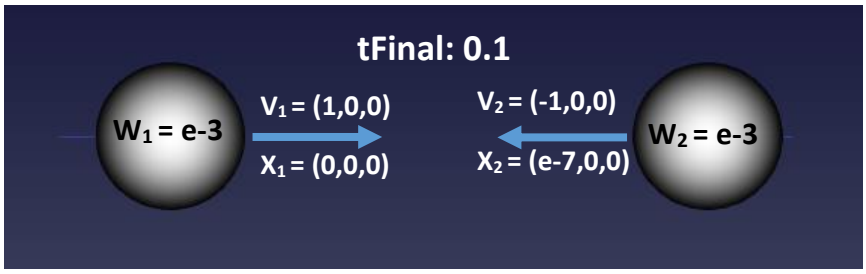
Computing Methodologies III – Space Bodies Summative

Video of work available at: https://youtu.be/FwuJtQq3OWU

Numerical experiments:

Consistency/convergence:

Input data: ./ summativeCode 0.1 0 0 0 1 0 0 0.0001 0.0000001 0 0 -1 0 0 0.0001



The x co-ordinate of the point of collision of these 2 bodies should be 0.000000045.

Adaptive timestep OFF:

TIMESTEP SIZE	X CO-ORDINATE OF COLLISION	ERROR
1e-9	No Collision	N/A
1e-10	0.000000054775998458544998015067	0.000000009775998458545
1e-11	0.000000045105922873334791313611	0.000000000105922873334
1e-12	0.000000045049514022543391783917	0.000000000049514022543
1e-13	0.000000045007906612829063503775	0.000000000007906612829
1e-14	0.000000045000902821281281807021	0.000000000000902821281

Adaptive timestep method:

$Equation\ used: While\ \left(\frac{\left|\frac{txF}{m}\right|}{v} > \varepsilon\right)\{t = \frac{t}{2}\}$ The method reduces the timestep size if any body has a change of velocity greater than a given parameter ε

Adaptive timestep ON (Initial timestep of 1e-3):

Ɛ SIZE	X CO-ORDINATE OF COLLISION
100 (1e+2)	No Collison
10 (1e+1)	0.000000053432581544186985804690
1 (1e+0)	0.000000048833470534421960515172
0.1 (1e-1)	0.000000045355109225711221780013
0.01 (1e-2)	0.000000045008107391863546673621
0.001 (1e-3)	0.000000045004617204994909012963
0.0001 (1e-4)	0.000000045000514341397421183701

Convergence order:

If C is roughly constant for each timestep, p is the convergence order.

$$|F^{(i+1)} - F^{\infty}| \leq C|F^{(i)} - F^{\infty}|^{p}$$

Using $(\log(e_{new}/e_{old}))/(\log(h_{new}/h_{old}))$ for the convergence order, inputting 1e-10 and 1e-14 yields the result
p= (log(0.000000000000902821281/0.00000000975998458545)/log(1e-14/1e-10))
p=1.00863
F(i+1) = position ends up for time step size

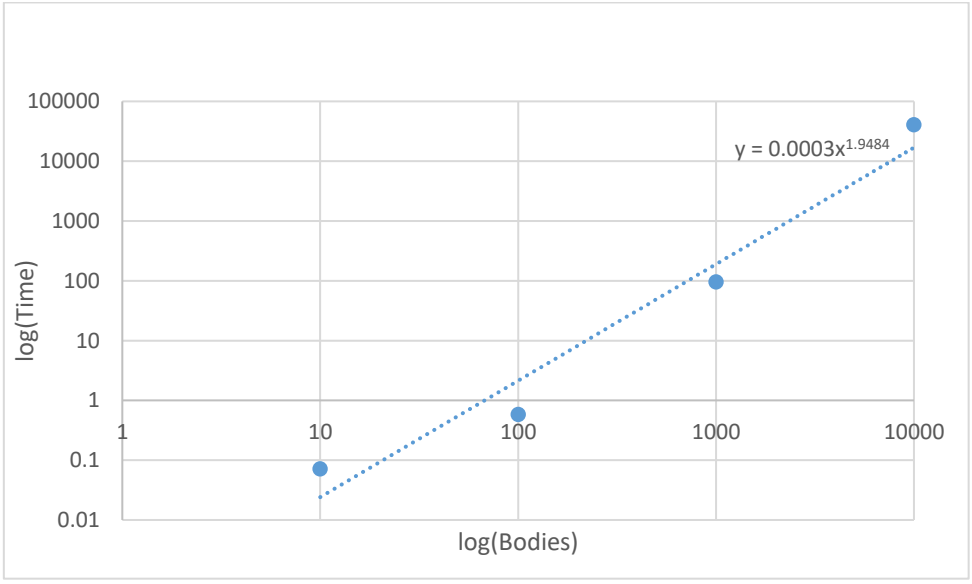
Complexity:

For the following results, each part was run 5 times and averages to produce more accurate results

NUMBER OF BODIES	RUNTIME
10	0.072
100	0.58
1000	95.66
10000	41047.60

Runtime complexity: The most complex part of my code is a nested for loop used to calculate the distances between each body to calculate the force for the next time step. This leads to a runtime complexity of **$O(n^2)$** where n is the number of bodies in the simulation.

As shown by the graph below, the data backs up this claim, with the log of the table above suggesting a x^1.9 corrclation which is roughly equal to x^2.



Statistics:

For my code and the test data I used, the only collisions that occurred did so at the beginning of the simulation at the first increment of the bodies.

NUMBER OF BODIES	COLLISIONS IN FIRST STEP	COLLISIONS AFTER FIRST STEP
10	0	0
100	0	0
1000	1	0
10000	49	0

Scaling experiments:

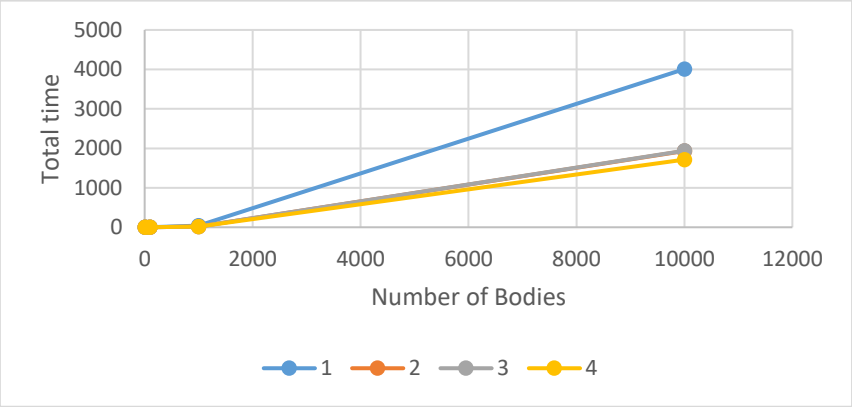
How does the scalability for very brief simulation runs depend on the total particle count?

NUMBER OF CORES	10 BODIES	100 BODIES	1000 BODIES	10000 BODIES
1	0.013	0.36	39.95	4008.45
2	0.022	0.55	19.87	1932
3	0.041	0.69	20.15	1939.7
3	0.031	0.84	19.66	1714.12

The data I collected showed that the code was no more efficient in parallel for a small number of bodies. In fact, for 100 bodies and fewer, the code ran more effectively on one core. This is probably dues to the time spent on allocating tasks and collected results outweighing the time saved performing the calculations in parallel.

A surprising result occurred when comparing 2 and 3 threads, showing next to no difference in performance.

As shown in the graph below, 4 cores yielded the best results for 10000 bodies.



Calibrating Gustafson's law

Using the following formula the data below was collected to show the time spent in serial and parallel over a 100 second simulation:

$$t(1) = f \cdot t(p) + (1 - f)t(p) \cdot p$$

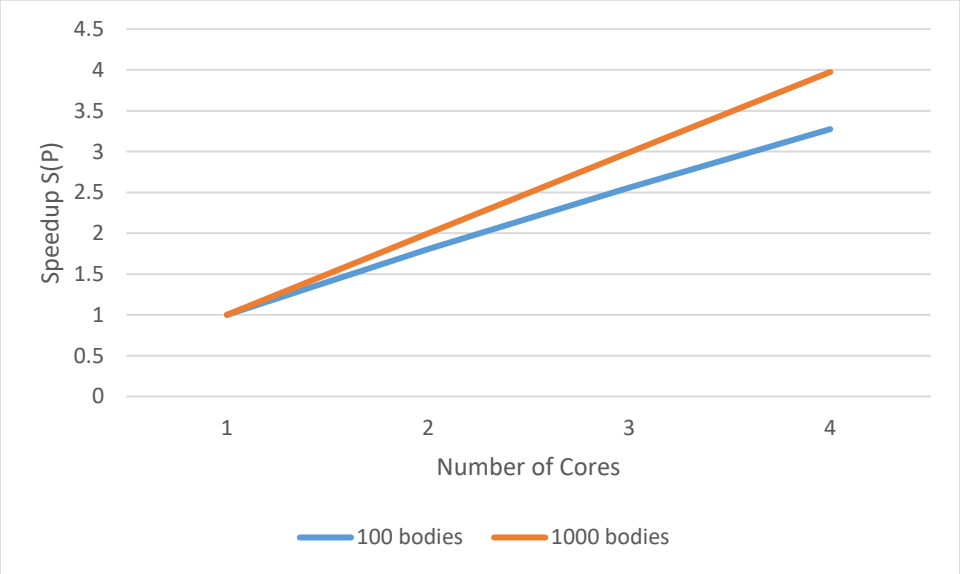
CORES	100 BODIES PARALLEL	1000 BODIES PARALLEL	100 BODIES SERIAL	1000 BODIES SERIAL
1	88.41	99.74	11.59	0.26
2	80.4	99.62	19.6	0.38
3	77.83	99.37	22.17	0.63
4	75.82	99.15	24.18	0.85

Using this speedup equation, the following values were calculated for each different number of cores:

$$S(p) = f + (1 - f)p$$

CORES	100 BODIES	1000 BODIES
1	1	1
2	1.804	1.9962
3	2.5566	2.9874
4	3.2746	3.9745

When put into graph form, the data displays Gustafson's law:



Due to the running time of $O(n^2)$, the total speed up has less of an effect than a larger, say $O(n^4)$ would have, as for larger simulations the parallel sections have less time to improve the efficiency of the program

How does the parallel efficiency change over time studying a long-running simulation?

For a 10000 body simulation, the data below was gathered:

	$tFinal_{short} = 0.1$	$tFinal_{long} = 10$	$tFinal_{long} / tFinal_{short}$
1 Core	4008.45	19807.63	4.94
4 Core	1714.12	3948.88	2.30

As shown in the $tFinal_{long} / tFinal_{short}$ column, the rate at which the time increases for a longer simulation is much greater for 1 core than 4 cores (4.94>2.30). This suggests that a parallel simulation becomes more efficient over a longer simulation compared to a sequential one.