Scaling in nature and in the machine

Direct numerical simulation of sediment transport in fluids

These are curriculum materials for the Blue Waters Project: they are intended to be used by an instructor, familiar with the content, to teach computational science.

J. Russell Manson
The Richard Stockton College of New Jersey

Aims of this lesson

- To understand some fundamental physical processes governing sand movement in rivers
- To implement a model for simulating these processes using C
- To learn how to visualize these simulations using Paraview
- To see how massive simulations can be achieved using supercomputers

Prerequisites

- A multi-core computer running Linux
- The GNU C compiler installed
- Paraview installed
- MPI installed

Lesson 1: Background

Sediment deficits in the Duero River in Portugal, resulted in sediment hungry waters. Excess hydraulic energy then caused massive channel incision resulting in the collapse of a bridge, killing more than seventy people in March 2001.

Sedimentation in reservoirs causes reductions in water storage estimated at US $9 billion in replacement costs per year.

Several hundred million cubic yards of sediment are must be removed and treated annually as dredged material in maintaining and developing European waters for shipping.

Sediment is one of the main tools in coastal zone management. Massive amounts of sediment are being used for flood protection (including beach nourishment), and habitat and wetland protection

Many nutrients, contaminants and pollutants (such as phosphorus, heavy metals and radionuclides) adsorb to sediment particles and are thus transported and deposited with sediment with attendant risks to society and the environment

Understanding how and why sediment is transported by moving fluids is important

Fundamental Understanding

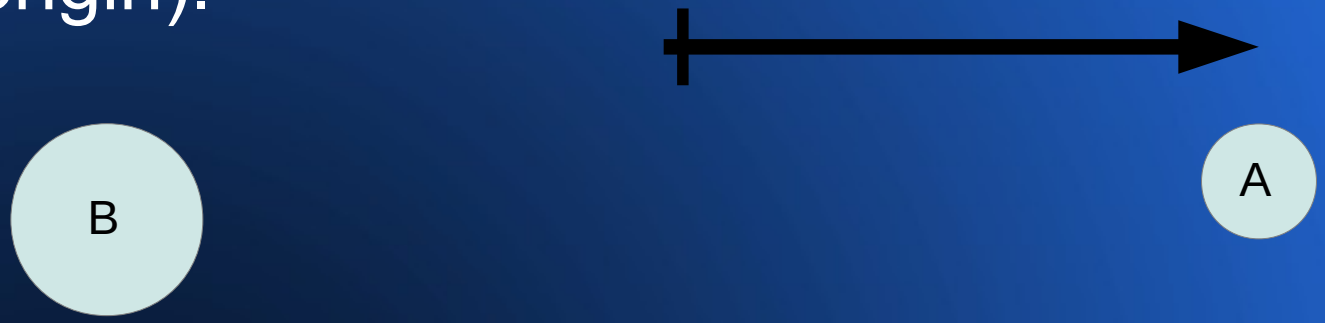
Understanding sediment transport at a fundamental level requires theories that describe:

- the location, velocity and acceleration of every sediment particle in the fluid.
- how every sediment particle in a fluid moves and interacts with other particles

We will develop these in one dimension in the coming lessons but note that when we do simulations we will use fully three dimensional models.

Lesson 2: Position, velocity and acceleration

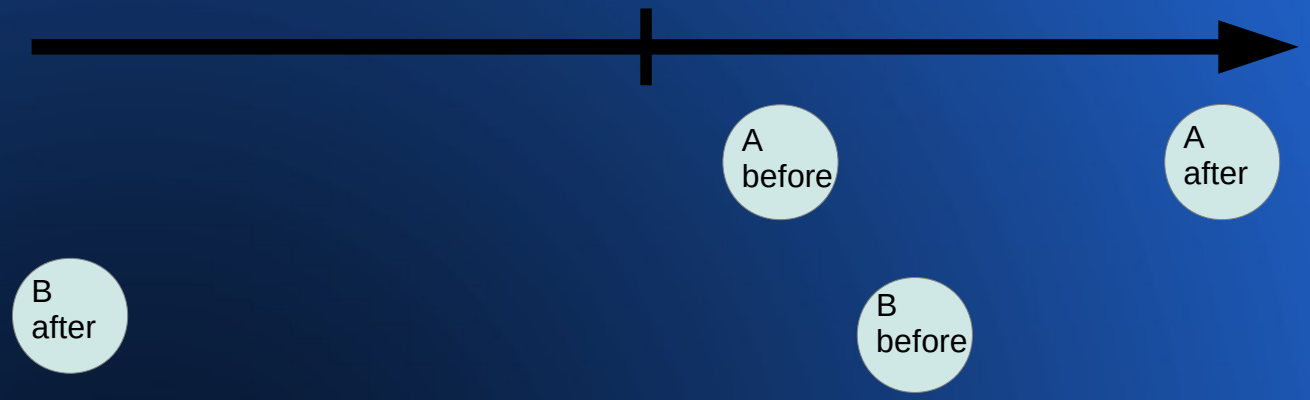
- Position (x) describes where a sediment particle is relative to some fixed point (the origin).



- Particle A has position x = +10 m. Particle B has position x = -9 m.

Lesson 2: Position, velocity and acceleration

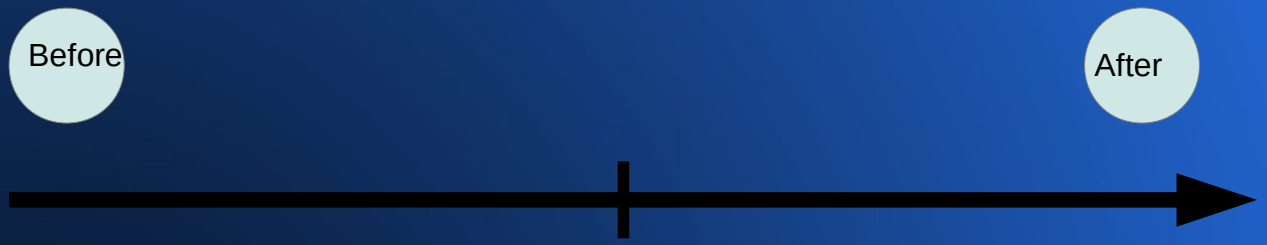
- Displacement (Δx) describes the change in a particle's position over some time interval.



- Particle A has experienced a displacement of $\Delta x = +9$ m. Particle B has experienced a displacement of $x = -14$ m.

Lesson 2: Position, velocity and acceleration

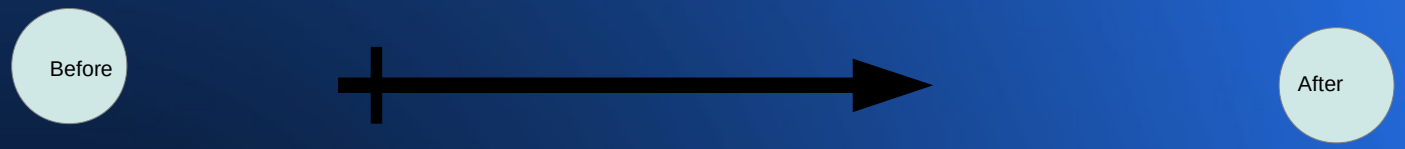
- Average velocity (v) is displacement divided by time interval over which the displacement takes place.



Particle A moves from x=-5m. to x=+5m. in a time 10s. Its displacement is $\Delta x = +10m$. Its average velocity is +1 m/s.

Lesson 2: Position, velocity and acceleration

Acceleration occurs when a particle changes velocity over some time interval. The average acceleration is given by the difference in velocity divided by the time over which the velocity change takes place.



The particle shown is traveling at v=+5m/s and its velocity changes to v=+1m/s in 1 s. Its acceleration is therefore $a=-4m/s^2$ It is important to note that while the velocity is positive the acceleration can be negative. Also the velocity can be zero when the acceleration is not, as is the case when a ball thrown into the air is at its apogee.

Lesson 3: Kinematic relations

If we consider acceleration to remain **constant** over some **small** time increment (Δt) then we can derive equations to predict the new position and velocity from the old:

$$v_{new} = v_{old} + a\,\Delta t$$

$$x_{new} = x_{old} + v_{new}\,\Delta t + 0.5\,a\,\Delta t^2$$

This is basically a Newton-Stormer-Verlet (NSV) method for solving the differential equations of Newtonian mechanics. The term in red is optional.

Lesson 4: Newton's Second Law

Newton's Second Law states that the acceleration (a) of a particle is proportional to the external unbalanced force (F) acting on it and inversely proportional to its mass (m) or,

$$a = \frac{F}{m}$$

If we can quantify the forces acting on a particle we can therefore use this equation to work out the particle's acceleration. We can then use this acceleration in the kinematic equations to predict changes in velocity and position over some small time increment, Δt.

Lesson 5: Forces on a sediment particle

There are several important forces acting on a sediment particle in the sand size range.

1 - - - - - gravity (weight)

2 - - - - - buoyancy

3 - - - - - drag force

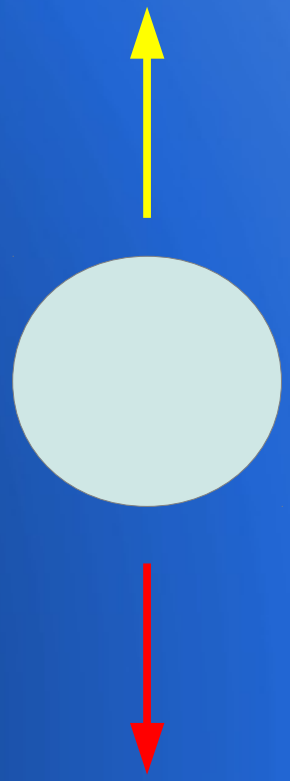
4 - - - - - lift force due to velocity shear

Other forces may be important but these are not considered in this lesson.

Lesson 5.1: The body forces

- Gravity (acting downwards)
- $$\rho_s \; g \frac{\pi}{6} d^3$$
- Buoyancy (acting upwards)

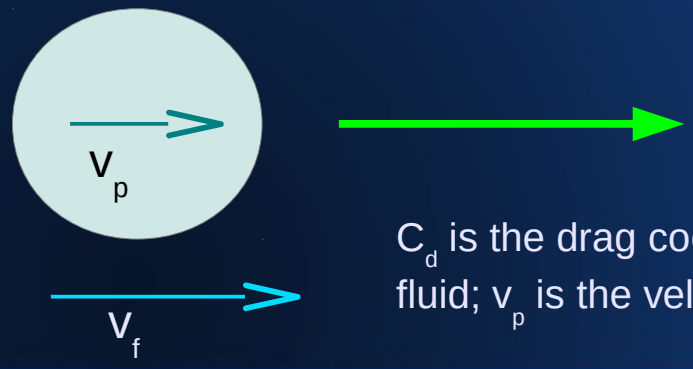
$$\rho_f \; g \frac{\pi}{6} d^3$$



d is the diameter of the sphere; ρ_s is the density of the particle; ρ_f is the density of the fluid; g is gravitational acceleration

Lesson 5.2: Velocity dependent forces

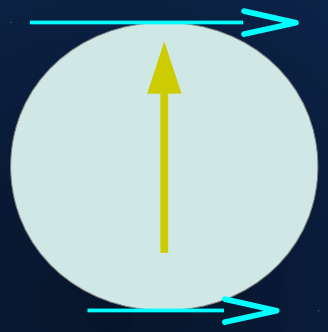
- When fluid flows around a particle it creates pressure and shear stresses differentially around the particle.
- The net result is a drag force acting on the particle.



$$\frac{1}{2} C_d \rho_f \frac{\pi}{4} d^2 (v_p - v_f)^2$$

C_d is the drag coefficient of the spherical particle; ρ_f is the density of the fluid; v_p is the velocity of the particle; v_f is the velocity of the fluid

- In a shear flow such as in a river the velocity varies from zero at the river bed to a maximum at the river surface.
- This velocity gradient induces a pressure differential which causes a net upward lift force.



$$\frac{1}{2} C_L \rho_f \frac{\pi}{4} d^2 [\Delta v_t^2 - \Delta v_b^2]$$

C_L is the lift coefficient of the spherical particle; ρ_f is the density of the fluid; Δv_t is the velocity of the particle relative to the fluid at its top; Δv_b is the velocity of the fluid relative to the fluid at the particle bottom

Lesson 6: Other forces

- There are other forces that may be present but we will leave those for a future lesson. They include: the Magnus force (if the particle is spinning); the “added mass” effect; Basset's forces.
- Copies of useful references are attached.

Exercise 1: Running the code

- You have been given a C code to compute the trajectory of a single particle.
- Open the code with your favorite Linux text editor or print it out.
- Look through the code. Try to understand how it works.
- Compile the code: gcc single_particle.c -lm -o single_particle
- Execute the code. ./single_particle

Exercise 2: Test the code

Look through the code. Find the places where the particle and fluid properties and initial conditions are defined.

Design and run several simulations to demonstrate the effect of changing parameters such as particle diameter, initial position and velocity, particle density.

Prepare a short report (described on the next slide) for the single particle model; it will be assessed according to the given rubric.

The short report: guidelines

Address the following points in your report:

A) Discuss your choice of values for simulation parameters

B) Show how the sediment particle behaves for differing parameter values

You are expected to show results graphically.

Grading rubric for short report

Category	Evaluation criteria
A. Choice of values	An excellent answer (5 points) will choose a range of parameter values for each parameter that are physically sensible. A poor answer (1 point) will choose one parameter value for each parameter or ranges that are physically nonsense.
B. Show model behavior for varying parameter values	An excellent answer (5 points) will show graphically how the particle trajectory differs as each parameter is systematically adjusted while holding all others constant. All graphs will be correctly labeled, titled and captioned with corresponding units. A poor answer (1 point) will show one or two simulations with no systematic investigation of differing model behavior and/or graphs will lack enough information to convey the message.

Lesson 7: Scientific visualization

- We will use Paraview to visualize our results.
- The trajectory code produces as output a file describing the position of the particle for each timestep.
- Start Paraview; click on file-open in the file menu; now select the line that begins:

  + trajectory..

- The plus indicates that there are multiple same files in a time sequence. All plotting operations will pertain to the entire sequence thus giving a movie.

Scientific visualization

- You will now have an object in your pipeline called trajectory*
- Click Apply to load the data
- To the right you will probably see a spreadsheet of your data; click in the little X in the top right of that window.
- You can now click to choose 3D View
- Return to the pipeline browser and highlight trajectory*
- Now go to filters - - alphabetical and select Table to Points

Scientific visualization

- Now for x select x in the pull down menu to the right
- Now for y select y in the pull down menu to the right
- For z select z in the pull down menu to the right
- Hit Apply and you will see some small dots appear in the 3D View
- Return to the pipeline browser and highlight Table to Points
- Now go to filters - - alphabetical and select Glyph
- Choose sphere for the type and hit Apply

Scientific visualization

- You may want to play with the radius size and other plotting properties. Remember to hit Apply each time you make a change.
- When you are happy with how the particles look then hit the green play button in the top middle of the main window. You will see the particles move.
- The instructor will show you other tricks for Paraview.

Lesson 8: When particles collide

When particles collide we apply an event version of Newton's Laws by solving the two equations below simultaneously. The coefficient of restitution, e, is a model parameter describing the amount of energy lost in the collision. e = 1 represents a perfectly elastic collision.

- Conservation of momentum

$$m_A v_A^{\text{before}} + m_B v_B^{\text{before}} = m_A v_A^{\text{after}} + m_B v_B^{\text{after}}$$

- Restitution

$$e = \frac{v_B^{\text{after}} - v_A^{\text{after}}}{v_B^{\text{before}} - v_A^{\text{before}}}$$

Exercise 4: Colliding particles

- You have been given another code which computes multiple particles (50) with collisions.
- Compile the code:

```
gcc many_particles.c -lm -o many_particles
```

- Execute the code. It generates a file describing the position and velocity of 50 particles for each time step.
- You can visualize this scenario using the same approach in Paraview as before.

A note about vectors

- Until now we have introduced concepts using only one dimension. These concepts transfer readily to multiple dimensions using vectors.
- Most good mechanics texts will give the necessary information.

Lesson 9: The complete model

The complete model then consists of:

1. Initialize all particle positions and velocities
2. Advance all particle positions and velocities forward Δt in time
3. Look through all particle pairs to see if collisions occur: if so apply the collision formula.
4. Return to step 2 until simulation time is reached.

Lesson 10: Petascale simulation

- If we want to understand how large scale sediment structures such as river bed dunes “emerge” from these fundamental laws then we will need to simulate millions or billions of particles.
- Such an approach is termed direct numerical simulation and requires computations in amounts in excess of one quadrillion per second or petascale simulation

Achieving petascale

- We achieve the petascale performance by assigning the calculations for smaller sub-groups of particles (movement and collision) to multiple CPUs (or cores).
- So if we had one million particles to simulate then we could assign 1000 particles each to 1000 cores.
- Blue Waters will have 380,000 AMD x86 cores.

Multiple cores

- We achieve this “parallel” computation using a programming library for C called MPI
- MPI includes various commands that can be placed in a C program to allow it to run on multiple cores simultaneously and also to pass data (messages) between the cores as the computation proceeds.
- The basic paradigm is that the code executes on all cores at the same time.
- In any MPI code there is computation in each core and communication between cores.

The parallel code

- The parallel code that you have been given uses the most simple of parallelization techniques.
- In this technique all particle information is sent to every core.
- Then each core computes the movement of a subset of the particles. Then they send back the new information which is collated on a “manager” core.
- Note that in this code we have done something that you would almost never do in a production code and that is write out results every timestep. We have done this to create data files for the Paraview visualization

Lesson 10: The parallel algorithm

The complete model then consists of:

1. Initialize all particle positions and velocities
2. Send all latest particle information to all cores.
3. Within each core advance a portion of particle positions and velocities forward Δt in time
4. Look through all particle pairs to see if collisions occur: if so apply the collision formula.
5. Gather all new information on particles. Return to step 2 until simulation time is reached.

Exercise : The parallel code

The parallel code is called trajectory_mpi.c

Compile it using the mpicc command:

```
mpicc multiple_particles_mpi.c -lm -o mp_mpi
```

Then run using the mpirun command:

```
mpirun -np 4 ./mp_mpi 16
```

The 16 is an argument which tells the code how many particles to simulate. The number after -np (4 in this case) is also an argument that tells the code how many cores to use.

Exercise : The parallel code

Investigate using the parallel code.

Does the parallel code run faster for small problems? i.e. when the number of particles is relatively small (say about 50 per core) .

Remember that all parallel codes involve computation and communication. For a code to be worthwhile to parallelize then it must have much more computation than communication.

Amdahl's Law

I ask you to complete a task for me that consists of eating a cooked pizza.

I guess it would take you about 20 mins to eat it.

If I ask you to get 9 friends to help you eat it, then with ten of you eating I estimate it would take you all about 2 minutes.

Eating a pizza is an example of a problem that is easily parallelized. In this case the speed-up (time for one person to complete the task divided by the time for many people to complete the task) is about 10.

Amdahl's Law

Now If I ask you to complete a new task for me that consists of heating a frozen pizza and eating it then it will probably take you about 40 minutes.

It would take about 20 minutes to heat it and then about 20 minutes to eat it as before.

Employing 9 friends to help eat it does not have the same effect in reducing the task completion time however because the time to heat the pizza cannot be reduced by adding more “workers”. That portion of the task is not parallelizable.

So even employing 9 friends as before the task will still take 22 minutes (20 minutes to heat the pizza and 2 minutes to eat it). The speed-up is just 40/22 or 1.8 !

Amdahl's Law

The pizza example is a real world example of a computing science law known as Amdahl's Law after its original proposer, Gene Amdahl.

Amdahl stated that if you have a computer algorithm that has a fraction of it which is parallelizable (f) and a fraction that is not parallelizable (1-f) then the potential speed-up, S, is given by the following formula:

$$S(p) = \frac{p}{f + (1 - f)p}$$

Note that S is a function of p (the number of cores).

So that if an algorithm is 100% parallelizable (f=1) then 10 cores will give you a speed up of 10. (Just like the pizza problem!)

Exercise : The parallel code

Study the timing of the parallel code.

To do this design a set of computer runs that involve varying the number of cores used and the total number of particles involved.

Specific tasks:

1) Run the parallel code using only 1 core for a fixed number of particles (say 200).
2) Rerun this case using 2 cores, then 4 then 8.
3) Use the Linux time command to time each run.
4) Repeat steps 1-3 for 400, 800, 1200 particles.

Exercise : The parallel code

Compute speedup:

$$SpeedUp = \frac{Time\ taken\ on\ one\ core}{Time\ taken\ on\ many\ cores}$$

Do not expect to get linear speedup (i.e. a speedup of X for X cores).

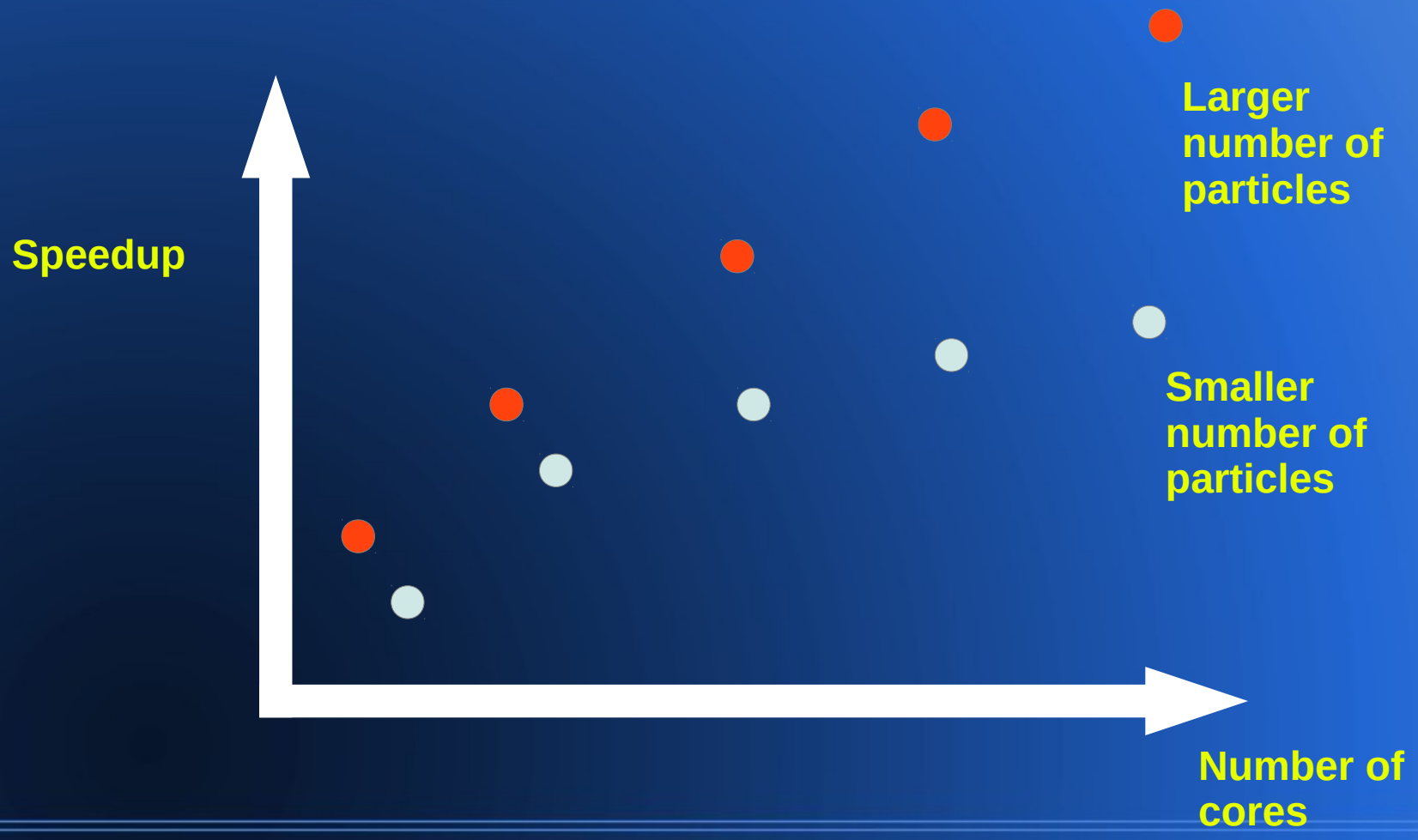
Your graphed results should show however that as you increase the number of particles (and hence the amount of computation) the parallel code becomes much more efficient relative to the serial code.

Scalability

Strong scaling: In strong scaling the problem size remains constant while the number of cores is increased and the time to complete would decrease as the number of cores increase. In general it is difficult to maintain strong scaling as the number of cores increase due to communication overheads.

Weak scaling: In weak scaling the problem size is increased commensurate with the additional cores employed. So if you are using 1000 particles on one core you would use 2000 on two cores and 3000 on three cores (i.e. 1000 particles per core). In the case of weak scaling the time to complete should remain constant as the workload is increased.

For grading: What we expect results should look like



Further exercises for students

1. Examine different size distributions of sediments.
2. Study different density distributions of sediments.
3. Examine different fluid densities and viscosities.
4. Study different velocity profiles.
5. Investigate more complicated force models (Magnus force).

Future extensions

The most obvious and interesting future extension of the codes and lesson is to “soft particles” through concepts of the discrete element method.

A second extension would be to coupling the lesson with a lesson on openFOAM for generating velocity fields.