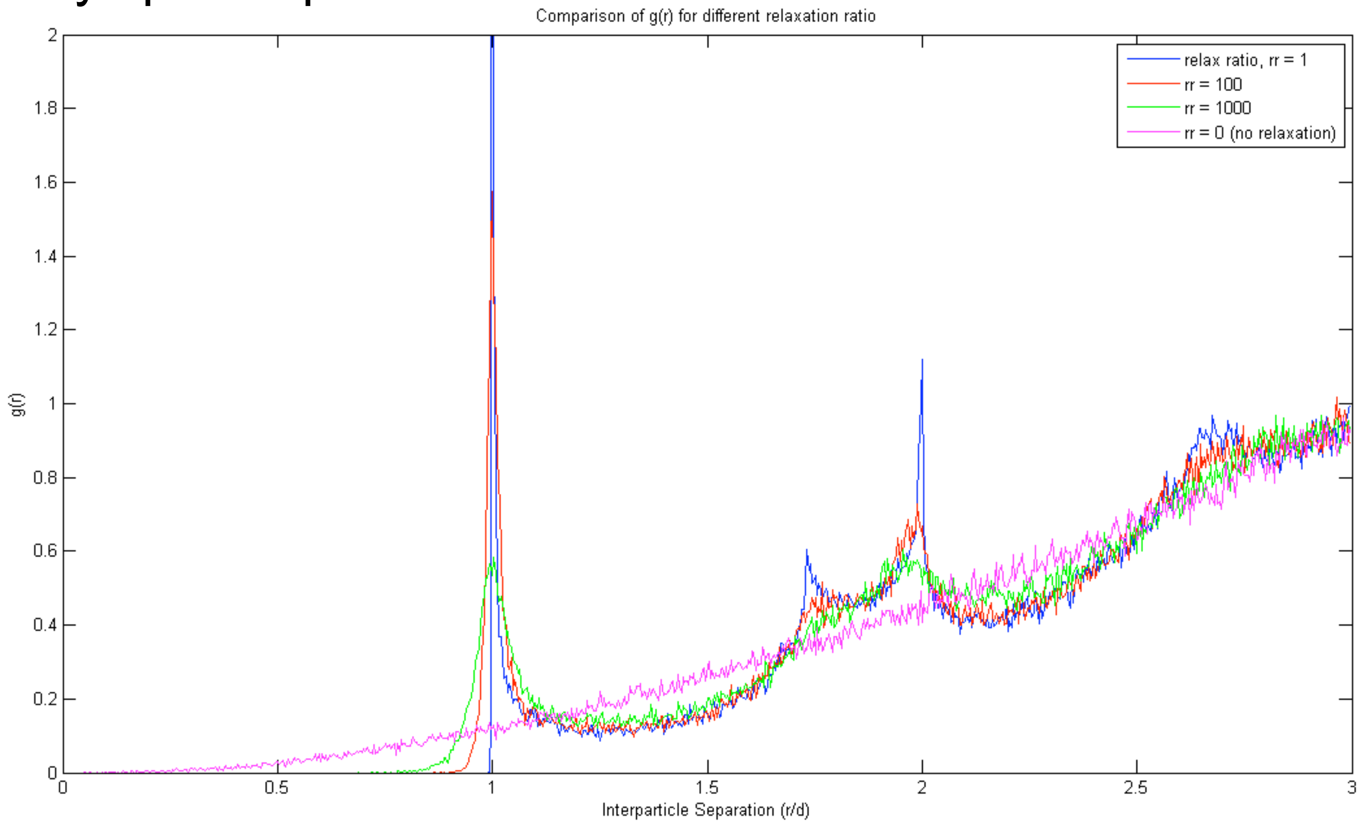
# of slaves	Processor Grid	Time per frame (s)	Speedup	Efficiency
(Serial)	1x1x1	1256	1	1
2	1x1x2	821	1.529	50.99%
3	1x1x3	674	1.864	46.59%
4	1x1x4	569	2.207	44.15%
5	1x1x5	515	2.439	40.65%
6	1x1x6	476	2.639	37.70%
7	1x1x7	446	2.816	35.20%
8	1x1x8	425	2.955	32.84%
9	1x1x9	406	3.094	30.94%
10	1x1x10	387	3.245	29.50%

Distributed Algorithm

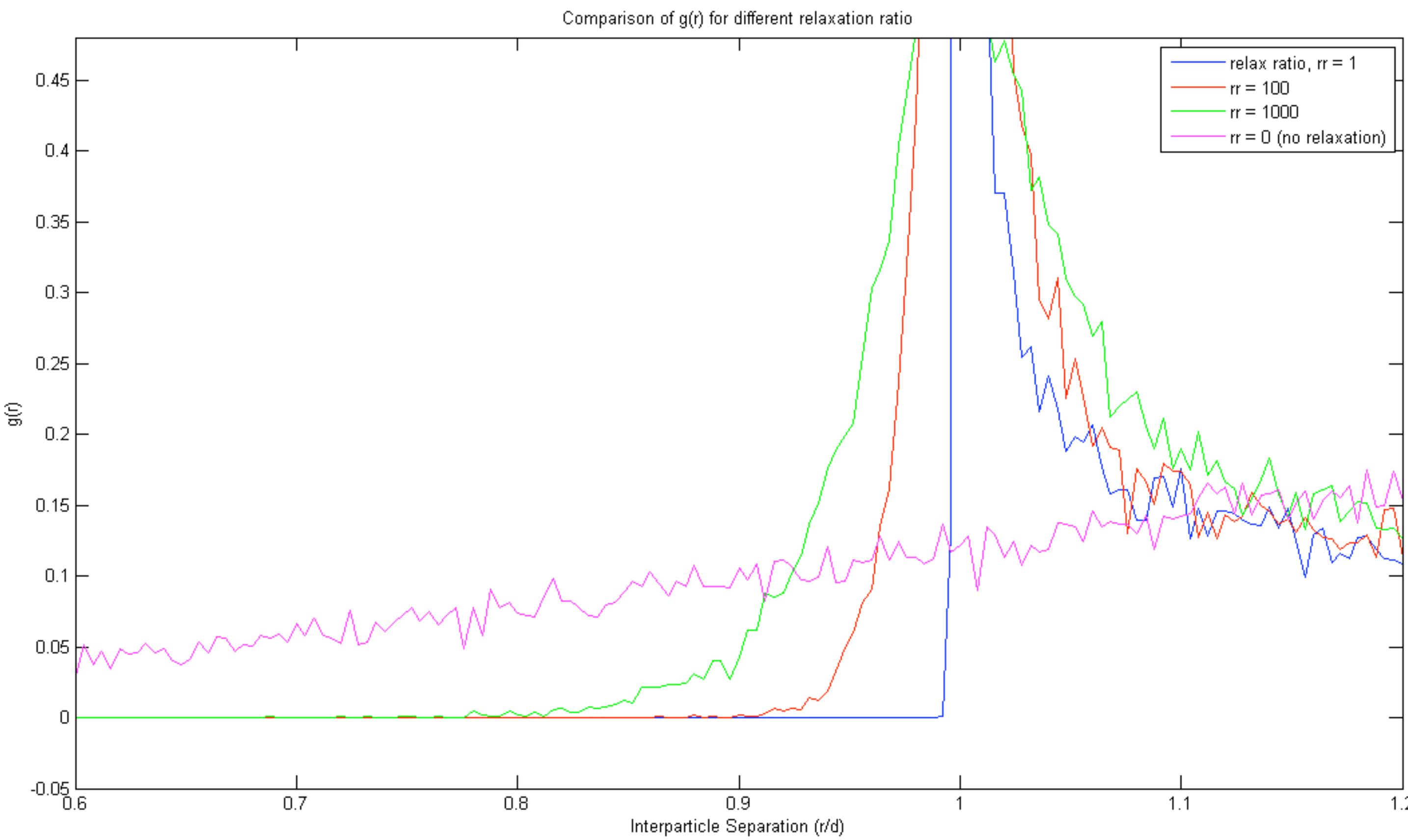
- Much better speedup compared with master/ slave method, but still not optimal
- Bottleneck: Overlapping Spot Motion
  - One slave needs to transfer its particles to another slave, then wait for the computation and receives back particles that are in the region it controls

A Faster Distributed Algorithm

- Motivation: The elastic relaxation step can “magically” fix a lot of the unphysical packings, even if we do not apply relaxation every spot step.



A Faster Distributed Algorithm



A Faster Distributed Algorithm

- For overlapping spot motion, both slaves responsible for the region of the spot influence carry out spot computation independently, and exchange particles that are out of range if necessary
- May not be 100% accurate, but significantly reduce waiting time and size of messages being exchanged between slaves

A Faster Distributed Algorithm

- Timing results: (implemented and run on SiCortex)

# of slaves	Processor Grid	Time per frame (s)	Speedup	Efficiency
(Serial)	1x1x1	1256	1	1
2	1x1x2	687	1.827	60.91%
3	1x1x3	458	2.745	68.63%
4	1x1x4	334	3.757	75.13%
5	1x1x5	254	4.950	82.50%
6	1x1x6	207	6.054	86.48%
7	1x1x7	176	7.134	89.18%
8	1x1x8	151	8.319	92.44%
9	1x1x9	132	9.502	95.02%
10	1x1x10	116	10.86	98.75%

A Faster Distributed Algorithm

- Significant speedups and very good scalability with number of slaves
- Problems with this approach occur near the boundaries of regions owned by each slave. Larger errors with increasing number of processors since the container is divided into more regions.

Conclusion

- Master/slave method didn’t do so well
- Distributed Algorithm gave satisfactory results
- Significant speedup by Faster Distributed Algorithm, but balance between accuracy and speed
- Possible future work considering other algorithms