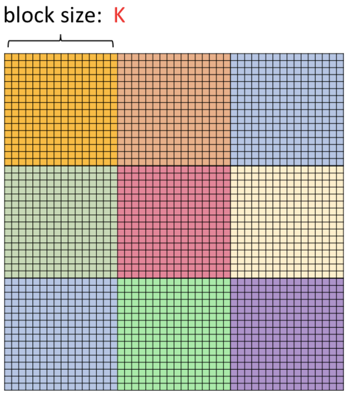
Parallel Programming hw4-1 ----Blocked Folyd-Warshall algo.

學號: 109062639　姓名: 葉哲欣

Implementation:

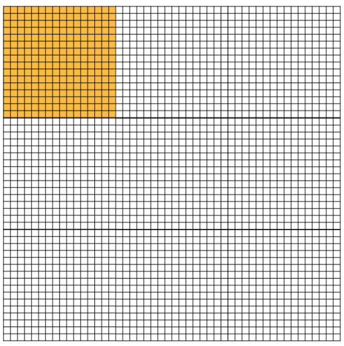


K:32

將 distance matrix(data)依照每個 block size（K*K）來分割 data。並依照演算法 3 個 kernel function(phase)用到的 data dependency block 來宣告 block 數以及 share memory 數。

共需執行 Round = ceil(V/k) 次，Round*Round 個 block

Phase1



phase1 每個 round 只需運算一個 block(黃色)的大小 data (block$_{(k,k)}$)

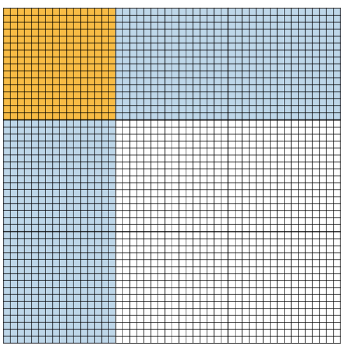
故 phase1 所需 thread 及 block：

dim3 ThreadPerBlock(k,k)

dim3 BlockPerGrid(1,1)

並且每個 block 所使用到的 share memory size = K*K

Phase2



Phase2 每個 round 需運算一個 2*Round 個 block(藍色+黃色)

故 phase2 所需 thread 及 block：

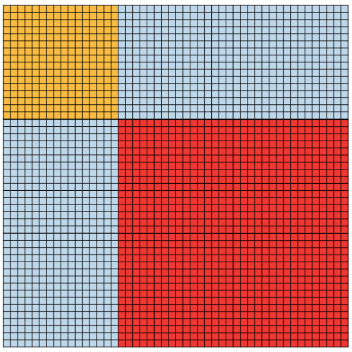
dim3 ThreadPerBlock(k,k)

dim3 BlockPerGrid(round,2)

而每個 block 會需要用到自己及 phase1 的 data

故每個 block 所使用到的 share memory size = 2*K*K

Phase3



Phase3 每個 round 需運算一個(round-1)*(round-1)個 block(藍色+紅色)
故 phase3 所需 thread 及 block：

dim3 ThreadPerBlock(k,k)

dim3 BlockPerGrid(round,round)

而每個 block 會需要用到自己及 phase2 的 row 及 col data
故每個 block 所使用到的 share memory size = 3*K*K

Profiling Results

Occupancy

kernel	Achieved occupancy(avg.)
Phase1	0.496326
Phase2	0.942633
Phase3	**0.896393**

sm efficiency

kernel	sm efficiency
Phase1	3.66%
Phase2	98.39%
Phase3	**99.97%**

Share memory load/store throughput

kernel	Load throughput	Store throughput
Phase1	57.631GB/s	19.755GB/s
Phase2	2429.2GB/s	1249.9GB/s
Phase3	**2970.5GB/s**	**92.716GB/s**

Global load/store throughput

kernel	Load throughput	Store throughput
Phase1	364.46MB/s	741.18MB/s
Phase2	16.554GB/s	44.787GB/s
Phase3	**19.902GB/s**	**56.468GB/s**

Experiment and Analysis

System Spec: hades server

Time distribution:

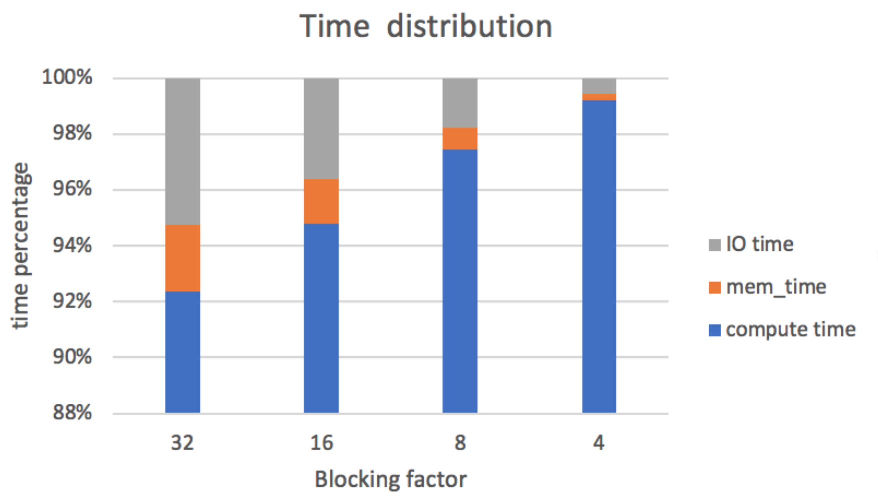
採用 nvprof 來查詢 kernel compute 以及 memry htd,dth time

使用 timespec 來紀錄 I/O time

(a.)

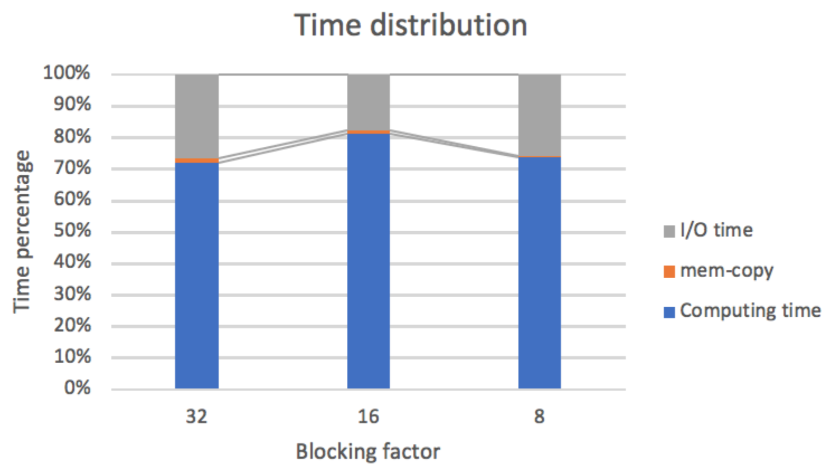
Test case:p21k

Block-factor	Computing time	copy time	I/O time
Block =32	20.813 s	0.53733 s	1.185324 s
Block =16	33.0087 s	0.54889 s	1.199476 s
sBlock =8	68.9008 s	0.55201 s	1.256658 s
Block=4	227.112 s	0.54833 s	1.29825 s



(b.)

Test case:p31k



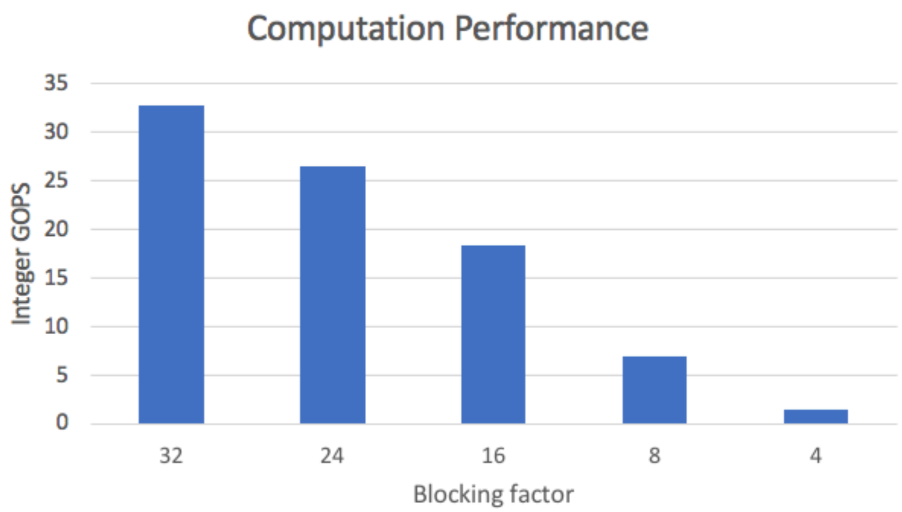
Block-factor	Computing time	copy time	I/O time
Block =32	64.9908	1.21186	23.973353
Block =16	100.281	1.20125	22.019499
Block =8	203.924	1.20078	71.302255

Blocking factor:

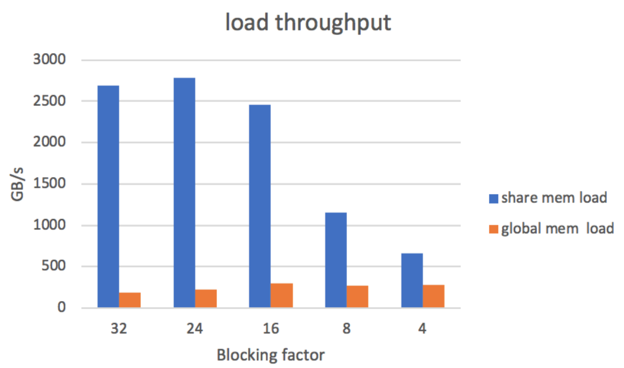
採用 phase3 作為 compute time 標準

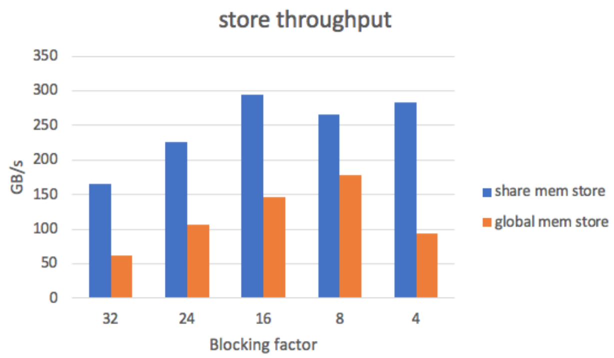
Computation Performance:

GOPS: $\frac{\#Instruction \times 4byte}{compute\ time \times 10^9}$



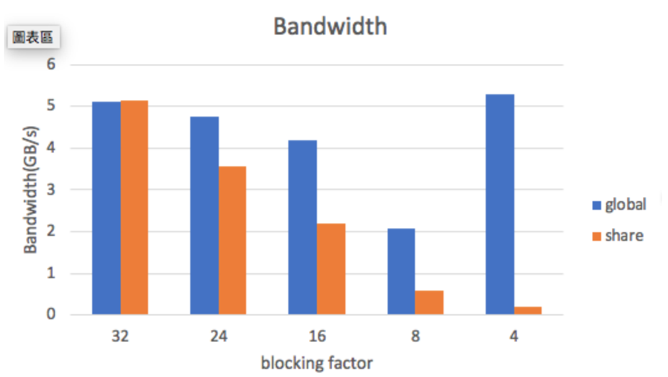
Global memory memory / shared memory load/store throughput :





Global memory memory / shared memory load/store bandwidth :

$$\frac{(\text{Load transcation} + \text{load transcation}) \times 32(\text{bytes})}{compute\ time}$$

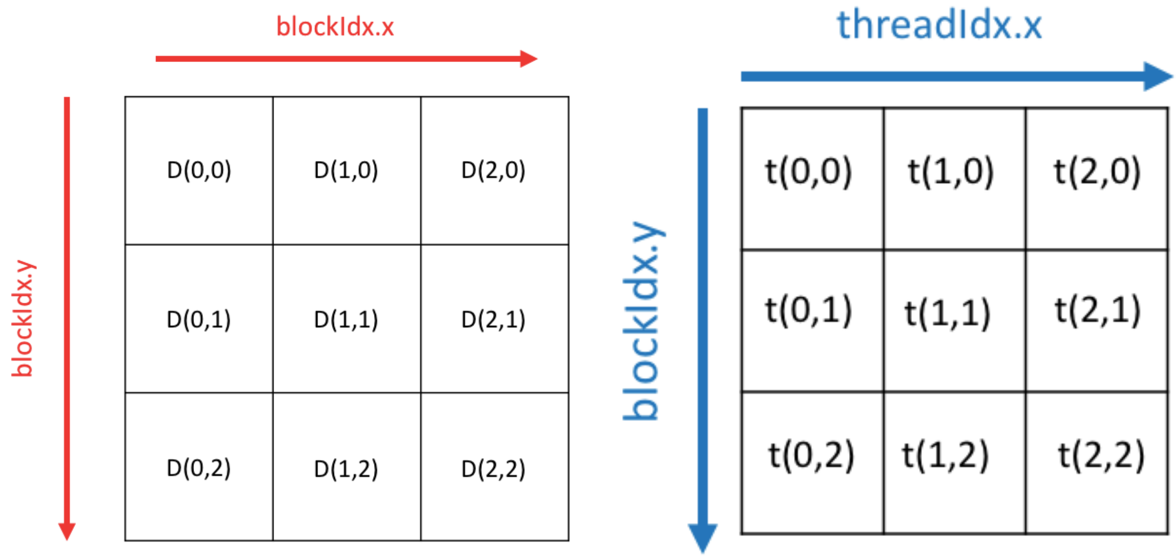


Optimization :

1. using 2 dim block

2. memory coalesced

   memory access 為 column major,而 threadId 編號是以 x 為 major,若直接以 threadIdx.x,threadId.y 轉換為 x,y 座標，則會發生不連續 memory access

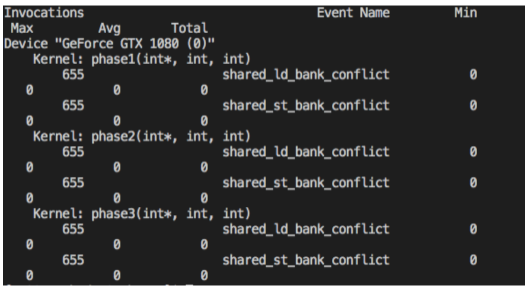


3. shared memory

   使用 share memory 來提升存取速度

4.avoid bank conflict

用 memory padding(share memory:block_size*(block_size+1)) 方式來避免



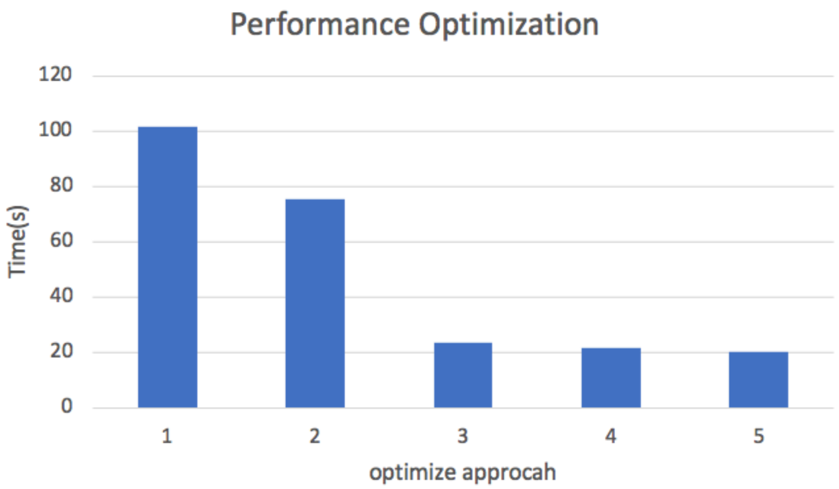
```
Invocations                                Event Name         Min
 Max          Avg        Total
Device "GeForce GTX 1080 (0)"
    Kernel: phase1(int*, int, int)
        655                    shared_ld_bank_conflict          0
   0            0            0
        655                    shared_st_bank_conflict          0
   0            0            0
    Kernel: phase2(int*, int, int)
        655                    shared_ld_bank_conflict          0
   0            0            0
        655                    shared_st_bank_conflict          0
   0            0            0
    Kernel: phase3(int*, int, int)
        655                    shared_ld_bank_conflict          0
   0            0            0
        655                    shared_st_bank_conflict          0
   0            0            0
```

5.unrolling

使用 pragma unroll 來展開迴圈

1	2D	101.599
2	2D + share_memory	75.4432
3	2D + share_memory +memoy coalsed	23.3685

4	2D + share_memory + bankconflict+memoy coalsed	21.477
5	2D + share_memory + bankconflict+memoy coalsed +unroll	20.3403



Experiment and Analysis

從這次作業我學到如何運用 cuda 來做 gpu 的運算，來加速程式運算。
透過存取 Share memory 方式凸顯 global memory 存取速度上的差異性，並且可以透過 coalesced memory ,memory padding 等等各式各樣的優化方式讓效能最佳化。本次作業困難點就是 block 與 block 彼此之間資料的傳輸，data 該如何分割，如何透過 blockIdx,以及 thredIdx mapping 到 data distance 上 ，並存入 shared memory 。另外發現若使用過多的 share memory 效能不一定會越好，過多的 thread 在 syncthreads 會嚴重影響效能。 因此我覺得還可以改進的部分就是減少 syncthreads 時間。