Parallel Programming

N-Body & Single Roller Coaster Problem

Peter Huang

1、 Design— the implementation for Barnes-Hut Algorithm

A、 How to parallelize with Barnes-Hut Algorithm on building tree phase

1. Assign the same amount of data to N processes or N threads (total data / #process)
2. Each processor is responsible for adding its data into the ONLY quadrant tree whose each node contains Q1,Q2,Q3 and Q4 tree node
3. Each node assigns with its mutex_lock for preventing synchronization problem when building the tree
4. Before finding to suitable location and insert new node, the program adds the node's mass into existing node when traverse the tree.
5. Go through step4, and when no any tree node is existing in the current tree, adding itself into the tree node and also recording the current min X, max X , min Y, max Y value for the new added node.

B、 How to partition the task

1. Assign the same amount of data to N processes or N threads (total data / #process)
2. Processor or Thread will do its own task involving Building tree、Calculating Force、Update new location and speed

C、 How to prevent the synchronization problem

1. Building tree: each node has its own mutex_lock, before any updating actions like updating data or inserting node, the process should get the node’s mutex_lock first.
2. Calculating force and Update location, speed: Any global data’s operation should limited in critical section (like getting mutex_lock in advance)
3. How main program signify threads or threads signify main program. I adopt spin lock 、condition variable or muetex_lock for preventing synchronization problem. For example: main program get mutex_lock and waits for all threads which are doing assigned task, after all threads finish its task, all thread signify main program to execute next task.

D、 What technique is used to reduce execution time and increase scalability

.

1. Averagely assign the same amount of data to each processes, which should be the key for Barnes-Hut Algorithm. The reason is that, based on the experiment, we know, build tree (log N) & I/o take a little time only, the most of time is taken on going thorough each node to calculate its new location and new speed. So if we get more and more threads for it, the time complexity will be reduce largely, because the data has been partition to smaller unit for each processor, so the processor can finish its job quickly. (the bottleneck on this algorithm: traverse tree node take N log N)

2. Build tree. Assign each node with mutex lock, before any operation in the node like update value or inserting its sub-node, we should get mutex_lock first, after finished the operation, go through to next node, we release its mutex_lock for other process use, and other processer can add new node by get individual node’s mutex_lock. (By the way, building tree takes little consuming time and just takes log N to build tree.)

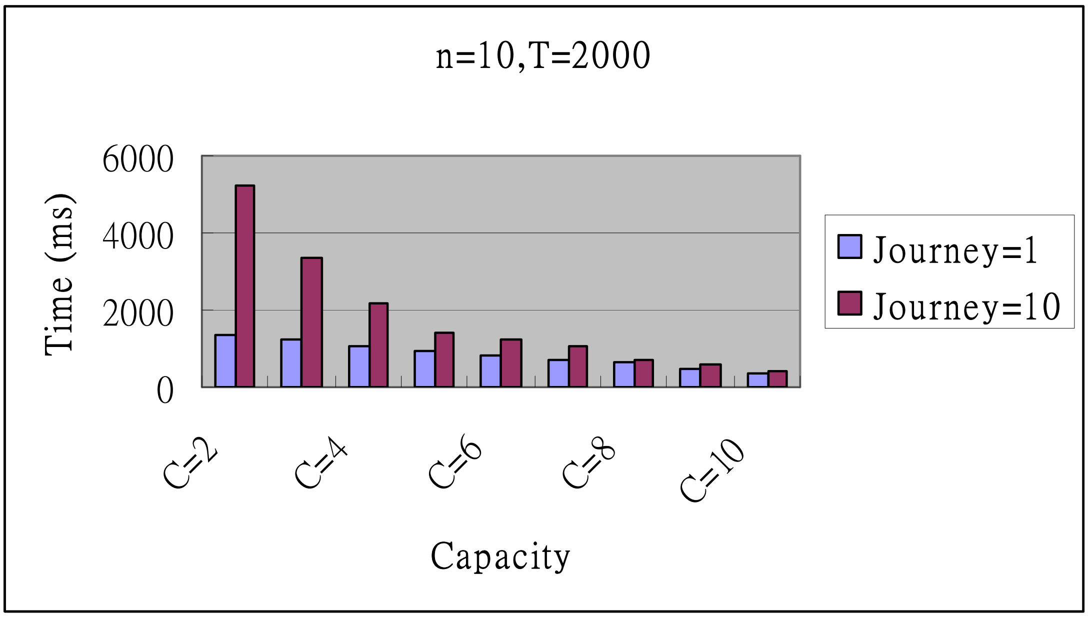
E、 Other efforts you do in the program

1. Decrease the action of thread creation and thread destroy action to reuse the existing thread, which means the programs just create thread the ONLY one time. However, finally there are not obvious speed-up effect for the implementation.

2、 Performance Analysis on Single Roller Coaster Car Problem

A、 Plot the average waiting time v.s. the input arameters C or T.

(1) Adjust C value, and fixed other parameters



Fixed N=10, T=2000 , We just adjust variable C value.
Input parameter: XXX.out 10 C 2000 1 or XXX.out 10 C 2000 10

Capacity	Journey=1,**(Total waiting person:10)**	Journey=10, **(variable total waiting persons)**
C=2	1346.247	5254.809
C=3	1235.194	3364.537
C=4	1064.049	2183.538
C=5	955.988	1408.380
C=6	823.840	1238.447
C=7	715.128	1043.756
C=8	669.020	682.799
C=9	477.277	590.2544
C=10	377.490	386.54025

When the Capacity is increased, the average waiting becomes reduced.

Analysis:

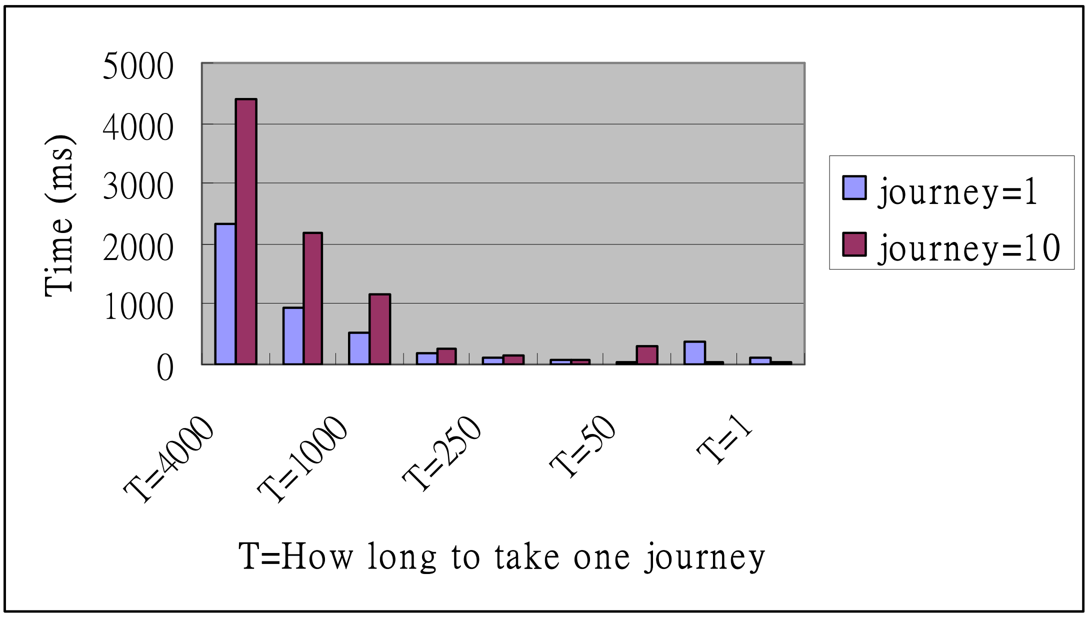
First ,I set the number of journey=1, T which at least is larger than t and the way is to reduce the variable condition of experiments. (Because of t is random time, however, when T is equals with t, the experiment will be influenced by t value, because if T is equals with t or smaller than t , the waiting time will be dominated by t, because people just take a ride right away.)

Total waiting time / waiting person. ➔ ((n-c)* T +initial time) / (c+n-c) ➔ ((n-c)*T+ initial time) / waiting person (n). Based on the c, when c is larger, the waiting time will reduced at the only journey.

Finally, in generally, based on the above rule, we also can get the conclusion, when we fixed other variable, and make **T is larger than t, when C become larger, the waiting time will be reduced.**

However, why journey 10 is not the 10 times with journey 1, because finally, our total waiting time should be divided by total waiting persons. Because we cannot get the same proportions on the waiting persons, which means the total waiting persons is variable. For example, in the last time if the car is finished earlier than the person who just finished the ride and is approaching to take a ride, the total waiting person will be less. However, if the person who have been finishing the ride and is approaching to take a ride and finally comes into the waiting queue, which happens earlier than car finished the last ride, there are more the number of waiting persons. Because the variable condition, that is why the waiting of the journey=10 is not equally time with the waiting time of journey=1.

(2) Adjust T value, and fixed other parameters



Fixed N=5, C=2 , We just adjust variable T value.
Input parameter: XXX.out 5 2 T 1 or XXX.out T 2 T 10

T	**journey=1**	**journey=10**
T=4000	2345.152	4381.092
T=2000	954.76	2190.837
T=1000	519.946	1176.88
T=500	206.052	271.12
T=250	103.719	143.62
T=100	67.321	65.541
T=50	26.64375	293.823
T=10	393.693	51.345
T=1	102.7408	50.286

Analysis:
When journey=1 and T is larger than t, please following previous formula: Total waiting time / waiting person. ➔ ((n-c)* T +initial time) / (c+n-c) ➔ ((n-c)*T+ initial time) / waiting person (n). So we get (n-c) T /n ➔ So the form T=1000~4000, match the value, T will dominate the result.

And the situation also happens in journey=10, because when T is larger than t, when passenger comes in to waiting Queue which take a little time, and the most of waiting time is dominated by T's value, because he waiting on the waiting queue for taking a ride. So based on the diagram when T is >= 1000, we found out that the T, the time car spends for one ride, is dominated the result.

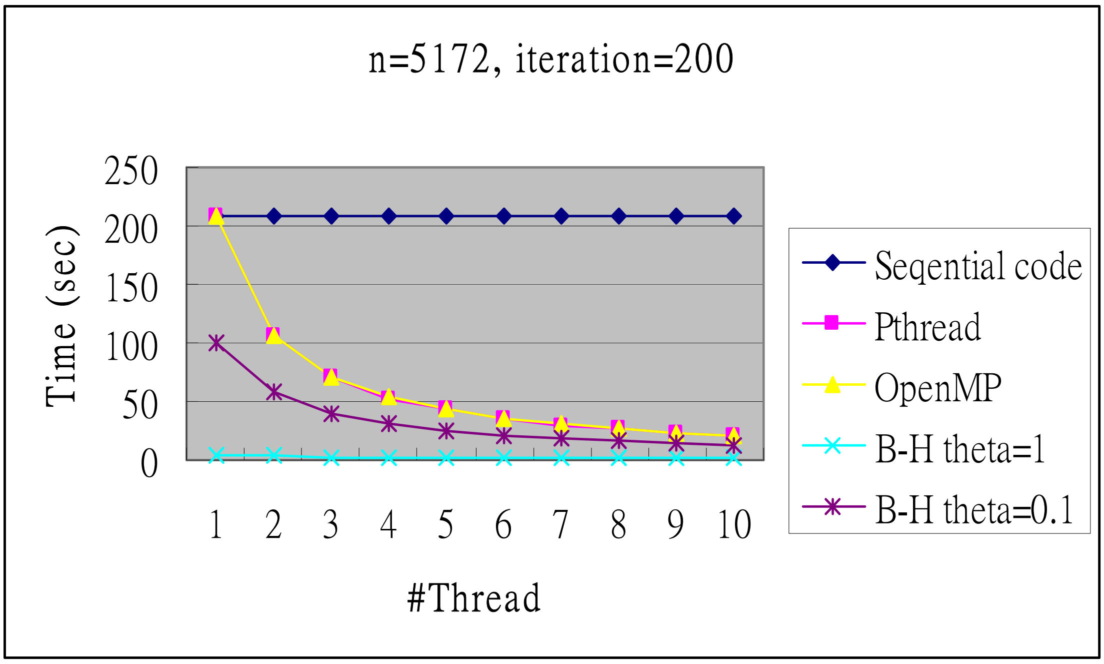
However, when T is less than 1000 or even less than 500ms, we found that the average waiting time is around 1~500 ms. Why we get so little waiting time, even if the journey takes more than 10 times, I suspect that it is due to the passenger's wandering time is large than T, the time car spends for one ride. On the situation, the passenger's wandering time is dominated the case. So the waiting time will represents on how long the passenger's wandering around the park. After wandering around the park, they spend a little time to waiting for one ride, which is why we get smaller value on waiting time. Finally, we get the conclusion, when t is larger than T, t value will dominated the experiments. However, t is randomized around 1~ 1000 ms. These why we sometimes get little waiting time and sometimes get more waiting time. However, on matter which we get, when t is larger than T, the waiting time will be under 1000ms.

3、 Performance Analysis on N-body Problem

A、 Strong Scalability

(1) Fix N, T and manipulate # Threads.

Input parameter : XXX.out 2 1 200 1 test3.txt 1 disable
N=5172, T=200



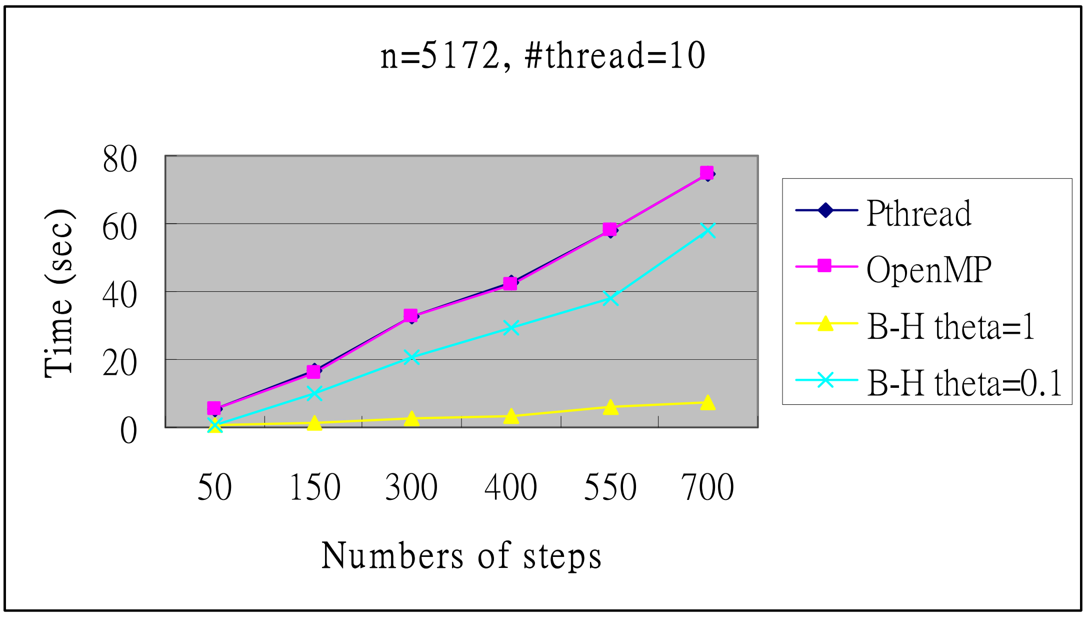
#thread (sec.)	1	2	3	4	5	6	7	8	9	10
Sequential code	208.423	208.423	208.423	208.423	208.423	208.423	208.423	208.423	208.423	208.423
Pthread	208.325	105.997	70.4	53.038	43.523	34.953	29.946	26.205	23.323	21.499
OpenMP	208.334	105.939	70.13	53.224	43.642	35.115	30.26	26.545	23.437	21.222
B-H theta=1	4.528	3.179	2.501	2.102	1.94	1.854	1.782	1.611	1.579	1.514
B-H theta=0.1	99.583	57.298	40.439	30.788	25.209	21.283	18.52	16.28	14.716	13.197

Analysis: By the diagram, we can see all the programs run faster, when it gets more threads. The total execution time will be reduced by providing more threads, and the original task will be partitioned to the less amount of data. (total data / #thread). B-H=1 is the fastest, because it can filter the most of subnodes, which theta is less than 1. B-H=0.1 is slower than B-H1, because B-H=0.1 is filter less data, and the only data it can viewed as single mass, is less than theta 1. So it definitely B-H=1 will be faster than B-H=0.1. However, when B-H=0.1 is faster than Pthread and OpenMp version, because the two of them need to go through all nodes, and no any data can be filtered, which is why the execution-time likes "B-H=1 > B-H=0.1 > Pthread and Openmp > sequential codes"

B、 Weak Scalability

(1) Fix N, #Threads and manipulate T iteration.

Input parameter : XXX.out 10 1 700 1 test3.txt 1 disable
N=5172, #Thread=10



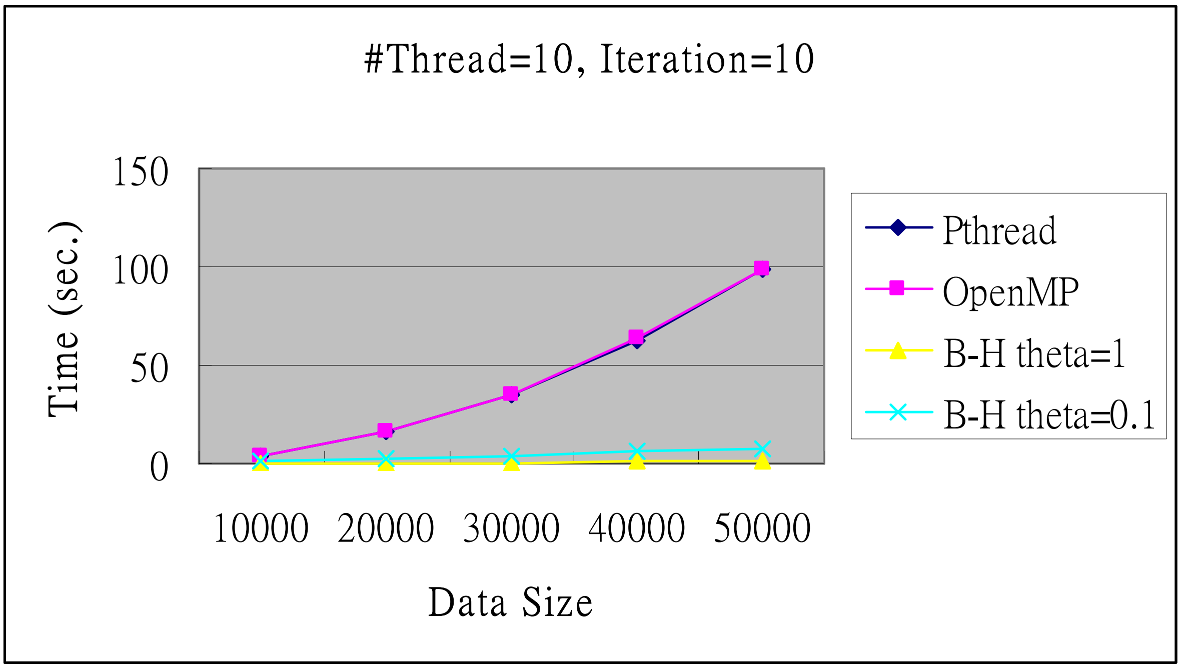
Iterations	50	150	300	400	550	700
Pthread	5.283	16.907	32.387	42.437	57.904	74.511
OpenMP	5.26	15.724	32.496	42.162	57.686	74.885
B-H theta=1	0.397	1.363	2.412	3.403	5.768	7.541
B-H theta=0.1	0.4	10.261	20.922	29.407	38.185	58.132

Compared with sequential code, sequential code running 50 iterations with 1 thread take: 52.111 sec.

Analysis: When number of steps increases, which represents more iterations should be done by each programs, it is obvious for all programs to increase execution time by more iterations. However, the B-H=1.0 increases the less execution time, because in the data, there are many masses that it is not need to be traversed, because these nodes's parent or parent-parent node whose theta is less than theta=1. So a lot of data has been skipped, that is why it is fastest with these version. However, B-H=0.1 is slower than B-H=1.0, which is also the same reason that the data can be filter is less than B-H=1.0. And, the B-H=0.1 is faster than pthread and openmp, because pthread and openmp will traverse each nodes, no any nodes will be skipped when traversing tree., so it take the most of time.

(2) Fix T, #Threads and manipulate N size

Input parameter: XXX.out 10 1 10 1 test-big.txt 1 disable
#Thread=10, T=200



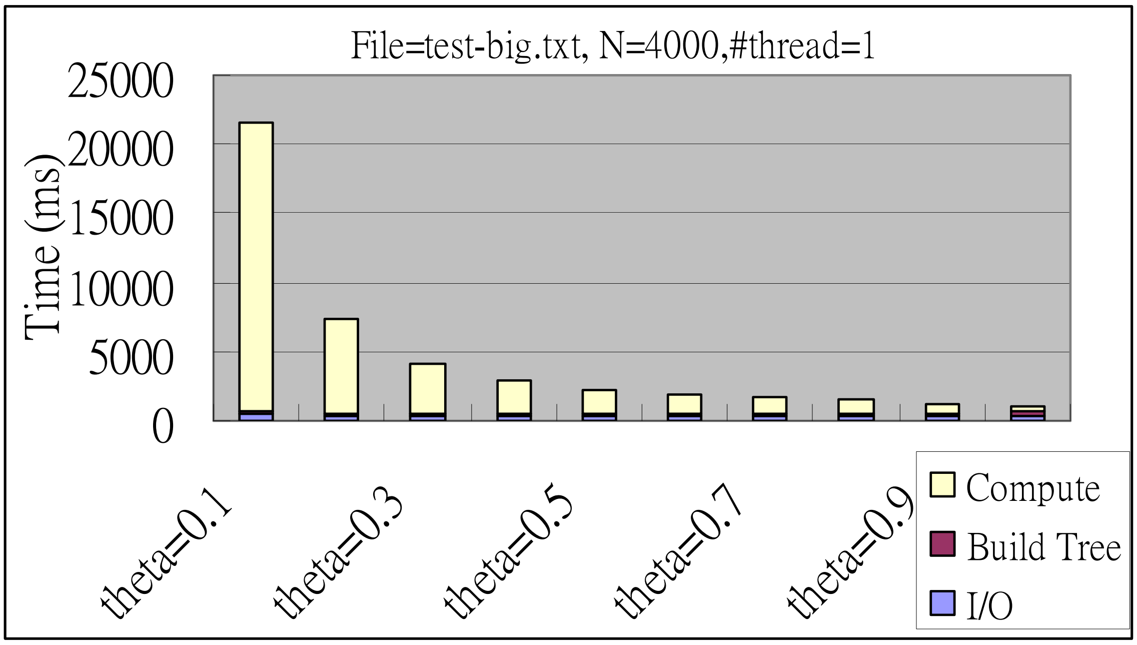
Compared with sequential code, Sequential code running 10 iterations with 1 thread takes: 39.147 sec when n =10000

Data Size	**10000**	**20000**	**30000**	**40000**	**50000**
Pthread	3.997	15.714	35.488	62.679	98.413
OpenMP	3.998	15.926	35.492	63.639	99.087
B-H theta=1	0.19	0.387	0.524	0.743	0.91
B-H theta=0.1	1.444	2.423	4.12	5.704	7.727

Analysis: When fixed #thread and iteration, the case is also the same with previous case, all the consuming time complexity is dominated by how many nodes can be filtered, when traversal the tree. So based on test-big.txt file, there are many data is viewed as single mass node. Hence, it causes B-H=1.0 and B-H=0.1 has impressively result. However, if the data content has been changed, there are have different result or speed-up factors. Anyway the total spending time in Pthread and OpenMp are extremely higher, because it should go through each nodes to finish Force calculation and get new location and speedup factors.

C、 Barnes-Hut Algorithm

Input parameter: XXX.out 1 1 5 0.01 test-big.txt 0.5 disable
N=4000, #thread=1, iteration=5



theta	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
I/O	434	411	412	412	412	412	410	410	412	411
Build Tree	181	181	182	181	181	181	181	181	181	191
Compute	20900	6810	3598	2361	1707	1335	1081	880	659	479

Analysis:

Based on Barnes-Hut Algorithm, if theta is zero, it likes brute forces method to traverse each nodes. However, when theta is larger than zero, it will filter some data whose far enough to be viewed as single mass. So because of theta value, we do not need to traverse these data individually, just directly calculated its total mass to get forces result, which may save a lot time complexity when calculating mass's force. In the diagrams, we find out that in the tset-big.file, when theta is 0.3, it filter the most of data. However, the rest of data, when we add theta 0.1, the data will be filter proportionally after adding more 0.1 theta value until theta is 1. It 's because we have filtered the most of part of data, and the rest of data have averagely distributed form 0.4~1.

In generally, we know the build tree action will take log(N), and the calculation action will take N log(N) , because of going through each nodes. However, when the theta is larger, there are more likely to finding out some internal node which is far enough from current calculated node, so the nodes will save more time to calculate its force, by utilizing internal node’s result directly without traverse its sub-nodes. Finally, we observe the diagram. We discover that I/O and build tree will not take too much time. However, when theta become smaller, the compute time has been reduced largely, which benefits form avoiding going thorough its sub-nodes, as we mentioned before.

D、 Other experiments to compare the performance

1. Compare create thread many time with the reusable thread, which created once only.

Input parameter: XXX.out 10 1 400 1 test-3.txt 1 disable

N= 5172, #Thread=10

iteration	**200**	**300**	**400**
Create Pthread thread each phase (sec)	21.432	31.703	42.328
Create pthread once (sec)	21.358	31.554	41.977

Analysis:

How disappointed with the idea. When we re-created thread each phase, the program even gets nice time-consuming time, which means the create action in Pthread is very soon and without taking too much time. And why the reusable thread will be slower, which may be due to that our main programs should take extra effort to wait to be signified by running thread (The communication time between thread and main programs). It may be the reason why it has worse execution time.

4、 Experience

A、 What I have learned for this assignment?

In shared memory mechanism, I can use pthread and openmp to parallelize the programs to reduce its execution time. And learning how to manipulate mutex_lock and condition variable is useful skills for me, because the current systems is heavy-loading and need to solve task as soon as possible, so using the tools I can make the programs solve job quickly.

B、 What difficulty did I encounter when implementing this assignment?

1. Because of not yet study in senior school, I take the most of my time to validate if my formula of nbody is right or not. And understand why the formula can get the correct result. However, it is good chance for me to learn something about physics.
2. In roller coaster problem. The difficult I met is how to control the process order, although I can signify the process for waiting to running, the pthread, however, cannot guarantee that once you called signify to process, you can get mutex_lock right away. So I take many…many time to think how to solve them. The following are my idea to solve it.

   a. Round robin to signify the process, until the process ID is match the ID that I want to assign. (For example, ID is in the first index in my Queue.) With the **same condition variable** for **CAR and Passenger.**

(PS. The mechanism is like send and receive, the wait function in CAR is cirtical, because it seems to force the car process to release mutex_lock right away, and the passenger can get it, once passengers have been waiting there.)

[Car]

```
  while( isMathchProcessID ){
        pthread_cond_signal(&queueReset_cv);
        pthread_cond_wait(&queueReset_Result_cv, &lock_mutex);
 }
```

[Passenger]

```
while (isMathchProcessID ){
   pthread_cond_wait(&queueReset_cv, &lock_mutex);
        if (threadInfo->id == waitingQueue[indexID]){
                     isMathchProcessID= false;
         }
        pthread_cond_signal(&queueReset_Result_cv);
 }
```

b. Solve it with different condition variable for different passengers. Because of different condition variable, I can signify or wake up different passengers by call the specific condition variable.

3. Build Quadrant Tree. It's a good chance for me to learn how to build the tree and traversal the tree with recursive call or for loop implementation. When I first write it, I cannot find out the stop condition for my cursive function, because of not knowing which level my point will be located it. However, after search for the google, I find out the solution, and don't care which level on tree will stop, just pay attention, when node traverses in new tree node, and the node should stop it to traverse more, and insert it into tree.

C、 Any feedback

Just my personal suggestion only, forgive me, If the course could release sequential Nbody calculation, it will be useful for someone like me, because of never learning about physics, so for a longer while I always .. always…always draw wrong diagram, when compared with the version TA released, which is a huge nightmare for me and make me frustrated, because I don’t know where I calculate them wrongly. After asking TA about its formula, I can draw it correctly. I really appreciate TA’s help also.