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 between the position of the error of the new item. 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 others would follow a very similar pattern.

Timestep	Error at x	Collision?
0.001 (Adaptive)	-0.00000000351134	yes
0.000000005	-0.00000000353880	yes
0.0000000025	-0.00000000337228	yes
0.00000000125	-0.00000000481655	yes
0.000000000625	-0.00000000509412	yes
0.0000000003125	-0.00000000540662	yes
0.00000000015625	-0.00000000649210	yes

This is the caption See figure .

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 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?
2. Calibrate Gustafson's law to your setup and discuss the outcome. Take your considerations on the algorithm complexity into account.
3. How does the parallel efficiency change over time if you study a long-running simulation?

Distributed Memory Simulation