實驗九 CUDA Programming

9617145 資工 4C 許晏峻
9617167 資工 4C 蔡孟儒

1. 實驗目的

以 CUDA programming language 了解 GPU 如何加速 parallel program 的計算。

2. 步驟過程

2.1. 基本題

- 在 host 部分：

首先必需先分配空間給 CUDA 儲存計算的值，並且將兩個矩陣複製一次：

```
cudaMalloc(&d_A, size);
cudaMalloc(&d_B, size);
cudaMalloc(&d_C, size);
cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
cudaMemcpy(d_B, h_B, size, cudaMemcpyHostToDevice);
```

接著將 CUDA 需要使用的 Grid、Block、Thread 做初始化：

```
dim3 dimGrid(1, N);
dim3 dimBlock(1, N);
```

最後執行 kernel function，完成後將計算的值複製回矩陣中，再將空間釋放：

```
MatrixMulKernel<<<dimGrid, dimBlock>>>(d_A, d_B, d_C, N);
cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);
cudaFree(d_A);
cudaFree(d_B);
cudaFree(d_C);
```

- 在 device 部分：

宣告一個__global__ function 作為 host 呼叫給 device 的 kernel function：

```
__global__ void MatrixMulKernel(int* d_A, int* d_B, int* d_C, int width){
    int Cvalue=0;
    for(int k=0; k<width; ++k){
        int Aelement = d_A[blockIdx.x*width+k];
        int Belement = d_B[k*width+threadIdx.y];
        Cvalue += Aelement * Belement;
    }
    d_C[blockIdx.x*width+threadIdx.y] = Cvalue;
}
```

以上利用 blockIdx.x 是 0~N-1、而 threadIdx.y 是 0~N-1 來對應到矩陣位置的運算。

2.2. 進階題

- pthread 程式部分：

首先宣告 pthread 工作函式需傳遞變數的 struct：

```
typedef struct
{
    int tid;
    int *a, *b, *c;
} parm;
```

接者，改寫原本的 array_mul function 成 thread 工作函式；只要將 for 迴圈的起始值和每次 iteration 遞加的數字改成 thread 個數，完成分工。

最後，在 main function 部分，則是宣告 pthread 需要的各個變數，在一一呼叫 pthread_create，最後再一一呼叫 pthread_join 即可。

- 各版本程式執行效能比較圖部分：

在各種版本的程式上加上執行時間的計算，然後以不同的 thread 數執行每種版本的程式，把執行時間記錄下來，最後再和原始版本程式唯一劇作效能比較。

測試數據放在 3.數據結果中。

3. 數據結果

- Origin program

	1000*1000	100*100	10*10	2*2	1*1
1 thread	8.082575811	0.006787883	0.000009719	0.000000378	0.000000771

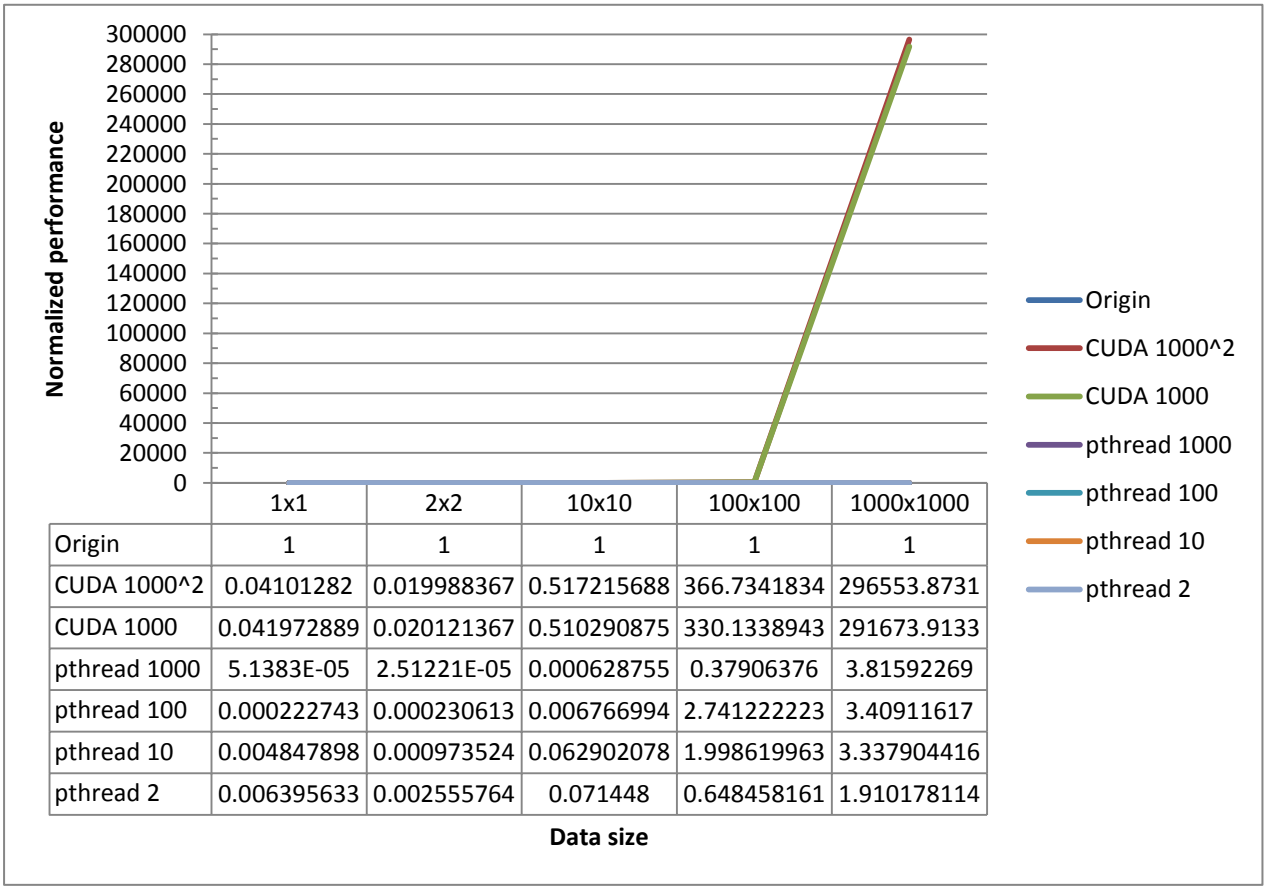
- CUDA

	1000*1000	100*100	10*10	2*2	1*1
1000^2 threads	0.000027255	0.000018509	0.000018791	0.000018911	0.000018799
1000 threads	0.000027711	0.000020561	0.000019046	0.000018786	0.000018369

- pthread

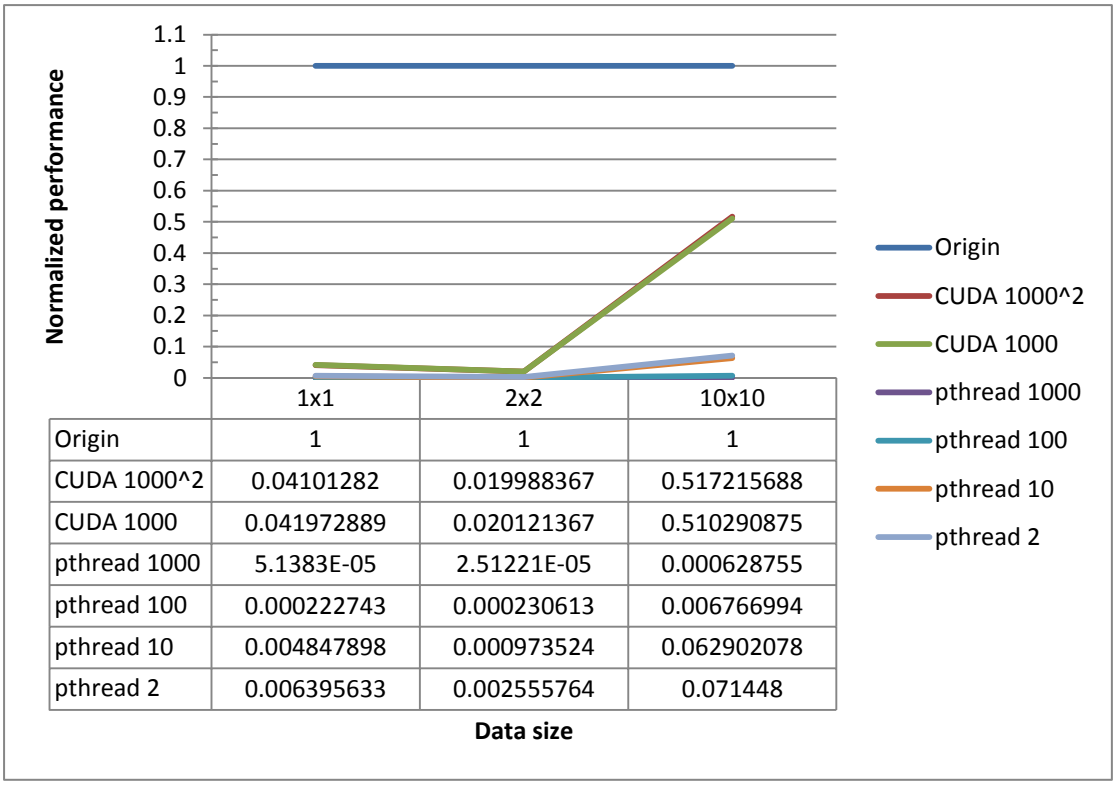
	1000*1000	100*100	10*10	2*2	1*1
1000 threads	2.118118334	0.017906969	0.015457536	0.015046503	0.015004959
100 threads	2.370871337	0.002476225	0.001436236	0.001639107	0.003461390
10 threads	2.421452146	0.003396285	0.000154510	0.000388280	0.000159038
2 threads	4.231320499	0.010467727	0.000136029	0.000147901	0.000120551

- 比較圖 1 (總覽)：



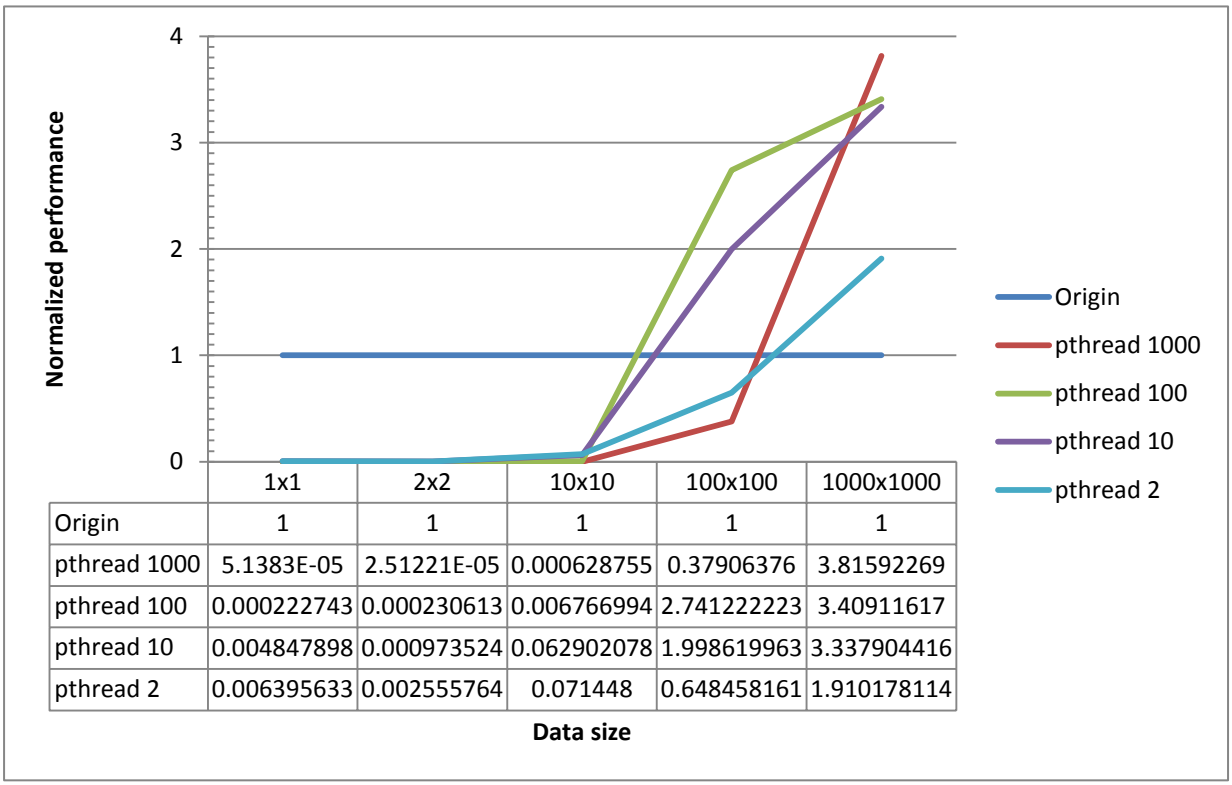
	1x1	2x2	10x10	100x100	1000x1000
Origin	1	1	1	1	1
CUDA 1000^2	0.04101282	0.019988367	0.517215688	366.7341834	296553.8731
CUDA 1000	0.041972889	0.020121367	0.510290875	330.1338943	291673.9133
pthread 1000	5.1383E-05	2.51221E-05	0.000628755	0.37906376	3.81592269
pthread 100	0.000222743	0.000230613	0.006766994	2.741222223	3.40911617
pthread 10	0.004847898	0.000973524	0.062902078	1.998619963	3.337904416
pthread 2	0.006395633	0.002555764	0.071448	0.648458161	1.910178114

- 比較圖 2 (縮小縱座標間隔，取小於 10x10 data size)：



	1x1	2x2	10x10
Origin	1	1	1
CUDA 1000^2	0.04101282	0.019988367	0.517215688
CUDA 1000	0.041972889	0.020121367	0.510290875
pthread 1000	5.1383E-05	2.51221E-05	0.000628755
pthread 100	0.000222743	0.000230613	0.006766994
pthread 10	0.004847898	0.000973524	0.062902078
pthread 2	0.006395633	0.002555764	0.071448

- 比較圖 3 (縮小縱座標間隔，不取 CUDA(因資料差異間隔太大))：



	1x1	2x2	10x10	100x100	1000x1000
Origin	1	1	1	1	1
pthread 1000	5.1383E-05	2.51221E-05	0.000628755	0.37906376	3.81592269
pthread 100	0.000222743	0.000230613	0.006766994	2.741222223	3.40911617
pthread 10	0.004847898	0.000973524	0.062902078	1.998619963	3.337904416
pthread 2	0.006395633	0.002555764	0.071448	0.648458161	1.910178114

由以上三張圖，可看到當 data size 極小的時候，origin 程式效能最好，因為它沒有對 CPU 做建立 thread 所需的 overhead，也沒有對 GPU 做建立 grid、block

和 thread 所需的 overhead。然而，隨著 data size 的提高，可看出多執行緒的程式會有較好的效能，normalized performance 甚至差上好幾萬倍。

4. 結論心得

這次實驗相當有趣，接觸到了使 CPU 以外的運算程式—CUDA programming。從實驗中也了解到了寫 CUDA 程式時必須考慮到 device 和 host 之間的溝通；還有就是在 device 端無法用 printf 等方法做 output，因此 debug 的難度又提升了，必須先想清楚其中的邏輯概念。

而實驗中較令我不解的是有關於一個 block 可以執行幾個 thread 的問題，測試中不同的主機上，有些可以開到 N*N 個 thread，效能表現上也較好，而有些則無法，必須動到 grid 多執行幾個 block 來減少一個 block 需要執行的 thread 數，效能上當然是稍微差了。

而還有一個較不清楚的概念是有關 blockIdx 和 threadIdx 的計算方式，在寫 CUDA 程式的過程中，跑出來的值似乎和預期有所不同，造成了有些認為不對的邏輯執行結果卻是正確的。

在多次測試中我發現當其中一次答案正確後，之後不管怎麼改在執行也是正確的，實在令我感到不解；但關機後再開機執行，答案又會變成錯誤的(只有在 CUDA programming 出現此問題)；因此，實驗中我盡量每次都重新開機執行來得到不同 thread 的結果。

希望在以後幾個 CUDA programming 實驗中可以更加瞭解 CUDA 的觀念，並且能夠更熟練 CUDA programming 的技巧。