基于 cuda 编程的等离子体数值模拟

一．前言：

在等离子体模拟中，需要有大规模的计算，特别是计算磁流体力学方程组，需要在空间上进行取点，并进行计算，其相应的算法，很适合进行并行计算，因此采用 gpu 编程可以大大提高程序的运行速度，同时也提高了程序运行的效率。大大降低了等待的时间。

二．原理介绍：

1.磁流体力学方程组

在等离子体中，需要研究磁流体物理方程，磁流体方程如下：

$$\frac{\partial \rho}{\partial t} = -\nabla(\rho \mathbf{u})$$

$$\frac{\partial \mathbf{u}}{\partial t} = -\mathbf{u}\cdot\nabla\mathbf{u} - \rho^{-1}[\nabla p - (\nabla\times\mathbf{B})\times\mathbf{B}]$$

$$\frac{\partial p}{\partial t} = -\mathbf{u}\cdot\nabla p - \gamma p\nabla\cdot\mathbf{u}$$

$$\frac{\partial B}{\partial t} = \nabla\times(\mathbf{u}\times\mathbf{B})$$

假设在绝热情况下：

$$\mathrm{p}\rho^{-\gamma} = \mathrm{C}$$

则方程可以简化为：

$$\frac{\partial \rho}{\partial t} = -\nabla(\rho \mathbf{u})$$

$$\frac{\partial \mathbf{u}}{\partial t} = -\mathbf{u}\cdot\nabla\mathbf{u} - \rho^{-1}[v_s^2\nabla\rho - (\nabla\times\mathbf{B})\times\mathbf{B}]$$

$$\frac{\partial B}{\partial t} = \nabla\times(\mathbf{u}\times\mathbf{B})$$

其中 $\mathrm{v}_s^2 = \frac{p\gamma}{\rho^2}$

并假设为平板位形，在 x 方向上为长为 a 周期边界，宽为 2b，边界为理想导体边界，存在沿 y 轴方向的固定磁场 B0。

设磁场可以改写成为：$\mathrm{B} = \mathrm{B}_0 + \nabla\times A_z$

忽略二阶小量，得到的公式如下：

$$\frac{\partial\rho}{\partial t}=-\rho_0(\frac{\partial v_x}{\partial x}+\frac{\partial v_y}{\partial y})$$

$$\frac{\partial v_x}{\partial t}=-\frac{v_s^2}{\rho_0}\frac{\partial\rho}{\partial x}+(\frac{\partial^2 A}{\partial x^2}+\frac{\partial^2 A}{\partial y^2})\cdot B_0$$

$$\frac{\partial v_y}{\partial t}=-\frac{v_s^2}{\rho_0}\frac{\partial\rho}{\partial y}$$

$$\frac{\partial A}{\partial t}=-B_0\cdot v_x$$

在边界条件，上满足

$$\frac{\partial\rho}{\partial x}|_{x=0,l}=0,\frac{\partial\rho}{\partial y}|_{y=0,l}=0,v_x|_{x=0,l}=0,v_y|_{y=0,l}=0$$

2.计算模型建立：

假设任何小量 h(x,y)，我们在时间和空间上进行取点操作，对于导数可以写成：

$$\frac{\partial f(i,j)}{\partial x}=\frac{f(i,j)-f(i-1,j)}{2dx}$$

$$\frac{\partial^2 f(i,j)}{\partial x^2}=\frac{f(i+1,j)+f(i-1,j)-2f(i,j)}{dx^2}$$

故将上述式子带入方程 得到：

$$\frac{\rho^{n+1}(i,j)-\rho^{n\text{-}1}(i,j)}{dt}=-\rho_0(\frac{v_x^n(i+1,j)-v_x^n(i-1,j)}{dx}+\frac{v_y^n(i,j+1)-v_y^n(i,j-1)}{dx})$$

$$\frac{v_x^{n+1}(i,j)-v_x^{n\text{-}1}(i,j)}{dt}=-\frac{v_s^2}{\rho_0}\frac{\rho^n(i+1,j)-\rho^n(i-1,j)}{dx}+$$

$$2(\frac{2A^n(i,j)-A^n(i+1,j)-A^n(i-1,j)}{dx^2}+\frac{2A^n(i,j)-A^n(i,j+1)-A^n(i,j-1)}{dy^2})\cdot B_0$$

$$\frac{v_y^{n+1}(i,j)-v_y^{n\text{-}1}(i,j)}{dt}=-\frac{v_s^2}{\rho_0}\frac{\rho^n(i,j+1)-\rho^n(i,j-1)}{dx}$$

$$\frac{A^{n+1}(i,j)-A^{n\text{-}1}(i,j)}{dt}=-B_0\cdot v_x^n$$

其中 n 表示 n 时刻，可以看出 n+1 时刻的关系取决于 n-1 时刻状态的量和 n 时刻的量，通过 n-1 时刻和 n 时刻的量可以计算 n+1 时刻的状态。从而我们可以通过迭代计算得到每个时刻的值，从而模拟出方程发展的规律。

三．程序设计：

采用 visual2015 结合 cuda8.0 作为实验平台。对于程序设计，我们采用三个子 kenerl 函数，分别用于计算(calcculate)，复制()，边界条件 设置，并循环启动三个 kernel 函数，A 为（4,nx+2,ny+2）表示第 n 时刻状态，B 为（4,nx,ny）数组代表 n+1 时刻参数，nx 表示在 x 方向上的节点数目，ny 表示在 y 方向的节点数目。其代码如附录所示。采用的是笔记本电脑的显卡，显卡型号为 gtx960M，其运行的参数如下图所示：

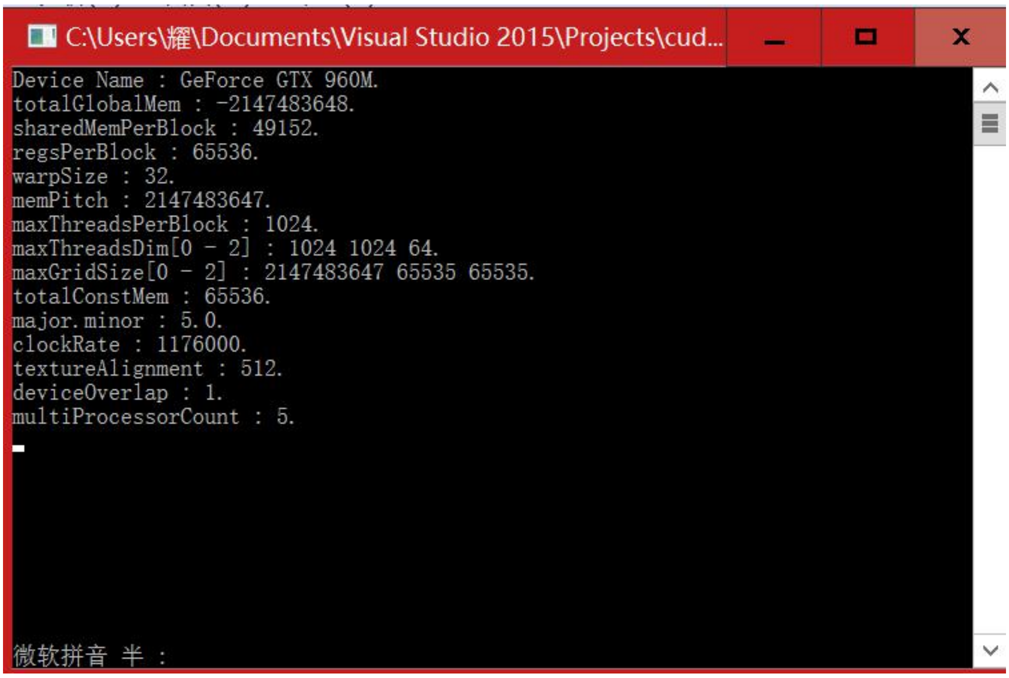

图 1.gpu 状态。

采用的三维的 grid 为（nx），block 维度为（ny）。通过 gpu 和 cpu 计算，计算其误差和运行时间。我们可以看到两种计算方式，计算结果完全一致。

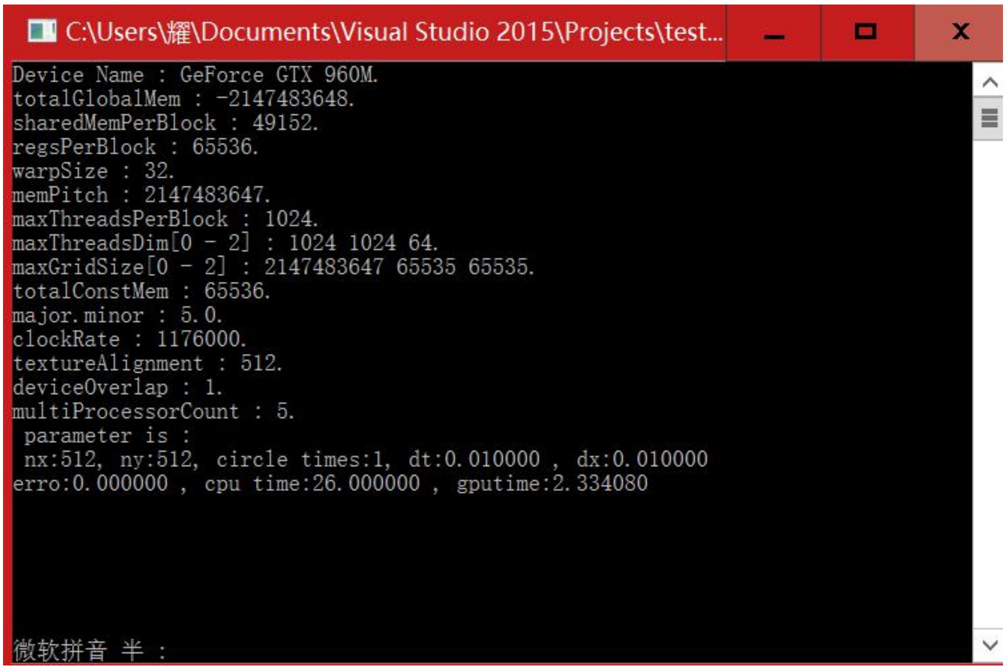

图 2.运行结果

四．实验结果：

1.物理图像的验证

在物理上，当产生一个正弦波扰动时，那么会产生相应的波，如阿尔芬波和声波，其在物理上表现为，相应物理会产生周期性变化。

我们设置：nx=ny=256,dt=0.01,dx=dy=0.01。当输入扰动的波形为：$vx = \sin(2\pi x)$，经过

50000dt 时间后，观察压强和磁场在（nx/2,ny/2）的幅度随时间的变化，可以得到如下图所示的结果。即产生了

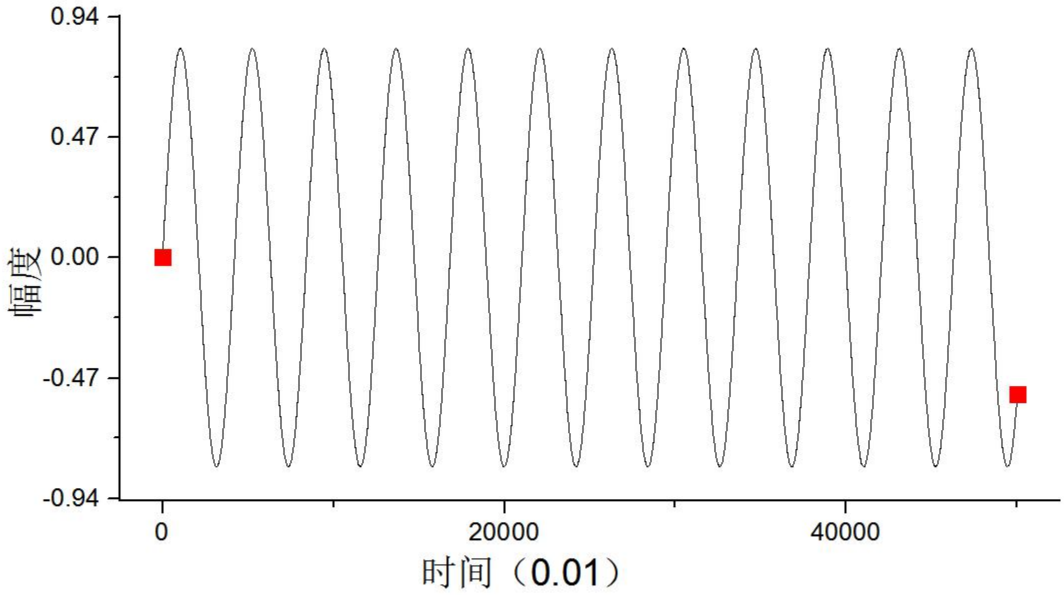

图 3.磁声波

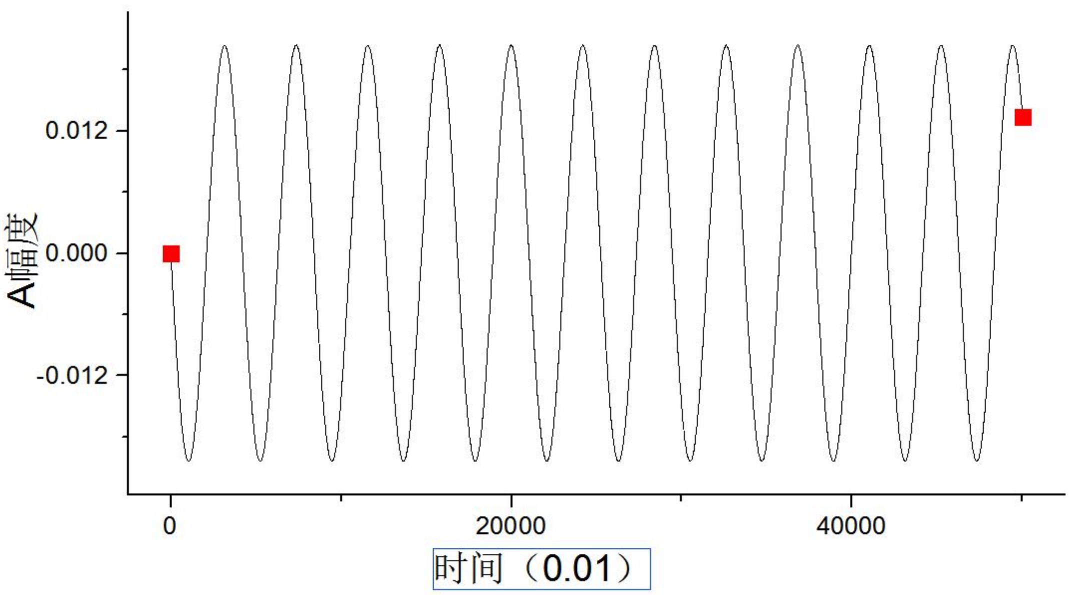

图 4.阿尔芬波

可以看出磁场和压强在此点都随时间变化而产生周期变化，即产生了阿尔芬波和声波。结果与理论分析结果相同。

2. 采用不同的 nx，ny 对于运行时间的影响

设定计算次数 t=100，当 nx 和 ny 取不同值时，gpu 和 cpu 运行时间的对比。

	Nx=ny=32	64	128	256	512
Gpu（ms）	23	39	39	88	283
Cpu（ms）	10	37	156	706	2557

表一.nx=ny 取值不同时计算时间

Nx=128	Nx=ny=32	64	128	256	512
Gpu（ms）	27	32	35	49	87
Cpu（ms）	29	80	126	291	586

表二.nx=128,ny 取值不同时计算值

可以看出当 nx=ny<64 时，cpu 花费的时间显著少于 gpu 时间，当 nx=ny>64 时，gpu 花费的时间小于 gpu 的时间，且随着 nx 增加，其加速效果愈加明显。

Function Name	Grid Dimensions	Block Dimensions	Start Time (μs)	Duration (μs)
calcculate	{512, 1, 1}	{512, 1, 1}	299,598.906	1,446.080
updatevalues	{512, 1, 1}	{512, 1, 1}	369,472.634	692.096
initialboundarycondition	{1, 1, 1}	{512, 1, 1}	432,369.210	20.800

图 5.nsight 监测各个核函数表现

通过 Nsight 测量在一个周期内，各个函数的表现，观发现计算的核函数花费时间最长达到了 1446us,因此可以对其多进行优化。简单的赋值也占用了比较长的时间。

四．结论：

当节点数目比较大时，gpu 多 block 多核的运行能够成倍的减少运行时间，在此情况下采用 cuda 编程可以极大地提高程序的运行速度，然而在节点数比较小的情况下，运行效率比不上 cpu，在此情况下采用 cpu 编程，更加适合。

附录：

程序代码：

```
#include "cuda_runtime.h"
#include "device_launch_parameters.h"
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include  "device_functions.h"
#include <fstream>
using namespace std;
const int nx = 512;
const int ny = 512;
const float pi = 3.1415926;
const int times = 1;  //total circle time .循环次数
const float dt = 0.01;//时间间隔
const float dx = 0.1;//
float a[4][nx+2][ny+2];//represent the value of all variables in  time=n 代表 n 时刻的值
float b[4][nx][ny];   //represent the value of all variables in  time=n+1 代表 n+1 时刻的值
float bp[4][nx][ny];  //represent the value of all variables in  time=n-1 代表 n-1 时刻的值
float ca[4][nx+2][ny+2]; //use to store the a data from gpu
float br[4][nx][ny];  //use to store the b data from gpu
float rt[times][4];   //store the history of one point in the metrix 储存某个点的历史参数值
                        //use gpu compute the b in time n+1;计算 n+1 时刻的值
__global__ void calcculate(float *a, float *b,float *c)
{
    int width = ny;
    int lenth = nx * ny;
    int awidth = ny+2;
    int alenth = (nx+2)* (ny+2);
    float dt = 0.01;
    float dx = 0.1;
    //p
    int x = blockIdx.x;
    int y = threadIdx.x;
    int bl, bc, br, bu, bd, al, ar, au, ad, ac;
    bc = x*width + (y);
    ad = (x + 1)*awidth + (y);
    au = (x + 1)*awidth + (y + 2);
    ar = (x + 2)*awidth + (y + 1);
    al = x*awidth + (y + 1);
```

```
    ac = (x + 1)*awidth + (y + 1);

    bc = x*width + (y);
    ad = (x + 1)*awidth + (y);
    au = (x + 1)*awidth + (y + 2);
    ar = (x + 2)*awidth + (y + 1);
    al = x*awidth + (y + 1);
    ac = (x + 1)*awidth + (y + 1);
    b[bc] = -(a[ alenth + ar] - a[alenth + al])*dt/dx
        -(a[2 * alenth + au] - a[2 * alenth + ad])*dt/dx
        +c[bc];
    b[lenth + bc] = -(a[ar] - a[al])*dt / dx
        - 2*(a[3 * alenth + ar] + a[3 * alenth + au] + a[3 * alenth + ad] + a[3 * alenth
+ al] - 4 * a[3 * alenth + ac])*dt / (dx*dx)
        + c[lenth + bc];
    b[2 * lenth + bc] = -(a[au] - a[ad])*dt / ( dx) + c[2 * lenth + bc];
    b[3 * lenth + bc] = -a[1 * alenth + ac] * dt + c[3 * lenth + bc];
    __syncthreads();
}
//update all values 更新各个状态量
__global__ void  updatevalues(float *a, float  *b,float *c)
{
    int width = ny;
    int lenth = nx * ny;
    int awidth = ny + 2;
    int alenth = (nx + 2)* (ny + 2);
    int x = blockIdx.x;
    int y = threadIdx.x;
    int bl, bc, br, bu, bd, al, ar, au, ad, ac;
    bc = x*width + y;
    ac = (x + 1)*awidth + y + 1;
    c[bc] = a[ac];
    c[1 * lenth + bc] = a[1 * alenth + ac];
    c[2 * lenth + bc] = a[2 * alenth + ac];
    c[3 * lenth + bc] = a[3 * alenth + ac];
    __syncthreads();
    a[ac] = b[bc];
    a[1 * alenth + ac] = b[1 * lenth + bc];
    a[2 * alenth + ac] = b[2 * lenth + bc];
    a[3 * alenth + ac] = b[3 * lenth + bc];
    __syncthreads();
}
//deal with the boundary condition ,边界条件进行处理
__global__ void initialboundarycondition(float *a, float *b)
```

```
{
    int m = threadIdx.x;

    int width = ny;
    int lenth = nx * ny;
    int awidth = ny + 2;
    int alenth = (nx + 2)* (ny + 2);
    int ml, mr, mu, md, nl, nr, nu, nd;

    md = (m + 1)*awidth;//[m+1][0]
    mu = (m + 2)*awidth - 1;//[m+1][ny+1]
    ml = m + 1;//[0][m+1]
    mr = (awidth - 1)*awidth + m + 1;//[nx+1][m+1]
    nd = m*width;//[m][0]
    nu = (m + 1)*width - 1;//[m][ny-1]
    nl = m;//[0][m]
    nr = (width - 1)*width + m;//[nx-1][m]
    //p
    a[ml] = b[nl];
    a[mr] = b[nr];
    a[mu] = b[nu];
    a[md] = b[nd];

    a[1 * alenth + ml] = 0;
    a[1 * alenth + mr] = 0;
    a[1 * alenth + mu] = b[1 * lenth + nu];
    a[1 * alenth + md] = b[1 * lenth + nd];

    a[2 * alenth + mu] = 0;
    a[2 * alenth + md] = 0;
    a[2 * alenth + ml] = b[2 * lenth + nl];
    a[2 * alenth + +mr] = b[2 * lenth + nr];

    a[3 * alenth + ml] = b[3 * lenth + nl];
    a[3 * alenth + +mr] = b[3 * lenth + nr];
    a[3 * alenth + mu] = b[3 * lenth + nu];
    a[3 * alenth + md] = b[3 * lenth + nd];

    __syncthreads();
}

void randefloat(float *a, unsigned int size) //generate the random seris of a array;随机产生一组随机数
{
```

```
    for (int i = 0; i < size; i++)
    {
        *(a + i) = (float)rand();
    }
}
void printDeviceProp(int divice_no = 0) //to display information about gpu 展示 gpu 信息
{
    cudaDeviceProp prop;
    cudaGetDeviceProperties(&prop, divice_no);
    printf("Device Name : %s.\n", prop.name);
    printf("totalGlobalMem : %d.\n", prop.totalGlobalMem);
    printf("sharedMemPerBlock : %d.\n", prop.sharedMemPerBlock);
    printf("regsPerBlock : %d.\n", prop.regsPerBlock);
    printf("warpSize : %d.\n", prop.warpSize);
    printf("memPitch : %d.\n", prop.memPitch);
    printf("maxThreadsPerBlock : %d.\n", prop.maxThreadsPerBlock);
    printf("maxThreadsDim[0 - 2] : %d %d %d.\n", prop.maxThreadsDim[0],
prop.maxThreadsDim[1], prop.maxThreadsDim[2]);
    printf("maxGridSize[0 - 2] : %d %d %d.\n", prop.maxGridSize[0], prop.maxGridSize[1],
prop.maxGridSize[2]);
    printf("totalConstMem : %d.\n", prop.totalConstMem);
    printf("major.minor : %d.%d.\n", prop.major, prop.minor);
    printf("clockRate : %d.\n", prop.clockRate);
    printf("textureAlignment : %d.\n", prop.textureAlignment);
    printf("deviceOverlap : %d.\n", prop.deviceOverlap);
    printf("multiProcessorCount : %d.\n", prop.multiProcessorCount);

}
//use array a and array b to generate a  Metrix like c=aXb 用两个数组产生一个矩阵 C=A X B
void initialvalue(float *a, float *b, float *c, unsigned int rows, int  cols)
{
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            a[i*rows + j] = b[i] * c[j];
        }
    }
}
//use x as a  axis array to generate a array b use function pfun ,like b=pfun(x) 利用数
轴 x 和函数 pfun 产生数组 b,即 b=pfunc(x)
void initialfunc(float *x, float *b, int size, float(*pfunc)(float))
{
    for (int i = 0; i < size; i++)
```

```
    {
        float tempx = x[i];
        b[i] = pfunc(tempx);
    }
}
//set array a all values to zero ; 将数组所有的值设为0.
void zeros(float *a, int size)
{
    for (int i = 0; i < size; i++)
    {
        a[i] = 0;
    }

}
//use cpu to solve equation 利用cpu进行计算
void cpucompute()
{
    int width = ny;
    int lenth = nx * ny;
    int awidth = ny + 2;
    int alenth = (nx + 2)* (ny + 2);


    for (int i = 0; i < times; i++)
    {
        //p
        for (int x = 0; x < nx; x++)
        {
            for (int y = 0; y < ny; y++)
            {
                b[0][x][y] = -(a[1][x+2][y+1] - a[1][x][y + 1])*dt / ( dx)
                    -(a[1][x+1][y+2]-a[1][x+1][y])*dt /(dx)
                    +bp[0][x][y];
                b[1][x][y] = -(a[0][x + 2][y + 1] - a[0][x][y + 1])*dt/ (dx)
                    -2*(a[3][x+2][y+1]+a[3][x][y+1] + a[3][x+1][y] + a[3][x+1][y+2]-4*
a[3][x+1][y+1])*dt/(dx*dx)
                    + bp[1][x][y];
                b[2][x][y] = -(a[0][x + 1][y + 2] - a[0][x + 1][y])*dt / dx + bp[2][x][y];
                b[3][x][y] = -a[1][x + 1][y + 1] * dt + bp[3][x][y];

            }
        }

        for (int x = 0; x < nx; x++)
```

```
        {
            for (int y = 0; y < ny; y++)
            {
                bp[0][x][y] = a[0][x + 1][y + 1];
                bp[1][x][y] = a[1][x + 1][y + 1];
                bp[2][x][y] = a[2][x + 1][y+ 1];
                bp[3][x][y] = a[3][x + 1][y + 1];
                a[0][x + 1][y + 1] = b[0][x][y];
                a[1][x + 1][y + 1] = b[1][x][y];
                a[2][x + 1][y+ 1] = b[2][x][y];
                a[3][x + 1][y + 1] = b[3][x][y];
            }
        }


        for (int m = 0; m < nx; m++)
        {
            a[0][0][m+1] = b[0][0][m];
            a[0][nx + 1][m + 1] = b[0][nx-1][m];
            a[0][m + 1][ny + 1] = b[0][m][ny-1];
            a[0][m + 1][0] = b[0][m][0];

            a[1][0][m + 1] = 0;
            a[1][nx + 1][m + 1] = 0;
            a[1][m + 1][ny+1] = b[1][m][ny-1];
            a[1][m + 1][0] = b[1][m][0];

            a[2][0][m + 1] = b[2][0][m];
            a[2][nx + 1][m + 1] = b[2][nx-1][m];
            a[2][m + 1][ny+1] = 0;
            a[2][m + 1][0] = 0;

            a[3][0][m + 1] = b[3][0][m];
            a[3][nx + 1][m + 1] = b[3][nx-1][m];
            a[3][m + 1][ny+1] = b[3][m][ny-1];
            a[3][m + 1][0] = b[3][m][0];
        }
        int pointx = 256;
        int pointy = 256;
        rt[i][0] = a[0][pointx][pointy];
        rt[i][1] = a[1][pointx][pointy];
        rt[i][2] = a[2][pointx][pointy];
        rt[i][3] = a[3][pointx][pointy];
    }
```

```
}
//compare two results,calcutate total erro 比较两个结果，计算总体误差。
float caculateErro(float *a, float *b, int size)
{
    float total, erro;
    total = 0;
    erro = 0;
    for (int i = 0; i < size; i++)
    {
        total = (a[i] + b[i])*(a[i] + b[i]) + total;
        erro = (a[i] - b[i])*(a[i] - b[i]) + erro;
    }
    return  sqrt(erro / total);
}
int main()
{
    int bdatasize = 4 * nx*ny*sizeof(float);  //b data's size ; 变量 b 数据大小
    int adatasize = 4 * (nx + 2)*(ny + 2)*sizeof(float);//a data's size; 变量 a 数据大小
    float *d_a;
    float *d_b;
    float *d_bp;
    float cputime, gputime;
    float erro1;
    //generate  the initial value of all vairables ; 产生所有变量的初始值。
    float axis[nx + 2];
    float ax[nx + 2];
    float ay[nx + 2];
    zeros(&b[0][0][0], 4 * nx*ny);
    zeros(&a[0][0][0], 4 * (nx + 2)*(ny + 2));
    zeros(&bp[0][0][0], 4 * (nx)*(ny));
    for (int i = 0; i<nx + 2; i++)
    {

        ax[i] = sin(i * 2 * pi / (1.0*(nx + 1)));
        ay[i] = 1.0;
    }
    int awidth = ny + 2;
    int alenth = (nx + 2)* (ny + 2);
    initialvalue(&a[1][0][0], ax, ay, nx + 2, ny + 2);
    for (int i = 0; i < nx; i++)
    {
        for (int j = 0; j < ny; j++)
        {
            bp[0][i][j] = a[0][i + 1][j + 1];
```

```
            bp[1][i][j] = a[1][i + 1][j + 1];
            bp[2][i][j] = a[2][i + 1][j + 1];
            bp[3][i][j] = a[3][i + 1][j + 1];
        }
    }
    //use gpu to calculate ;利用 gpu 进行计算。

    dim3 Dimgrid(nx);//set grid size;设置 grid 大小
    dim3 Dimblock(ny);//set block size;设置 block 大小
    cudaEvent_t start, stop;
    cudaEventCreate(&start);
    cudaEventCreate(&stop);

    //we allocate memary in  gpu .在 gpu 上分配存储。
    cudaMalloc((void **)&d_b, bdatasize);
    cudaMalloc((void **)&d_a, adatasize);
    cudaMalloc((void **)&d_bp, bdatasize);

    //copy data to gpu.将数据复制到 gpu 上。
    cudaMemcpy(d_a, a, adatasize, cudaMemcpyHostToDevice);
    cudaMemcpy(d_b, b, bdatasize, cudaMemcpyHostToDevice);
    cudaMemcpy(d_bp, bp, bdatasize, cudaMemcpyHostToDevice);
    //compute the equation .计算方程。
    cudaEventRecord(start, 0);
    for (int t = 0; t < times; t++)
    {
        calcculate << <Dimgrid, Dimblock >> >(d_a, d_b,d_bp);
        cudaThreadSynchronize();
        updatevalues << <Dimgrid, Dimblock >> >(d_a, d_b,d_bp);
        cudaThreadSynchronize();
        initialboundarycondition << <1, nx >> >(d_a, d_b);
        cudaThreadSynchronize();
    }
    cudaEventRecord(stop, 0);
    cudaEventSynchronize(stop);

    //calcutate the time use in gpu .计算在 gpu 中花费的时间。
    float duringtime;
    cudaEventElapsedTime(&duringtime, start, stop);
    cudaEventDestroy(start);
    cudaEventDestroy(stop);
    cudaMemcpy(ca, d_a, adatasize, cudaMemcpyDeviceToHost);
    cudaMemcpy(br, d_b, bdatasize, cudaMemcpyDeviceToHost);
```

```
    //release the gpu memory .释放 gpu 内存。
    cudaFree(d_a);
    cudaFree(d_b);
    cudaFree(d_bp);
    //use cpu to compute the equation .用 cpu 进行计算
    clock_t startc, endc;
    startc = clock();
    cpucompute();
    endc = clock();
    //计算 cpu 花费的时间
    cputime = (float)(endc - startc)*1000.0 / CLOCKS_PER_SEC;
    //compute the erro between the result of  cpu and gpu , 计算 gpu 和 cpu 结果误差
    erro1 = caculateErro(&a[0][0][0], &ca[0][0][0], nx*ny * 4);
    //print the reslult .打印结果
    printDeviceProp();
    printf(" parameter is :\n nx:%d, ny:%d, circle times:%d, dt:%f , dx:%f \n", nx, ny,times,
dt, dx);
    printf("erro:%f , cpu time:%f , gputime:%f", erro1, cputime, duringtime);

    //we save one point's  variable history .记录一个点状态的历史轨迹。
    ofstream SaveFile("data.txt");
    for (int i = 0; i < times; i++)
    {
        SaveFile << i << " " << rt[i][0] << " " << rt[i][1] << " " << rt[i][2] << " " <<
rt[i][3] << endl;
    }
    SaveFile.close();
    //avoid the display windows close automatically.防止显示窗口自动关闭
    int s = 0;
    scanf("%d", &s);
#
}
```