Parallel Boids Simulation in Python

Imad Abziouad

HH Wills Physics Laboratory, University of Bristol, Bristol BS8 1TL

1 Abstract

A boid or bird-oid programme is a piece of code that attempts to model the beautiful and intricate flocking behaviour observed in the natural world in flocking bids and schooling fish. The flocking characteristics can be broken down into three main parameters; for birds, birds will try to match the velocity of nearby birds, birds will try to keep a separation to avoid collisions and birds will try to gravitate towards the average position of the local flock. Using these three parameters it is possible to model flocking in python, and by using MPI and Cython it is possible to parallelise the programme across many CPU's to simulate more birds more quickly.

2 Flocking In Nature

Flocking is the collective motion of individually motivated animals. It is an emergent behaviors observed as a result of collective animal behaviour. It can be seen in birds, fish, insects and more recently bacteria. [1]

Figure 1. Photograph of flocking in starlings, called a murmuration. [2]

It is thought to occur either during foraging, that is searching for food, or as a defence mechanism to avoid predators. By following the pack, any success in one member of the flock finding food can easily be shared amongst the flock, increasing the likelihood of collective survival. Equally, an alone individual must spend more time watching for predators when not in a flock. If an alone individual is confronted by a predator, it must spend all its time fleeing and watching if the predator is following. An individual in a pack however can spend more time eating and socialising, with many individuals sharing the burden of being lookout. It can be seen in experiments with Laughing Doves in Africa that the greater the number of Doves in a flock, the quicker the reaction of the birds away from the model predatory hawk. [1]

This however breaks down in really large flocks where the prevalence of infighting makes reactions slower, not to mention the alluring prospect to predators of such a large food source decreasing flock survival probabilities. But larger flocks of animals with defence capabilities do have an advantage working together to defeat or even bamboozle predators.

3 Boids

In 1986 Craig Reynolds published a paper entitled 'Flocks, Herds, and Schools: A Distributed Behavioral Model'. His paper described a boid programme, short for bird-oid object, in which he could accurately simulate the flocking of birds using three simple rules. [3] The rules are as follows; Alignment, steer to move towards the average position of local flock-mates. Cohesion, attempt to match velocity with the nearby flock-mates. Separation, steer to avoid colliding with local flock-mates. Flying towards the average position of the flock ensures that the birds always remain flocking and near each other without straying too far. The separation is clearly to ensure the birds don't collide with each other, and the velocity matching also contributes to this, as if the birds share speed and direction they are unlikely to collide. [3]

4 Parallel computing

Parallel computing describes a form of computing where multiple programs are executed simultaneously. This enables a computer to do more within a given time frame. Early on, computers had one processing core within the central processing unit or CPU. This meant that when given a set of instructions a computer could only execute them serially, that is one at a time. With the advent of multiple cores and threads computers can now execute multiple instructions and calculations at once, speeding up programmes and saving time and money. [4]

A set of processes could now be run on multiple cores simultaneously instead of one core. This meant that one programme

taking its time to load wouldn't massively hamper the progress of another programme running. For example in the past opening a web browser while running a computer game would cause the game to stutter and crash. Now the computer can allocate certain applications to certain cores and smoothly run multiple processes at once. [4]

The link to gaming is an important one, as games typically need to run thousands of calculations simultaneously to accurately simulate the environment in which the player will be immersed, with the number of processes increasing dramatically going from 1D to 2D to 3D. The need for so many computational cores led to the invention of graphics cards, which typically now have around 5,000 cores to run vector calculations simultaneously. [5] Only graphics cards can handle creating the complex 3D environments typical for games selling today, which include complex artificial intelligence, light ray tracing and particle simulations. [6]

But this may not be enough for serious computational tasks such as physics simulations or large mathematical operations. For more intensive computational loads, a supercomputer is needed.

5 Supercomputing

A supercomputer is a computer which much more processing power compared to a home computer. [7] Supercomputers achieve such processing power by employing a great number of motherboards. A home computer will have a single motherboard, which typically holds a processor and some RAM modules among other things. Supercomputers commonly house thousands of motherboards in server shelves that fill a warehouse, with each motherboard commonly holding two specialised CPU's and multiple specialised ECC RAM modules.[7]

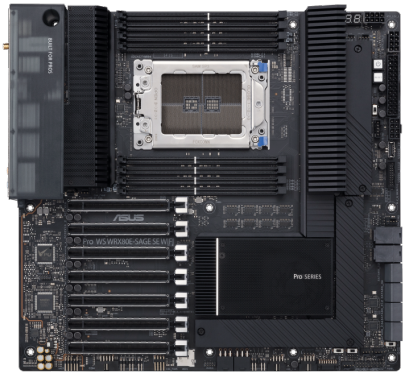

Figure 2. Latest generation of server and supercomputer grade motherboards, capable of hosting eight RAM modules, seven PCIe graphics cards and an AMD Ryzen 64 core processor.[8]

A supercomputer can therefore harness thousands of processors and memory modules all with the task of executing complicated scientific calculations and simulations in parallel.[7]

The supercomputer used in this project is Bluecrystal 3, which has 223 motherboards each with 16 x 2.6 GHz Intel SandyBridge cores, 4GB of RAM per core and a 1TB SATA disk. [9]

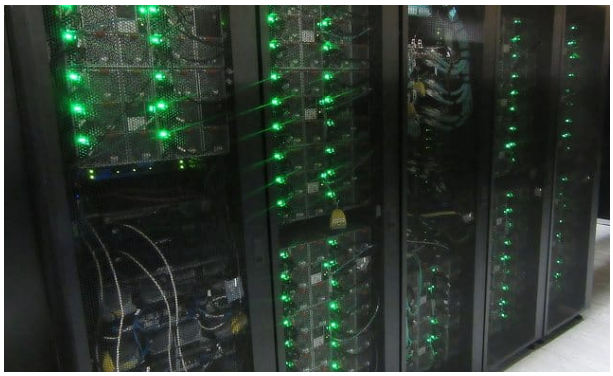

Figure 3. Photograph of University of Bristol Bluecrystal Supercomputer. [10]

6 MPI

MPI (for Message Passing Interface) is a language package first released in June 1994 that enables the execution of a code or programme over multiple computing cores, which do not need any shared memory. [11] This makes MPI incredibly useful in situations requiring high performance and scalability, such as supercomputers. How MPI works is as follows. Within a programme, a master core is designated by the user to monitor and send tasks to a number of user designated worker cores. [11] The worker cores will receive the task given by the master core and execute it, reporting the result back to the master core which will collect and manipulate or store the data. [11] The load balance and core/worker numbers are not automatic and have to be specified by the user upon executing the programme, which could make it less efficient in instances of very large core counts. The version of MPI used in this project is that for python, which is called mpi4py. [11]

7 Cython/OpenMP

Cython is a compiled language released in July 2007 that is written mostly in python with elements of C-like syntax. It aims to be a faster version of python crossed with C. [12] Cython compiles python code into a binary executable wrapper so that it can be run faster like a compiled language, instead of a slower interpreted language like pure python. On top of this, cython allows the user to analyse the code's underlying C script to find areas of sub-optimal speed. [12] These can be as simple as python having to determine the variable type (string, integer, float etc..), which can then be defined by the user (cdef) as being a certain type of variable, allowing for a speedup of that segment of code. [12] Further optimisation of a python

code can be carried out, such as replacing the notoriously slow python square root calculation with a much faster C code for the calculation. [12]

8 Method

8.1 Physics

The physics of the boids was calculated using numpy. To start, a three-dimensional volume was created to be the bounds of where the boids would spawn, a volume of 2000 cubed. Positions were then created by making a numpy array of x, y and z values for each boid within the bounds. Similarly x, y and z velocities were created in an array for each boid to determine how each value of position would change over time. The boids were treated as point-like particles with equality.

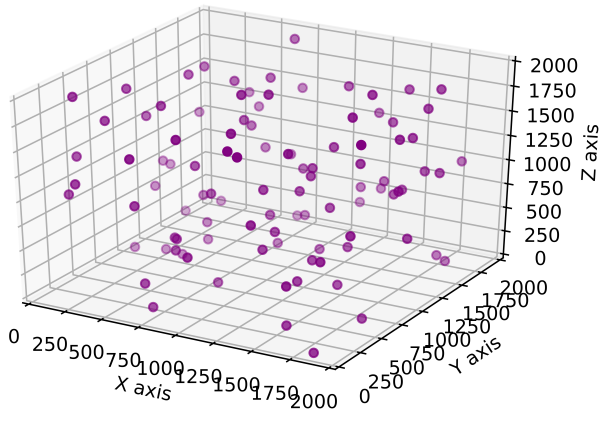

Figure 4. Initial spawn positions of 100 point-like boids within the simulation volume.

A function called 'updateboids' was created to cycle through seconds which were called frames for reasons that will become clear soon. To implement the three fundamental rules of boid flocking, loops were used. The first loop was alignment to ensure all boids gravitated towards the centre of mass position of the flock. This involved finding the average x, y and z value of all the boids to find the centre position, and then subtracting this centre position from the velocities of each boid multiplied by an arbitrary 'strength to centre' constant for simplicity. During every iteration of the loop, the velocities would update to ensure the boids flocked towards the centre of mass.

The second loop is for collision avoidance. This loop would check the separation between each boid's x, y and z coordinate by finding the difference between each value in the positions array. If the distance between the boids (or difference in values) was less than a particular 'awareness' value, the boid would steer in the opposite direction to the closest boid at a velocity called 'avoidspeed'. This would change the velocity array values. The greater the 'awareness', the more likely they are to keep away from each other. The greater the 'avoidspeed', the faster they fly away from each other.

Lastly is velocity matching. Again if a boid is within a certain distance 'awarenessv' of another boid or several boids, the average velocity of the neighbouring boids is calculated and added to the velocity of the group of boids multiplied by a scaling constant called 'coherence'. The greater the value of 'coherence', the more the boids will velocity match every second. The smaller the 'awarenessv' value, the closer they have to be before they start velocity matching.

At the end, the positions array was then updated with the manipulated velocities array. The function 'updateboids' then loops from the start for a duration set by the user.

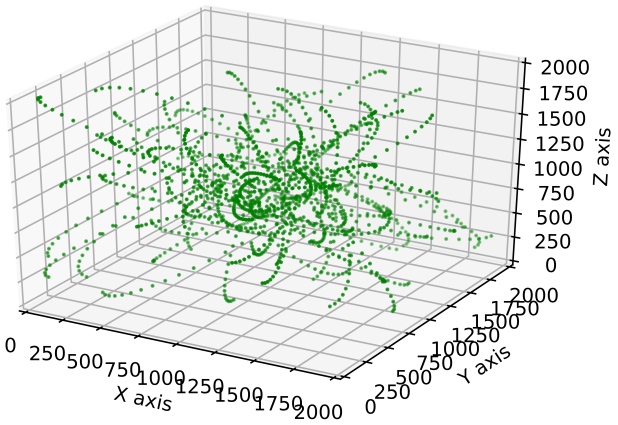

Figure 5. Motion tracks of 100 boids gravitating towards the centre of mass of the flock for 15 frames/seconds.

8.2 Serial Python

In standard python, the number of times the function is looped corresponds to the number of seconds the simulated boids have been flying. The number of seconds to run the simulation is however set by the matplotlib animation call. Within the call, the number of frames can be set by the user which corresponds to the number of seconds the simulation runs for, with 'frames=300' being 15 seconds at 20fps for example. The number of boids to be simulated can also be changed. This will run the code in series and output an mp4 h.264 video file, showing the user a video of the 3D simulation containing the number of boids and duration specified. This was done for convenience and the video making capability can always be replaced by a simple loop over a user-defined number of seconds.

8.3 mpi4py

In parallel the execution of the code is slightly different to the serial case. The user sets the total number of boids to be simulated and the code will divide that number as equally as possible across the number of worker cores that will be assigned for the execution of the programme. The user also sets the number of frames they wish the simulation to run for. The master core then sends the function 'updateboids' to the worker cores.

A worker core will loop over the number of frames and after every second append the position values to lists of x, y and z. Once the worker cores have finished their calculations and appending for a subset of the total number of boids, the resulting lists of position vs time are sent back to the master via 'comm.send()'. The master core will compile these lists of positions vs time for all the worker cores to an array, to create frames of all the boids' positions vs time. When all the workers have finished, a function will then animate these boid positions using matplotlib again, to create a video file of the 3D simulation containing all boids.

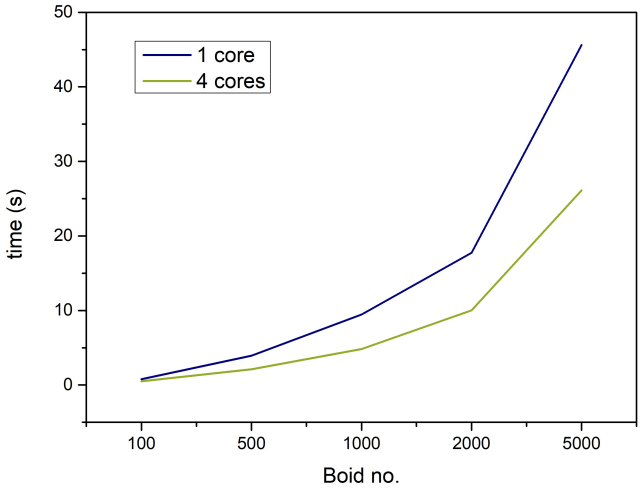

Figure 6. Graph of boid number versus time taken for single and multi-core calculation on a home computer.

9 Results

The results for serial python vs mpi4py for a quad core home computer can be seen in the figure below. For a small number of boids the difference is negligible, as the benefits of parallel computing have either no effect or a negative effect for quick tasks. As expected, for more intensive boid numbers the serial python version of the boids algorithm is slower compared to the 4 core MPI code. At one thousand boids the parallel MPI code reaches double the speed of the serial code, and the speed boost just keeps increasing from there. This is because each of the four cores is sharing the work-load of computing the motion for a subset of the total number of boids. Executing multiple cores to record the motion of a fraction of the boids in separate smaller arrays and then combining them at the end, is much faster than asking one core to keep track of all the boids in a large array and list. This is most evident at 5000 boids as the serial code takes almost 46 seconds whereas the quad-core computation takes but 26 seconds.

10 Conclusion

A boids programme was successfully implemented in python to closely resemble the behaviour of flocking animals. The boids were made to gravitate towards the centre of the group, avoid collisions and try and match the direction and speed of the nearby birds. This simulation can then be animated to give a video of the boids flocking. The programme was then parallelised using mpi4py to speed up the process of simulating the motion of the flocking boids, and successfully implemented on multi-core CPU's. This parallel simulation showed great promise in speeding up the simulation for a larger number of boids over 300 seconds.

Unfortunately given to unforeseen circumstances it was not possible to successfully re-wrap the programme into C binary using Cython within the given time. Also due to some bluecrystal3 issues it was not possible to run the code on a supercomputer within the time frame, but that is still planned to happen, thanks to the help of bluecrystal technician Callum Wright.

Future attempts at this will include the folowing improvements; encoding the programme in C using cython, running both the parallel mpi4py and cython/openMP programmes on bluecrystal3 and also implementing better communication between cores/nodes in mpi4py to ensure the birds all flock to the same centre-of-mass position.

References

[1] "Why birds flock together." https://www.rspb.org.uk/birds-and-wildlife/natures-home-magazine/birds-and-wildlife-articles/features/why-birds-flock-together/.

[2] "Why birds flock together." https://www.howitworksdaily.com/why-do-birds-flock-together/.

[3] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '87, (New York, NY, USA), p. 25–34, Association for Computing Machinery, 1987.

[4] G. S. Almasi and A. Gottlieb, *Highly Parallel Computing*. USA: Benjamin-Cummings Publishing Co., Inc., 1989.

[5] "Geforce rtx 3070." https://www.nvidia.com/en-gb/geforce/graphics-cards/30-series/rtx-3070/.

[6] “Nvidia in the driver’s seat for deep learning.” https://insidehpc.com/2016/05/nvidia-driving-the-development-of-deep-learning/.

[7] “Inside the titan supercomputer.” https://www.anandtech.com/show/6421/inside-the-titan-supercomputer-299k-amd-x86-cores-and-186k-nvidia-gpu-cores.

[8] “Pro ws wrx80e-sage se wifi mother-board.” https://www.asus.com/us/Motherboards-Components/Motherboards/All-series/Pro-WS-WRX80E-SAGE-SE-WIFI/techspec/.

[9] “Bluecrystal phase 3.” https://www.acrc.bris.ac.uk/acrc/phase3.htm.

[10] “Bristol powers up its latest multi-million pound supercomputer.” https://www.techspark.co/blog/2017/05/26/bristol-powers-latest-multi-million-pound-supercomputer/.

[11] “mpi4py documentation.” https://mpi4py.readthedocs.io/en/stable/.

[12] “Cython’s documentation.” https://cython.readthedocs.io/en/latest/.