Space Bodies Assignment

cmkv68

February 19, 2018

Numerical Experiments

Consistency & Convergence

The table below describes the information related to the two bodies used for the collision experiment.

Body #	$s(x)$	$s(y)$	$s(z)$	$v(x)$	$v(y)$	$v(z)$	Mass
1	+0.1	+0.1	+0.1	-2.0	-2.0	-2.0	$1e^{-11}$
2	-1.0	-1.0	-1.0	+2.0	+2.0	+2.0	$1e^{-11}$

The bodies collide at $(x, y, z) = (0.155, 0.155, 0.155)$ at time $t = 0.275$. The error is calculated through calculating the difference between the position of one body and the other body upon collision. The following table shows the timestep used to calculate the collision, and the error value left over. We calculate the error of only one dimension; x, as the other dimensions (y, z) would follow the same values.

Timestep	Error	Position of x
0.0001 (Adaptive)	0.000199998	-0.4498
0.000078125000	0.000156250000	-0.449844
0.000039062500	0.000078125000	-0.449922
0.000019531300	0.000039062500	-0.449961
0.000010000000	0.000020000000	-0.44998
0.000003906250	0.000007812500	-0.449992
0.000001953130	0.000003906250	-0.449996
0.000001000000	0.000001999990	-0.449998
0.000000610352	0.000001220720	-0.449999
0.000000305176	0.000000610365	-0.449999
0.000000152588	0.000000305098	-0.45

Our convergence scheme

The adaptive timestep works suitably and utilises less timesteps than $ts = 5e^{-}9$ by a significant margin, whilst having relatively comporable error residues.

Complexity

Under the assumption that the timestep and time limit is fixed, the most dominant function `updateBodies()` which utilises a nested loop that iterates through the number of bodies intiated. For each iteration, a force for a given body is calculated by comparing its position against every other body in space. This results in `updateBodies()` to run in $O(n^2)$.

Procedures have been taken to reduce the constant; Each body only calculates its force against bodies that precede

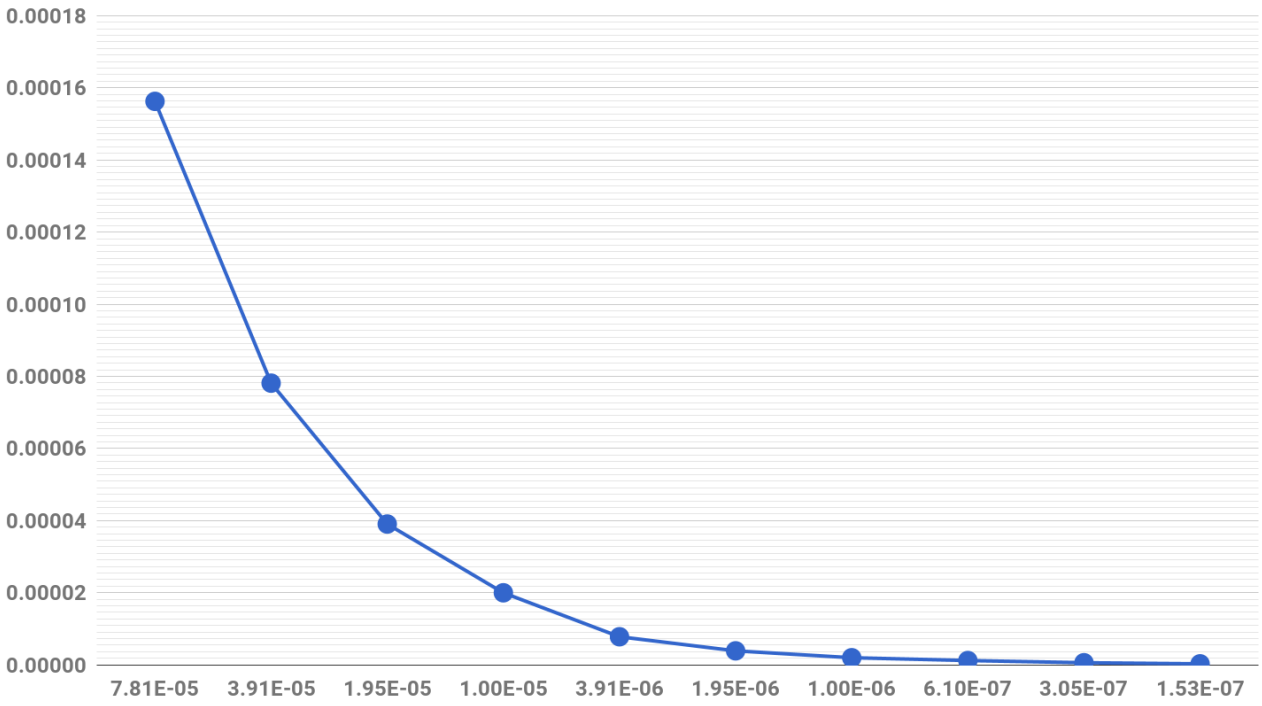

Figure 1: A chart showing the the timestep used against the error produced. The chart shows a clear convergence.

them in the order of initiation i.e. Body 2 calculates force from Body 1 and Body 0 whereas Body 3 calculates from $0, 1$ and 2. A pseudocode describes this method:

```
for (i=0; i<n):
  for (j=0; j<i):
    if (i != j):
      distance = distance between body i and body j

      m = i.mass*j.mass/distance/distance/distance;

      for (k = 0; k < 3; k++):
        i.force[k] += (i.x[k]-j.x[k]) * m;
        j.force[k] += (i.x[k]-j.x[k]) * m;
```

Whilst `updateBodies()` would continue to run in $O(n^2)$, the hidden constant would be drastically reduced to a factor of $\frac{1}{2}$ of the original number of calculations needed.

Statistics

Scaling Experiments

The machine used consists of Durham's MIRA machine, which is a 128 core intel xeon distributed system. At runtime, the program consumes less than a megabyte of memory.

Questions

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

The overhead involved in initiating a large number of particles for a set of parallel processors may take more time than the simulation itself.

2. Calibrate Gustafson's law to your setup and discuss the outcome. Take your considerations on the algorithm complexity into account.

Gusafson's Law: Gustafson estimated the speedup S gained by using N processors (instead of just one) for a task with a serial fractions (which does not benefit from parallelism) as follows: S=N+(1-N)s

3. How does the parallel efficiency change over time if you study a long-running simulation?

There

Distributed Memory Simulation