HW3 Report

111065541 張騰午

Implement

Hw3-1

做 Floyd-Warshal，時間複雜度 O(n^3)，不過在 i,j 兩個迴圈增加 omp 指令來平行化。

```
    for (int k = 0; k < vertex; ++k){
        #pragma omp parallel for schedule(guided, 1) collapse(2)
        for (int i = 0; i < vertex; ++i){
            for (int j = 0; j < vertex; ++j){
                if (Dist[i][j] > Dist[i][k] + Dist[k][j] && Dist[i][k] != INF)
                    Dist[i][j] = Dist[i][k] + Dist[k][j];
            }
        }
    }
}
```

另外在 initial 部分使用 fill 一次性填入 INF 並用 omp 指令來增加速度

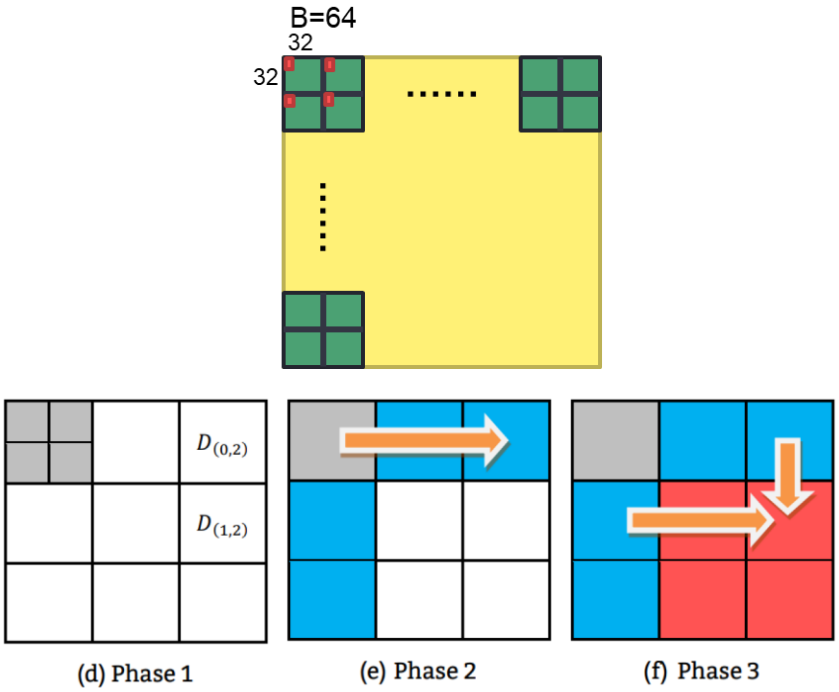
```
#pragma omp parallel for schedule(guided, 1)
for (int i = 0; i < vertex; ++i) {
    fill(Dist[i].begin(), Dist[i].end(), INF);
    Dist[i][i] = 0;
}
```

Hw3-2

使用了助教提供範例 blocked-Floyd-Warshall 演算法，input 時修改使用 int *fpt = (int*)mmap(NULL, 2*sizeof(int), PROT_READ, MAP_PRIVATE, file, 0);如此直接將所有資料一次性讀取，之後再寫入 V*V 的 Dist 中，如此可以減少 I/O 時間。

而演算法計算方面思路如下:

blocking factor 64，而一個 GPU 的 block 為 32 threads*32 threads，因此整個演算法把矩陣分成數個 64*64 block，每個 block 使用一個 GPU 32*32 的 block 下去算要計算 4 次，一個 thread 在一個 B*B block 要算 4 個點，如下圖紅色標示處，4 個點距離左右、上下皆為 32，



三個 Phase 分別用一個 32*32 block_dim threads、兩行 ceil(V/B)個 32*32 block_dim threads 以及 ceil(V/B)*ceil(V/B)個 32*32 block_dim threads 平面進行計算，宣告方式如下:

```
  int blocks = ceil(V, B);
  dim3 block_dim(BLOCK_SIZE , BLOCK_SIZE);//BLOCK_SIZE=32
  dim3 grid_dim(blocks, blocks);
  dim3 grid_dim2(blocks, 2);


for (int r = 0; r < round; ++r) {
    // phase 1
    Phase1<<<1, block_dim>>>(dst, r, V);
    // phase 2
    Phase2_<<<grid_dim2, block_dim>>>(dst, r, V);
    // phase 3
    Phase3<<<grid_dim, block_dim>>>(dst, r, V);
    //Phase3_<<<grid_dim, block_dim>>>(dst, r, V);
}
```

Phase 中需要計算的點座標如下(用一維方式表示)，

Phase 1(僅 pivot block)

```
__global__ void Phase1(int *dst, int Round, int V){
    int i = threadIdx.y;
    int j = threadIdx.x;
    int i_B = i + BLOCK_SIZE;
    int j_B = j + BLOCK_SIZE;
    //B為64，一次處理一個64*64，但block dim僅為32*32，因此一個大block分成四個小block，一個thread要計算四個小block四個點
    // 1 2
    // 3 4
    int offset = Round * B * (V+1);
    int blk_pt1 = offset + i * V + j;
    int blk_pt2 = offset + i * V + j_B;
    int blk_pt3 = offset + i_B * V + j;
    int blk_pt4 = offset + i_B * V + j_B;
```

Phase 2(下左，依據 blockId.y 的值判斷計算 pivot row 的 block 或是 pivot column 的 block)

Phase 3(下右，使用同 row 還有同 column 的 pivot row/column block 來計算值)

```
int blk_pt1 = offset + i * V+ j;
int blk_pt2 = offset + i * V + j_B;
int blk_pt3 = offset + i_B * V + j;
int blk_pt4 = offset + i_B *V + j_B;
//for same col block
if(blockIdx.y == 0) {
    offset_rc = blockIdx.x * B * V + Round * B;
    pivot_rc_blk1 = offset_rc + i * V + j;
    pivot_rc_blk2 = offset_rc + i * V + j_B;
    pivot_rc_blk3 = offset_rc + i_B * V + j;
    pivot_rc_blk4 = offset_rc + i_B * V + j_B;
}else {//for same row block
    offset_rc = Round * B * V + blockIdx.x * B;
    pivot_rc_blk1 = offset_rc + i * V + j;
    pivot_rc_blk2 = offset_rc + i * V + j_B;
    pivot_rc_blk3 = offset_rc + i_B * V + j;
    pivot_rc_blk4 = offset_rc + i_B * V + j_B;
}
```

```
int offset_ = blockIdx.y * B * V + blockIdx.x * B;
int blk_pt1 = offset_ + i * V + j;
int blk_pt2 = offset_ + i * V + j_B;
int blk_pt3 = offset_ + i_B * V + j;
int blk_pt4 = offset_ + i_B * V + j_B;
//same row
int offset_r = blockIdx.y * B * V + Round * B;
int row_blk1 = offset_r + i * V + j;
int row_blk2 = offset_r + i * V + j_B;
int row_blk3 = offset_r + i_B * V + j;
int row_blk4 = offset_r + i_B * V + j_B;
//same col
int offset_c = Round * B * V + blockIdx.x * B;
int col_blk1 = offset_c + i * V + j;
int col_blk2 = offset_c + i * V+ j_B;
int col_blk3 = offset_c + i_B * V+ j;
int col_blk4 = offset_c + i_B * V + j_B;
```

三個 phase 結束後先在 gpu 上用 devicetodevice 的 CudaMemcpy 把本來 V*V 存在 dst 的資料複製到另一個 vertex*vertex 大小的 dst_，

```
cudaMalloc(&dst_, vertex*vertex*sizeof(int));
for(int i = 0; i < vertex; ++i) {
    cudaMemcpy(dst_ + i*vertex, dst + i*V, sizeof(int)*vertex, cudaMemcpyDeviceToDevice);
}
cudaMemcpy(Dist, dst_, size_vertex, cudaMemcpyDeviceToHost);
```

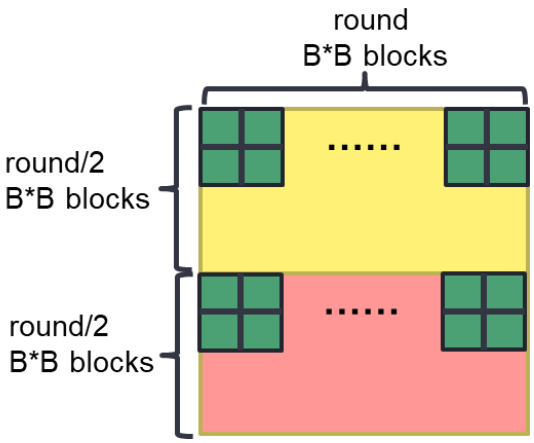
之後 output 時就可以直接 fwrite(Dist, sizeof(int), vertex*vertex, outfile);一次性把 vertex*vertex 量的資料全部寫入 output file 中減少 I/O 時間。

用__shared__ int share[B][B];創建 share memory，來讀取後續比較還有計算需要的值，每個 thread 將自己負責的值存入 share memory，並進行後續更新值的計算，存取修改 share memory 後要__syncthreads();進行同步，最後把 share memory 中的值存回 dst 中。
另外在進行比較計算時，發現使用__device__函數再調用 cuda 內建函數 min，讓計算在 GPU 上進行會比直接用 if 判斷式更快。

```
__device__ int min_(int a, int b) {return min(a, b);}
```

Hw3-3

3-2 的程式大致不變，查詢相關資料並和同學討論後只在 phase 3 使用兩 GPU 平分執行，因為有了 start_point，phase 3 計算座標時要多加入 y 軸方面 offset。
平分方式如下圖，整個矩陣本來能分成 round*round 個 B*B block，在 y 軸上分成上下兩部份給兩個 GPU 做。



程式在執行三個 phase 時用#pragma omp parallel num_threads(2)來執行兩 gpu，接著取得 thread_id 以及 neighbor thread id，並計算雙方需要做的數量還有起始點，若是 round 無法被整除，則將多的一個 round 加在 thread_id=0 的那邊

```
unsigned int thread_id = omp_get_thread_num();
cudaSetDevice(thread_id);
// neighbor
unsigned int neighbor_thread_id;
if(thread_id){
    neighbor_thread_id = 0;
}else{
    neighbor_thread_id = 1;
}
```

```
unsigned int rounds_to_do = round / 2;
unsigned int rounds_to_do_neighbor = round / 2;
if (thread_id == 0) {
    rounds_to_do += round % 2;
}else{rounds_to_do_neighbor += round % 2;}
```

```
unsigned int start_point;
if (thread_id) {
    start_point = rounds_to_do_neighbor;
} else {
    start_point = 0;
}
```

進入迴圈後每次都得進行迴圈 r 對應的那一行 block 數據交換(pivot row)，通過 #pragma omp barrier 來確保兩邊都已經完成數據操作。

```
for (int r = 0; r < round; ++r) {
    if (r >= start_point && r < (start_point + rounds_to_do)) {
        cudaMemcpy(dst[neighbor_thread_id] + r * B * V, dst[thread_id] + r * B * V, B * V * sizeof(int), cudaMemcpyDefault)
    }
    #pragma omp barrier
```

Experiment & Analysis

i、 Methodology

a. System Spec

在課程提供的hades上進行測試

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 384.81                 Driver Version: 384.81                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GeForce GTX 1080    On   | 00000000:4B:00.0 Off |                  N/A |
| 32%   47C    P8    14W / 216W |      2MiB /  8114MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  GeForce GTX 1080    On   | 00000000:4D:00.0 Off |                  N/A |
| 10%   47C    P8    18W / 216W |     11MiB /  8113MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
```

b. Blocking Factor (hw3-2)

GOPS

c21.1 進行 --metrics inst_integer 計算再除以執行時間

B=64

```
[pp23s71@hades01 hw3-2]$ srun -p prof -N1 -n1 --gres=gpu:1 nvprof --metrics inst_integer ./hw3-2 /home/pp23/share/hw3-2/cases/c21.1 B64.out
==222856== NVPROF is profiling process 222856, command: ./hw3-2 /home/pp23/share/hw3-2/cases/c21.1 B64.out
==222856== Profiling application: ./hw3-2 /home/pp23/share/hw3-2/cases/c21.1 B64.out
==222856== Profiling result:
==222856== Metric result:
Invocations                               Metric Name                    Metric Description         Min         Max         Avg
Device "NVIDIA GeForce GTX 1080 (0)"
    Kernel: Phase2_(int*, int, int)
         79                           inst_integer                    Integer Instructions    98244608    98244608    98244608
    Kernel: Phase1(int*, int, int)
         79                           inst_integer                    Integer Instructions      610304      610304      610304
    Kernel: Phase3(int*, int, int)
         79                           inst_integer                    Integer Instructions  2859898880  2859898880  2859898880
```

```
[pp23s71@hades01 hw3-2]$ srun -p prof -N1 -n1 --gres=gpu:1 nvprof ./hw3-2 /home/pp23/share/hw3-2/cases/c21.1 B64.out
==222921== NVPROF is profiling process 222921, command: ./hw3-2 /home/pp23/share/hw3-2/cases/c21.1 B64.out
==222921== Profiling application: ./hw3-2 /home/pp23/share/hw3-2/cases/c21.1 B64.out
==222921== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   73.15%  128.03ms        79  1.6207ms  1.5063ms  1.6559ms  Phase3(int*, int, int)
                    9.10%  15.930ms         1  15.930ms  15.930ms  15.930ms  [CUDA memcpy HtoD]
                    8.94%  15.641ms         1  15.641ms  15.641ms  15.641ms  [CUDA memcpy DtoH]
                    7.62%  13.334ms        79  168.78us  152.77us  178.27us  Phase2_(int*, int, int)
                    1.19%  2.0850ms        79  26.392us  24.128us  27.169us  Phase1(int*, int, int)
      API calls:   62.23%  174.58ms         2  87.288ms  15.953ms  158.62ms  cudaMemcpy
                   37.41%  104.94ms         1  104.94ms  104.94ms  104.94ms  cudaHostRegister
                    0.22%  624.42us       237  2.6340us  2.0330us  84.972us  cudaLaunchKernel
                    0.05%  147.82us         1  147.82us  147.82us  147.82us  cudaFree
                    0.05%  143.90us       101  1.4240us     103ns  60.857us  cuDeviceGetAttribute
                    0.03%  93.803us         1  93.803us  93.803us  93.803us  cudaMalloc
                    0.00%  10.844us         1  10.844us  10.844us  10.844us  cuDeviceGetName
                    0.00%  7.8010us         1  7.8010us  7.8010us  7.8010us  cuDeviceGetPCIBusId
                    0.00%  1.1290us         3     376ns     167ns     738ns  cuDeviceGetCount
                    0.00%     600ns         1     600ns     600ns     600ns  cuModuleGetLoadingMode
                    0.00%     493ns         2     246ns     129ns     364ns  cuDeviceGet
                    0.00%     267ns         1     267ns     267ns     267ns  cuDeviceTotalMem
                    0.00%     190ns         1     190ns     190ns     190ns  cuDeviceGetUuid
```

B=32

```
[pp23s71@hades01 hw3-2]$ srun -p prof -N1 -n1 --gres=gpu:1 nvprof --metrics inst_integer ./hw3-2 /home/pp23/share/hw3-2/cases/c21.1 B32.out
==123134== NVPROF is profiling process 123134, command: ./hw3-2 /home/pp23/share/hw3-2/cases/c21.1 B32.out
==123134== Profiling application: ./hw3-2 /home/pp23/share/hw3-2/cases/c21.1 B32.out
==123134== Profiling result:
==123134== Metric result:
Invocations                               Metric Name                    Metric Description         Min         Max         Avg
Device "NVIDIA GeForce GTX 1080 (0)"
    Kernel: Phase2_(int*, int, int)
        157                           inst_integer                    Integer Instructions    26118656    26118656    26118656
    Kernel: Phase1(int*, int, int)
        157                           inst_integer                    Integer Instructions       78848       78848       78848
    Kernel: Phase3(int*, int, int)
        157                           inst_integer                    Integer Instructions  1563894272  1563894272  1563894272
```

```
[pp23s71@hades01 hw3-2]$ srun -p prof -N1 -n1 --gres=gpu:1 nvprof ./hw3-2 /home/pp23/share/hw3-2/cases/c21.1 B32.out
==222785== NVPROF is profiling process 222785, command: ./hw3-2 /home/pp23/share/hw3-2/cases/c21.1 B32.out
==222785== Profiling application: ./hw3-2 /home/pp23/share/hw3-2/cases/c21.1 B32.out
==222785== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   78.39%  158.09ms       157  1.0070ms  984.43us  1.0314ms  Phase3(int*, int, int)
                    7.79%  15.701ms         1  15.701ms  15.701ms  15.701ms  [CUDA memcpy HtoD]
                    7.66%  15.452ms         1  15.452ms  15.452ms  15.452ms  [CUDA memcpy DtoH]
                    5.48%  11.043ms       157  70.334us  65.345us  73.793us  Phase2_(int*, int, int)
                    0.68%  1.3765ms       157  8.7670us  8.0960us  9.4080us  Phase1(int*, int, int)
      API calls:   65.80%  200.66ms         2  100.33ms  15.724ms  184.94ms  cudaMemcpy
                   33.67%  102.69ms         1  102.69ms  102.69ms  102.69ms  cudaHostRegister
                    0.41%  1.2393ms       471  2.6310us  2.0380us  82.213us  cudaLaunchKernel
                    0.05%  140.89us       101  1.3940us     102ns  61.210us  cuDeviceGetAttribute
                    0.04%  128.59us         1  128.59us  128.59us  128.59us  cudaFree
                    0.03%  86.052us         1  86.052us  86.052us  86.052us  cudaMalloc
                    0.00%  11.598us         1  11.598us  11.598us  11.598us  cuDeviceGetName
                    0.00%  8.4310us         1  8.4310us  8.4310us  8.4310us  cuDeviceGetPCIBusId
                    0.00%  1.1830us         3     394ns     180ns     817ns  cuDeviceGetCount
                    0.00%     704ns         2     352ns     124ns     580ns  cuDeviceGet
                    0.00%     312ns         1     312ns     312ns     312ns  cuModuleGetLoadingMode
                    0.00%     311ns         1     311ns     311ns     311ns  cuDeviceTotalMem
                    0.00%     283ns         1     283ns     283ns     283ns  cuDeviceGetUuid
```

B=16

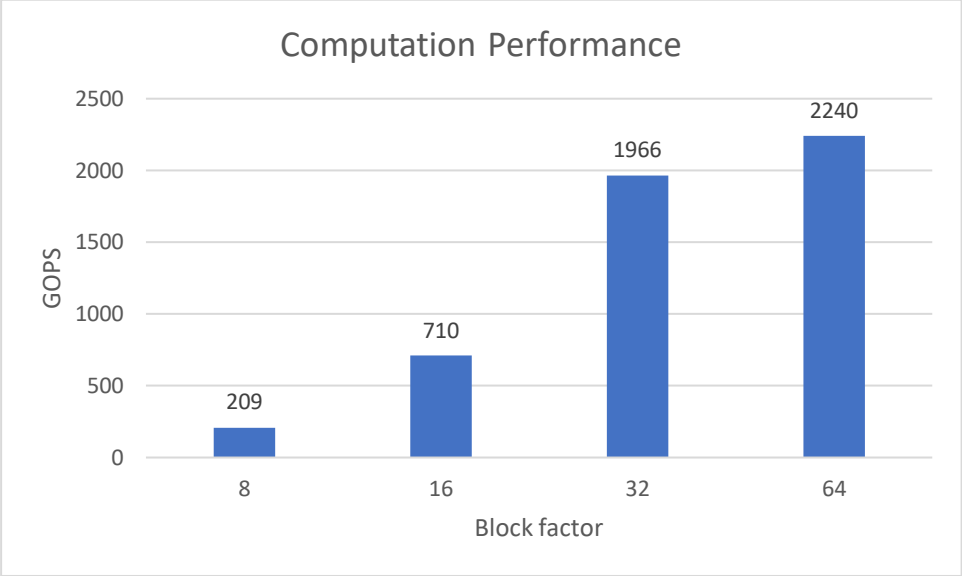
```
[pp23s71@hades01 hw3-2]$ srun -p prof -N1 -n1 --gres=gpu:1 nvprof --metrics inst_integer ./hw3-2 /home/pp23/share/hw3-2/cases/c21.1 B16.out
==158005== NVPROF is profiling process 158005, command: ./hw3-2 /home/pp23/share/hw3-2/cases/c21.1 B16.out
==158005== Profiling application: ./hw3-2 /home/pp23/share/hw3-2/cases/c21.1 B16.out
==158005== Profiling result:
==158005== Metric result:
Invocations                               Metric Name                        Metric Description         Min         Max         Avg
Device "NVIDIA GeForce GTX 1080 (0)"
    Kernel: Phase2_(int*, int, int)
        313                           inst_integer                    Integer Instructions     7308416     7308416     7308416
    Kernel: Phase1(int*, int, int)
        313                           inst_integer                    Integer Instructions       10496       10496       10496
    Kernel: Phase3(int*, int, int)
        313                           inst_integer                    Integer Instructions   866052224   866052224   866052224
```

```
[pp23s71@hades01 hw3-2]$ srun -p prof -N1 -n1 --gres=gpu:1 nvprof ./hw3-2 /home/pp23/share/hw3-2/cases/c21.1 B16.out
==223809== NVPROF is profiling process 223809, command: ./hw3-2 /home/pp23/share/hw3-2/cases/c21.1 B16.out
==223809== Profiling application: ./hw3-2 /home/pp23/share/hw3-2/cases/c21.1 B16.out
==223809== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   91.29%  422.51ms       313  1.3499ms  1.2889ms  1.4326ms  Phase3(int*, int, int)
                    3.41%  15.770ms         1  15.770ms  15.770ms  15.770ms  [CUDA memcpy HtoD]
                    3.31%  15.309ms         1  15.309ms  15.309ms  15.309ms  [CUDA memcpy DtoH]
                    1.67%  7.7137ms       313  24.644us  22.528us  27.137us  Phase2_(int*, int, int)
                    0.33%  1.5164ms       313  4.8440us  4.4160us  5.3760us  Phase1(int*, int, int)
      API calls:   80.50%  460.93ms         2  230.47ms  15.854ms  445.08ms  cudaMemcpy
                   19.03%  108.99ms         1  108.99ms  108.99ms  108.99ms  cudaHostRegister
                    0.40%  2.2719ms       939  2.4190us  2.0280us  84.584us  cudaLaunchKernel
                    0.03%  158.05us         1  158.05us  158.05us  158.05us  cudaFree
                    0.02%  138.60us       101  1.3720us     105ns  60.827us  cuDeviceGetAttribute
                    0.02%  91.753us         1  91.753us  91.753us  91.753us  cudaMalloc
                    0.00%  9.2720us         1  9.2720us  9.2720us  9.2720us  cuDeviceGetName
                    0.00%  8.3130us         1  8.3130us  8.3130us  8.3130us  cuDeviceGetPCIBusId
                    0.00%  1.1020us         3     367ns     149ns     759ns  cuDeviceGetCount
                    0.00%     567ns         2     283ns     125ns     442ns  cuDeviceGet
                    0.00%     547ns         1     547ns     547ns     547ns  cuModuleGetLoadingMode
                    0.00%     338ns         1     338ns     338ns     338ns  cuDeviceTotalMem
                    0.00%     196ns         1     196ns     196ns     196ns  cuDeviceGetUuid
```

B=8

```
[pp23s71@hades01 hw3-2]$ srun -p prof -N1 -n1 --gres=gpu:1 nvprof --metrics inst_integer ./hw3-2 /home/pp23/share/hw3-2/cases/c21.1 B8.out
==223926== NVPROF is profiling process 223926, command: ./hw3-2 /home/pp23/share/hw3-2/cases/c21.1 B8.out
==223926== Profiling application: ./hw3-2 /home/pp23/share/hw3-2/cases/c21.1 B8.out
==223926== Profiling result:
==223926== Metric result:
Invocations                               Metric Name                        Metric Description         Min         Max         Avg
Device "NVIDIA GeForce GTX 1080 (0)"
    Kernel: Phase2_(int*, int, int)
        625                           inst_integer                    Integer Instructions     2076704     2076704     2076704
    Kernel: Phase1(int*, int, int)
        625                           inst_integer                    Integer Instructions        1344        1344        1344
    Kernel: Phase3(int*, int, int)
        625                           inst_integer                    Integer Instructions   566971424   566971424   566971424
```

```
[pp23s71@hades01 hw3-2]$ srun -p prof -N1 -n1 --gres=gpu:1 nvprof ./hw3-2 /home/pp23/share/hw3-2/cases/c21.1 B8.out
==158503== NVPROF is profiling process 158503, command: ./hw3-2 /home/pp23/share/hw3-2/cases/c21.1 B8.out
==158503== Profiling application: ./hw3-2 /home/pp23/share/hw3-2/cases/c21.1 B8.out
==158503== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   97.76%  1.75083s       625  2.8013ms  2.5571ms  2.9828ms  Phase3(int*, int, int)
                    0.87%  15.496ms         1  15.496ms  15.496ms  15.496ms  [CUDA memcpy HtoD]
                    0.85%  15.214ms         1  15.214ms  15.214ms  15.214ms  [CUDA memcpy DtoH]
                    0.41%  7.2831ms       625  11.652us  10.880us  12.928us  Phase2_(int*, int, int)
                    0.11%  2.0435ms       625  3.2690us  3.0720us  3.9360us  Phase1(int*, int, int)
      API calls:   52.10%  985.55ms         2  492.77ms  15.534ms  970.02ms  cudaMemcpy
                   42.60%  805.82ms      1875  429.77us  1.6900us  2.9847ms  cudaLaunchKernel
                    5.28%  99.946ms         1  99.946ms  99.946ms  99.946ms  cudaHostRegister
                    0.01%  161.96us         1  161.96us  161.96us  161.96us  cudaFree
                    0.01%  120.33us       101  1.1910us      91ns  52.429us  cuDeviceGetAttribute
                    0.00%  71.254us         1  71.254us  71.254us  71.254us  cudaMalloc
                    0.00%  8.1560us         1  8.1560us  8.1560us  8.1560us  cuDeviceGetName
                    0.00%  6.3610us         1  6.3610us  6.3610us  6.3610us  cuDeviceGetPCIBusId
                    0.00%     916ns         3     305ns     150ns     598ns  cuDeviceGetCount
                    0.00%     648ns         2     324ns     115ns     533ns  cuDeviceGet
                    0.00%     454ns         1     454ns     454ns     454ns  cuModuleGetLoadingMode
                    0.00%     314ns         1     314ns     314ns     314ns  cuDeviceTotalMem
                    0.00%     174ns         1     174ns     174ns     174ns  cuDeviceGetUuid
```



從圖中可發現隨著 blocking factor 增加，每秒的 operation 也在增加，推測這是因為並行化程度增加，同時進行的計算操作變多的緣故。

global/shared memory bandwidth

B64*64

```
[pp23s71@hades01 hw3-2]$ srun -p prof -N1 -n1 --gres=gpu:1 nvprof --metrics shared_store_throughput,shared_load_throughput,gld_throughput,gst_throughput ./hw3-2 /home/pp23/share/hw3-2/cases/c21.1 B64_64.out
srun: job 803094 queued and waiting for resources
srun: job 803094 has been allocated resources
==226715== NVPROF is profiling process 226715, command: ./hw3-2 /home/pp23/share/hw3-2/cases/c21.1 B64_64.out
==226715== Some kernel(s) will be replayed on device 0 in order to collect all events/metrics.
==226715== Profiling application: ./hw3-2 /home/pp23/share/hw3-2/cases/c21.1 B64_64.out
==226715== Profiling result:
==226715== Metric result:
Invocations                               Metric Name                        Metric Description         Min         Max         Avg
Device "NVIDIA GeForce GTX 1080 (0)"
    Kernel: Phase2_(int*, int, int)
          79                    shared_store_throughput           Shared Memory Store Throughput  913.39GB/s  1051.4GB/s  1032.6GB/s
          79                     shared_load_throughput            Shared Memory Load Throughput  2228.1GB/s  2564.8GB/s  2519.0GB/s
          79                             gld_throughput                   Global Load Throughput  27.678GB/s  31.861GB/s  31.292GB/s
          79                             gst_throughput                  Global Store Throughput  13.839GB/s  15.931GB/s  15.646GB/s
    Kernel: Phase1(int*, int, int)
          79                    shared_store_throughput           Shared Memory Store Throughput  37.464GB/s  42.869GB/s  42.309GB/s
          79                     shared_load_throughput            Shared Memory Load Throughput  111.24GB/s  127.29GB/s  125.63GB/s
          79                             gld_throughput                   Global Load Throughput  590.20MB/s  675.35MB/s  666.53MB/s
          79                             gst_throughput                  Global Store Throughput  590.20MB/s  675.35MB/s  666.53MB/s
    Kernel: Phase3(int*, int, int)
          79                    shared_store_throughput           Shared Memory Store Throughput  249.06GB/s  260.08GB/s  258.23GB/s
          79                     shared_load_throughput            Shared Memory Load Throughput  3051.0GB/s  3186.0GB/s  3163.3GB/s
          79                             gld_throughput                   Global Load Throughput  186.80GB/s  195.06GB/s  193.67GB/s
          79                             gst_throughput                  Global Store Throughput  62.266GB/s  65.020GB/s  64.558GB/s
```

B32*32

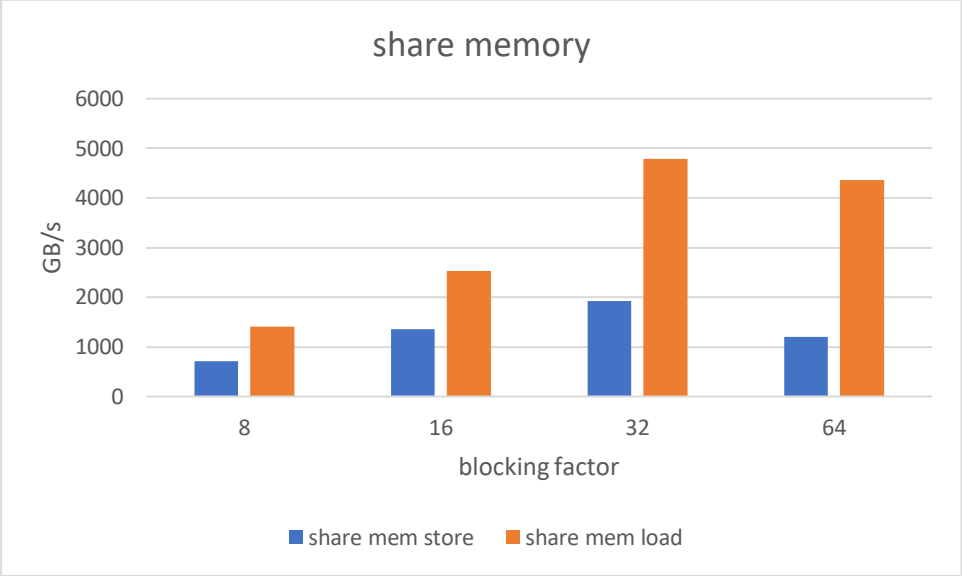
```
[pp23s71@hades01 hw3-2]$ srun -p prof -N1 -n1 --gres=gpu:1 nvprof --metrics shared_store_throughput,shared_load_throughput,gld_throughput,gst_throughput ./h
w3-2 /home/pp23/share/hw3-2/cases/c21.1 B32_32.out
==227347== NVPROF is profiling process 227347, command: ./hw3-2 /home/pp23/share/hw3-2/cases/c21.1 B32_32.out
==227347== Some kernel(s) will be replayed on device 0 in order to collect all events/metrics.
==227347== Profiling application: ./hw3-2 /home/pp23/share/hw3-2/cases/c21.1 B32_32.out
==227347== Profiling result:
==227347== Metric result:
Invocations                               Metric Name                         Metric Description         Min         Max         Avg
Device "NVIDIA GeForce GTX 1080 (0)"
    Kernel: Phase2_(int*, int, int)
        157                    shared_store_throughput            Shared Memory Store Throughput  1189.8GB/s  1266.2GB/s  1227.2GB/s
        157                     shared_load_throughput             Shared Memory Load Throughput  2414.6GB/s  2569.5GB/s  2490.6GB/s
        157                             gld_throughput                    Global Load Throughput  34.995GB/s  37.240GB/s  36.095GB/s
        157                             gst_throughput                   Global Store Throughput  17.497GB/s  18.620GB/s  18.048GB/s
    Kernel: Phase1(int*, int, int)
        157                    shared_store_throughput            Shared Memory Store Throughput  30.378GB/s  33.480GB/s  31.874GB/s
        157                     shared_load_throughput             Shared Memory Load Throughput  74.563GB/s  82.178GB/s  78.236GB/s
        157                             gld_throughput                    Global Load Throughput  471.31MB/s  519.45MB/s  494.53MB/s
        157                             gst_throughput                   Global Store Throughput  471.31MB/s  519.45MB/s  494.53MB/s
    Kernel: Phase3(int*, int, int)
        157                    shared_store_throughput            Shared Memory Store Throughput  707.15GB/s  725.08GB/s  717.46GB/s
        157                     shared_load_throughput             Shared Memory Load Throughput  2298.2GB/s  2356.5GB/s  2331.8GB/s
        157                             gld_throughput                    Global Load Throughput  265.18GB/s  271.91GB/s  269.05GB/s
        157                             gst_throughput                   Global Store Throughput  88.394GB/s  90.635GB/s  89.683GB/s
```

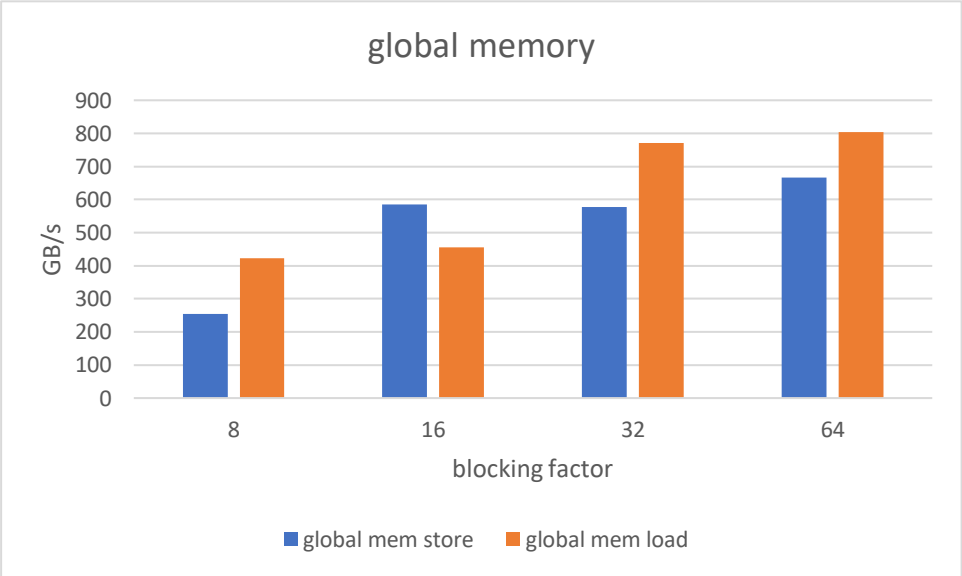
B16*16

```
[pp23s71@hades01 hw3-2]$ srun -p prof -N1 -n1 --gres=gpu:1 nvprof --metrics shared_store_throughput,shared_load_throughput,gld_throughput,gst_throughput ./h
w3-2 /home/pp23/share/hw3-2/cases/c21.1 B16_16.out
==160848== NVPROF is profiling process 160848, command: ./hw3-2 /home/pp23/share/hw3-2/cases/c21.1 B16_16.out
==160848== Some kernel(s) will be replayed on device 0 in order to collect all events/metrics.
==160848== Profiling application: ./hw3-2 /home/pp23/share/hw3-2/cases/c21.1 B16_16.out
==160848== Profiling result:
==160848== Metric result:
Invocations                               Metric Name                         Metric Description         Min         Max         Avg
Device "NVIDIA GeForce GTX 1080 (0)"
    Kernel: Phase2_(int*, int, int)
        313                    shared_store_throughput            Shared Memory Store Throughput  782.56GB/s  904.70GB/s  867.49GB/s
        313                     shared_load_throughput             Shared Memory Load Throughput  1521.6GB/s  1759.1GB/s  1686.8GB/s
        313                             gld_throughput                    Global Load Throughput  43.476GB/s  50.261GB/s  48.194GB/s
        313                             gst_throughput                   Global Store Throughput  43.476GB/s  50.261GB/s  48.194GB/s
    Kernel: Phase1(int*, int, int)
        313                    shared_store_throughput            Shared Memory Store Throughput  6.6663GB/s  8.1881GB/s  7.6793GB/s
        313                     shared_load_throughput             Shared Memory Load Throughput  16.078GB/s  19.748GB/s  18.521GB/s
        313                             gld_throughput                    Global Load Throughput  200.77MB/s  246.61MB/s  231.28MB/s
        313                             gst_throughput                   Global Store Throughput  401.55MB/s  493.21MB/s  462.57MB/s
    Kernel: Phase3(int*, int, int)
        313                    shared_store_throughput            Shared Memory Store Throughput  563.57GB/s  603.74GB/s  580.49GB/s
        313                     shared_load_throughput             Shared Memory Load Throughput  986.25GB/s  1056.5GB/s  1015.9GB/s
        313                             gld_throughput                    Global Load Throughput  211.34GB/s  226.40GB/s  217.68GB/s
        313                             gst_throughput                   Global Store Throughput  140.89GB/s  150.94GB/s  145.12GB/s
```

B8*8

```
[pp23s71@hades01 hw3-2]$ srun -p prof -N1 -n1 --gres=gpu:1 nvprof --metrics shared_store_throughput,shared_load_throughput,gld_throughput,gst_throughput ./h
w3-2 /home/pp23/share/hw3-2/cases/c21.1 B8_8.out
==161430== NVPROF is profiling process 161430, command: ./hw3-2 /home/pp23/share/hw3-2/cases/c21.1 B8_8.out
==161430== Some kernel(s) will be replayed on device 0 in order to collect all events/metrics.
==161430== Profiling application: ./hw3-2 /home/pp23/share/hw3-2/cases/c21.1 B8_8.out
==161430== Profiling result:
==161430== Metric result:
Invocations                               Metric Name                         Metric Description         Min         Max         Avg
Device "NVIDIA GeForce GTX 1080 (0)"
    Kernel: Phase2_(int*, int, int)
        625                    shared_store_throughput            Shared Memory Store Throughput  464.92GB/s  524.96GB/s  495.99GB/s
        625                     shared_load_throughput             Shared Memory Load Throughput  976.32GB/s  1102.4GB/s  1041.6GB/s
        625                             gld_throughput                    Global Load Throughput  92.983GB/s  104.99GB/s  99.198GB/s
        625                             gst_throughput                   Global Store Throughput  46.492GB/s  52.496GB/s  49.599GB/s
    Kernel: Phase1(int*, int, int)
        625                    shared_store_throughput            Shared Memory Store Throughput  1.2895GB/s  1.5778GB/s  1.5216GB/s
        625                     shared_load_throughput             Shared Memory Load Throughput  3.5820GB/s  4.3827GB/s  4.2267GB/s
        625                             gld_throughput                    Global Load Throughput  146.72MB/s  179.52MB/s  173.12MB/s
        625                             gst_throughput                   Global Store Throughput  146.72MB/s  179.52MB/s  173.12MB/s
    Kernel: Phase3(int*, int, int)
        625                    shared_store_throughput            Shared Memory Store Throughput  244.58GB/s  258.12GB/s  250.31GB/s
        625                     shared_load_throughput             Shared Memory Load Throughput  428.02GB/s  451.72GB/s  438.04GB/s
        625                             gld_throughput                    Global Load Throughput  183.44GB/s  193.59GB/s  187.73GB/s
        625                             gst_throughput                   Global Store Throughput  61.146GB/s  64.531GB/s  62.577GB/s
```



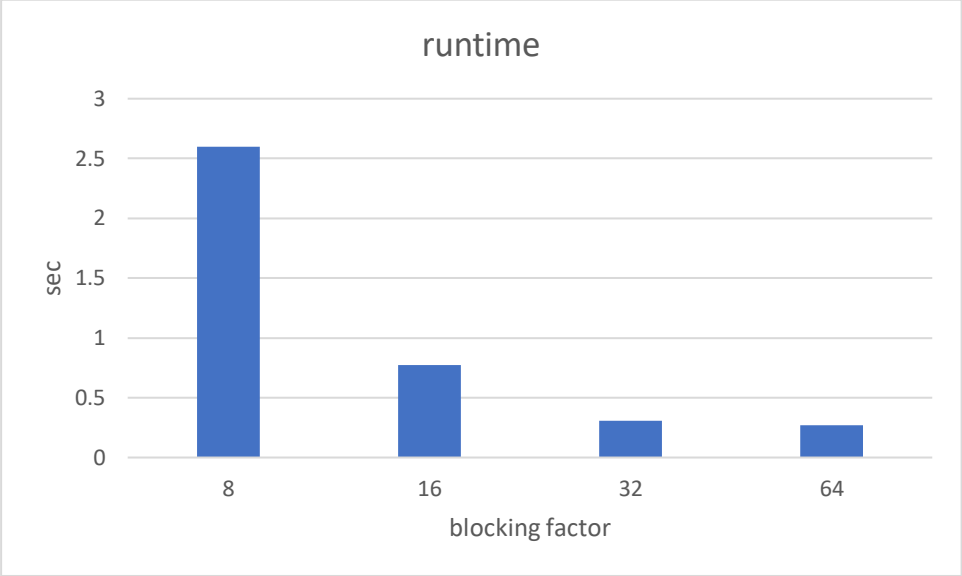


由於程式中對於 share memory 的存取計算，Share memory store 隨著 blocking factor 增加有非常明顯的差異，幅度上千，但 share memory 也有數量限制，因此不會一直上升。global 部份則沒有那麼劇烈的變化。

c. Optimization (hw3-2)

1. Large blocking factor

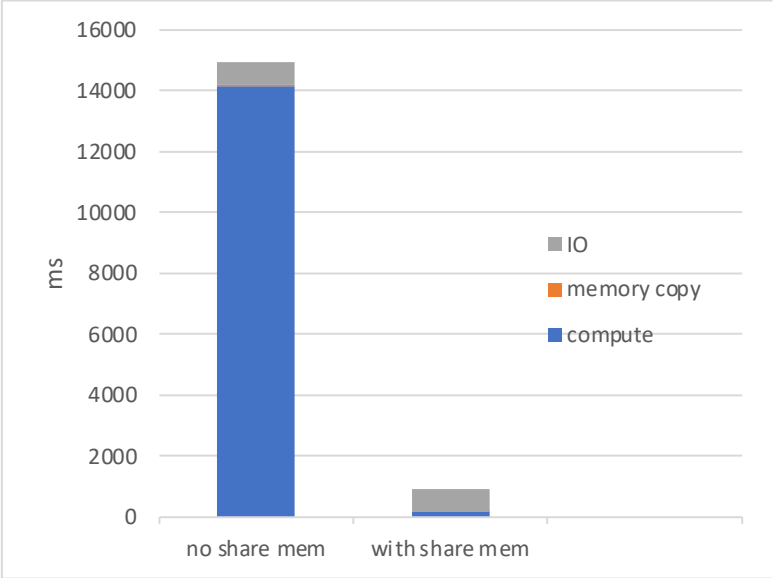
c21.1 進行測試四種 blocking factor 下的執行時間



較大的 blocking factor 的確可以明顯降低運行時間

2. share memory

使用 share memory 可以達到很好的優化效果，對 c21.1 比較使用 share memory 還有 no share memory，兩者差距是巨量的。



3. 比較計算優化

以 p16k1 V=16000 測資，在 block-floyd-warshall 計算時比較使用__device__函數

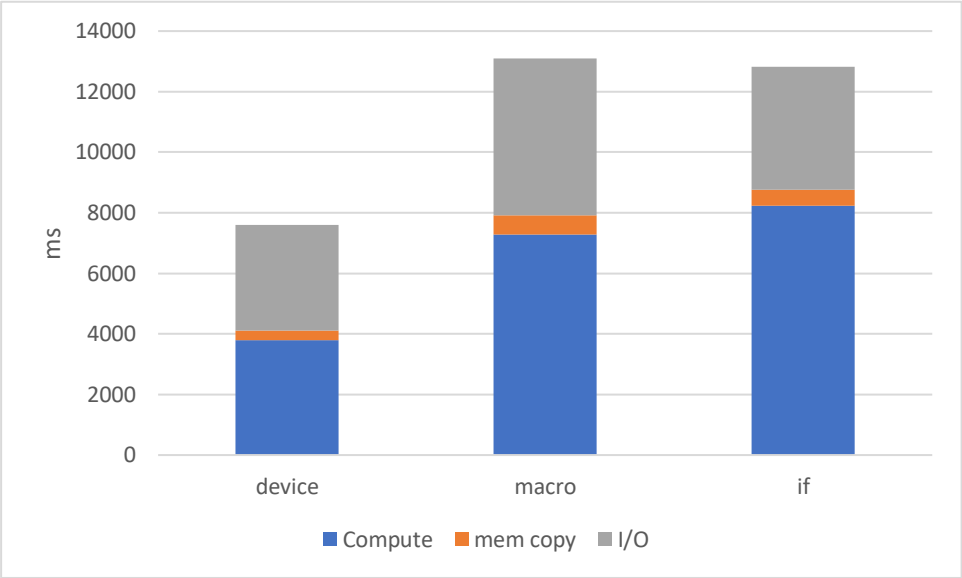
```
__device__ int min_(int a, int b) {return min(a, b);}
```

、在 GPU 上執行再調用 cuda 內建函數 min，以及直接使用 if

還有在#define 定義一個 macro：

#define min__(a, b) ((a) < (b) ? (a) : (b))

三種方式進行比較：



測量方式是使用 cudaEvent 計算每部分累計時間。可以清楚看到使用__device__與 cuda 內建 min 函數在 GPU 上執行是最快的，而以#define 寫一個比較大小的 macro 還有 if 則慢很多，差異接近兩倍。

4. IO 優化

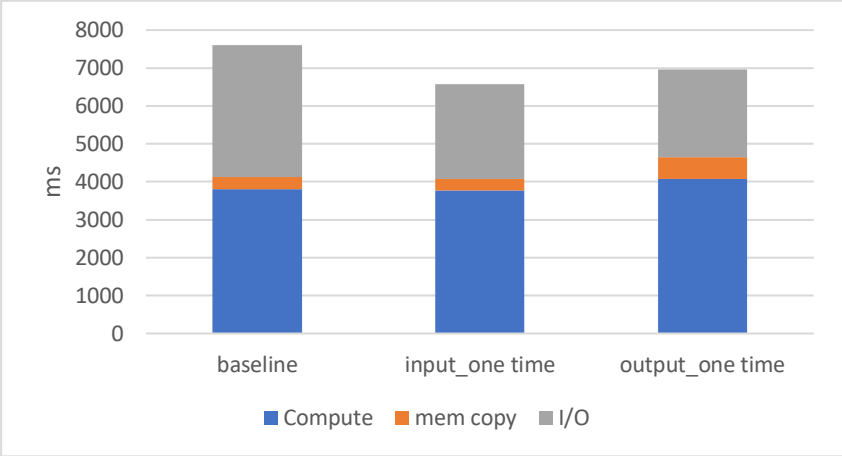
另外進行了 IO 相關優化實驗，同樣使用 p16k1 測資，測量方式是使用 cudaEvent 計算每部分累計時間。

Baseline 是使用原來助教範例程式中的 input 與 output 程式

Input_one time 是使用

int *fpt = (int*)mmap(NULL, 2*sizeof(int), PROT_READ, MAP_PRIVATE, file, 0);直接將所有資料一次性讀取，之後再寫入 V*V 的 Dist 中，

而 output_one time 則是在先三個 phase 結束後先在 gpu 上用 devicetodevice 的 CudaMemcpy 把本來 V*V 存在 dst 的資料複製到另一個 vertex*vertex 大小的 dst_， 之後 output 時就可以直接 fwrite(Dist, sizeof(int), vertex*vertex, outfile);一次性把 vertex*vertex 量的資料全部寫入 output file 中減少 I/O 時間。

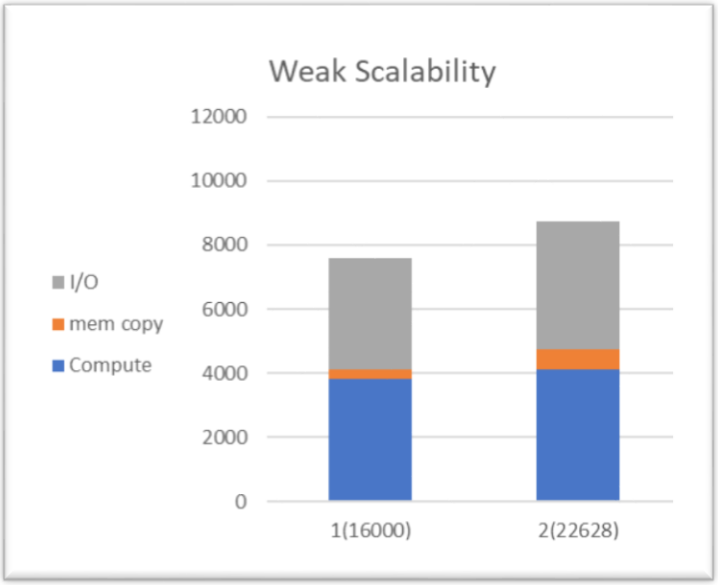


結果可看出 IO 時間都有降低，尤其 input_one time 十分顯著，output_one time 雖然也講了不少 IO 時間，但因為要做更多的 CudaMemcpy 所以 memory copy 時間也有增長。

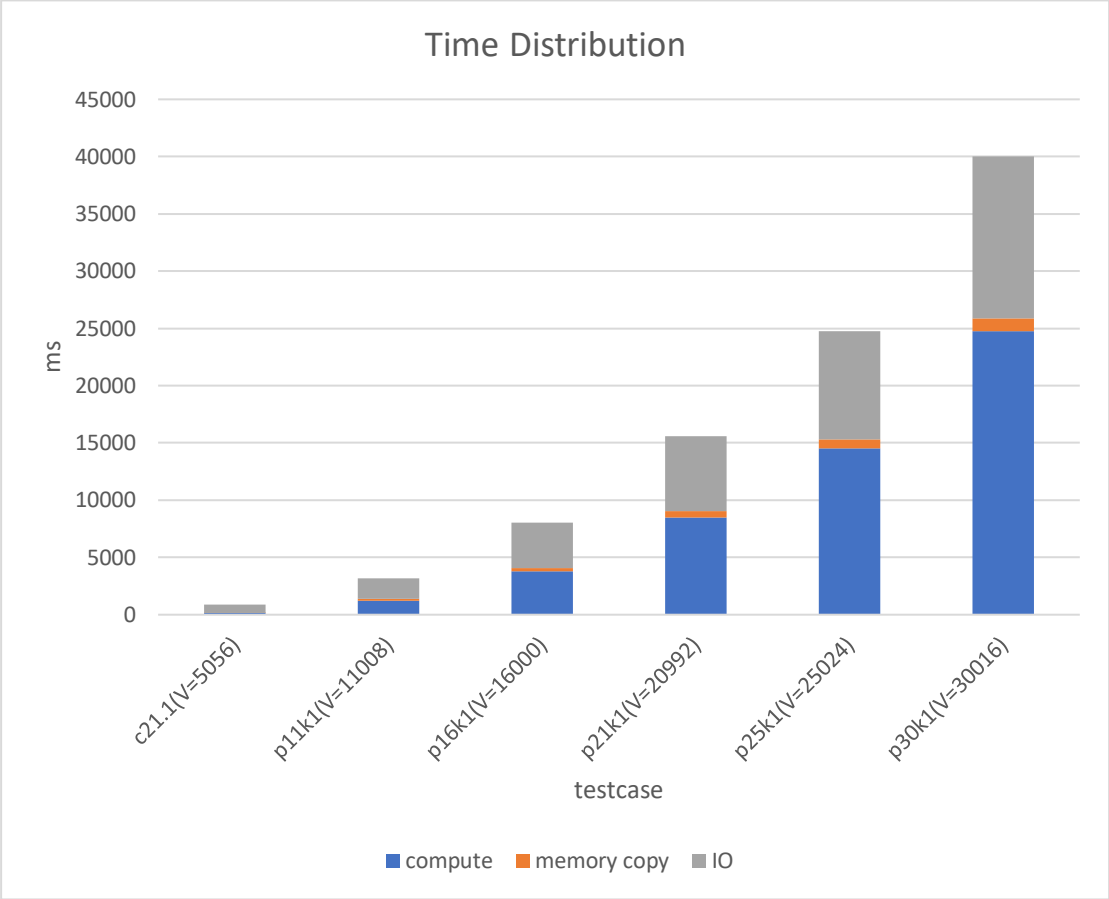
d. Weak scalability (hw3-3)

Weak scalability 是在每個計算單元計算量相同時比較不同數量計算單元能否在差不多時間內完成任務，在此使用數量為 1 與 2 的 GPU 進行測試，直接修改程式內 V 的值測試計算時間。測量方式是使用 cudaEvent 計算每部分累計時間。兩個 GPU 計算量應為一個的兩倍(V*V 兩倍)，故 V 應為$\sqrt{2}$倍，這裡單 GPU 使用 16000 與其$\sqrt{2}$倍也就是雙 GPU V 為約 22628 比較，單位為 ms。

可看出雙 GPU 仍高出單 GPU 一些時間，執行時間約為單 GPU 1.2~1.3 倍，可發現有較明顯增長的是 I/O 以及 memory copy，I/O 方面已經有使用一次性 mmap 毒入來節省時間，但後續將內容寫入 Dist 應仍會造成時間差異，此外雙 GPU 運行時每個迴圈要進行兩個 GPU 間 CudaMemcpy，這也造成 memory copy 時間明顯增長，而 compute 時間因為有兩個 GPU 分工是增長較不明顯的。



e. Time Distribution (hw3-2)



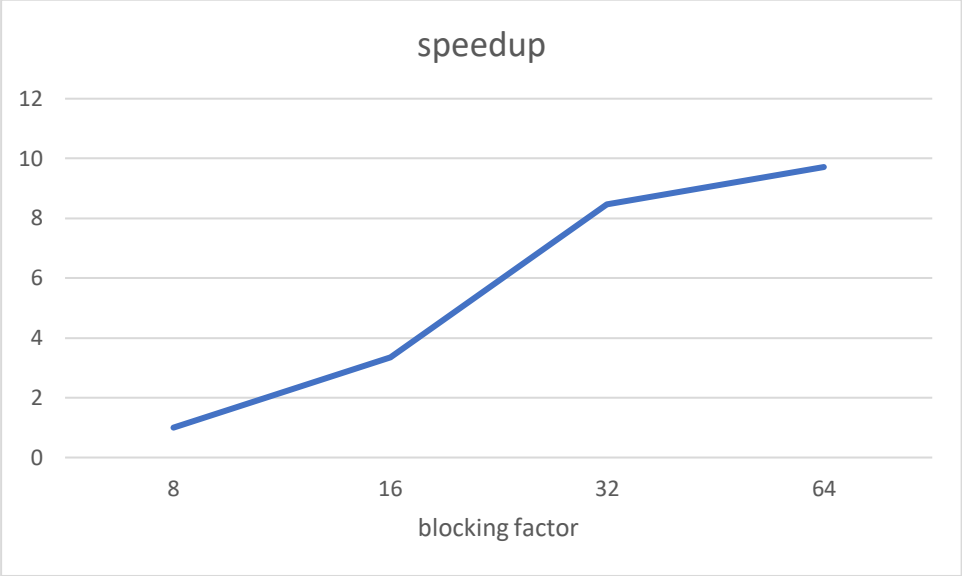
測試六個V值不同的testcase，測量方式是使用cudaEvent計算每部分累計時間。可看出Memory copy時間是佔比最少的，其上漲幅度也很低，主因是memory copy時間主要在執行cudaMemcpy時，此函數主要是在copy V*V的資料量，因此只和input問題的節點數量有關。

接著是I/O時間，I/O時間，同樣來自於input問題大小，由於input, output兩個函數要不斷進行fread,fwrite，其成長幅度高於memory copy。

最後是compute time，compute time不只和input節點數量有關，也和input距離矩陣內容有關，其內容會影響計算量，其時間上升幅度也是三者中最大的。

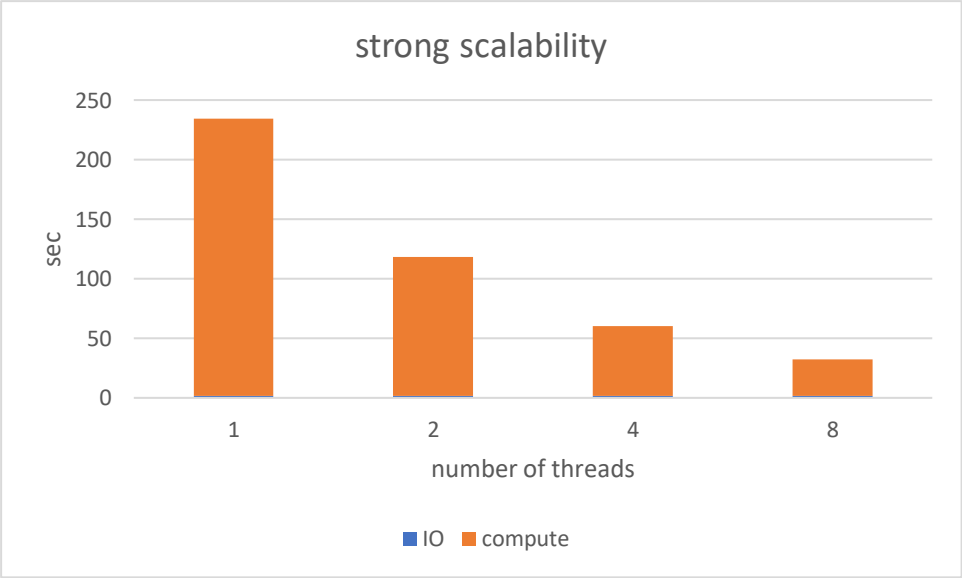
f. Others

1. 在 large blocking factor 部分可看出 large blocking factor 的確有益於執行時間，這裡進一步觀察 speedup 程度



其實我本來以為 blocking factor 的 speedup 理想應該會和 block 大小成比，也就是 16*16 會是 8*8 四倍…以此類推，但實驗做出來明顯不是這樣，但整體 speedup 其實仍算有維持差不多的斜率。

2. 對 hw3-1 部分有花點時間驗證其 strong scalability，使用 c20.1 測 edge:7594839 vertex:5000，可以看到 strong scalability 因為有 omp 平行化 Floyd Warshal 外層兩個迴圈關係表現很好。



Experience & Conclusion

本作業花很多時間在後面兩個 cuda 的部分上面，
在第一部分，其實很多測資都是屬於 edge 稀疏的矩陣，理論上 Floyd-Warshal 時間複雜度 $O(n^3)$，並不適合 edge 稀疏的矩陣，可能用 Johnson 更適合，但

Johnson 的實作複雜且不易加入平行化，我最終還是先選擇 Floyd-Warshal+平行化通過 judge，想說之後再回來實驗 Johnson，結果後來就花了大量時間在後面兩個關於 cuda 的部分上，沒時間回來弄 Johnson 了。
不過在 cuda 上面我覺得學習 cuda 程式帶給我一種寫程式時的「空間感」，好像以前寫的程式都是線性的，以前平行化想的都只是如何把 for 迴圈拆分。如今突然變成二維甚至三維的感覺，進入另一個新世界，還學到 share memory、注意 bank conflict 概念等等，一口氣前進好多步。