Parallel implementations for Barnes-Hut algorithm

Anas . Almousa

College of Information and Computer Science Department of computer Engineering
King Fahd University of Petroleum and Minerals
Dhahran 31261, Saudi Arabia
Email: { g200805960}@kfupm.edu.sa

Contents

1. Progress information

We expanded the algorithm to 3D model. And did measurements for the proportion of time the algorithm spends in each step:

We found that the dominant operation is the force computation step. Notice that currently only the force computation step is the parallelized step. Other steps have not been parallelized or can't be parallelized. For the parallelized step, we achieved a speedup of about 7.3 for 8 threads for this step, as seen in Figure 1 belowbelow.

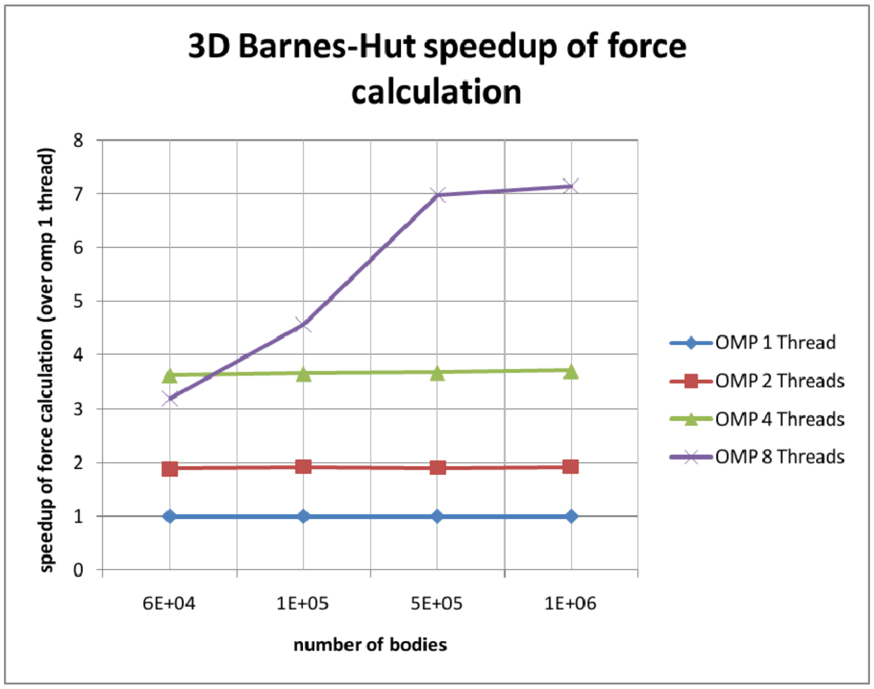


Figure 1: speedup of the parallelized force calculation step of 3D Barnes-Hut algorithm

Due to the parallelization of only the force calculation time; the above mentioned speedup will be somehow far from the speedup of total simulation time, which is charted in Figure 2 below.

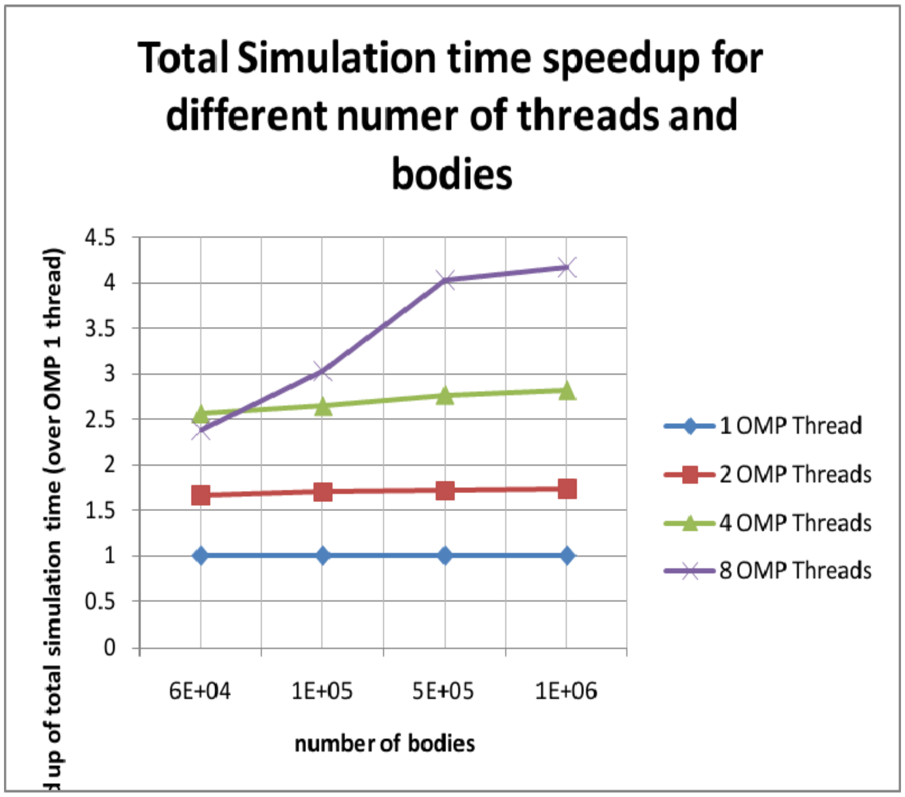
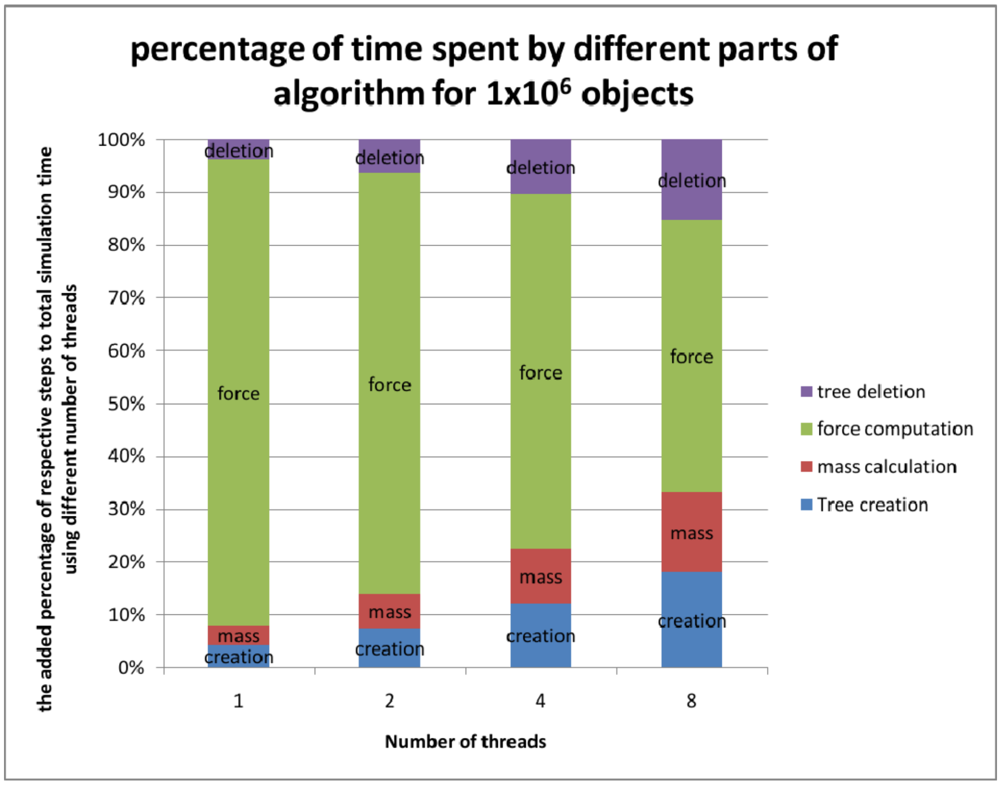
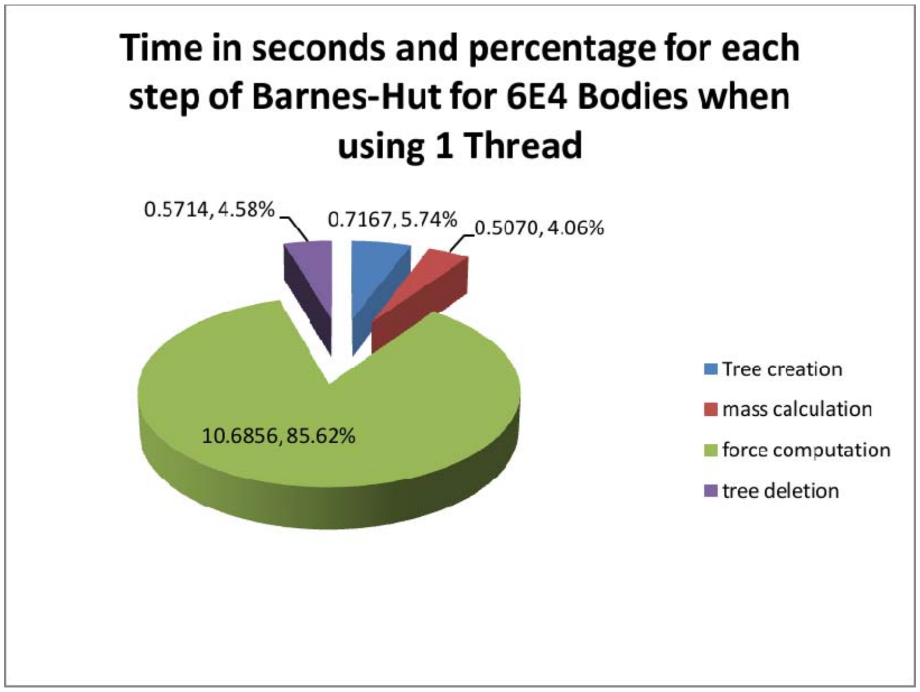


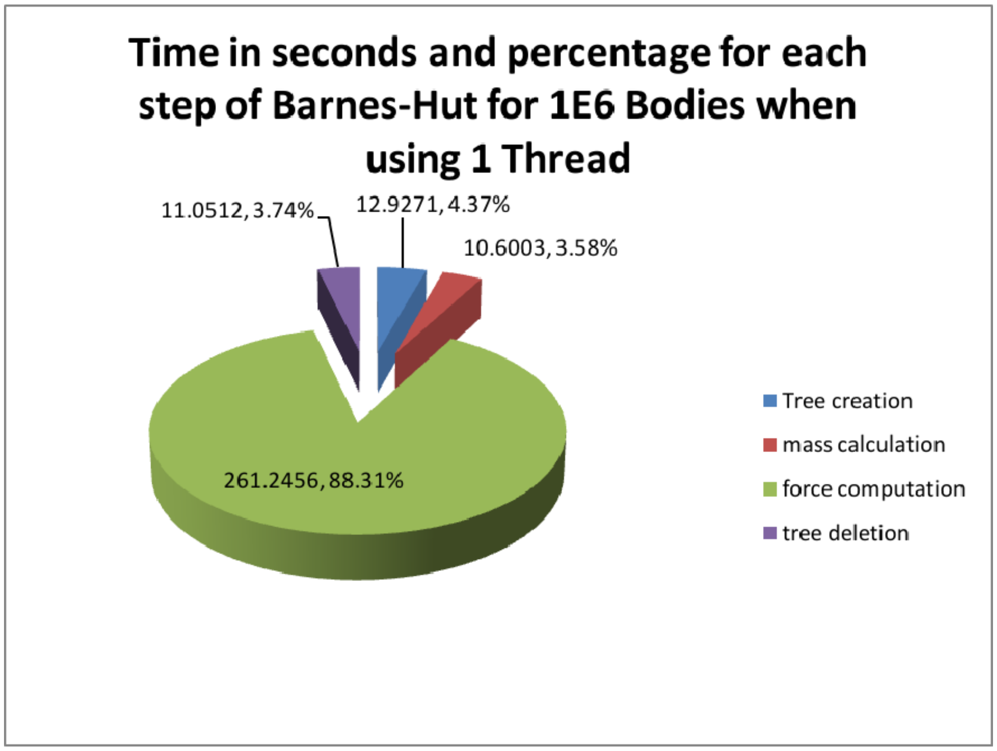
Figure 2: speedup of total simulation time of 3D Barns-Hut algorithm

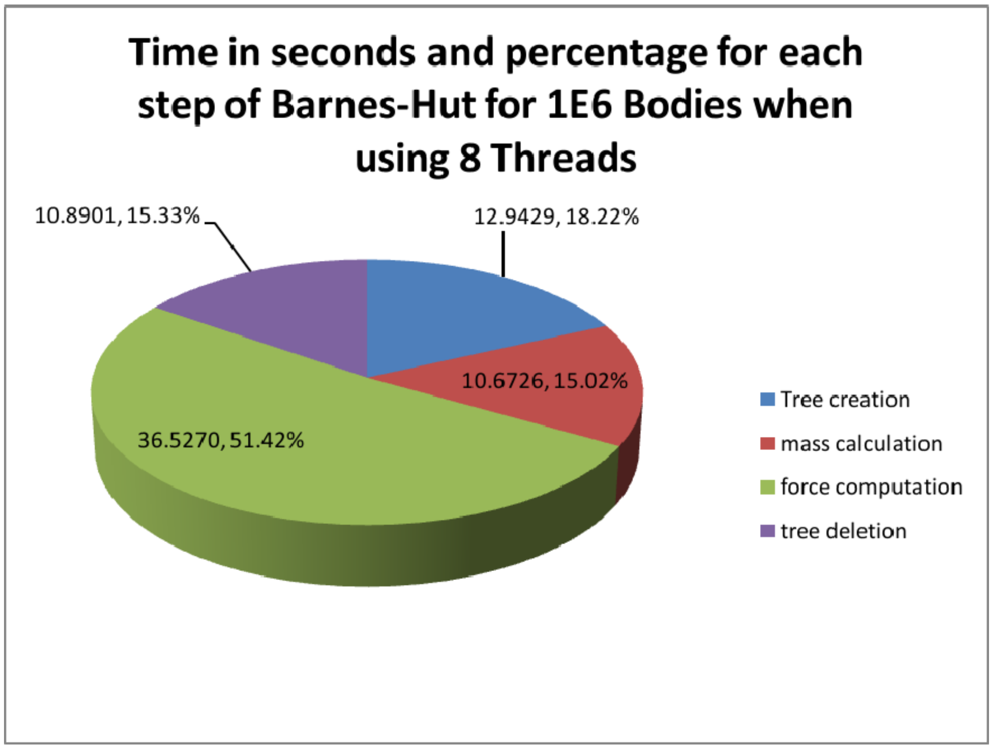
This would raise a question of what is the exact proportion of time each step is using. To sum up the change of proportions of total simulation time, we show a stacked chart that sums this information in the figure below. We see how the force computation step is taking less percentage of total simulation time as we increase the number of threads. This would raise the question of whether the parallelization of other steps shall be our concern. To know that, we show the charts that specify the exact timings of simulation steps with dofferent number of bodies and threads, hence we would see how the time consumed by these step would escalate with number of bodies.



The charts below show the timings for 1 thread and 8 threads for 2 different counts of bodies.







Notice how that for 1 thread, the difference in time for steps other than the force calculation step doesn`t escalate much between 6E4 and 1E6 bodies, they only approximately double. Also notice how for 8 threads, force computation becomes comparable to other steps (3 times slower) for 1E6 bodies.

Taking note of the above charts yields that those other steps does not escalate in a fast speed with number of bodies, which would eliminate the idea of investing time in their parallelization from our concern.

2. Load balancing information:

We have done some fast-small test to see how much work is being done by each thread, and we have seen that for all number of bodies the time being spent by each thread is only different by a fraction of a second from other threads. This is likely the effect of the initialization of bodies into space in a uniformly random manner.

3. Pseudo code for the implementation.

In this section we summarize the pseudo code for our implementation of Barnes-Hut algorithm, we repeate the following steps for a decided number of iterations (e.g. 30 iterations):

```
1. create the body structures, initializing them to random data as follows
    1. Initialize body positions in a uniformly distributed random manner inside a space of 400x400x400.
    2. Initialize velocity x, y and z components randomly between 0 and 10.
    3. Decide body mass based on random numbering between 1 and 10.
    4. Initialize timers and set number of threads
2. Create the tree as follows
    1. Create a single new empty node R (representing a single space partition) for the tree.
    2. For each body in space , Insert body B into tree under node R as follows:
        i. Do the insertion of body B under node R as follows.
            1. If tree is still empty, create a new node inside the tree with that body inside it; continue covering the bodies in step (Error! Reference source not found.) with node R being this created node.
            2. else, (if tree is not empty): if node R decided to receive the body is an internal node  in the tree
            3. Based on distance between center of node R and body B, decide on a Child C of node R's children nodes to insert body B into.
            4. Execute the insertion step (Error! Reference source not found.) above with the node R being the child node C, chosen above.
            5. Else (if R is a leaf node), do the following :
            6. The body represented by this node is preserved as body B1.
            7. Convert node R into an internal node.
            8. insert body B1 into (now internal) Node R by executing step (i) above for node R and the body being B1
            9. Insert body B into node R by executing step (i) above for node R and body B.
3. Starting from root of tree R, Compute masses in the tree as follows:
    1. If R is empty or a leaf, consider this instance of this step (step3.) to be finished.
    2. Else, for each child (C)of node R, compute the mass of the sub tree by executing an instance of step (3) above with R being the child C
    3. Calculate Total mass of node by summing all total masses of the all children of node R. and register it in node structure.
    4. Calculate x, y and z components of center of mass by summing all the multiplications total mass of node R by respective x, y, z components of all of R's children. And register those components in node's structure.
4. Now update the position components of the node structure.
    1. Assign the following two tasks for all bodies over threads
        i. For each body B do the following:
            1.  Reset the registered force components affecting body B to zero (of that body).
```

a. Starting from root node R of the tree, calculate the forces of the nodes in the tree affecting body B as follows :

b. If R is empty, consider this instance of step (a) above to be finished

c. Else calculate the distance between body B and center position of node R as distance D.

d. If D is larger than a certain threshold T, then for each child C of node R, calculate the force effecting body B by this node by executing step(a) above with R being the child C.

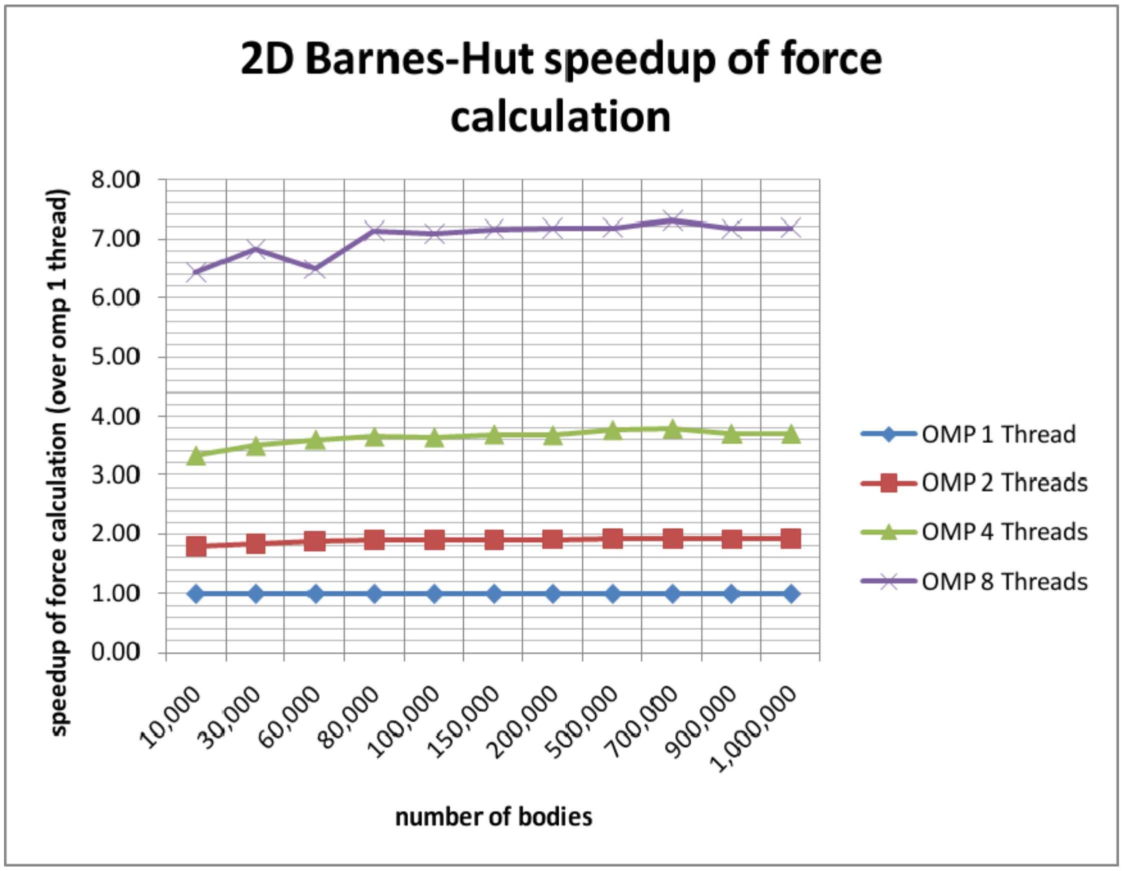
e. Else, (distance is smaller than threshold) then calculate the force affecting body B by Node R as a whole on behalf of its sub tree. And register force components in body B's structure.

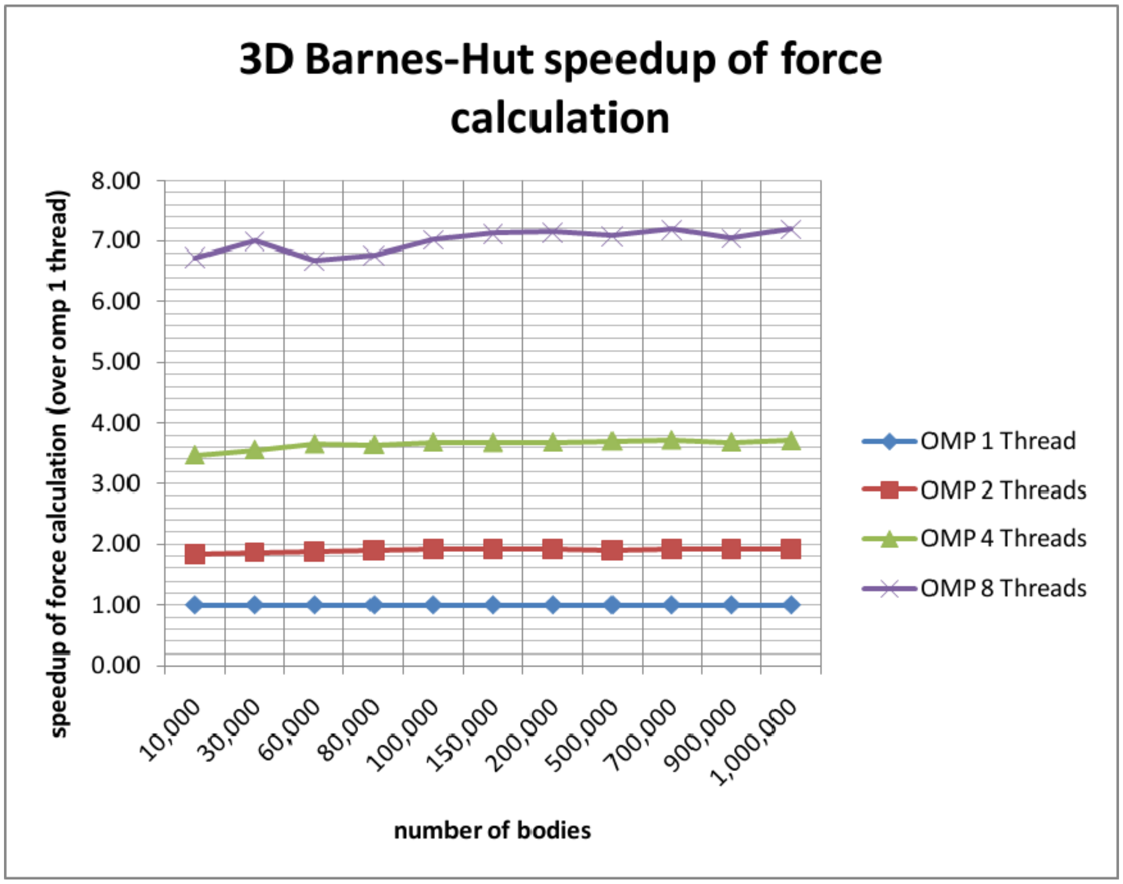
ii. For each body B

1. Update the velocity and position of body depending on the force components affecting this body calculated in previous step.

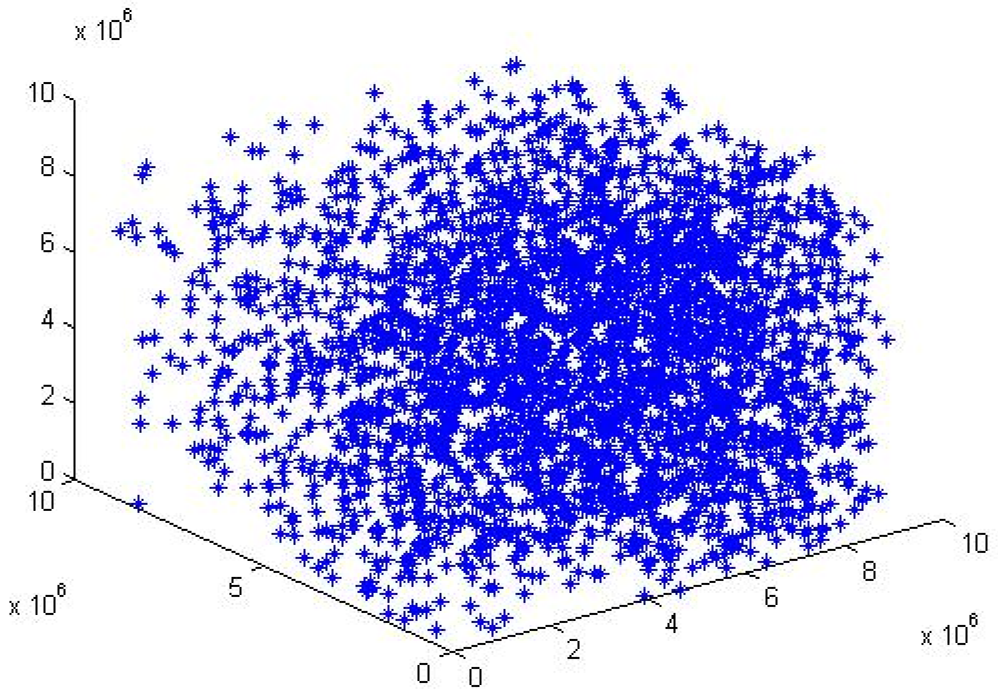
5. Delete the tree of nodes recursively from leafs to root, preserving only bodies that are still inside the space in the bodies structure.

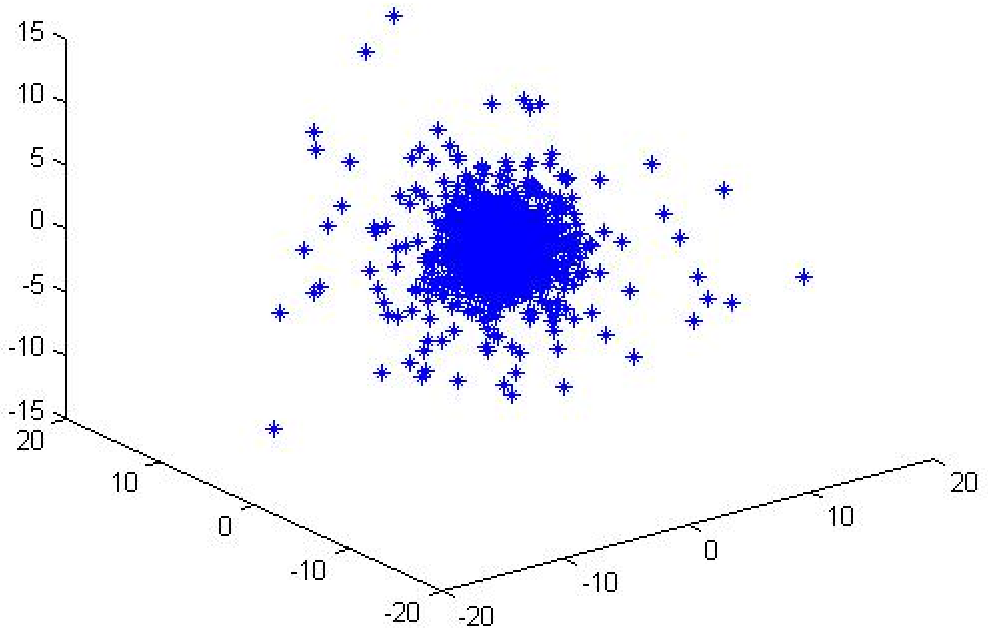
End of progress report


2D Barnes-Hut speedup of force calculation
speedup of force calculation (over omp 1 thread)
8.00
7.00
6.00
5.00
4.00
3.00
2.00
1.00
0.00
10,000
30,000
60,000
80,000
100,000
150,000
200,000
500,000
700,000
900,000
1,000,000
number of bodies
OMP 1 Thread
OMP 2 Threads
OMP 4 Threads
OMP 8 Threads


3D Barnes-Hut speedup of force calculation
speedup of force calculation (over omp 1 thread)
8.00
7.00
6.00
5.00
4.00
3.00
2.00
1.00
0.00
10,000
30,000
60,000
80,000
100,000
150,000
200,000
500,000
700,000
900,000
1,000,000
number of bodies
OMP 1 Thread
OMP 2 Threads
OMP 4 Threads
OMP 8 Threads

Distribution of bodies in the space using a Poisson polar probabilistic distribution





The algorithm for generation of 3D position according to the Poisson distribution is as follws:
1. Let's u be a random variable that is generated using a uniform distribution over [0,1]
2. Evaluate r= - r0*[Ln(1-u)], where r0 is the desired average distance to space center, Ln is Log neperian, and r is the generated radius or distance to center,
3. Also generate a and b using a uniform distribution over [0,1]*2Pi,
4. The mapping from spherical coordinate to Cartesian 3D:
X = r*cos(a)*cos(b)
Y = r*cos(a)*sin(b)
Z = r*sin(a)
Where r should follow a Poisson distribution, and a and b must be uniform within [0,2Pi].
Please try the above rules and try to see the generated points in a plan.

Forcasting the tread execution time in the next iteration using the Double Exponential forcasting algorithm:

The completion times of a series to be forcasted is a value Y(t) observed at time t. The estimated value for Y(t) is denoted as YY(t) and the forcasted value at time t+1 is YY(t+1).

L(t)=a*Y(t)+(1-a)*YY(t)
T(t)=b*(L(t)-L(t-1))+(1-b)*T(t-1)
YY(t+1)= L(t) + T(t)

Starting : YY(1)=Y(1); YY(2)=Y(2); L(1)=Y(1); and T(1)=0.
In other words, we need Y(1) and Y(2) to start the algorithm and produce a forcast YY(3).

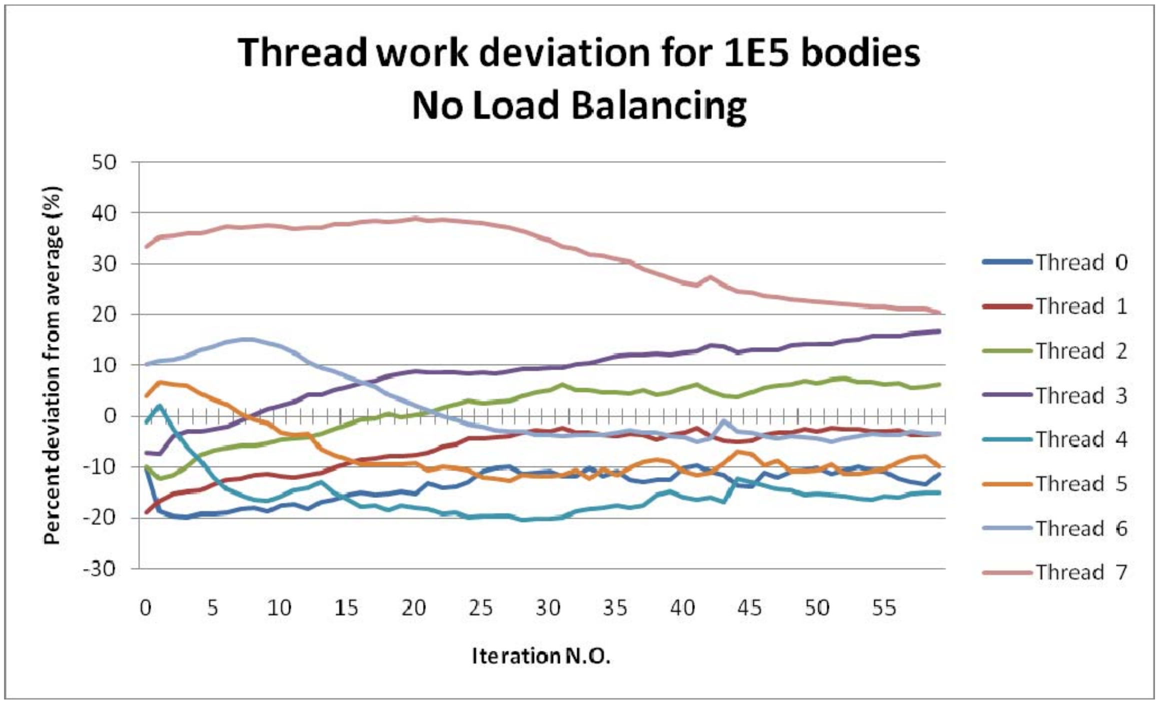
How do we choose the value of constant a and b:

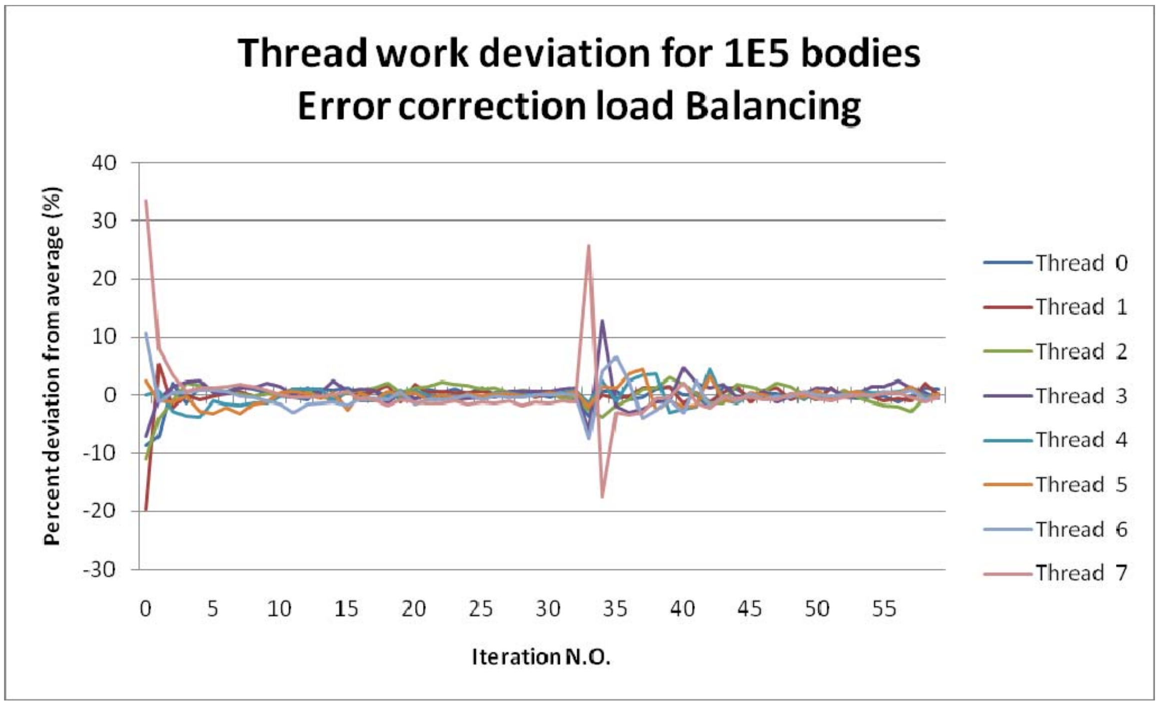
- Constant a is called the smoothing parameter, where 0 ≤ a ≤ 1.
- For noisy data the value of a must be small, and for smooth data the value of s must be large.
- In practice 0.05 ≤ a ≤ 0.35
- Constant b is called trend, where 0 ≤ b ≤ 1. For linear trend b must be small and for changing trend b must be large.
- One may choose a=b=0.35.

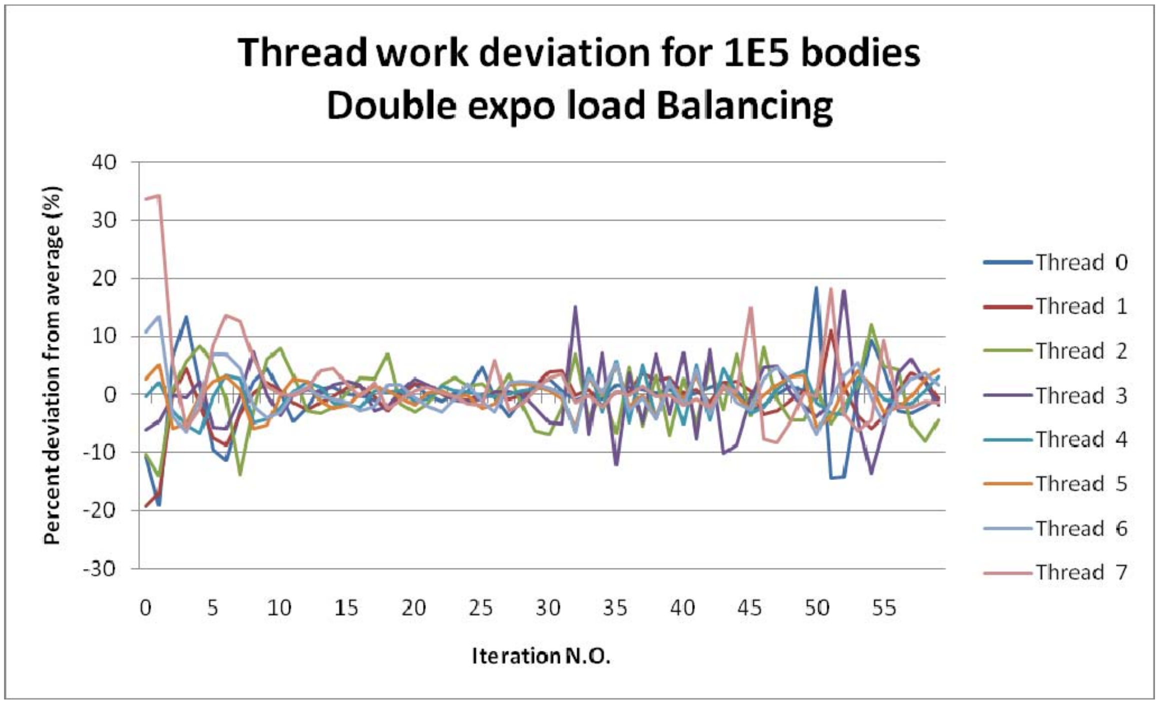
Dynamic Load Balancing based on forcasting the thread execution time and linear correction of the thread computational load:

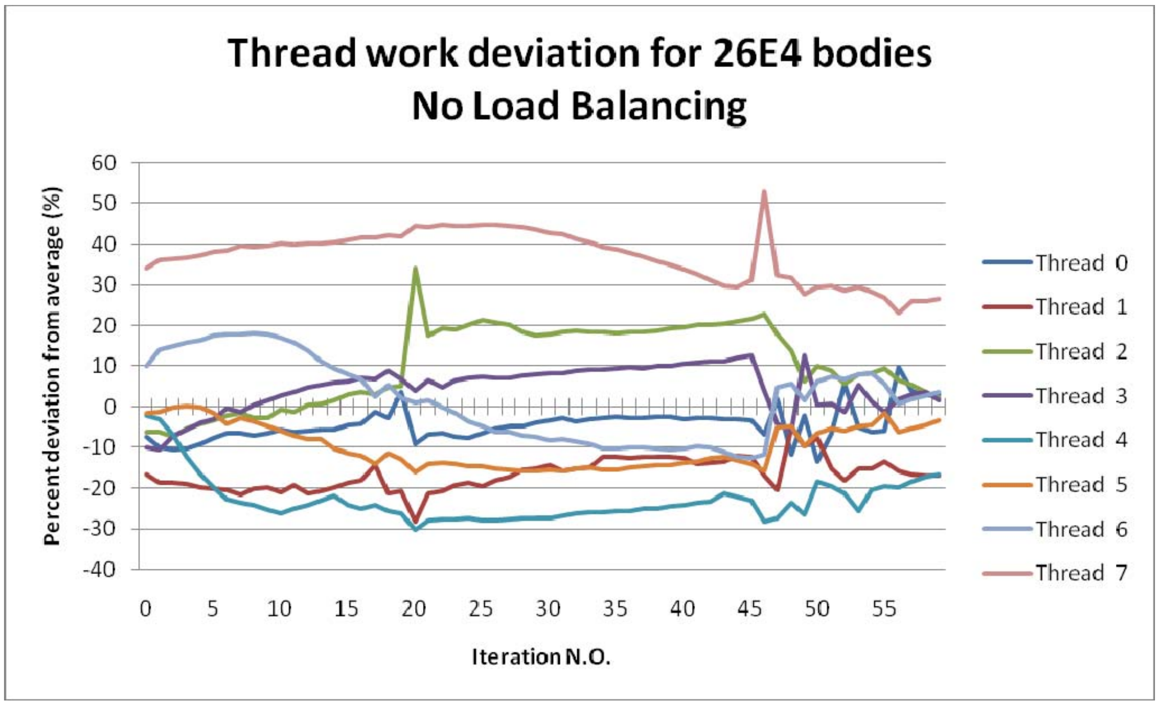
Reference to the need to eliminate the system fluctuations in the execution times due to OS time-sharing. The exact update on the number of bodies should be:

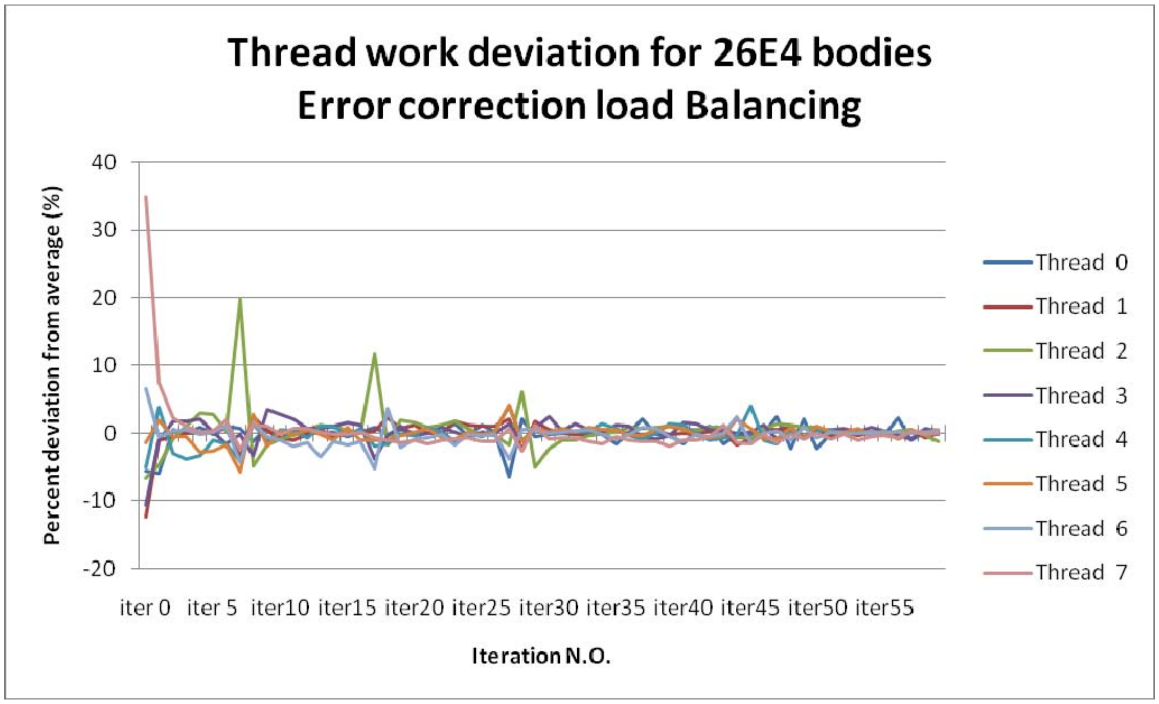
```
N = N - [100*(th-th')/th']*G*Nbodies/(8*100)
>>
>> Where   N is number of bodies for next iteration for a given thread
>>             th is forcasted on the thread time for this iteration
>>             th' is average thread time for this iteration,
>>             G is some gain which is typically 0.2 or 0.5
>>             Nbodies is the total number of bodies
>>
>> Thus  100*(th-th')/th' = percent deviation from average or simply the
>> forcast YY on the deviation from average for this iteration:
>>
>> N = N - YY*G*Nbodies/(8*100)
>>
>> Please check, before applying the correction, that the forcast YY is
>> correctly generated in a system without correction on the number of
>> bodies.
>>
```

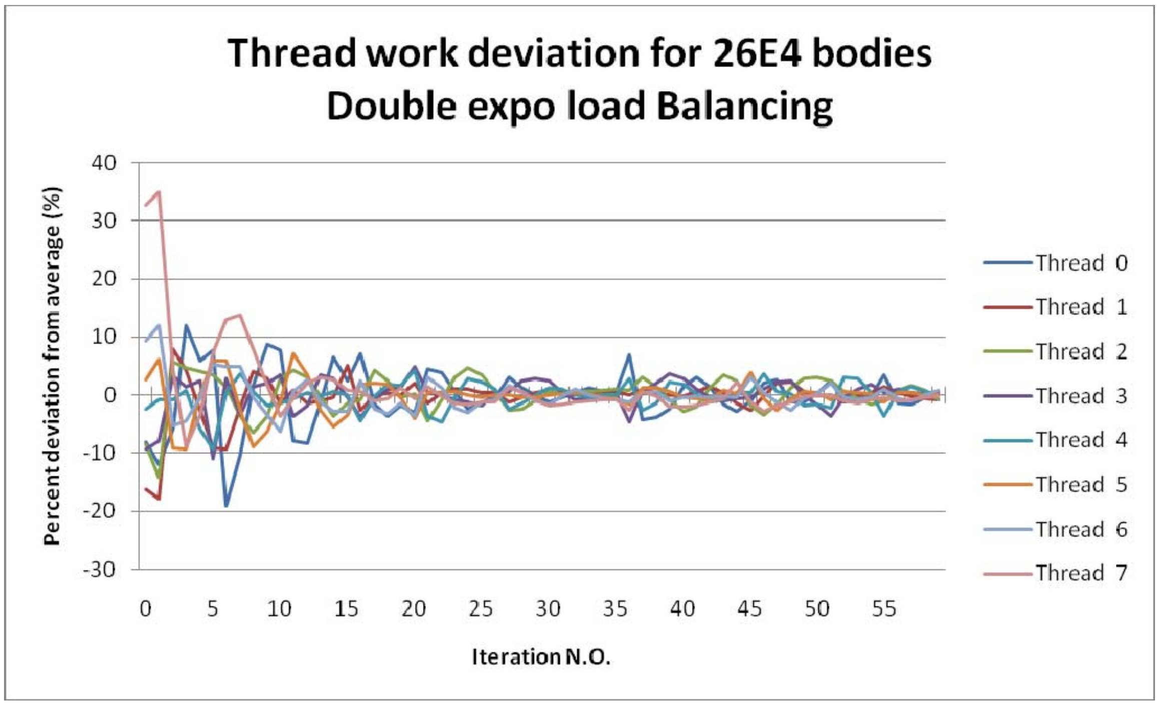

Thread work deviation for 1E5 bodies
No Load Balancing
Percent deviation from average (%)
Iteration N.O.
Thread 0
Thread 1
Thread 2
Thread 3
Thread 4
Thread 5
Thread 6
Thread 7
50
40
30
20
10
0
-10
-20
-30
0
5
10
15
20
25
30
35
40
45
50
55

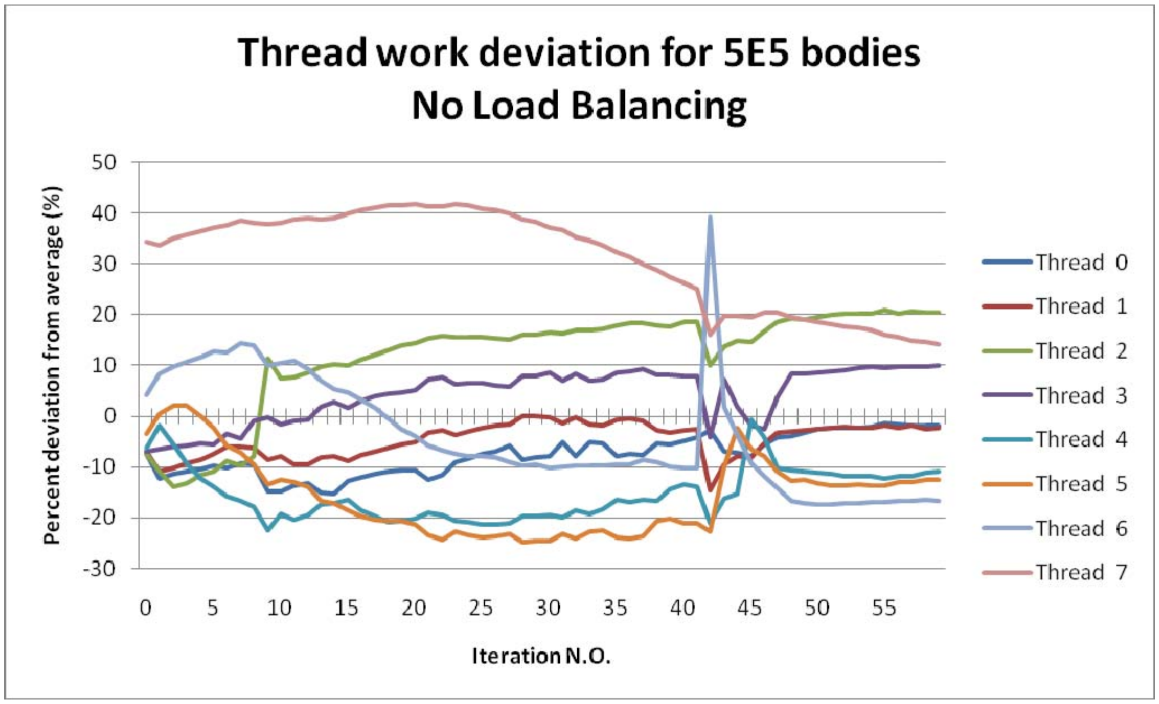

Thread work deviation for 1E5 bodies
Error correction load Balancing
Percent deviation from average (%)
Iteration N.O.
Thread 0
Thread 1
Thread 2
Thread 3
Thread 4
Thread 5
Thread 6
Thread 7
40
30
20
10
0
-10
-20
-30
0
5
10
15
20
25
30
35
40
45
50
55


Thread work deviation for 1E5 bodies
Double expo load Balancing
Percent deviation from average (%)
Iteration N.O.
40
30
20
10
0
-10
-20
-30
0
5
10
15
20
25
30
35
40
45
50
55
Thread 0
Thread 1
Thread 2
Thread 3
Thread 4
Thread 5
Thread 6
Thread 7


Thread work deviation for 26E4 bodies
No Load Balancing
Percent deviation from average (%)
Iteration N.O.
60
50
40
30
20
10
0
-10
-20
-30
-40
0
5
10
15
20
25
30
35
40
45
50
55
Thread 0
Thread 1
Thread 2
Thread 3
Thread 4
Thread 5
Thread 6
Thread 7


Thread work deviation for 26E4 bodies
Error correction load Balancing
Percent deviation from average (%)
40
30
20
10
0
-10
-20
iter 0
iter 5
iter10
iter15
iter20
iter25
iter30
iter35
iter40
iter45
iter50
iter55
Iteration N.O.
Thread 0
Thread 1
Thread 2
Thread 3
Thread 4
Thread 5
Thread 6
Thread 7


Thread work deviation for 26E4 bodies
Double expo load Balancing
Percent deviation from average (%)
40
30
20
10
0
-10
-20
-30
0
5
10
15
20
25
30
35
40
45
50
55
Iteration N.O.
Thread 0
Thread 1
Thread 2
Thread 3
Thread 4
Thread 5
Thread 6
Thread 7


Thread work deviation for 5E5 bodies
No Load Balancing
Percent deviation from average (%)
50
40
30
20
10
0
-10
-20
-30
0
5
10
15
20
25
30
35
40
45
50
55
Iteration N.O.
Thread 0
Thread 1
Thread 2
Thread 3
Thread 4
Thread 5
Thread 6
Thread 7