阅读

2022.04 chapter03

关于cuda计算时间

不可用如下形式，如下形式不可用并行计算的程序

```
clock_t start, finish;
start = clock();
// 要测试的部分
finish = clock();
duration = (double)(finish - start) / CLOCKS_PER_SEC;
```

采用如下形式

```
#include <sys/time.h>
double cpuSecond()
{
  struct timeval tp;
  gettimeofday(&tp,NULL);
  return((double)tp.tv_sec+(double)tp.tv_usec*1e-6);
}
```

使用方式如下

```
  double iStart,iElaps;
  iStart=cpuSecond();
  sumArraysGPU<<<grid,block>>>(a_d,b_d,res_d,nElem);
  cudaDeviceSynchronize();
  iElaps=cpuSecond()-iStart;
```

还有个工具是nvprof

```
nvprof [nvprof_args] <application>[application_args]
```

nvprof的使用参考文档

组织并行线程

考虑矩阵并行计算

2-3 谭升blog

用于加深grid网格（核函数）的块和线程的理解，二维度的块和二维度的线程

blog 是真的强，里面解释很清楚，关于二维其实也可以理解为一维，毕竟展开后，如下形式，线程的

ix=threadIdx.x+blockIdx.x*blockDim.x

threadIdx已经在每个block里的线程Idx，假设全局线程位于的坐标(ix，iy)

看block(0,1) 中块的线程thread(1,1)

参考上一个chapter blockDim和gridDim为核函数传递进去的thread块和block块

ix = threadIdx.x +blockIdx.x*blockDim.x=1+0x3=1

iy = threadIdx.y +blockIdy*blockDim.y = 1+1x3 = 4

nx = gridDim.x = 2

全局函数变量 idx = ix+iy*nx = 1+4x2=9

2,2

2+2*2=6

为了方便绘画，如下代表一个核函数中gird被拆分了block和thread块，分别实际坐标汇算方式仔细思考上述公式，

```
-------------------------------------------------------------------------------
---------------------->fx  (ix,iy)
             block(0,0)                                  block(1,0)
--------------------------------------  | --------------------------------------
----
|thread(0,0)||thread(1,0)||thread(2,0)|  |
|thread(0,0)||thread(1,0)||thread(2,0)|
|thread(0,1)||thread(1,1)||thread(2,1)|  |
|thread(0,1)||thread(1,1)||thread(2,1)|
|thread(0,2)||thread(1,2)||thread(2,2)|  |
|thread(0,2)||thread(1,2)||thread(2,2)|
--------------------------------------  | --------------------------------------
----
              block(0,1)                |                   block(1,1)
--------------------------------------  | --------------------------------------
----
|thread(0,0)||thread(1,0)||thread(2,0)|  |
|thread(0,0)||thread(1,0)||thread(2,0)|
|thread(0,1)||thread(1,1)||thread(2,1)|  |
|thread(0,1)||thread(1,1)||thread(2,1)|
|thread(0,2)||thread(1,2)||thread(2,2)|  |
|thread(0,2)||thread(1,2)||thread(2,2)|
--------------------------------------  | --------------------------------------
----
```

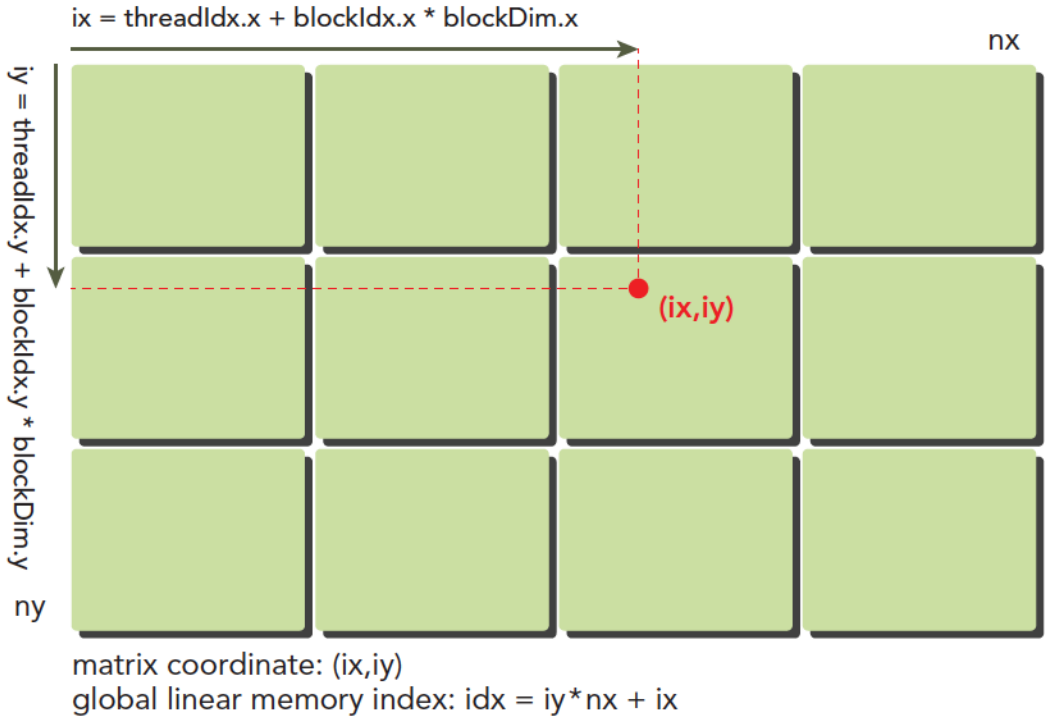
iy=threadIdx.y+blockIdx.y*blockDim.y

解释上述公式为何换算成线程，threadIdx为该block块当前的线程的idx，需要定位到当前block的偏移量，blockIdx则为切分的block的编号

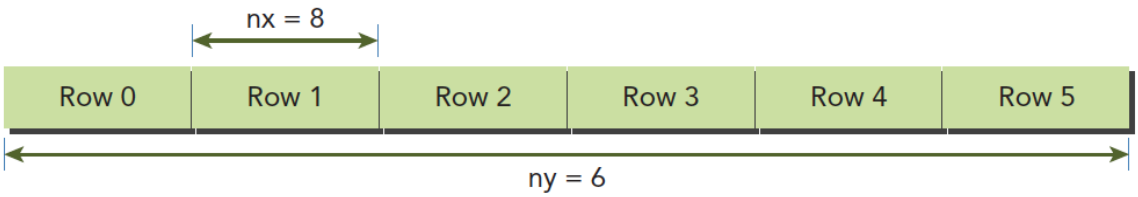
blockDim为线程块的x

idx = ix+iy*nx

二维一开始如此，后续理解展开为线性的内存分配，则全局地址如下展示所示



后续可理解为一维度global linear memory index = idx = iy*nx+ix nx=gridDim.x即grid被分割的block块的x维度



代码上理解谭升blog中

```
__global__ void printThreadIndex(float *A,const int nx,const int ny)
{
  int ix=threadIdx.x+blockIdx.x*blockDim.x;
  int iy=threadIdx.y+blockIdx.y*blockDim.y;
  unsigned int idx=iy*nx+ix;
  printf("thread_id(%d,%d) block_id(%d,%d) coordinate(%d,%d)"
         "global index %2d ival %2d\n",threadIdx.x,threadIdx.y,
         blockIdx.x,blockIdx.y,ix,iy,idx,A[idx]);
}
```

output

grid(2,3) gridDim.x = 2 =nx

block(4,2)

```
thread_id(0,0) block_id(1,0) coordinate(4,0)global index  4 ival 0 ix = 0+1*4=4
iy = 0+0*2=0 glo_idx = ix+iy*gridDim.x =4+0*2=4
thread_id(1,0) block_id(1,0) coordinate(5,0)global index  5 ival  0
thread_id(2,0) block_id(1,0) coordinate(6,0)global index  6 ival  0
thread_id(3,0) block_id(1,0) coordinate(7,0)global index  7 ival  0
thread_id(0,1) block_id(1,0) coordinate(4,1)global index 12 ival  0
thread_id(1,1) block_id(1,0) coordinate(5,1)global index 13 ival  0
thread_id(2,1) block_id(1,0) coordinate(6,1)global index 14 ival  0
thread_id(3,1) block_id(1,0) coordinate(7,1)global index 15 ival  0
thread_id(0,0) block_id(1,1) coordinate(4,2)global index 20 ival  0
```

```
thread_id(1,0) block_id(1,1) coordinate(5,2)global index 21 ival  0
thread_id(2,0) block_id(1,1) coordinate(6,2)global index 22 ival  0
thread_id(3,0) block_id(1,1) coordinate(7,2)global index 23 ival  0
thread_id(0,1) block_id(1,1) coordinate(4,3)global index 28 ival  0
thread_id(1,1) block_id(1,1) coordinate(5,3)global index 29 ival  0
thread_id(2,1) block_id(1,1) coordinate(6,3)global index 30 ival  0
thread_id(3,1) block_id(1,1) coordinate(7,3)global index 31 ival  0
thread_id(0,0) block_id(0,0) coordinate(0,0)global index  0 ival  0
thread_id(1,0) block_id(0,0) coordinate(1,0)global index  1 ival  0
thread_id(2,0) block_id(0,0) coordinate(2,0)global index  2 ival  0
thread_id(3,0) block_id(0,0) coordinate(3,0)global index  3 ival  0
thread_id(0,1) block_id(0,0) coordinate(0,1)global index  8 ival  0
thread_id(1,1) block_id(0,0) coordinate(1,1)global index  9 ival  0
thread_id(2,1) block_id(0,0) coordinate(2,1)global index 10 ival  0
thread_id(3,1) block_id(0,0) coordinate(3,1)global index 11 ival  0
thread_id(0,0) block_id(1,2) coordinate(4,4)global index 36 ival  0
thread_id(1,0) block_id(1,2) coordinate(5,4)global index 37 ival  0
thread_id(2,0) block_id(1,2) coordinate(6,4)global index 38 ival  0
thread_id(3,0) block_id(1,2) coordinate(7,4)global index 39 ival  0
thread_id(0,1) block_id(1,2) coordinate(4,5)global index 44 ival  0
thread_id(1,1) block_id(1,2) coordinate(5,5)global index 45 ival  0
thread_id(2,1) block_id(1,2) coordinate(6,5)global index 46 ival  0
thread_id(3,1) block_id(1,2) coordinate(7,5)global index 47 ival  0
thread_id(0,0) block_id(0,1) coordinate(0,2)global index 16 ival  0
thread_id(1,0) block_id(0,1) coordinate(1,2)global index 17 ival  0
thread_id(2,0) block_id(0,1) coordinate(2,2)global index 18 ival  0
thread_id(3,0) block_id(0,1) coordinate(3,2)global index 19 ival  0
thread_id(0,1) block_id(0,1) coordinate(0,3)global index 24 ival  0
thread_id(1,1) block_id(0,1) coordinate(1,3)global index 25 ival  0
thread_id(2,1) block_id(0,1) coordinate(2,3)global index 26 ival  0
thread_id(3,1) block_id(0,1) coordinate(3,3)global index 27 ival  0
thread_id(0,0) block_id(0,2) coordinate(0,4)global index 32 ival  0
thread_id(1,0) block_id(0,2) coordinate(1,4)global index 33 ival  0
thread_id(2,0) block_id(0,2) coordinate(2,4)global index 34 ival  0
thread_id(3,0) block_id(0,2) coordinate(3,4)global index 35 ival  0
thread_id(0,1) block_id(0,2) coordinate(0,5)global index 40 ival  0
thread_id(1,1) block_id(0,2) coordinate(1,5)global index 41 ival  0
thread_id(2,1) block_id(0,2) coordinate(2,5)global index 42 ival  0
thread_id(3,1) block_id(0,2) coordinate(3,5)global index 43 ival  0
```

由此转向二维矩阵的加法

```
__global__ void sumMatrix(float * MatA,float * MatB,float * MatC,int nx,int ny)
{
    int ix=threadIdx.x+blockDim.x*blockIdx.x;
    int iy=threadIdx.y+blockDim.y*blockIdx.y;
    int idx=ix+iy*ny;
    if (ix<nx && iy<ny)
    {
      MatC[idx]=MatA[idx]+MatB[idx];
    }
}
```

计算过程设置过程中 非常巧妙的用了比较

```
dimx =32
dimy =32
//Mat矩阵 1<<12 x 1<<12 维度
int nxy=nx*ny=1<<12 x 1<<12
dim3 block_0(dimx,dimy);
dim3 grid_0((nx-1)/block_0.x+1,(ny-1)/block_0.y+1);

dim3 block_1(dimx);
dim3 grid_1((nxy-1)/block_1.x+1);

dim3 block_2(dimx);
dim3 grid_2((nx-1)/block_2.x+1,ny);
```

output

```
CPU Execution Time elapsed 0.025741 sec
GPU Execution configuration<<<(128,128),(32,32)>>> Time elapsed 0.000888 sec
Check result success!
GPU Execution configuration<<<(524288,1),(32,1)>>> Time elapsed 0.001332 sec
Check result success!
GPU Execution configuration<<<(128,4096),(32,1)>>> Time elapsed 0.001334 sec
```