中山大学数据科学与计算机学院

《高性能程序设计基础》实验 7

（2018-2019 学年秋季学期）

学　　号：16337113

姓　　名：劳马东

教学班级：教务 2 班

专　　业：超算

一. 实验题目

1、 完成 cuda 的“Hello world”程序，编译运行 grid=（2,4），block=（8,16），给出输出结果文件。

2、 完成 CUDA 的两个矩阵乘法 A*B=C, 其中 A，B 是 5000*5000 的方阵。假设矩阵 A 的元素为 aij=i-0.1*j+1，矩阵 B 的元素为 bij=0.2*j-0.1*i。

二. 实验过程

（一）一些约定

假设 A、B 矩阵是 $WIDTH \times WIDTH$ 矩阵：

1、 grid 和 block 都是二维方阵，grid 的大小为 $M \times M$，block 大小为 $N \times N$；

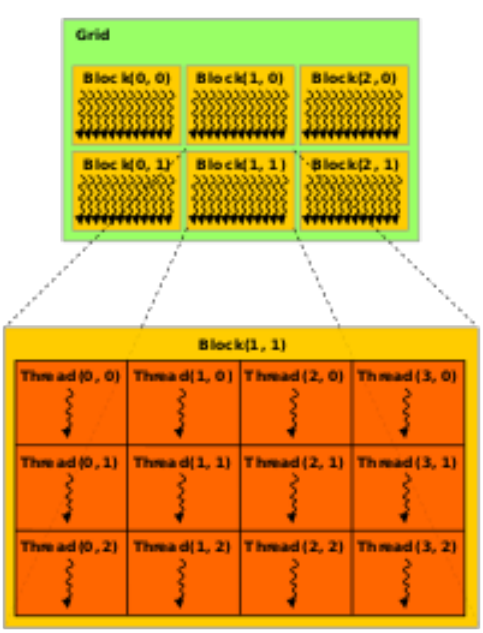
2、 $\frac{WIDTH}{M \times N}$ 是整数；

3、 一个 block 最多有 512 个线程 $\Rightarrow N \leq \lfloor \sqrt{512} \rfloor = 20$；

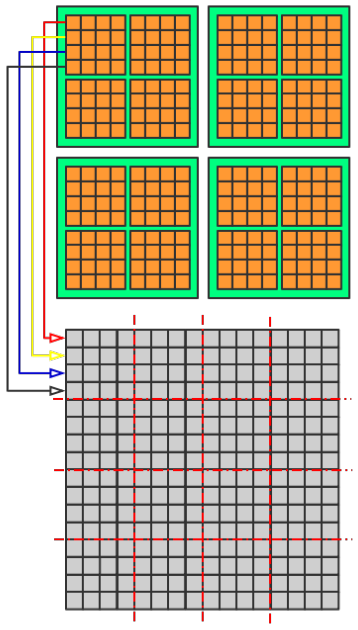
（二）矩阵分块

在 ___global___ 函数中，我们能够得到一个线程的 $blockIdx$ 和 $threadIdx$，如何根据这两个坐标，找到线程对应的子矩阵呢？在 CUDA 中，线程的分布情况如图1(a)。一个 grid 中有多个 block，它的 $blockIdx$ 中的 (x, y) 表示它的坐标，而不是行列号；同理，一个 block 中有多个 thread，它的 $threadIdx$ 同样是坐标。例如，在图1(a)中，绿色表示一个 grid，$(0, 0)$ block 右边是 $(1, 0)$ block，下面是 $(0, 1)$ block；橙色表示一个 block，$(0, 0$ thread 右边是 $(1, 0)$ thread，下面是 $(0, 1)$ thread。

基于线程分布，我们就可以将坐标相同的线程与矩阵块对应起来，如图1(b)，上半部分每一个橙色的小格是一个线程，下半部分每一个灰色的小格是矩阵的一个块（不一定是 1×1）。



(a) CUDA 中的线程层次



(b) 矩阵分块示意图

如此，一个线程对应的矩阵块的起始 (i, j) 可以这样计算（这里的 i、j 是绝对而不是相对）：

$$
\begin{aligned}
block\_i &= blockIdx.y \times \frac{WIDTH}{gridDim.x} \\
block\_j &= blockIdx.x \times \frac{WIDTH}{gridDim.x} \\
i &= block\_i + threadIdx.y \times \frac{WIDTH}{gridDim.x \times blockDim.x} \\
j &= block\_j + threadIdx.x \times \frac{WIDTH}{gridDim.x \times blockDim.x}
\end{aligned}
\tag{1}
$$

（三）子矩阵乘法

在这里，子矩阵乘法并不是线性代数中矩阵分块乘法。由于线程之间不涉及通信，每个线程需要计算它对应的结果子矩阵的每个元素的最终结果。因此，为了计算结果矩阵 (i, j) 位的一个元素，一个线程需要遍历矩阵 A 的第 i 行的所有元素和矩阵 B 第 j 列的所有元素，即。

算法 1 线程计算其结果子矩阵

输入: 线程的矩阵块的全局起始行列号 (i, j)，输入矩阵 A、B，结果矩阵 C

function $thread\_matrix\_mul(i, j, A, B, C)$
  $local\_width \leftarrow \frac{WIDTH}{gridDim.x \times blockDim.x}$
  for $local\_i \in [0, local\_width)$ **do**
    $global\_i \leftarrow i + local\_i$
    for $local\_j \in [0, local\_width)$ **do**
      $global\_j \leftarrow j + local\_j$
      $sum \leftarrow 0$
      for $k \in [0, WIDTH)$ **do**
        $sum \leftarrow sum + A[global\_i][k] * B[k][global\_j]$
      end for
      store sum to $C[global\_i][global\_j]$
    end for
  end for
end function

三. 关键代码

1、 矩阵分块

```
__global__ void matrix_mul_kernel(double* P, unsigned n) {
    size_t n_per_block = n/gridDim.x, n_per_thread = n_per_block/blockDim.x;
    size_t block_i = blockIdx.y*n_per_block, block_j = blockIdx.x*n_per_block;
    size_t thread_i = block_i + threadIdx.y * n_per_thread,
           thread_j = block_j + threadIdx.x * n_per_thread;
    ...
}
```

代码清单 1. 矩阵分块

2、 子矩阵乘法

```
__device__ void cal_one_ele(size_t i, size_t j, size_t n, double* P) {
    double sum = 0;
    for (int k = 0; k < n; ++k) {
        double a_ik = i - 0.1 * k + 1, b_kj = 0.2 * j - 0.1 * k;
        sum += a_ik * b_kj;
    }
    *(P + i * n + j) = sum;
}
__global__ void matrix_mul_kernel(double* P, unsigned n) {
    ...
    for (int i = 0; i < n_per_thread; ++i) {
        size_t ii = thread_i + i;
        for (int j = 0; j < n_per_thread; ++j) {
            cal_one_ele(ii, thread_j + j, n, P);
        }
    }
}
```

代码清单 2. 计算结果矩阵

3、 统计 GPU 时间

在并行程序中，想要统计程序整体的运行时间，就需要在计时前做同步。在 MPI 中，提供了 Barrier 机制实现同步，类似的，CUDA 有事件同步。在开始调用核函数之前，让时间记录下时间点，程序运行完成后，使用 cudaEventSynchronize 函数同步，记录此时的时间点。两次事件记录之间的时间差计时核函数运行的时间，可用 cudaEventElapsedTime 获得。值得一提的是，直接使用 clock 函数统计得到的是 CPU 的运行时间，因为 CPU 调用完核函数后，并不等待其运行完成，直接就返回了。

```
    float elapsed=0;
    cudaEvent_t start, stop;
    cudaEventCreate(&start);
    cudaEventCreate(&stop);

    cudaEventRecord(start, nullptr);

    matrix_mul_kernel<<<grid_dim, block_dim>>>(result.data().get(), n);

    cudaEventRecord(stop, nullptr);
    cudaEventSynchronize (stop);
    cudaEventElapsedTime(&elapsed, start, stop);
    cudaEventDestroy(start);
    cudaEventDestroy(stop);
```

代码清单 3. 用事件统计程序运行时间

四. 实验结果及分析

实验平台为 Google Colab（GPU Tesla K80），测得串行代码矩阵相乘部分的时间是 1882.77 秒；使用 sm_30 架构，并行时间如下。

grid	block	kernel 时间/s	加速比
250×250	20×20	1.27853	1472.6
125×125	20×20	1.25271	1502.9
500×500	10×10	1.37771	1366.6
1000×1000	10×10	1.64177	1146.8