Parallelizing the Spot Model for Dense Granular Flow

Yee Lok Wong

Department of Mathematics, MIT

May 15, 2008

Abstract

The spot model [1] provides a fast method to simulate flow in dense granular media, and this has been implemented as a serial algorithm in C++ for the case of granular drainage from a silo. We study previous work on parallelizing the spot model [7], which include two parallelization algorithms, one based on a master/slave scheme, and another based on distributed algorithm, both implemented with C++/MPI. With some loss in accuracy of packings, the distributed algorithm can be modified to run significantly faster, with simulation speedup scaling almost linearly with the number of processors. Further directions of studying a parallel spot algorithm are proposed.

1 Introduction

Granular materials have received a great amount of attention from the research community in the past twenty years[3, 4], motivated by the possibility of new physics to be discovered. Granular matrials are frequently viewed in connection with systems featuring strong confinement such as glasses, and its dense random packing dynamics are at the heart of condensed matter physics. Thus a better understanding of granular materials can also lead to a broad range of applications.

Granular materials cannot be characterized as a gas, liquid or solid, and they can exhibit behavior characteristic of each of these three phrases in certain situations [4]. The behavior of dilute, collisional granular media have been explained using a modified version of Boltzmann's kinetic theory to account for inelastic random collisions. At the opposite limit, densely packed static granular materials exhibit dislocations, voids and grain boundaries like crystals. But how about slow flow in a dense regime, where particles exhibit long-lasting many-body contacts? How do random packing in granular materials flow, and how do we describe such cooperative random motions?

Despite many attempts to derive a continuum model for dense granular flow, a general theory is still lacking. The MIT Dry Fluids group has been studying the regime of dense granular flow, such as the drainage of glass beads in a hopper. Surprisingly, there are very few models to accurately describe this simple situation, and even an accurate description of the mean flow is lacking. For more complex questions, such as how much particle diffusion we expect to see, or how the random packing of particles behaves, very little is known.

To study these phenomena, our group has made use of both experiments and simulations. However, experiments often have many drawbacks to study microscopic data, since it is very difficult to experimentally extract information from a fully three dimensional system. Either we are restricted

to looking at a two dimensional flow, which often exhibits very different behavior, or we are faced with the challenges of trying to accurately measure into the bulk of a three dimensional system.

Simulation is therefore an easier way to obtain complete information about a flow without any experimental complications. In the past few years, our group made use of the AMCL cluster to carry out Discrete Element Method (DEM), which is a code written by Sandia National Laboratories [5, 8]. Each particle is accurately modeled according to Newton's laws and a realistic friction model is employed to capture particle interactions. Since we are solving Newton's equations for each interacting pair in each time step, the system is very stiff and requires very small timesteps. The code can be run in parallel, and is typically executed on systems from 30-90 processors, handling up to half a million particles. However, the large computational cost of these simulations presents a significant drawback. Even for relatively small granular systems of practical importance, DEM simulations require a cluster of processors, and may take an extremely long time to complete. For example, silo drainage of 55000 particles can take up to a week on 24 processors on AMCL.

In 2001, the Dry Fluids Group has proposed the "Spot Model" [1], which is a cooperative mechanism for dense random-packing dynamics based on diffusing "spots" of interstitial free volume. When a spot moves, it induces a small, correlated motion of all particles within a range, followed by an internal relaxation with soft-core repulsion, which yields the net cooperative motion. This simple mechanism suffices to capture many results seen in granular drainage experiments, such as particle diffusion and spatial velocity correlations. This report studies the parallel computing aspect of the Spot Model. In section 2, we discuss the algorithm in more detail and its implementation as a serial code, and compare the results to those from DEM. Since flow in the spot model is made up of local group displacements on the length scale of several particle diameters, and includes no long-range effects, it lends itself very well to parallelization. In section 3, we review previous work on parallel implementation of the code by Chris Rycroft. In section 4, we discuss a faster distributed parallel algorithm, with a tradeoff between speed and slightly loss in accuracy of packing. In section 5 we conclude with possible future directions.

2 Spot Model

2.1 Microscopic Mechanism

Our intuition tells us that a particle in a dense random packing of flowing amorphous materials does not move individually, but moves together with its neighbors over short distances. Gradual cage breaking over longer distances then follows. This intuition is confirmed by experimental work by Choi et al. [2]. Using 3mm glass beads in a rectangular hopper with transparent walls, they carried out experiments of granular drainage. They were able to obtain very accurate information about particle velocities by using a high speed digital camera. In particular,they calculated the spatial velocity correlation function, $C(r)$, which measures how much the velocities of particles in the dense flow regime are correlated with their neighbors, as a function of interparticle distance. They found that a particle's motion was correlated with its neighbors on a length scale of several particle diameters. Also, the data suggests that in granular drainage, cage breaking occurs slowly, over time scales comparable to the exit time from the silo, so that cooperative motion is important throughout the container at the macroscopic scale.

Base on this previous work and observations, the MIT Dry Fluids Group has proposed the spot model [1, 6], shown in Figure 1, to describe the particle dynamics in a flowing amorphous material. Motion is mediated by 'spots', which represent a region of free interstitial space spread across several particle diameters, as shown by the blue circle in (a). When the spot moves according to the blue

arrow, it induces a small, correlated motion of all particles within range in the opposite direction. This simple mechanism suffices to capture many results seen in granular drainage experiments, such as particle diffusion and spatial velocity correlations.

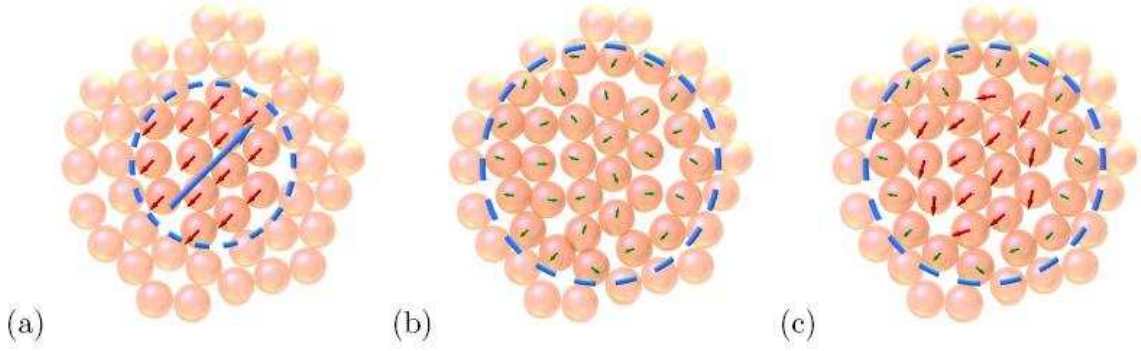

Figure 1: The mechanism for structural rearrangement in the Spot Model. The random displacement of a diffusing spot of free volume (dashed circle) causes affected particles to move as a block by an amount (a), followed by an internal relaxation with soft-core repulsion (b), which yields the net cooperative motion (c). (The displacements, typically 100 times smaller than the grain diameter, are exaggerated for clarity.) Figure reproduced from [7].

However, while this basic model remains simple enough for mathematical analysis, it is clear that it does not explicitly enforce packing constraints of the particles. Since the spots only displace a region of particles, the displace particles will overlap with some particles not displaced. In order to preserve valid packings, a very important second step has been proposed. After the block motion has been carried out, a small elastic relaxation step is applied, during which the particles and their nearest neighbors experience a softcore repulsion with each other, as shown in (b). Note that no physical parameters such as contact forces, energy, or momentum is considered. This is done on a purely geometrical basis. The net effect, as shown in (c), is then a co-operative local deformation, whose mean is roughly the original block motion.

2.2 Simulations with C++

Although the spot model mechanism is simple and intuitive, it is not clear *a priori* whether it will produce realistic flowing packings. Thus, the spot model has been studied by event-driven simulations using C++[6]. Spots are introduced at the silo exit following an exponential waiting time distribution to match the outflow of particles. Spots then move upwards through the packing following a random walk, moving in one of the horizontal directions with equal probability, and causing a corresponding downwards motion in the particles. Each spot moves according to an exponential waiting time distribution, and once it reaches the free surface, it is removed from the simulation.

The simulation was tested systematically using results by Discrete Element Method (DEM). The DEM simulation was first run, and then five parameters for the spot model including (1) the step size of the spot random walk, (2) the influence of a spot (how much particles are displaced by a spot), (3) the spot radius, (4) rate of introducing spot at the orifice, and (5) waiting time between

two spot movements, were calibrated off the DEM data.

The initial packing of 55000 particles in a $50d \times 8d \times 110d$ container from the DEM was imported as the initial packing for the spot model. The two simulations match to a high degree of accuracy, as shown in Figure 2, which shows a comparison of snapshots from both systems at different timesteps. Many other aspects, such as particle diffusion, velocity profiles, and random packing statistics also agree well [6].

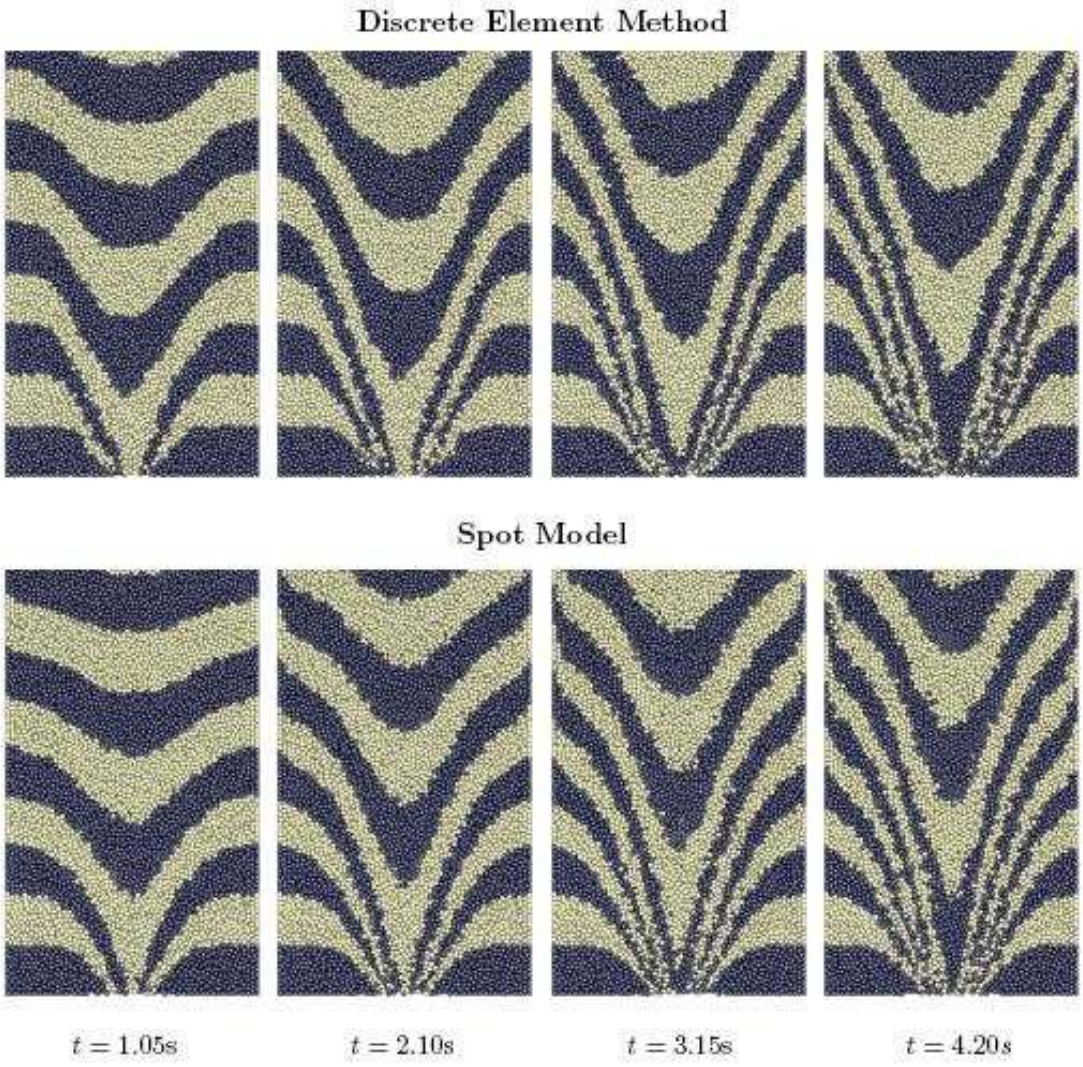

Figure 2: Time evolution of the random packing (from left to right) in DEM (top) and the spot simulation (bottom), starting from the same initial packing. Each image is a vertical slice through the center of the silo near the orifice well below the free surface. Figure reproduced from [7].

However, the computation times are significantly different for DEM and spot model C++ simulations. DEM took 3 to 7 days on 24 processors of AMCL to drain the whole container of 55000 particles, while spot model simulations took about 8 to 12 hours on a single processor of AMCL.

This speedup is of two orders of magnitude.

2.3 Important features of Serial code

The spot algorithm was originally implemented as a serial C++ code. The bulk of the code is built into the `container` class. The container constructor takes the form

```
container::container(float minx, float maxx, float miny, float maxy,
                     float minz, float maxz, int xn, int yn, int zn);
```

which initializes a cuboidal volume for a simulation. The first six parameters set the size of the volume. The volume is divided up into smaller regions, each of which handles the particles within that region, and the number of subdivisions in each direction is given by the remaining three parameters, `nx`, `ny`, and `nz`.

The container holds and manages all particles in the simulation. Simple routines such as `void put(vec &p, int n)` place a single particle with numerical identifier `n` and position `p` into the simulation, assigning it to the correct region. Other routines such as `void dump(char *filename)` dump the particle positions out to a file.

However, the two most important routines for the simulation are the ones responsible for the spot motion and the elastic relaxation. The routine `void spot(vec &p, vec &v, float r)` displaces the particles within a radius `r` of position `p` by an amount `v`. Since the influence of the spot is local, the code only tests the regions of the container which the spot can potentially influence, significantly improving computational efficiency. Any particles which are displaced from one region to another are reassigned.

The routine routine `void relax(vec &p, float r, float s, float force, float damp, int steps)` carries out an elastic relaxation on those particles within radius `s` of position `p`. The code considers all particle pairings, and if two particles overlap, they experience a normal replusion equal to `force` multiplied by the overlap. Once all particle pairings have been considered, each particle experiences a displacement equal to the sum of the forces on it. After each step, the force experienced by the particle is multiplied by a factor `damp`, before all particle pairings are considered again; this process is repeated `steps` times, before the particle positions in the container are updated. A shell of rarticles which have radii greater than `r` but less than `s` are kept fixed to avoid long range disruptions.

Using these routines, a code was written to carry out a spot simulation. The code first initializes a container object, then imports particle positions in from the DEM simulation. The simulation is event-driven, with spots introduced at the orifice according to an exponential waiting time distribution, and each spot in the container moving according another exponential waiting time distribution. Every time a spot moves, a `spot()` call is issued, followed by a `relax()` call. Once spots reach the top of the container, they are removed from the simulation. Snapshots of all the particles are saved to a file every 0.0349s, which can then be analyzed later.

Although the spot model running in serial is already significantly faster than DEM simulation, due to the local microscopic mechanism, it lends itself well for parallel computing. There is still room for faster simulations or better handling of larger systems with appropriate parallelization.

3 Review of Previous Work on Parallelizing the Spot Model

Chris Rycroft has studied two ways of parallelizing the spot model as the term project for 18.337 in Spring 2006 [7]. We will review the two algorithms, timing results and discuss the problems

encountered. We follow [7] for the discussion in this section.

3.1 Master/Slave Method

The first attempt was a master/slave method using MPI. During the computation, the entire state of the system (particle positions and spot positions) is held on the master node. The master node sequentially passes out jobs to the slave nodes for computation, and then receives them back. Since the spot motion events occur at random positions within the container, most of the time multiple relaxation events can be computed simultaneously.

However sometimes a spot motion takes place in a position which overlaps with a job already being computed on a slave node. A simple resolution to this problem is to wait until the conflicting job is finished before submitting the new one. But in many cases this may result in a large amount of wasted CPU time.

A queueing system was therefore implemented. If a job overlaps with any others which are already being computed by the slave nodes, then the job is added to the queue depending on the status of it. If the queue is not full, then the job is added to the queue, and the routine exits. If the queue is full, then jobs are pulled back from the nodes until at least one job from the queue can be submitted, and the original job is then be added to the queue. Each time a job is submitted to the slave nodes, the queue is scanned to see if any new jobs are now free; when the routine exits, it is guaranteed that every job which is stored on the queue cannot at that time be submitted.

To test the algorithm, several simulations of the configuration that is same as the serial simulation were carried out on AMCL. The first sixty snapshots of the drainage simulation were calculated, and the times to generate each snapshot (using `MPI_Wtime`) were logged to a file. In order to assess the rate of computation, we applied linear regression during the steady state region from frame thirty to frame sixty, the time when steady state flow was reached. Table 1 shows the rates of computation compared to those from the serial code.

Number of slaves	Time per frame (s)	Speedup	Parallel 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%

Table 1: Times for the computation of a single frame using the master/slave algorithm, computed on the AMCL

The speedup, S, and parallel efficiency, E, are calculated with the following formula:

$$\text{Speedup, } S(N) = \frac{t(N)}{t(\text{serial})} \tag{1}$$

$$\text{Parallel Efficiency, } E(N) = \frac{S(N)}{N}\,, \tag{2}$$

where N = number of processors and $t(N)$ is the time per frame for N processors. Note that number of processors N is greater than the number of slaves by 1 since there is one master node.

For the single step relaxation code, the parallel jobs are all slower than the serial code, suggesting that the amount of time spent communicating the particle information to the slaves is larger than the amount of time for the actual computation. The results show the problem of creating a parallel

algorithm using a master/slave configuration. In this algorithm, too much stress is placed on the master node. Very poor scalability with the number of prcessors is achieved, as the slaves often stand idle waiting for the master node to pass jobs to them.

3.2 Distributed Algorithm

A second parallelization scheme was considered, in which the 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, significantly reducing the amount of communication. Furthermore, many spot motions can be submitted to each slave simultaneously in a long worklist, minimizing the number of messages that need to be sent.

The drawback with this method is that in some cases a spot's region of influence may overlap with areas managed by other slaves. In that case, each slave must transmit particles to the slave carrying out the computation, and then receive back the displaced particles once the computation is carried out. Note however that since this communication happens between slaves, and not between the master and the slave, the workload is much more evenly distributed. The information about when slaves must pass their particles to neighboring slaves for computation can be passed to them by the master node as a type of job in the worklist.

In this algorithm, communication between nodes is expected to be significant bottleneck. The previous code by Rycroft looks at all possible ways to create a grid of slaves using the specified number of processors, and chooses the one which minimizes the shared surface area between nodes. From the results in [7], this always results in a grid of the form $1 \times 1 \times n$ for n slave nodes. The cases of, for example, $2 \times 1 \times 2$, $2 \times 1 \times 4$, and $3 \times 1 \times 3$ nodes were also investigated, but took longer time than the $1 \times 1 \times n$ configurations with the same number n of processors.

I implemented this algorithm using C++/MPI and run it on the SiCortex Machine SC648. Same as the master/slave method, the first sixty snapshots of the drainage simulation were calculated, and the times to generate each snapshot (using `MPI_Wtime`) were logged to a file. In order to assess the rate of computation, linear regression was applied during the steady state region from frame thirty to frame sixty, the time when steady state flow was reached. Table 2 shows the rates of computation compared to those from the serial code.

We see a much better speedup as compared with the master/slave method. But the scalability with number of processors is worse than linear, and the parallel efficiency also decreases with the number of processors. This may be an indication that inter-processor communication eventually becomes the most significant factor in computation. For larger number of processors, bottlenecking of jobs becomes more frequent. We will be guaranteed that one slave will always be running, but other slaves can potentially have multiple dependencies on others, as the area of influence of a spot overlap with regions controlled by several other slaves.

4 A Faster Distributed Algorithm

In this section, we study a revised distributed algorithm which turns out to run significantly faster than the one presented in the last section. This faster algorithm produces less accurate yet still qualitatively correct flowing packings.

Number of slaves	Prcoessor grid	Time per frame (s)	Speedup	Parallel Efficiency
(Serial)	$1 \times 1 \times 1$	1256	1	1
2	$1 \times 1 \times 2$	821	1.529	50.99%
3	$1 \times 1 \times 3$	674	1.864	46.59%
4	$1 \times 1 \times 4$	569	2.207	44.15%
5	$1 \times 1 \times 5$	515	2.439	40.65%
6	$1 \times 1 \times 6$	476	2.639	37.70%
7	$1 \times 1 \times 7$	446	2.816	35.20%
8	$1 \times 1 \times 8$	425	2.955	32.84%
9	$1 \times 1 \times 9$	406	3.094	30.94%
10	$1 \times 1 \times 10$	387	3.245	29.50%

Table 2: Times for the computation of a single frame using the distributed algorithm, computed on the SiCortex machine

4.1 Motivation

The elastic relaxation step of the spot model was tested to see how often the relaxation step needs to be applied to get a reasonably accurate packing, while still allowing a fast computation. We compared the radial distribution $g(r)$ of 3D spot model with full relaxation, and spot model with relaxation applied every 100th and 1000th spot step, i.e. a relax rate of 0.01 and 0.001. In general, we define the relaxation rate $rr = 1/k$, where we run the spot model with relaxation every kth step of spot motion.

The plot is shown in Figure 3. The $g(r)$ do not agree exactly, but their qualitative features are the same. There are in fact unrealistic packing generated if we do not relax every single step (nonzero $g(r)$ for $r < 1d$), but this overlapping contributes to a very small portion of the distribution. The overlap of particles is still very acceptable, even with relaxation every 1000th step, especially compared with the distribution obtained with no relaxation at all. This shows us that the elastic relaxation step can "magically" fix most of the overlapping particles to produce quite accurate packings, even if the relaxation rate is low.

4.2 The Algorithm

As discussed in the previous section, the bottleneck of the Distributd Algorithm is the overlapping spot motion. When the influence of a spot is over area owned by more than one slaves, one of the affected slaves needs to transfer its particles to another slave, then wait for the computation and receives back particles that are in the region it controls. This waiting and dependency on other slaves slows down the computation.

Motivated by the results about the frequency of elastic relaxation step, I modified the Distributed Algorithm for the case of overlapping spot motion. For overlapping spot motion, both slaves responsible for the region of the spot influence carry out spot computation (spot motion and relaxation) independently, and exchange particles that are out of range if necessary. This is not completely accurate because particles are only guaranteed not to overlap with the other particles in the area controlled by one slave. However, this significantly reduces waiting time since all the slaves that need to communciate with each other will carry out and finish computation at the same time. The size of messages being exchanged between slaves will also be much smaller, because with spot displacement small enough, only a thin layer of particles near the boundary will possibly need

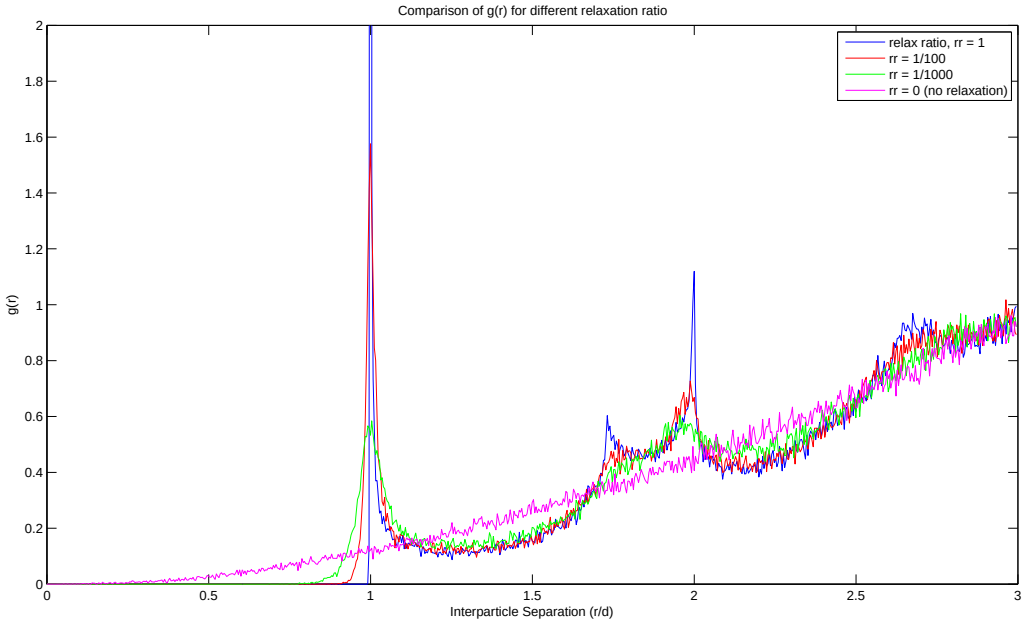

Figure 3: Radial Distribution function $g(r)$ for different relaxation ratio. Relaxation ratio is defined as $1/k$, where elastic relaxation is applied every k-th spot movement.

to be trasfered to another slave after spot computation.

4.3 Timing Results

To compare results more systematically, I implemented this algorithm using C++/MPI and run it on the same SiCortex Machine SC648. The first sixty snapshots of the drainage simulation were calculated, and the times to generate each snapshot (using `MPI_Wtime`) were logged to a file. Linear regression was applied during the steady state region from frame thirty to frame sixty, the time when steady state flow was reached. Table 3 shows the rates of computation compared to those from the serial code.

We observed significant speedups compare to the original distributed algorithm, and a nearly linear scalability with the number of processors. The speedup gets faster as the number of processors increases, because overlapping spot motion occurs more frequently as the container is divided into more regions. Number of particles involved in computations for an overlapping spot motion by each slave is smaller than a regular spot motion, since the particles inside the spot area are split between two processors. As the number of processors increases, the reduction in individual computing time is greater than the increase in communication cost, and consequently the speedup grows faster.

However, this feature also highlights the drawback of the modified distributed algorithm. We know that errors occur near the boundaries of regions owned by different slaves. Thus errors will be larger with increasing number of processors since the container is divided into more regions. This is shown in the snapshots of the two algorithms at the 60th snapshot for 4 slaves in Figure 4.

Also, even though we are doing relaxation every step, the errors in packing is still very visible. A possible reason is that errors is guaranteed to occur at fixed locations (the boundaries of different

Number of slaves	Prcoessor grid	Time per frame (s)	Speedup	Parallel Efficiency
(Serial)	$1 \times 1 \times 1$	1256	1	1
2	$1 \times 1 \times 2$	687	1.827	60.91%
3	$1 \times 1 \times 3$	458	2.745	68.63%
4	$1 \times 1 \times 4$	334	3.757	75.13%
5	$1 \times 1 \times 5$	254	4.950	82.50%
6	$1 \times 1 \times 6$	207	6.054	86.48%
7	$1 \times 1 \times 7$	176	7.134	89.18%
8	$1 \times 1 \times 8$	151	8.319	92.44%
9	$1 \times 1 \times 9$	132	9.502	95.02%
10	$1 \times 1 \times 10$	116	10.86	98.75%

Table 3: Times for the computation of a single frame using the faster distributed algorithm, computed on the SiCortex machine

regions), while in a serial computation in Section 4.1, the randomness of picking the k-th spot step to carry out relaxation fixes the packing randomly throughout the container.

5 Conclusion

We have studied several algorithms for parallelizing the spot model. The maste/slave method did not give satisfactory results, because too much stress is placed on the master node. The distributed algorithm gives speedups over the serial version, but the time of the simulations is still far from the optimal behavior for parallel algorithms. The modified distributed algorithm gives significant speedups and good scalability with the number of processors, but there is a tradeoff between the gain in speed and loss in accuracy of flowing packings.

There are still many possible parallel algorithm to consider for future work. One possible direction is to implement a fully distributed algorithm, where all the nodes are equal, and there is no master node to control things. Then instead of the event-driven nature of the spot motion, we can have all of them randomly walking forward each fixed timestep. As it has never been shown that the event-driven nature is crucial for the simulation to work, we can test this fixed timestep scheme in serial first. If it does not change significantly the simulation results compared to the event-driven version, we can then take into account spots and particles moving across boundaries, and study how this type of parallel algorithm will work out.

Another interesting direction is to parallelize the spot algorithm using the Lagrangian view. For the algorithms studied in this paper, we consider movement of spots with the container fixed and parallelize the algorithm dividing the containers into fixed regions (Eulerian). It would be interesting to see how we can work out a parallel or even simply serial algorithm that follows different spots (Lagrangian).

References

[1] M. Z. Bazant, The Spot Model for random-packing dynamics, Mechanics of Materials 38, 717-731 (2006).

[2] J. Choi, A. Kudrolli, R. R. Rosales, and M. Z. Bazant, Diffusion and mixing in gravity driven dense granular flows, *Phys. Rev. Lett.*, 92:174301, 2004.

[3] H. M. Jaeger and S. R. Nagel, Physics of the granular state, *Science*, 255:1523-1531, 1992.

[4] H. M. Jaeger, S. R. Nagel, and R. P. Behringer, Granular solids, liquids, and gases, *Rev. Mod. Phys.*, 68:1259-1273, 1996.

[5] J. W. Landry, G. S. Grest, L. E. Silbert, and S. J. Plimpton, Confined granular packings: structure, stress, and forces, *Phys. Rev. E*, 67:041303, 2003.

[6] C. H. Rycroft, M. Z. Bazant, J. Landry, and G. S. Grest, Physical Review E, 73, 051306 (2006)

[7] C. H. Rycroft, Parallelizing the Spot Model, Term project for 18.337 Parallel Computing, May 2006.

[8] L. E. Silbert, D. Ertas, G. S. Grest, T. C. Halsey, D. Levine, and S. J. Plimpton, Granular flow down an incline plane: Bagnold scaling and rheology, *Phys. Rev. E*, 64:051302, 2001.

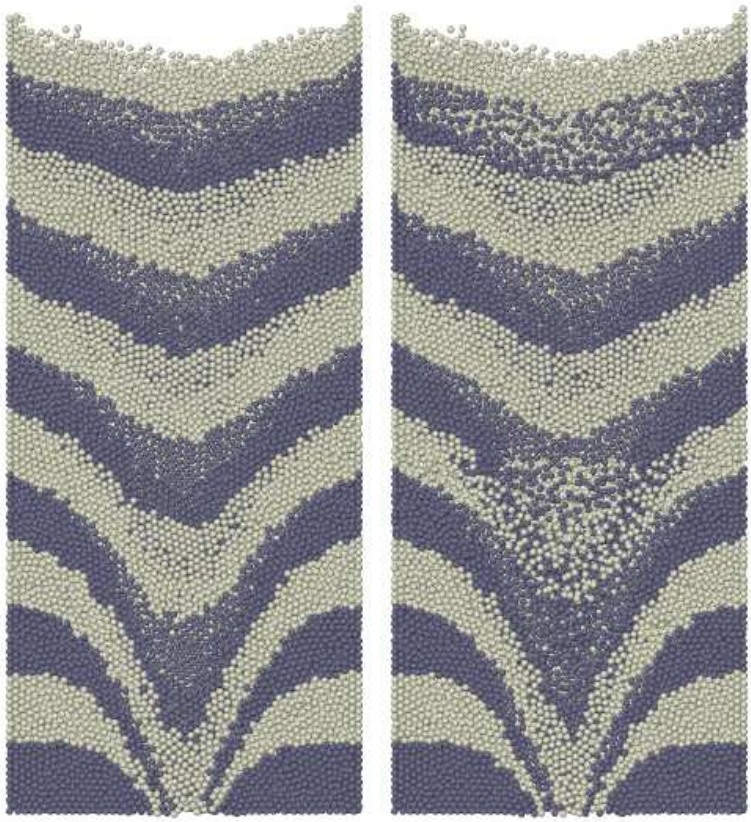

Figure 4: Snapshots of the original distributed algorithm (left) and modified algorithm (right), for number of slaves = 4.