**N-body实验报告**

黄 欢

（计算机科学与技术系 计33 2013011331）

1. **实验要求与内容**

Problem Description

Given N celestial bodies with the same mass m, each has its initial position (x , y) and velocity (vx , vy), assigned in input files. In this assignment, you need to simulate their movement T times by using t seconds between each step.

Input Format

**./a.out #threads m T t FILE θ enable/disable xmin ymin length Length**

- a.out: your execution file

- #threads: number of threads

- m: mass of each celestial body (float)

- T: number of steps

- t: time between each step (float)

- FILE: your input file

- θ: use in Barnes-Hut Algorithm

- enable/disable: enable or disable Xwindow

- xmin, ymin: the lower left coordinate of Xwindow

- length: the length of the coordinate axis

- Length: the Length of Window’s side (will be 10n times of length).

Notice that you don’t need to read the last 4 argument values if Xwindow is disabled.

For input file, first line contains a number N (1 ≤ N ≤ 107) as the number of celestial bodies. The following lines specifies the initial position (x , y) and velocity (vx , vy) of each body. We will provide 4 input files, you can copy from /home/ppc2015 directory (the input parameters is for your reference) :

test1.txt (N=945): ./a.out 12 1 1000 0.5 test1.txt 0 enable -0.3 -0.3 0.6 600

test2.txt (N=81921): ./a.out 12 1 1000 0.01 test2.txt 1 enable 0 -20 1 500

test3.txt (N=1923840): ./a.out 12 1 1000 0.01 test3.txt 1 enable 0 0 1 500

test4.txt (N=7283942): ./a.out 12 1 1000 0.001 test4.txt 1 enable 0 0 1 500

1. **实验方法与实现**
2. **Sequential Code**



经测试，用暴力线性的办法只能够将test1的图像较快地展示出来，当body数足够多时，这种方法所需的时间便是我们不能够等待的，所以需要我们采用并行程序来解决大量数据的相同运算问题。

1. **建树**

可以将所有的body按照空间建成一颗四叉树，便于寻找和简化计算。建树的方法如下：

1. If node x does not contain a body, put the new body b here.
2. If node x is an internal node, update the center-of-mass and total mass of x. Recursively insert the body b in the appropriate quadrant.
3. If node x is an external node, say containing a body named c, then there are two bodies b and c in the same region. Subdivide the region further by creating four children. Then, recursively insert both b and c into the appropriate quadrant(s). Since b and c may still end up in the same quadrant, there may be several subdivisions during a single insertion. Finally, update the center-of-mass and total mass of x.

由此容易写出建树部分的代码。

建树部分每个节点主要需要记录下以下参数：

int \*father; //父亲

int \*\*child; //孩子（二维数组）

int \*cnt; //该节点下的body数

double \*Sm; //该节点下所有body的质量和

double \*Ax, \*Ay; //该节点下所有body质心的x和y值

double \*X1, \*X2; //该节点表示的x坐标的范围

double \*Y1, \*Y2; //该节点表示的y坐标的范围

1. **计算力的总和**

可以由所见的树计算对每一个body万有引力的总和。方法如下：



此处算法的一大巧妙之处是，如果某个节点的质心离得足够远，就可以把该节点下的所有body看成一个点，算总作用力，即可简化很多运算，并且误差也不太大，可以忽略。

由于计算力的总和占用了大部分的时间，所以并行计算主要是在这个部分。

1. **Pthread思想**

设置一个锁，所有的thread通过抢这把锁来抢自己的body，并查找树计算该body所受的力的总和，其中body的范围为[0, N)。代码如下：

while (nowN < N) {

pthread\_mutex\_lock(&count\_mutex);

i = nowN++;

pthread\_mutex\_unlock(&count\_mutex);

//calcForce

int pi = 1;

stack[1] = 0;

while(pi>0) {

if (cnt[stack[pi]] == 0) {

pi--;

continue;

}

if (cnt[stack[pi]] == 1) {

//calc…

pi--;

continue;

}

if (cnt[stack[pi]] > 1) {

//calc…

}

}

}

pthread\_exit(NULL);

1. **Cuda思想**

//gpu

int THREADSPERBLOCK = 512;

int BLOCKSPERGRID = (int)ceil((double)N / 512);

//…

calc<<<BLOCKSPERGRID, THREADSPERBLOCK>>>(m, G, theta, d\_x, d\_y, d\_Fx, d\_Fy, d\_child, d\_cnt, d\_Sm, d\_Ax, d\_Ay, d\_X1, d\_X2);

设置合适的BlockPerGrid和ThreadPerBlock，每一个thread跑一个body，利用总thread数的优势，来达到高速并行的目的，减少运算的时间。

1. **一些细节上的考虑**
2. 如果两个body足够接近，其引力必定非常大。为了避免这种情况，给出特判：若两个body足够接近，则不算两者的作用力。
3. 如果body的x, y值超出了自己设定的边框，则将其坐标关于边框做对称操作。
4. 若许多body的距离都足够小（小于10的-10次方），则将其看做一个整体，用质心进行相关作用力的计算。
5. 为了方便将pthread改为cuda，程序中将所有的递归都改为了迭代，其中计算万有引力时，利用了堆栈来存储，直到堆栈为空（采用指针），计算结束。
6. 需要注意的是，要把所有body的Fx和Fy算出来之后，才能更新vx, vy, x, y的相关值。
7. 最终malloc的空间需要free掉，程序结束。
8. **画图**

画图采用xwindow和xlib库，注意范围和坐标转换，将body图形展示出来。

程序中bool inWindow(int i)用于判断body是否在显示的框内。

每一个body以其(x, y)为中心，2为边长画正方形，代码如下：

XFillRectangle (display, window, gc, (x[i] - xmin) / length \* Length - 1, (y[i] - ymin) / length \* Length - 1, 2, 2);

1. **Body的相关变量**

|  |  |
| --- | --- |
| 变量名 | 意义 |
| x | **X坐标** |
| y | **Y坐标** |
| Vx | **X轴方向上的速度** |
| Vy | **Y轴方向上的速度** |
| Fx | **X轴方向上的力** |
| Fy | **Y轴方向上的力** |
| G | **万有引力常量** |
| m | **质量** |

1. **时间**

在pthread中，运用clock\_gettime求时间，代码如下：

#include <time.h> //-lrt

//time

struct timespec time1 = {0, 0};

struct timespec time2 = {0, 0};

clock\_gettime(CLOCK\_MONOTONIC, &time1);

//do something

clock\_gettime(CLOCK\_MONOTONIC, &time2);

printf(" CLOCK\_TIME: %lf ms\n", (time2.tv\_sec - time1.tv\_sec) \* 1000 + (time2.tv\_nsec - time1.tv\_nsec) / 1e6);

在cuda中，运用cudaEvent求时间，代码如下：

#include <cuda\_runtime.h>

//time

cudaEvent\_t begin, stop;

cudaEventCreate(&begin);

cudaEventCreate(&stop);

//Get begin time

cudaEventRecord(begin, 0);

//do something on GPU

calc<<<BLOCKSPERGRID, THREADSPERBLOCK>>>(m, G, theta, d\_x, d\_y, d\_Fx, d\_Fy, d\_child, d\_cnt, d\_Sm, d\_Ax, d\_Ay, d\_X1, d\_X2);

cudaThreadSynchronize();

//Get stop time

cudaEventRecord(stop, 0);

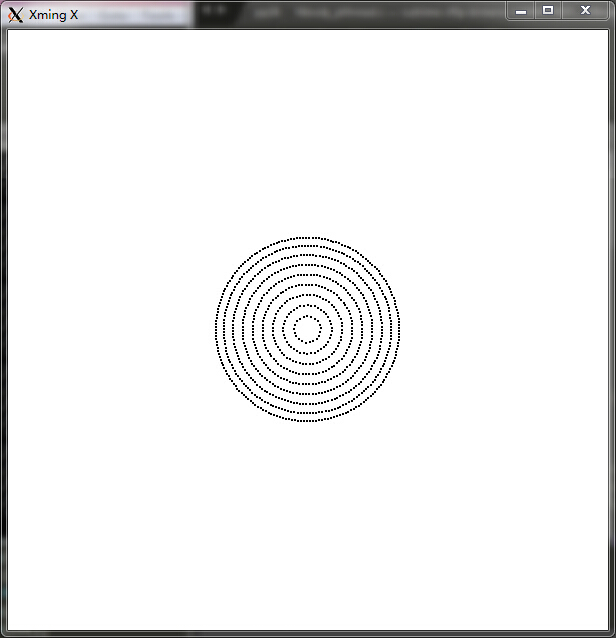
cudaEventSynchronize(stop);

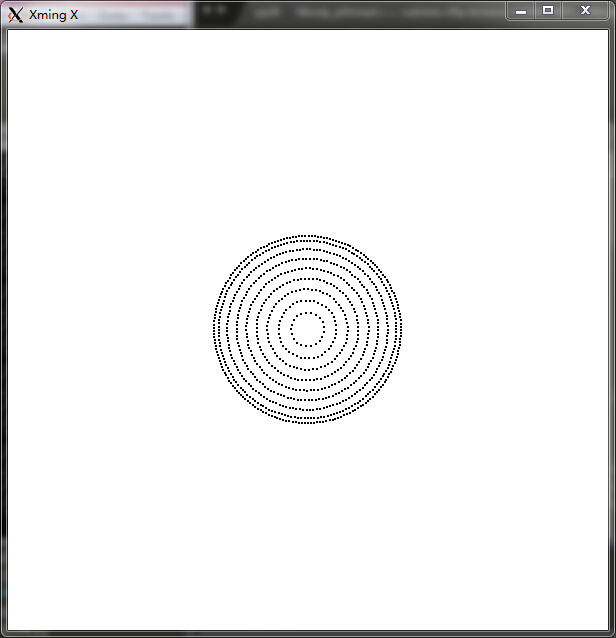
float elapsedTime;

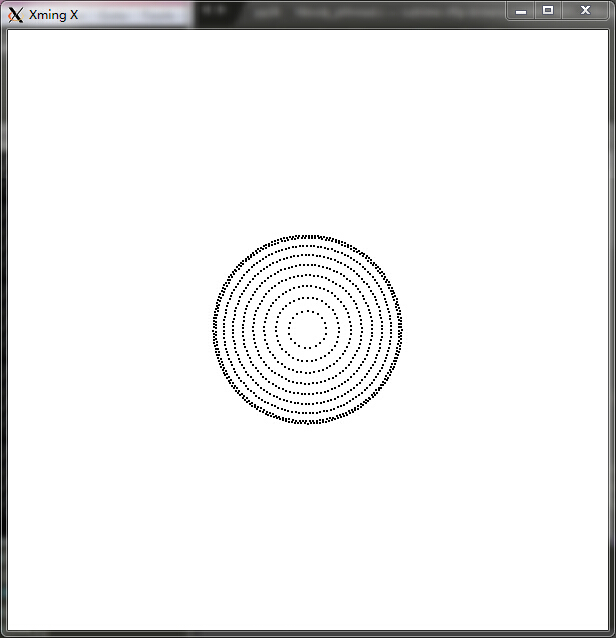
cudaEventElapsedTime(&elapsedTime, begin, stop);

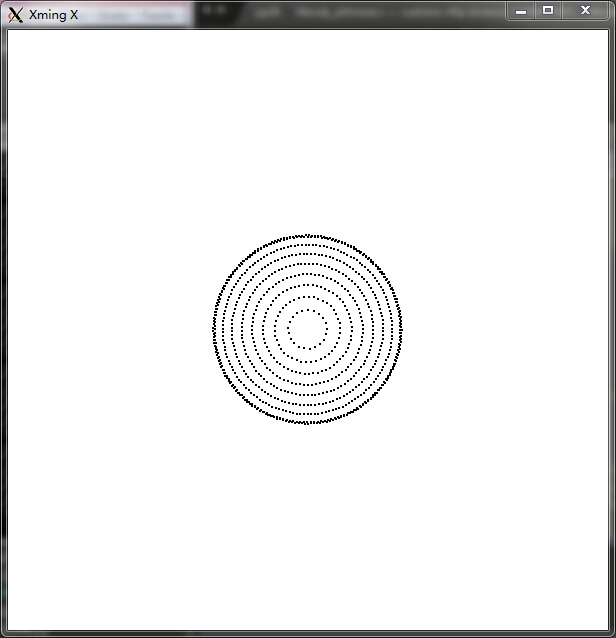
printf(" Execution Time on GPU: %3.20f s\n", elapsedTime/1000);

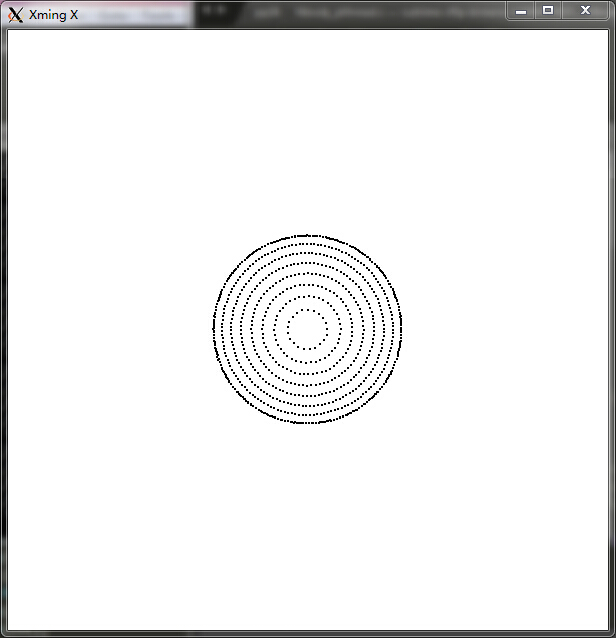
1. **实现功能效果图与对比**
2. **图像（以test1为例，时间顺序）**

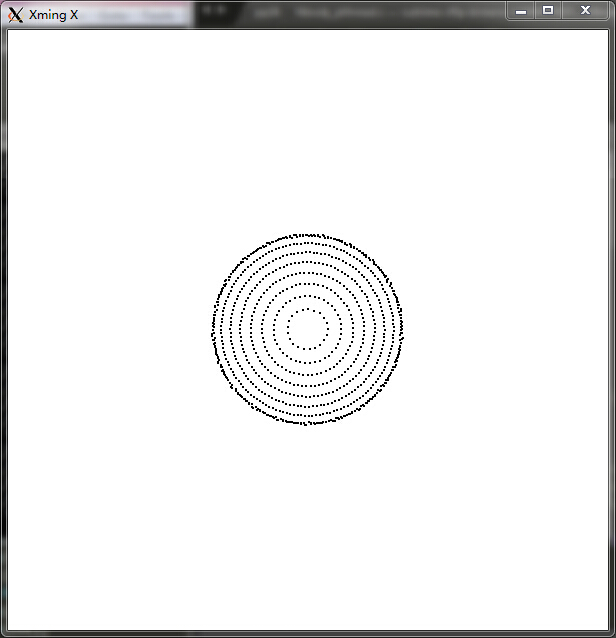


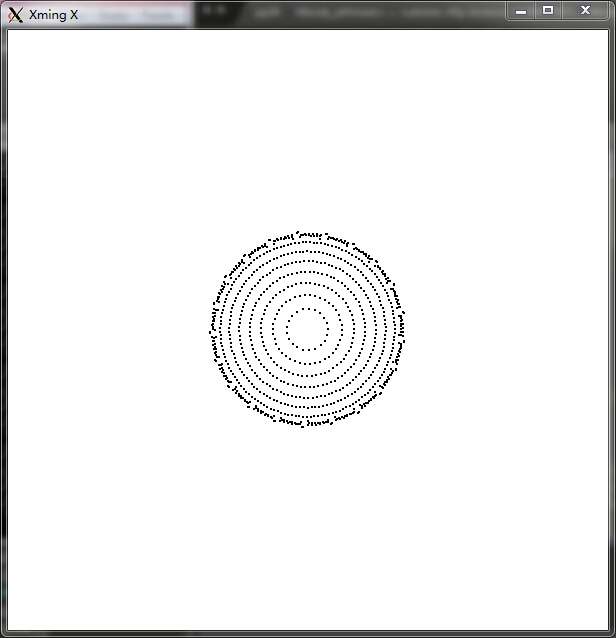


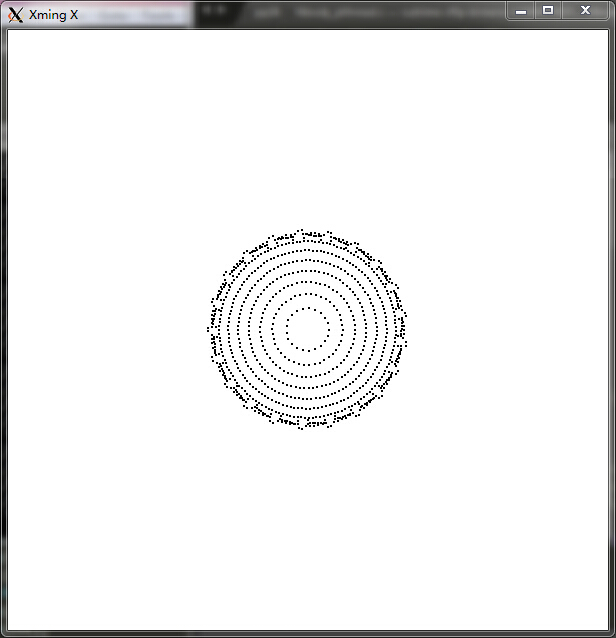


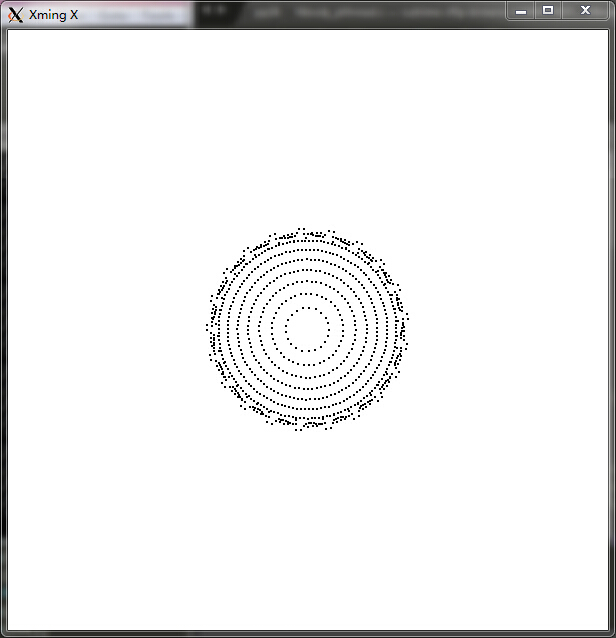


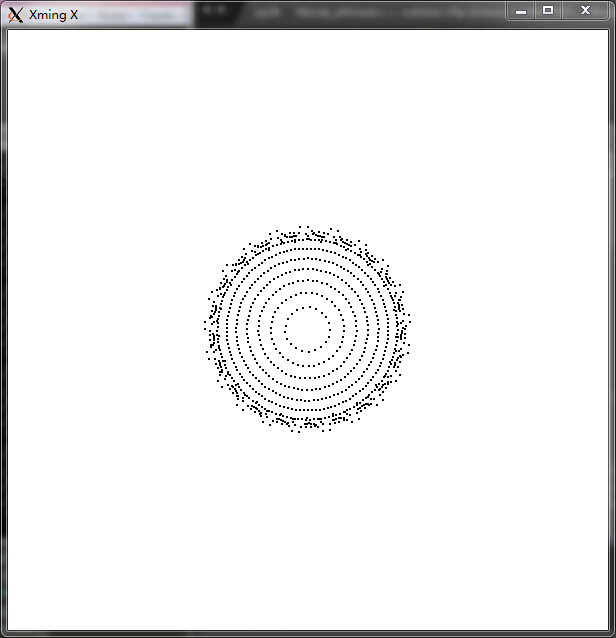


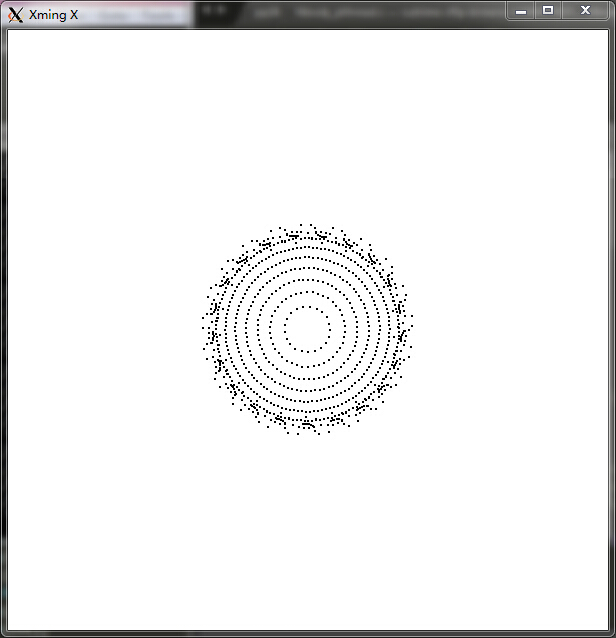


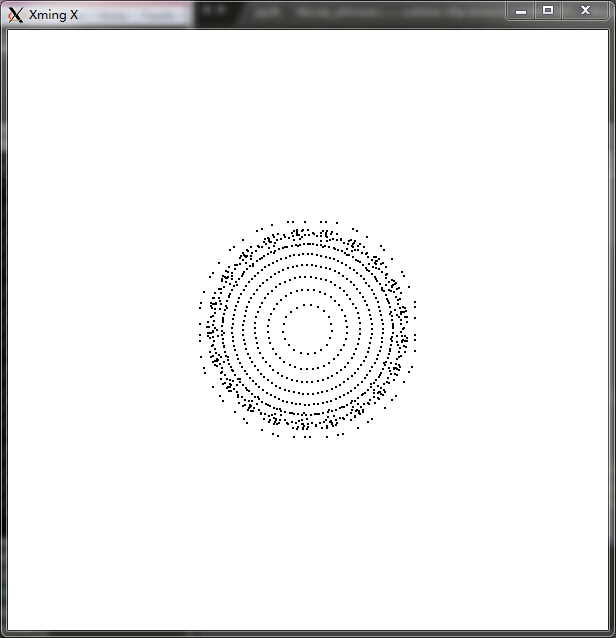


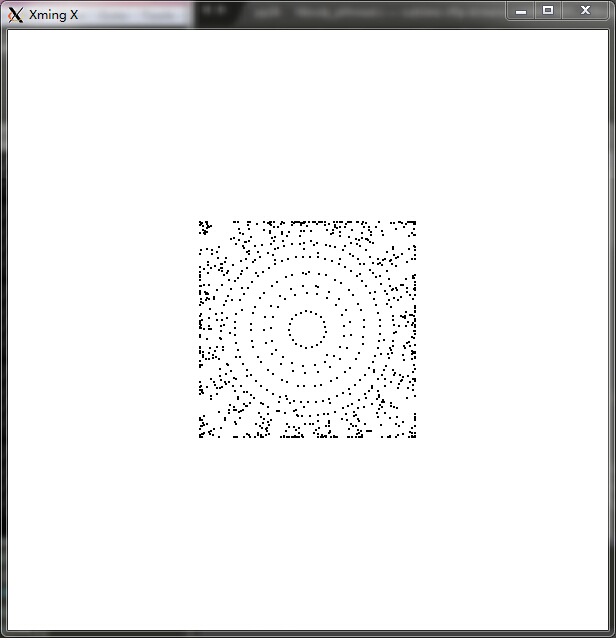












1. **运行时间及对比**

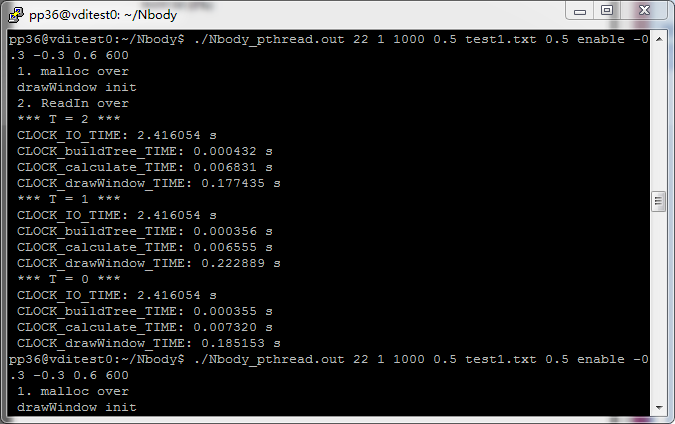
由表格后面截图整理如下：

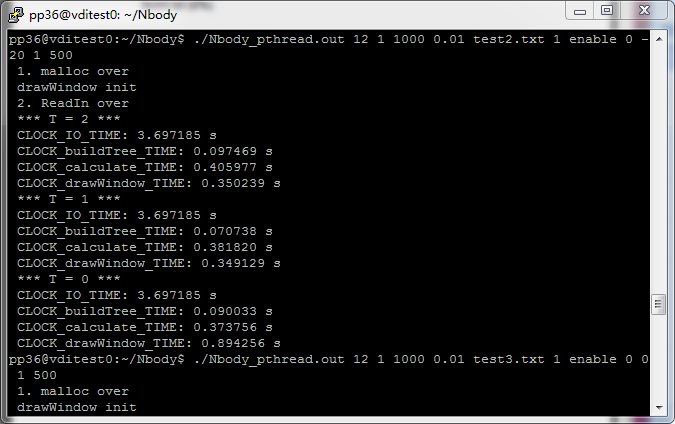
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| For one time  单位：s | IO-time | BuildTree time | Computing time | Drawing time |
| Pthread Text1 | 2.416054 | 0.000355 | 0.007360 | 0.351281 |
| Pthread Text2 | 3.697185 | 0.090033 | 0.381820 | 0.350239 |
| Pthread Text3 | 11.959448 | 3.827989 | 9.405483 | 1.458264 |
| Pthread Text4 | 30.765485 | 18.031463 | 26.641121 | 1.233580 |
| Cuda Text1 | 2.407322 | 0.000410 | 0.004217599 | 0.266860 |
| Cuda Text2 | 2.777865 | 0.069121 | 0.000010592 | 0.271728 |
| Cuda Text3 | 10.660789 | 2.794378 | 0.000745440 | 0.883880 |
| Cuda Text4 | 30.406189 | 14.009214 | 0.000935008 | 1.427213 |

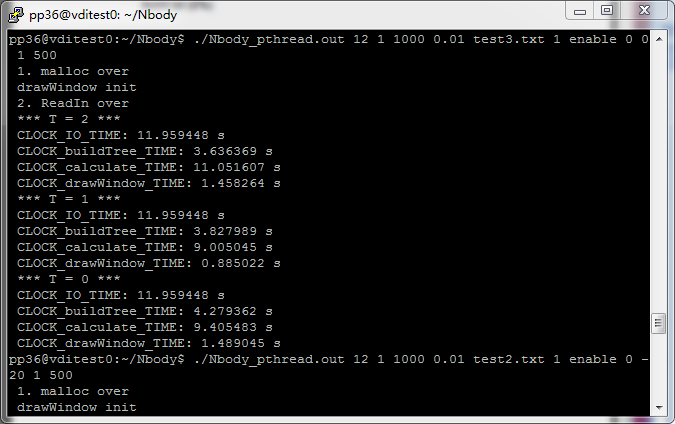
由于运行所需的时间会受到环境因素的影响，每一次的运行结果都不一样，但是相差也不会很多，所以表中数据仍有一定参考价值。

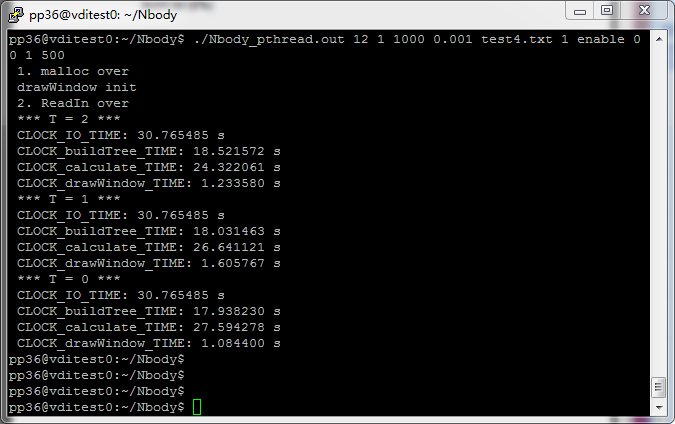
由表中数据可知，随着body数目的增加，各项活动的时间都有增加，符合常理。而cuda与pthread的不同在于计算时间明显减少。

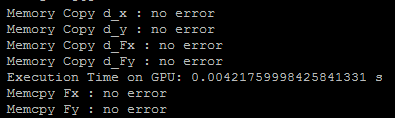
截图如下：

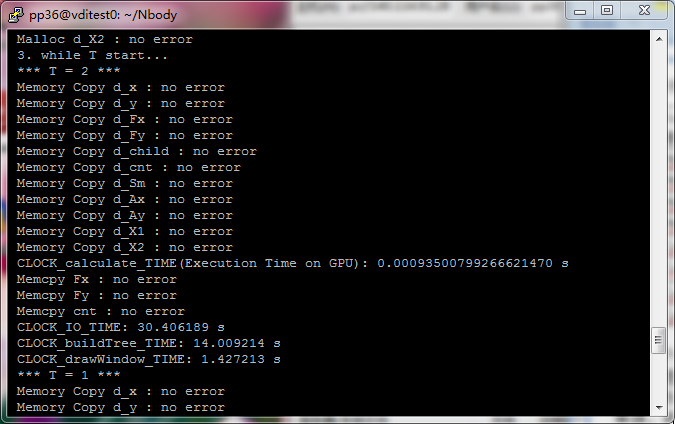












1. **实验实现总结与心得体会**

通过N-body实验，我学习并掌握了N-body算法的核心思想，以及将串行改为并行（pthread和cuda）的一些实现。这个实验很有趣，同时也很考验我们。在不断改进自己的算法和调程序的过程中，我学会了很多知识，提升了代码能力，收获很大。并行程式在实际生活中有着广泛应用，为我们节省了大量时间，相信随着科技的发展，并行程序设计将有更大的突破。

最后，非常感谢老师、助教的辛勤指导，我会继续探索、继续努力！