Assignment 1

Due: 1 October 2012, 8.00 pm

Number of weeks allocated: 7

Hand in: Submit 4-5 pages report (excluding appendices) (A4, 12 ppt, single spacing) and your code (including the executable file) to LMS.

Weight: 30%

Marking criteria:

Total	30 marks
Completion and correctness of Task 1	10
Completion and correctness of Task 2	10
Completeness and well-formness of the report	10
Completion and correctness of Task 3	20 (extra credit!)

Task

1. Modify the single-thread n-body program to make it multi-threaded using Pthreads. Compute forces, velocities and positions for each body in a separate thread for each body with multiple threads running in parallel. Use *bodiesfield-small* file to initialize the field of bodies. You should take care of a possible occurrence of racing conditions, making sure that no data is read or written by a thread, while the other thread is using this data. Submit this version of program as *nbody-mthread*. Present you code in the Appendix A of the report. Explain your multi-treading solution in the report.

2. Modify the single-thread n-body program to achieve the same result as in Task 1, only now using OpenMP. Use *bodiesfield-small* file to initialize the field of bodies. Submit this version of program as *nbody-openmp*. Present you code in the Appendix B of the report. Explain your OpenMP solution in the report.

3. Modify your OpenMP program using Barnes-Hut algorithm for clustering. The aim is to enable simulations for a larger system of bodies using the same computational resources available. Remember, that clustering itself requires computations too. Use *bodiesfield-large* file to initialize the field of bodies. Submit this version of program as *nbody-bh*. Present you code in the Appendix C of the report. Explain your clustering solution in the report.

Note: Task 3 is optional for extra credit.

Background

Gravitational N-body problem is referred to the task of finding positions and movements of bodies in space subject to gravitational forces from other bodies, using Newtonian laws of physics. Gravitational force between two bodies of masses m_a and m_b is:

$$F = \frac{Gm_a m_b}{r^2}$$

where *G* is the gravitational constant and *r is* the distance between the bodies *a* and *b*.

Subject to forces, a body accelerates according to Newton's 2nd law:

$$F = ma$$

where *m* is mass of the body, *F* is force it experiences, and *a* the resultant acceleration.

Let the time interval be Δ*t*. For a body of mass *m*, the force is:

$$F = \frac{m(v_{t+\Delta t} - v_t)}{\Delta t}$$

New velocity is:

$$v_{t+\Delta t} = v_t + \frac{F \cdot \Delta t}{m}$$

where $v_{t+\Delta t}$ is the velocity at time $t + \Delta t$ and v_t is the velocity at time *t*.

Over time interval Δ*t*, position changes by

$$x_{t+\Delta t} - x_t = v \cdot \Delta t$$

where x_t is its position at time *t*.

Once bodies move to new positions, forces change, so the computations have to be repeated.

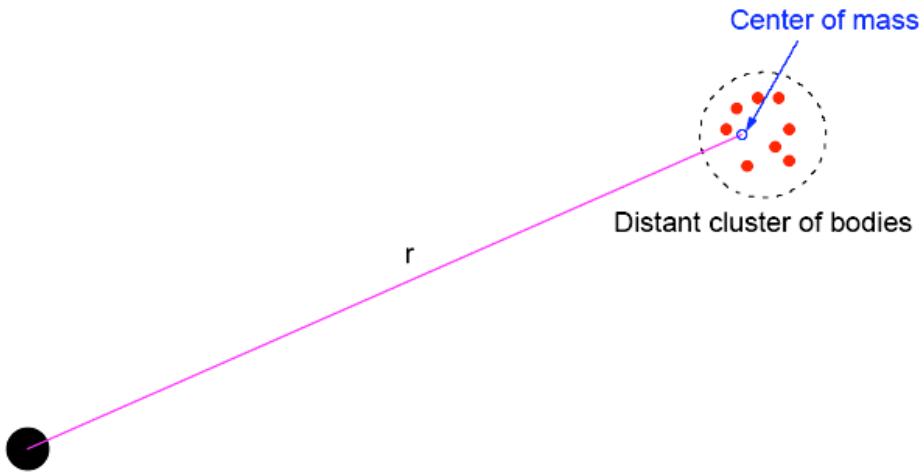
Overall gravitational *N*-body computation can be solved in sequential pseudo-code as:

```
for (t = 0; t < tmax; t++){              /* for each time period */
        for (i = 0; i < N; i++) {         /* for each body */
            Fxyz = Force_routine(i);                  /* compute net force of ith body */
            Vxyz_new[i] = Vxyz[i] + Fxyz * dt / m;     /* compute new velocity */
            xyz_new[i] = xyz[i] + Vxyz_new[i] * dt;   /* and new position */
            }
        for (i = 0; i < N; i++) {          /* for each body */
            xyz[i] = xiz_new[i];                /* update velocity & position*/
            Vxyz[i] = Vxyz_new[i];
            }
```

}

The sequential algorithm is an O(N^2) algorithm (for one iteration) as each of the *N* bodies is influenced by each of the other *N* - 1 bodies. It is not feasible to use this direct algorithm for most interesting *N*-body problems where *N* is very large.

Clustering and Barnes-Hut algorithm. Time complexity can be reduced approximating a cluster of distant bodies as a single distant body with mass sited at the center of mass of the cluster:



Barnes-Hut algorithm uses divide-and-conquer strategy to find clusters of particles in the N-body problem. It helps to reduce the computational complexity of the problem to O(*N Log N*).

A. *Constructing a tree.*

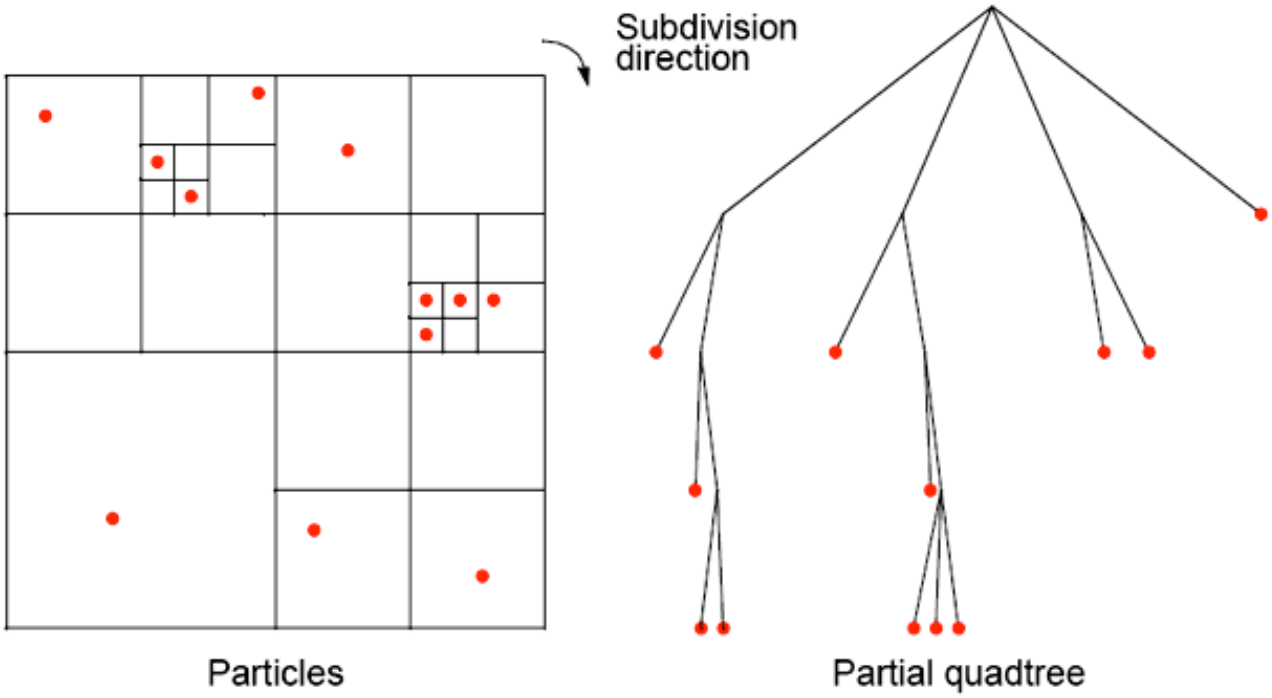
Start with whole space as a cube containing all the bodies.
1. First, this cube is divided into eight sub-cubes.
2. If a sub-cube contains no particles, sub-cube deleted from further consideration.
3. If a sub-cube contains one body, the sub-cube is retained.
4. If a sub-cube contains more than one body, it is recursively divided until every sub-cube contains only one body.

For three-dimensional space the process creates an octal-tree with up to eight edges from each node. The leaves of the tree represent cells, each containing one body.

B. Calculating and storing the total mass and the centre of mass.

After the tree has been constructed, the total mass and center of mass of the sub-cube is stored at each node. There are two types of nodes in the tree; internal and external. An

external node has no children and is either empty or represents a single body. Each *internal* node represents the group of bodies beneath it, and stores the centre of mass and the total mass of all its children bodies.



The figure demonstrates such recursive division in 2-dimensional space.

C. Traversing the tree.

The net force on each body is obtained by traversing tree, starting at root and stopping at a node when the clustering approximation can be used.

If the center of mass of an internal node is sufficiently far from the body, the bodies contained in that part of the tree are treated as a single body whose position and mass is respectively the center of mass and total mass of the internal node. If the internal node is not sufficiently far from the body, the process is repeated for each of its children.

Whether a node is or isn't sufficiently far away from a body, depends on the quotient s/d, where s is the width of the region represented by the internal node, and d is the distance between the body and the node's center of mass. The node is sufficiently far away when this ratio is smaller than a threshold value θ, which is typically 1.0 or less. The parameter θ determines the accuracy of the simulation; larger values of θ increase the speed of the simulation but decreases its accuracy. If $\theta = 0$, no internal node is treated as a single body and the algorithm degenerates to a direct-sum algorithm.

Thus, the algorithm can be described by the following:

```
for each time interval {
        Build_Octal_Tree();
        Compute_Total_Mass_and_Center();
```

```
    Traverve_Tree_and_Compute_Forces();
    Update_Position_Velosity();
  }
```