MNRAS **000**, 000–000 (0000) Preprint 1 March 2019 Compiled using MNRAS LATEX style file v3.0

Parallelising the N-Body direct Gravitational Calculation.

Bertie White

January 2019

1 DIRECT N-BODY GRAVITY

1.1 Background Physics

The N-body Newtonian gravitational simulation involves simulating N different particles and the forces between them, calculating the force on the ith particle is given by

$$\mathbf{F_i} = \sum_{j \neq i}^{N-1} -\frac{GM_iM_j}{|r_{ij}^3|}\mathbf{r_{ij}} = \sum_{j \neq i}^{N-1} \mathbf{F_{ij}}. \quad (1)$$

This cannot be solved analytically for any case with $N > 2$, instead Newtons second law can be used to form a differential equation for $\mathbf{r_i}$ which can be integrated using an approximating numerical integrator (discussed in 1.2). The divergence at $r_{ij} = 0$ causes non-physical results for systems were the particles pass through or very close to each-other, such as the simulations of dark matter haloes in the formation of galaxies. Many systems require the consideration of more than just gravity causing additional computational complexity, to avoid this the solar system was modeled. Since all the orbits are well separated the divergence is not an issue, the Kuiper belt will also be modelled which may see close passes but objects are observed leaving the Kuiper belt.

The orbits of all the planets all execute on a similar plane, this means it is well approximated by a 2-d simulation. However, inclusion of the Kuiper belt, a group of asteroids lying beyond Neptune, requires the consideration of the 3rd dimension as these extend out of that plane significantly. Within the simulations the units are set such that $M_\oplus = AU = d = 1$, with $M_\oplus$ as the mass of the earth, AU as astronomical units, d as days. This is the case throughout this report and simulation, all values quoted and plotted in graphs are in these units. Data supplied by Horizons Ephemeris gives the position of solar system objects which allows for distances from the sun to be calculated. Assuming a circular orbit an initial velocity can then be found simply using:

$$|\mathbf{v_i}| = \sqrt{\frac{GM_\odot}{r_i}}. \quad (2)$$

This velocity vector is required to be perpendicular to position vector, this can be enforced simply by taking the initial conditions to be:

$$\mathbf{r_i} = |\mathbf{r_i}|(sin(\theta), cos(\theta), \epsilon); \mathbf{v_i} = |\mathbf{v_i}|(cos(\theta), -sin(\theta), 0). \quad (3)$$

Where ϵ is defined by $r\epsilon = z$ with z as the maximum distance off the planetary plane. Theta in this case is taken as $\frac{\pi}{2}$ for the OpenCL simulations and are randomly generated for the C code; OpenCL has no in built random number generating tools. This process is repeated for all the planets and N-9 asteroids. The sun is then placed at the origin with a net velocity calculated such that the total momentum of the system is **0**. If this is not done there is a drifting of the system, this has no effect on physics but does make the visualizations generated less clear.

[h]

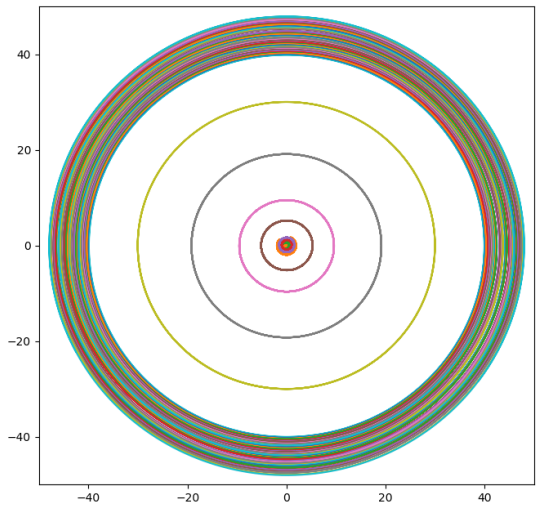

Figure 1. 50 Particles simulated for time steps of 4 hours, for 200000 time steps trajectory plotted in the x-y plane.

When both gravitational potential and kinetic energy are considered total energy should be conserved, this can be used to test for accuracy as any loss or gain of the total energy is nonphysical and comes from computational errors most likely from the integrator, rather from inaccuracies from the force calculations. The total energy can be simply written as:

$$E_{tot} = \sum_{j \neq i}^{N-1} -\frac{GM_iM_j}{r_{ij}} + \frac{1}{2}\sum_{i}^{N-1} m_i{\mathbf{v_i}}^2. \quad (4)$$

1.2 Computational Physics

This force calculation is inherently $\mathcal{O}(N^2)$, approximations exist that reduce the problem to $\mathcal{O}(N \log(N))$ which is significantly more computationally efficient. These methods often rely on more abstract and complex memory systems which are difficult to use on GPUs, this means that in order to make comparisons the direct method has to be used throughout.

To reduce the memory required (which leads to memory issues when dealing with large particle numbers) forces matrices are not

[h]

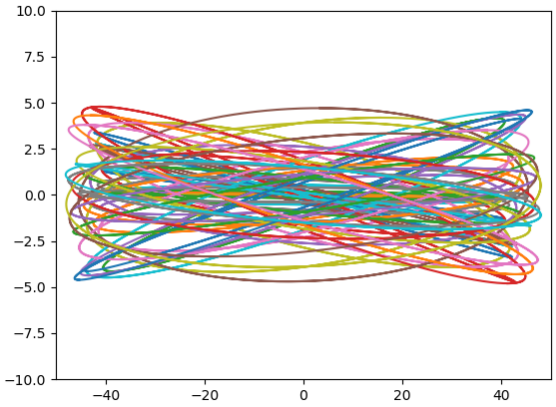

Figure 2. 50 Particles simulated for time steps of 4 hours, for 200000 time steps trajectory plotted in the y-z plane. The main view of this is the orbits of the Kuiper belt.

[h]

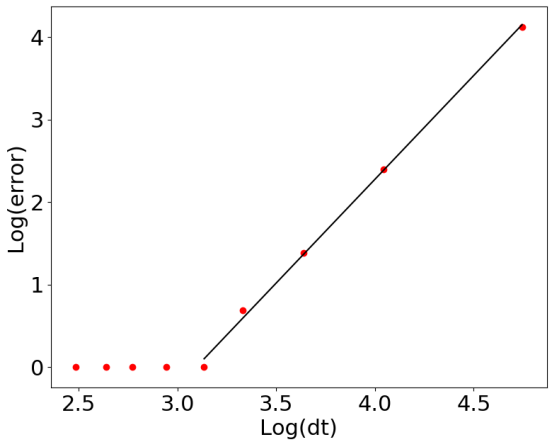

Figure 3. 2000 particles simulated for a fixed total time (165 years 1 orbit of Neptune) with a varying time step and number of time steps.

created but the force vector is immediately calculated. It should be noted that Newtons third law, $\mathbf{F_{ij}} = -\mathbf{F_{ji}}$, allows for the number of forces directly calculated to be halved. There can be difficulties implementing this with parallelisation as the calculation of $\mathbf{F_i}$ is not longer independent of $\mathbf{F_j}; \forall i \neq j$, meaning it is not always used.

The integrator used throughout is the velocity Verlet method. It assumes a constant acceleration through the time step (valid for a sufficiently small dt). The equations applied (2) are:

$$\mathbf{r_i}(\mathbf{t_{i+1}}) = \mathbf{r_i}(\mathbf{t_i}) + dt\frac{d\mathbf{r_i}(t_i)}{dt} + \frac{dt^2}{2}\frac{d^2\mathbf{r_i}(t_i)}{dt^2} \tag{5}$$

$$\frac{d\mathbf{r_i}(t_{i+1})}{dt} = \frac{d\mathbf{r_i}(t_i)}{dt} + \frac{dt}{2}\left(\frac{d^2\mathbf{r_i}(t_i)}{dt^2} + \frac{d^2\mathbf{r_i}(t_{i+1})}{dt^2}\right). \tag{6}$$

[h]

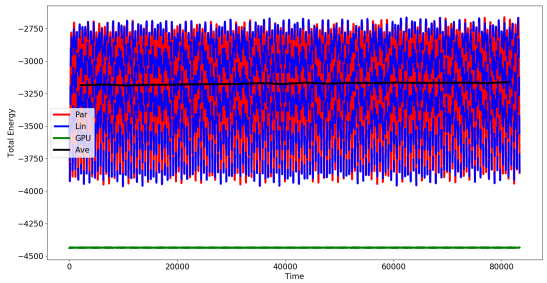

Figure 4. 50 particles simulated for time steps of 1 hour over 50000 time steps and the energy found for each time step and plotted.

[h]

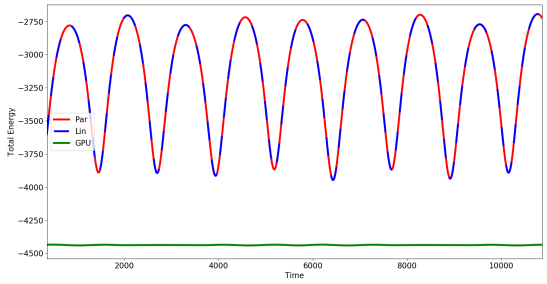

Figure 5. A closer look at one period of the oscillation seen within figure 4.

This is of order $\mathcal{O}(N)$ which when combined with the $\mathcal{O}(N^2)$ force calculation time means the total problem is still $\mathcal{O}(N^2)$. The set up times can also be ignored for the overall complexity as this set up is only performed once and operates with $\mathcal{O}(N)$.

1.3 Basic Results

The methods described successfully reproduce the orbits expected, a 2d projection in the x-y (which is taken as the orbital) plane and y-z plane are of the 3d trajectories can be seen in figure 1, and figure 2.

There is a slow in fall due to the systematic errors from the integrator,over estimating the fall into the well. The simulation used to observe this are off a larger total time simulated than is standard, this effect is normally negligible. The Energy of the system should be preserved, it is a very sensitive measure of the accuracy and it is never explicitly set within the simulations, unlike total momentum set to 0 at the start by the sun.

A plot of all total energies for all 3 methods is shown in figure 4, with a closer examination of 1 of the periods shown in figure 5. The average total energy (averaged over several periods) being almost perfectly conserved with only a small loss over the time scale, the oscillation is non-physical, and comes from a systematic error and the elliptical path. Figure 6 shows the exponential convergence to a minimal error, this hard limit is the inherent floating point errors for the 64 bit floats used in the calculation. This means that if

[h]

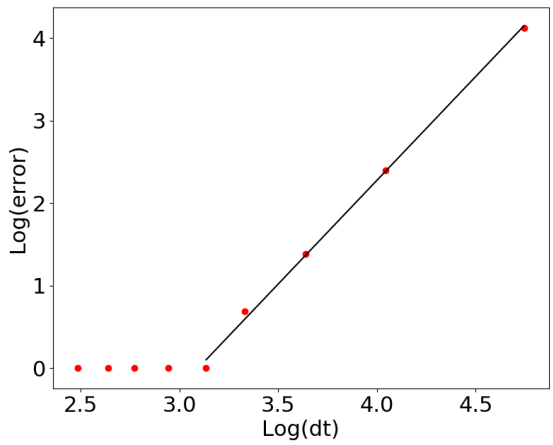

Figure 6. The standard deviation of energy oscillations against dt, the same total time was simulated whilst, number of steps and dt was changed.

a certain level of accuracy is required dt can simply be chosen to ensure that accuracy.

2 PARALLELISATION METHODS CONSIDERATIONS

The parallelisation software used is OpenMP for CPUs and OpenCL for GPUs. These will be compared to a linear implementation written in c which uses Newton's third law to reduce the calculations required. The timings used throughout the rest of this report all include the complete process including initialisation times.

2.1 OpenMP

OpenMP works by multiple CPU's tackling different independent iterations of a for loop. In order to spread the load so each loop has a similar quantity of work to split the work load F_{ij} is calculated if $(i+j)\%2 = 0$ and $i > j$ and F_{ji} is implied from that, equally if $i < j$ and $(i+j)\%2 = 1$. Used through out this is the c implementation of OpenMP. The c implementation allows for multiple for loops to be collapsed into one, for parallelisation allowing for massive speed ups in the force calculation. The inclusion of a parallelised integrator becomes advantageous at around 100 particles (depending on the number of processors), since this is at the bottom of the scales investigated the parallelised integrator has been used throughout. OpenMP uses the shared data model, where all the processors share an address space and transfer memory across processors. The job queue created by OpenMP loops can be handed out to the processors in multiple different ways, either the work load can be split at the start and each core can be given equal sized chunks, static scheduling. Cores can take part of the job from the queue and keep returning back to it, this can be done with variable chunk sizes, called dynamic if the chunk size is constant, guided if it decreases in order to better balance the work load.

All 3 scheduling systems were tested across both the force calculation loops and integrator, the default guided was at least as efficient as any user set schedules, it was chosen to keep the scheduling at the default setting as a low risk way of getting the most efficient schedule.

2.2 OpenCL

For OpenCL there are kernels that are run by multiple threads on a GPU; Blue Crystal 4's Tesla P100s have 2048 threads available, in two blocks of 1024 (1), throughout only 1 block is used. Synchronization is required so that every particle has 1 force calculation and then a time step before the next force calculation, as the forces rely on the positions. This can be done in 2 ways either 1 kernel that does both steps, or 2 kernels that tackle force calculation and a second for taking the time step. The 2 kernel method allows for the use of Newtons third law whilst the 1 kernel method does not, however forces are calculated in local OpenCL memory rather than being stored within shared memory. On a test of 46656 particles across 4096 time steps the 1 kernel method took 255.02s, whilst 2 kernel took 1495.90s. The 1 kernel method clearly outperforms at this size, the time expense in repeatedly reading and writing into arrays for the 2 kernel method grows as $\mathcal{O}(n^2)$, so a second test at a particle number of 20 was performed. The 2 kernel method took 3.81s, whilst the 1 kernel method took 6.45. So across the range of particle numbers studied here the 1 kernel method out performs the 2 kernel method. OpenCL has a tiered memory system the code used here stores the arrays being read and rewritten at a global level, which is similar to the shared memory system used by OpenMP. However, this is the slowest of the memory systems, a speed up is still seen but there is room for an even greater speed up. (3)

3 COMPARING ALL THE METHODS

Despite the differences in the 3 methods used two key values can be varied to explore the efficiency of the systems. That is varying the number of time steps and vary the number of particles. Used throughout this analyses two different OpenMP codes are run for 4 and 14 processors to see how OpenMP runs at both a low number of processors and high number. An analyses of how varying the number of processors is found later.

3.1 Varying Time Steps

Varying the number of time steps gives an insight into how the problem scales but also how the initialisation times vary, since this is a minimal part of simulation for the OpenMP implementations this was not parallelised. The results are shown in figure 8. For both OpenMP implementations and the linear implementation the initialisation times are negligible even at 10 time steps, this is due to the initialisation times being of $\mathcal{O}(N)$, furthermore the operations required are simpler than the $\mathcal{O}(N)$ process of the integration with only $6N$ calls into the arrays compared to the $30N$ calls required for the integration. The 4 core version of OpenMP is offset by 1 on the plot this corresponds to an $\approx$ 2.36x speed up since this corresponds to the coefficient infront of the N^2. Furthermore the 14 core version seems to have a further $\approx$ 2.36x speed up and finally the GPU has a $\approx$ 4.48x speed up over the 14 core OpenMP.

This same initialisation ideas can be applied to OpenCL with two important considerations, firstly the suns initial velocity is dependent on all other velocities, this requires an enforced synchronicity step, which is very inefficient in OpenCL (in OpenMP this is just done in a linear region). Secondly is that all OpenCL programs start by selecting devices and platforms, compiling code and other processes, this is an unavoidable constant time. At low time steps these initialisation times dominate, the speed doesn't

[h]

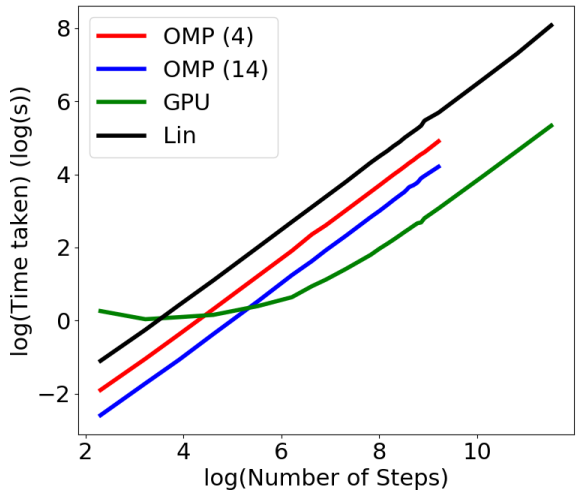

Figure 7. 2000 particles simulated for a varying number of time steps, plotted for the GPU code OpenMP with 4 and 14 cores and Linearly.

[h]

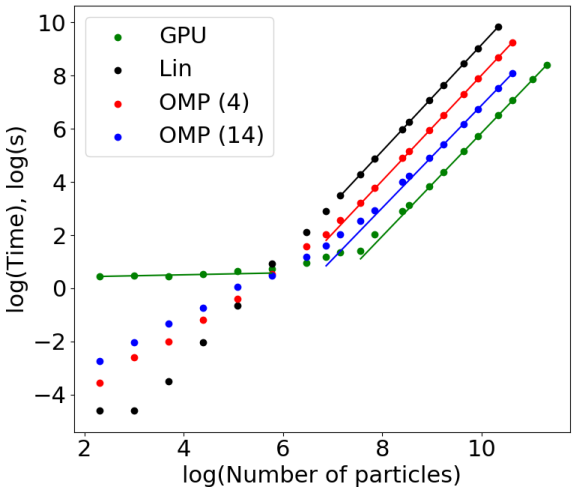

Figure 8. Different particle numbers simulated for 2000 time steps. Different linear regions have been plotted, all 4 methods have had their last 8 points fitted for a straight line aswell as the first 6 points for the GPU.

converge to $\mathcal{O}(N_{TimeSteps})$ until roughly 1000 time steps however the individual steps are much faster, as can be seen by the linear region being lower. The GPU method becomes faster between 40 and 80 time steps, very low compared to a standard number of time steps being used. Since this is dominated by the constant initialisation time as N increases more time is spent on force calculation decreases the number of time steps required before GPU is the quickest method.

3.2 Varying Particle Number

Plotting the log of the time taken versus the log of the particle number should reveal a straight line of gradient 2 for all the implementations of the problem if the problem is indeed $O(N^2)$. The different speeds to calculate 1 time step will be represented in the offset of these slopes. Also of interest is the speed at which they converge to the linear region.

Observed is the linearity expected setting in rapidly for both the linear code and both OpenMP this is much slower for OpenCL due to the previously mentioned constant set up time. OpenCL's behaviour at low particle number is more complex and will be discussed at the end of this subsection. Initially the linear method is the fastest as it requires no set up times to establish the shared memory system of OpenMP, or the initialisation time of the GPU. At the region of high N all methods have a gradient of roughly 2, 1.94 for the GPU, 1.98 and 1.93 for OpenMP with 4 and 14 cores respectively and 1.9998 for linear. The lower gradients of OpenMP and OpenCL show that the asymptote of $\mathcal{O}(N^2)$ is still yet to be reached since the higher N is still reducing the relative time spent on communication as the communication is $\mathcal{O}(N)$; Only once the time taken by all operations, except the $\mathcal{O}(N^2)$ force calculation, become negligible will the gradient of 2 be reached. Communication is more expensive for a 14 cores than for 4, which explains why the 14 core has a less steep gradient. OpenCL has multiple $\mathcal{O}(N)$ processes including communication and setting up command queues, as well as having a much quicker execution time for force calculations meaning that the time taken on processes other than force calculation are larger and so a higher N is required for these to become negligible and thus at the region study the gradient is lower. At this high N limit the GPU is the quickest then the 14 and 4 core OpenMP finally the linear, this is the reverse order than is seen at low N meaning that at some N the parallel methods switch to being quicker than the linear method, this is discussed in their respective sections.

The GPUs on Blue Crystal 4, the Tesla P100s, have a maximum thread number of 1024. Assuming OpenCL uses all of these and knowing each thread deals with 1 particle, it is not until a particle number of 1024 is used that each thread is used multiple times, since each thread needs to execute $\mathcal{O}(N)$ there should be a slight increase in time until $N = 1024(log(1024) = 6.9)$ where the scaling can start to settle in. This is explored in 4.1.1. Finally an estimation of the set up times can be calculated, the flat region at low particle numbers of ~ 0.4 which implies a set up time of $\sim 1.5s$.

4 INDIVIDUAL METHODS

4.1 OpenMP

One of the most critical parts of OpenMP is the choice of when to use it. In section 3.1 it was seen that for large N a greater number of processors (p) reduced computation time. This process (if perfect parallelisation is reached) the relationship between p and execution time is expected to follow Amdahl's law;

$$T_p = T_1\left(F_l(N) + \frac{F_p(N)}{p}\right). \quad (7)$$

T_p is the time for p processors to complete the code $F_l(N)$, $F_p(N)$ are the fractions of the code which are linear and parallel, the dependence these fractions have on N has been made explicitly clear. The linear sections of the program have $\mathcal{O}(N)$ whilst the parallel

[h]

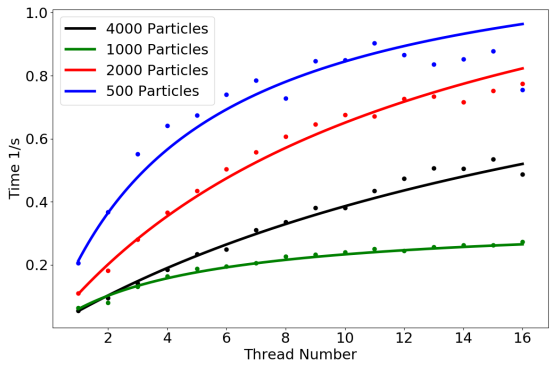

Figure 9. $500, 1000, 2000, 4000$ particles simulated repeatedly with different numbers of processors. Fitted is Amdahl's law shown is the inverted data. Not all the data was simulated other equal times, due to time constraints this could not be corrected so the times are scaled to be at similar points on the axes.

regions have $\mathcal{O}(N^2)$ this means F_l should be

$$\approx \frac{bN}{AN^2 + BN} \tag{8}$$

whilst by definition $F_p = 1 - F_l$. This means that as N is increased the fraction of the code is parallel increases meaning that the the speed varies closer to $\frac{1}{p}$ The dependence of the particle time has been plotted at $N = 500, N = 1000, N = 2000$ and $N = 4000$ with Amdahls law fitted.

By fitting Amdahl's law (taking advantage of $F_p = 1 - F_l$) it was found that $F_l = 0.169, 0.174, 0.073, 0, 043$ for $500, 1000, 2000$ and 4000 particles respectively. The decrease expected is observed especially at the higher particle numbers, however at the lower particle number times there is a deviation away from this. This could be due to the data collection having varying number of time steps meaning initialisation times at the lower particle numbers could be increasing the fraction of the time spent linearly. This trend is continued through to a similar process for data at 400000 particles producing a fraction of 0.0413. This small change could imply a hard limit, although more data at higher N values would have to be taken.

A perfect $\frac{1}{p}$ relation would appear linearly on this plot, observed is a movement away from this setting in at lower p for lower particle numbers, this is due to the reduced force calculation and integration time and so a larger fraction of the time is spent on the initialisation and linear regions of the code.

An important point of interest is where it becomes favourable to use OpenMP over linear code. Assuming that during this low particle number the $\mathcal{O}(N)$, a relation slower than $\mathcal{O}(N^2)$ is required to allow for it to 'catch up', dominates at low particle numbers the two regimes go as

$$T_{Lin} = cN^2; T_{Par} = c'N\left(F_l + \frac{F_p}{p}\right) \tag{9}$$

equating these times, such is the case at $N_{CrossOver}(N_{CO})$ gives the relation $N_{CO} \propto \frac{1}{p}$. However when the data is collected for this the stochastic behaviour at low N hides any relation. This is shown in figure 10.

All N_{CO} appear to be at $\approx$ 250 with any underlying

[h]

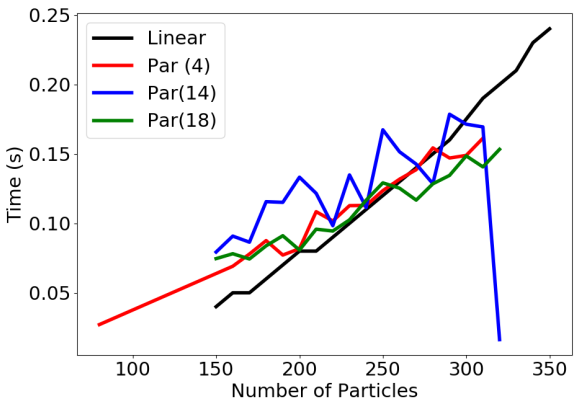

Figure 10. Time taken for 200 time steps at varying particle numbers.

[]

Local Size (2048)	Time Taken	Local Size (16384)	Time Taken
2	29.46	2	382.55
4	14.49	4	202.92
8	9.82	8	114.07
16	9.44	16	54.16
32	9.24	32	35.11
64	9.24	64	35.42
128	9.26	128	34.64
256	9.53	256	44.22
512	11.8	512	44.73
		1024	85.65

$N_{CrossOver} \propto \frac{1}{p}$ requiring either repeats or higher number of time steps to average out the behaviour. It should be noted that this could affect the value of N_{CO} as the initialisation times become more negligible. Finding the value of c in $N_{CrossOver} \propto \frac{1}{p}^c$ will give some insight into the behaviour (and thus the relative significance of the $\mathcal{O}(N)$ and $\mathcal{O}(N^2)$ processes).

4.1.1 OpenCL

Within OpenCL the kernels needing to be executed are gathered within a 'Global works pace' this is broken up into 'Local works pace' each local works paces are executed by 1 of the blocks within the GPU, these blocks contain a set number of threads. The size of both of these are set by the code with the global work space dictating the number of kernels run. It is trivial that the global work space should be dictated by the size of the system simulated whilst the local work space has to perfectly divide the global work space. This leaves limited choice for the local work space sizes. It should be noted up to a 3-dimensional work space can be used however there is no advantage seen for that in this case as all it does is limit the choices of work-sizes which are usable. Using a 1-dimensional work space elevates these issues and means simply the only particle numbers not usable are prime numbers, and even then a work around could be found. The limited local work spaces were investigated for 2 different global work spaces, 2048 and 16384.

These show that there is a large range of local sizes where the executed code is close to optimal. Both systems have a max at roughly Global work space$^{\frac{1}{3}}$ a more in depth look into this would require a much deeper understanding of GPU architecture and a large volume of testing which is beyond the scope for this work.

[h]

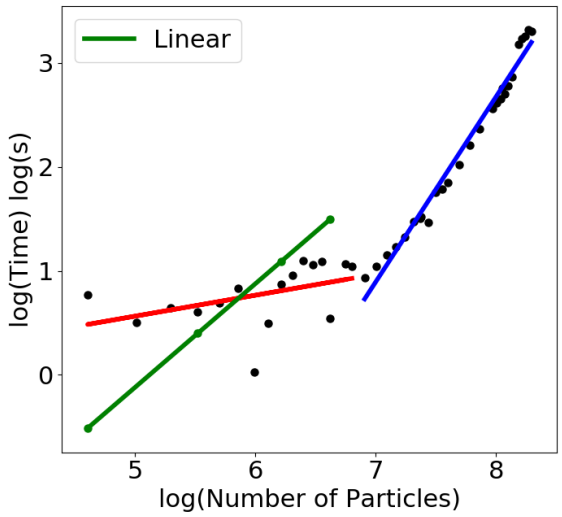

Figure 11. Time taken for 2000 time steps at different particle numbers focusing on the low N region.

Throughout this the local work space was held at 2 to try to remove this as an influence, the GPU has been shown to outperform its counterparts even without this level of optimisation.

As mentioned in section 3.2 this architecture within the GPU means that at low particle numbers the $\mathcal{O}(N^2)$ operation will not be and is not observed. A detailed search of the low N region is presented in figure 11.

The lower N slope has a gradient of 0.21 and the larger N value slope has a gradient of 1.8. The cross over of these regions occurs at $\approx$ 1000. This is very close to the 1024 threads usable by the GPU, this suggests that it is not until threads have the time to run multiple times that does regular behaviour begin to settle in. This implies that there is some set up cost on the initial running of the threads. Especially as the operation done by each threads is $\mathcal{O}(N)$ and a gradient significantly lower than 1 is observed in the region below 1000. Furthermore the region around this disconnect seems to produce a significant variation within the data.

The cross over where OpenCL becomes more optimal occurs at about $e^{5.8} \approx 330$ this will change as N is varied. This time will depend on the number of steps and will occur at lower particle numbers for high step counts, a bulk of this time is in the initialisation, if more time is spent in force calculation the more time the OpenCL code has to catch up. (3)

5 CONCLUSION

Overall it was observed that the accuracy and the results from these methods does not vary significantly allowing for a study of speed to be the sole distinguishing factor amongst the processors. The linear code dominates at low particle numbers, however this is not the scale of the systems usually used in N-body simulations, off interest tend to be the largest possible systems. At these high numbers OpenCL, even when being far from fully optimized, dominates in terms of speed. However the exact behaviour is more complex and developing the work to use the GPUs produces many hindrances. OpenMP tends to be very predictable and reliable whilst also massively speeding up the process. The ease of scaling of OpenMP allows for it's use over huge number of cores if available, whilst this would be difficult to implement with GPU processing. With that all said it is still evident why many advanced N-Body simulations base themselves on GPU processing.

References

Inside Pascal: NVIDIA's Newest Computing Platform (2018). **URL:** *https://devblogs.nvidia.com/inside-pascal/*

Swope, W. C., Andersen, H. C., Berens, P. H. and Wilson, K. R. (1982), 'A computer simulation method for the calculation of equilibrium constants for the formation of physical clusters of molecules: Application to small water clusters', *The Journal of Chemical Physics* **76**(1), 637–649.

Tay, R. (2013), *OpenCL parallel programming development cookbook*, Packt Publishing.