# Parallel Programming HW3 report

# Blocked floyd warshall

蔡茗鈞
110030014

---

## Implemetation

a. Which algorithm do you choose in hw3-1?
   sequential version of blocked floyd warshall.

b. How do you divide your data in hw3-2, hw3-3?
   將dist matrix以block size 32*32 分割成數個block, 分給kernel的block執行
   若n不為32的倍數, 將n補至32的倍數, 並在多出來的部分填入INF(padding)。

c. What's your configuration in hw3-2, hw3-3? And why? (e.g. blocking factor, #blocks, #threads)
   blocking factor = 32, 因為kernel的max thread per block為1024(32*32)
   #block = (n/32)^2 (此時n為32的倍數)
   #threads = 32*32 (per block)

d. How do you implement the communication in hw3-3?
   一開始兩個gpu先各自利用cudaMemcpy(, , cudaMemcpyHostToDevice);
   取得上半部與下半部的dist matrix, 進入round的for loop之後的每個 iteration一開始, 利用cudaMemcpyPeer間gpu之間傳送, 各自取得剩下一半的dist matrix再開始計算, 因為iteration r, phase3只會計算一半的blocks (gpu 0 算上半部, gpu1計算下半部)。

e. Briefly describe your implementations in diagrams, figures or sentences.
   n = padding後的vertex數
   B = 32
   round = n/B
   每個round r分成phase1, phase2 phase3。
   phase1 計算block(r,r) (1個block)

phase2計算除了block(r,r)以外，row = r或column= r的所有block(2*(round-2)個block)
phase3計算剩餘的所有block((round-1)^2個block)
每個phase的每個block將各自的dependency blocks從global memory複製進per block shared memory.
phase1的dependcy block只有block(r,r)
phase2的dependency block有自身的block(i,j)與block(r,r)
phase3的dependemcy block有自身的block(i,j)與block(r,j)和block(i,r)

# Profiling Results (hw3-2)
# on testcase c21.3, n=5000 , kernel = phase3

a. global load/ store throughput (Min Max Avg)
Global Load Throughput 15.607GB/s 15.739GB/s 15.726GB/s

```
Invocations                               Metric Name                         Metric Description         Min         Max         Avg
Device "NVIDIA GeForce GTX 1080 (0)"
    Kernel: phase1(int*, int, int)
          157                            gld_throughput                   Global Load Throughput  304.41MB/s  344.83MB/s  319.46MB/s
    Kernel: phase3(int*, int, int, int)
          157                            gld_throughput                   Global Load Throughput  15.607GB/s  15.739GB/s  15.726GB/s
    Kernel: phase2(int*, int, int, int, bool)
          314                            gld_throughput                   Global Load Throughput  12.068GB/s  13.398GB/s  13.072GB/s
```

Global Store Throughput 24.590GB/s 24.677GB/s 24.651GB/s

```
Invocations                               Metric Name                         Metric Description         Min         Max         Avg
Device "NVIDIA GeForce GTX 1080 (0)"
    Kernel: phase1(int*, int, int)
          157                            gst_throughput                   Global Store Throughput  544.96MB/s  589.71MB/s  583.64MB/s
    Kernel: phase3(int*, int, int, int)
          157                            gst_throughput                   Global Store Throughput  24.590GB/s  24.677GB/s  24.651GB/s
    Kernel: phase2(int*, int, int, int, bool)
          314                            gst_throughput                   Global Store Throughput  19.889GB/s  22.733GB/s  22.129GB/s
```

b. shared memory load/ store throughput (Min Max Avg)
Shared Memory Load Throughput 1578.3GB/s 1585.9GB/s 1584.7GB/s

```
Invocations                               Metric Name                         Metric Description         Min         Max         Avg
Device "NVIDIA GeForce GTX 1080 (0)"
    Kernel: phase1(int*, int, int)
          157                     shared_load_throughput              Shared Memory Load Throughput  51.090GB/s  55.825GB/s  54.865GB/s
    Kernel: phase3(int*, int, int, int)
          157                     shared_load_throughput              Shared Memory Load Throughput  1578.3GB/s  1585.9GB/s  1584.7GB/s
    Kernel: phase2(int*, int, int, int, bool)
          314                     shared_load_throughput              Shared Memory Load Throughput  1087.9GB/s  1462.1GB/s  1419.6GB/s
n = 5000
```

Shared Memory Store Throughput 49.272GB/s 54.807GB/s 52.072GB/s

```
Invocations                               Metric Name                         Metric Description         Min         Max         Avg
Device "NVIDIA GeForce GTX 1080 (0)"
    Kernel: phase1(int*, int, int)
          157                    shared_store_throughput             Shared Memory Store Throughput  8.4238GB/s  21.033GB/s  19.499GB/s
    Kernel: phase3(int*, int, int, int)
          157                    shared_store_throughput             Shared Memory Store Throughput  49.272GB/s  54.807GB/s  52.072GB/s
    Kernel: phase2(int*, int, int, int, bool)
          314                    shared_store_throughput             Shared Memory Store Throughput  38.582GB/s  50.950GB/s  46.546GB/s
n = 5000
```

c. occupancy (Min Max Avg)
Achieved Occupancy 0.975517 0.975901 0.975632

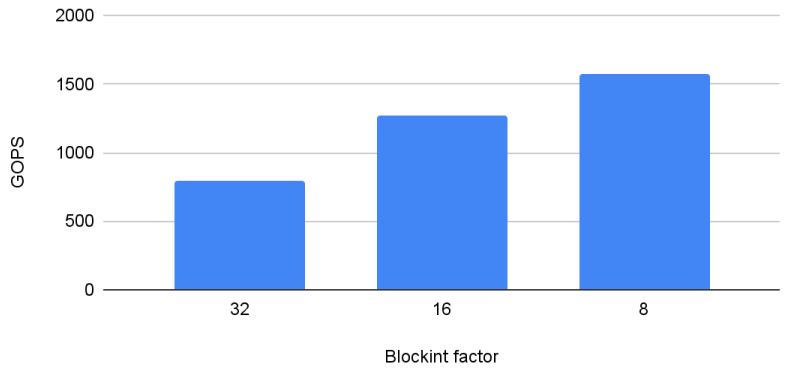
```
Invocations                               Metric Name                         Metric Description         Min         Max         Avg
Device "NVIDIA GeForce GTX 1080 (0)"
    Kernel: phase1(int*, int, int)
        157                         achieved_occupancy                    Achieved Occupancy    0.497012    0.497279    0.497118
    Kernel: phase3(int*, int, int, int)
        157                         achieved_occupancy                    Achieved Occupancy    0.975517    0.975901    0.975632
    Kernel: phase2(int*, int, int, int, bool)
        314                         achieved_occupancy                    Achieved Occupancy    0.932410    0.947622    0.938690
n = 5000
```

d. sm efficiency (Min Max Avg)
Multiprocessor Activity 99.62% 99.92% 99.89%

```
Invocations                               Metric Name                         Metric Description         Min         Max         Avg
Device "NVIDIA GeForce GTX 1080 (0)"
    Kernel: phase1(int*, int, int)
        157                          sm_efficiency                  Multiprocessor Activity       3.41%       3.63%       3.57%
    Kernel: phase3(int*, int, int, int)
        157                          sm_efficiency                  Multiprocessor Activity      99.62%      99.92%      99.89%
    Kernel: phase2(int*, int, int, int, bool)
        314                          sm_efficiency                  Multiprocessor Activity      88.11%      89.96%      89.28%
```

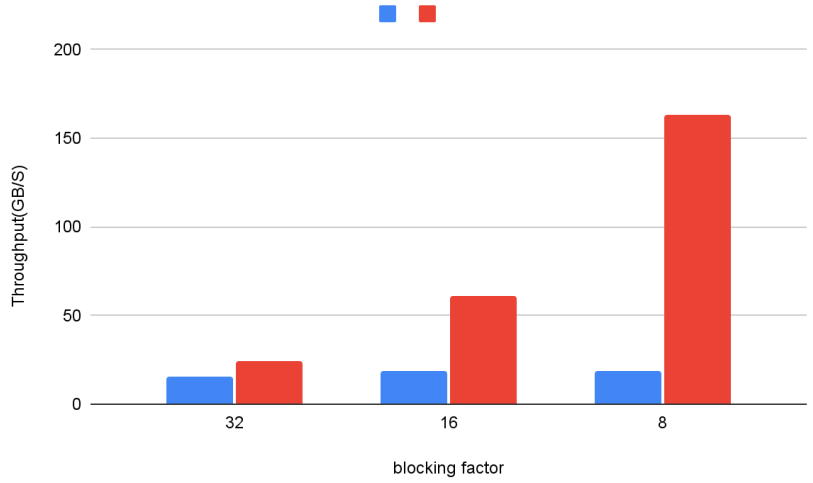
# Experiment & Analysis(hw3-2)
# on testcase c21.3, n=5000

a. System Spec: Hades
b. Blocking Factor (hw3-2)
GOPS
利用--metric instr_integer取得每個kernel得平均instruction count
然後計算 GOPS =
(#phase1_instr*#phase1_calls + #phase2_instr*#phase2_calls + #phase3_instr*#phase3_calls)/(phase1_totaltime + phase2_totaltime + phase3_totaltime)



Global memory throughput ( kernel = phase3)

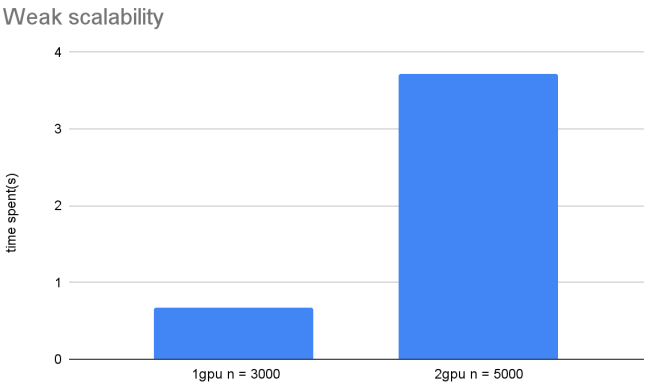
紅色:gloabl memory store 藍色:global memory load




c. Optimization(hw3-2)

   a. shared memory:
      for each block copy its dependency block to per block shared memory
   b. Reduce communication time:
      use cudaHostRegister(Dist, n*n*sizeof(int), cudaHostRegisterDefault);
      to pinned host memory for faster memcpy
   c. Handle bank conflict:
      use int j = threadIdx.x; int i = threadIdx.y; instead of
      int j = threadIdx.y;int i = threadIdx.x; to reduce bank conflict

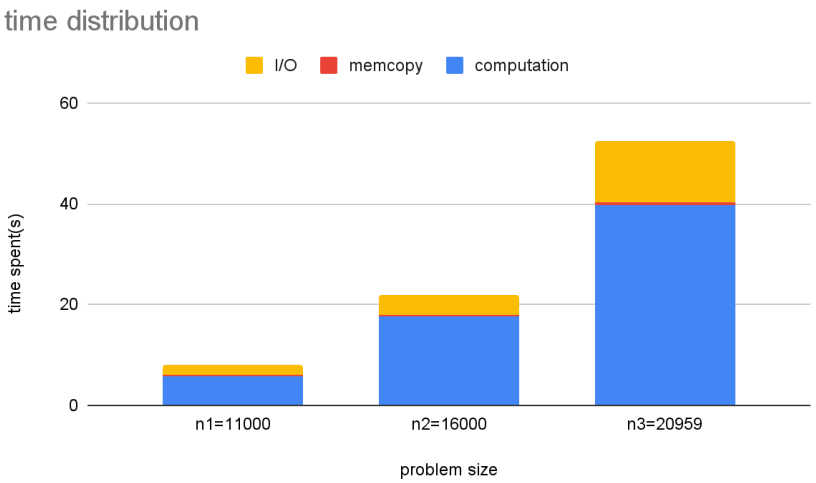
d. Weak Scalability(hw3-3)




(5000/3000)^2 ≈ 2

the time significantly increased when i used 2 gpus

e. Time Distribution(hw3-2)




computation is the main bottleneck for my program

## Experience & conclusion

Through this homework I learnt how to better utilize gpu to compute these kinds of computation. Although I didn't manage to perfect the result, I have some ideas to improve. For example, although I set the thread per block to 1024, which is the maximum number to set, I didn't fully use the shared memory, which is the first thing I would try to improve if I had to.