M. Troyer, P. Koumoutsakos
ETH Zürich, HIT G 31.8
CH-8093 Zürich

Set 5 - N-body application for gravity

Issued: March 23, 2015
Hand in: April 13, 2015

In this exercise we write a n-body solver for gravity. The gravitational force $\vec{F}_i$ acting on a planet having mass m_i and position $\vec{r}_i$ is defined as

$$\vec{F}_i = \sum_{j=1,i\neq j}^{N} -G\frac{m_i m_j}{|\vec{r}_i - \vec{r}_j|^3}(\vec{r}_i - \vec{r}_j), \tag{1}$$

where G is the gravitational constant.

The n-body code requires N^2 operations to compute the force acting on all particles. Since gravity is a long-range force we cannot use the cell list algorithm that we have seen last semester. Fortunately, the complexity can still be reduced to $O(N \log N)$ and even $O(N)$, but only with approximative methods. In this exercise we implement the Barnes-Hut algorithm, which reduces the complexity to $O(N \log N)$.

Question 1: Barnes-Hut Algorithm

The Barnes-Hut algorithm uses a tree structure to obtain an hierarchy of particles based upon which it is possible to efficiently approximate the gravitational field of a cluster of planets being far apart. In other words, instead of computing all particle-particle interactions, we will implement a particle-cluster interaction. Here we look the gravity problem in two dimensions using a quad-tree.

Quad-tree In a quad-tree the configuration space is divided in four quadrants (north-east, north-west, south-west, south-east). If the quadrant contains more than one particle, it is split again in four regions. This splitting continues until we are left with only one particle per quadrant. An example configuration is depicted in Figure 1. The resulting data structure is well represented as a tree with four children.

Each node of the tree is a cluster of masses for which we store the total mass contained in that subspace and its center-of-mass:

$$m_{\text{node}} = m_1 + m_2 + m_3 + m_4, \tag{2}$$
$$\vec{r}_{\text{node}} = [\vec{r}_1 m_1 + \vec{r}_2 m_2 + \vec{r}_3 m_3 + \vec{r}_4 m_4]/m_{\text{node}}, \tag{3}$$

with m_1, m_2, m_3, m_4 the mass of the children of the node.

To create the quad-tree we will use the following recursive algorithm. To insert a planet p in a node x:

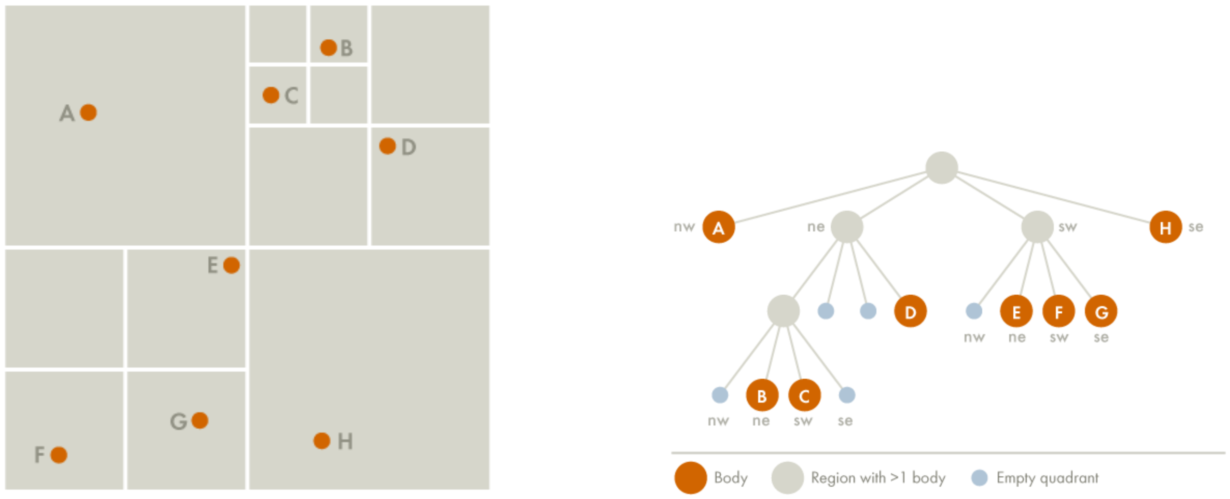

Figure 1: (left) Example configuration space split in quadrants, until there is only one particle per quadrant. (right) The quad-tree data structure corresponding to the example configuration. Credits: arborjs.org.

1. If node x does not contain a planet, insert p here.
2. If node x has children nodes, update the center-of-mass and total mass of x. Recursively insert p in the node representing the appropriate quadrant.
3. If node x is a planet node (let us call q the original planet), then subdivide the region further by creating four new quadrants. Then, recursively insert both q and p into the appropriate subregion. Finally, update the center-of-mass and total mass of x.

Force calculation Once we have the tree structure we can recursively traverse it to compute the forces. For each planet p we will iterate through the whole tree and, at each node, we compute the ratio $s = l/d$ between the distance d from the p to the center-of-mass of the node and the node's length l. The Barnes-Hut algorithm sets a threshold value θ for the ratio s: if the value is smaller than the threshold it is safe to consider the planet p being far apart from the cluster of planets. In the exercise we will use $\theta = 0.5$, in case of stability problems this value should be reduced.

The actual algorithm for a planet p reads:

1. If the current node is an actual planet (and it is not planet p), calculate the force exerted by the current node on p, and add this amount to p's net force.
2. Otherwise, calculate the ratio $s = l/d$. If $s < \theta$, treat this cluster as a single body, and calculate the force it exerts on planet p, and add this amount to p's net force.
3. Otherwise, run the procedure recursively on each of the current node's children.

Tasks

a) Complete the skeleton code by implementing:

   1. The N^2 force calculation to be used as a benchmark.
   2. The function to insert new elements in the tree.

3. The $N \log N$ approximative force calculation.
4. The Verlet time evolution scheme. (see previous semester)

b) Compare the scaling with the number of particles of the improved algorithm against the brute-force approach.

c) Parallelize the code using OpenMP and compute the scaling of your implementation. Discuss possible load balance problems.

   Note: On March 30 you will get part of the solution for the previous subquestions.

References

You might find useful to look at the example shown on this website `http://arborjs.org/docs/barnes-hut`.

Summary

Summarize your answers, results and plots into a short PDF document. Furthermore, elucidate the main structure of the code and report possible code details that are relevant in terms of accuracy or performance. Send the PDF document and source code to your assigned teaching assistant.