CS542200 Parallel Programming
Homework 4: FlashAttention
Due: Dec 7 23:59, 2025

name: 莊岱倫 student ID:114062616

1. Implementation

1.a FlashAttention Forward Pass CUDA 實作概述

整體流程描述

1. 載入 **Q row tile**（每個 **thread** 處理一小段 **d** 維度）
    - 使用 float4 vectorized load，加快 Q 的載入速度。
    - 同一 warp 中分成 2 個 group（16 threads/group），每個 group 計算 1 row（總共兩行：my_row_0, my_row_1）。
2. 迴圈掃過所有 **K/V column tiles**（共 **tc = N/bc** 個 **tiles**）
    - 每次處理一塊 bc × d 的 K/V 子矩陣
    - 把它們載入 shared memory
    - 同一 tile 中每個 key 向量依序被拿來與 Q 做 dot product + softmax streaming update
3. 採用 **streaming softmax**（**row-wise**）更新：**m**、**ℓ**、**O row**
    - 計算 score（Q·K）使用warp shuffle

```
score = dot(q_vec[row], k_val);
```

    - 更新 row-wise 最大值 m

```
m_new = max(m_prev, score);
```

    - 更新 ℓ（denominator accumulation）

```
exp_s = exp(score - m_new);
exp_m = exp(m_prev - m_new);
l[row] = l[row] * exp_m + exp_s;
```

    - 更新未歸一化的 O row

```
o_vec[row] *= exp_m;       // 舊的整行 O 按比例縮小
```

```
o_vec[row] += exp_s * v_val;  // 加上新的權重 * v
```

- 所有 K/V tiles 處理完後進行 final normalize

```
float inv_l = 1.0f / l[row];
o_vec[row] *= inv_l;
```

4. 迴圈結束後寫回最終結果：**O_row = O_row / ℓ**

1.b Q/K/V Block Partitioning

Q 的 tile (br × d)

- 每個 block 處理 br=128 個 Query rows
- 每個 warp 負責 4 rows

K/V 的 tile (bc × d)

- 每次迭代載入 bc=32 行的 K 與 V
- 這 32 行資料存放於 shared memory：
  - s_K 大小 = bc × d
  - s_V 大小 = bc × d

都使用 float4 vectorized load，因此每次載入 4 float。

1.c Blocking Factors B_r 與 B_c 的選擇理由

- Blocking factor的選擇我是根據實驗選出，br = 128 bc =32在讓KV重用性跟整體sm資源利用率達到平衡，可以大量使用很快的shared memory中的K V tile去做內積，並且也讓compute & load之間的時間有較好的匹配，所以整體pipeline不會因為資料不夠而暫停IPC也達到4的數值，整體平行度高且沒有bubble

1.d CUDA Kernel Launch Configurations

- Threads per block

  blockDim = (128 / 4) * 32 = 32 * 32 = 1024 threads per block

- Grid dimensions（依 batch × Q_tiles × K_tiles）

  不同testcase有不同 B N

int tr = (N + 128 - 1) / 128;

dim3 grid(B * 32);

- Shared memory 分配：

  | K_tile (bc × d floats) | V_tile (bc × d floats) |

  shared_mem = (2 * bc * d) * sizeof(float)

  shared(d = 64) = 2 * 32 * 64 * 4 bytes = 16KB

  shared(d = 32) = 2 * 32 * 32 * 4 bytes = 8KB

- Register 數：54

1.e Justification

- block 大小如何符合 shared memory 限制

  在d = 64的情況shared memory allocate = 16KB < 48KB

  不只符合GTX 1080的最大容量限制甚至因為只放了1/3, 所以有機會三個block

  在同個sm一起運行

- 為什麼 chosen B_r/B_c 帶來最佳效能

  經過實驗發現

  br=128

  一個 block 處理更多 Q rows（128 rows）提升 arithmetic intensity也可以

  減少 block scheduling overhead因為我讓每 warp 負責 4 rows → 利用率高

  bc=32

  K/V tile 足夠大（32 keys）讓資料重複存取很快的SHM且 shared memory = 16KB, 不會壓低 occupancybc 設大到 64 → warp reuse 不成比例增加導致 time 反而變慢

- 如何避免 shared memory bank conflicts
  我的shared meory layout：

  float4 * s_K_f4 = (float4*)s_K;

  index = t * (d/4) + lane_in_group

  float4 為 16B

shared memory bank size = **4 bytes**所以一個 float 剛好1 **bank** 但我是**verctorize**

一次讀**4 float** 所以跨四個 **banks**

32 threads 同時訪問時:

第 i 個 thread : bank group = (base_addr + i*16) / 4 % 32

因為存取都有對齊bank做連續存取，所以完全沒有bank conflict

- 如何提升 occupancy、IPC

  occupancy

  把SHM 跟 register 用量控制在合理範圍讓 gpu schedular 可以將多個 block 同時在 SM 運行

  IPC

  降低指令之間的相依性，讓越多的指令可以獨立運行在pipeline的不同stage

---

2. Profiling Results

Profile instruction

```
srun nvprof --metrics
achieved_occupancy,sm_efficiency,shared_load_throughput,shared_store_thr
oughput,gld_throughput,gst_throughput ./hw4 ./testcases/t28 ./out
```

```
Invocations                               Metric Name                        Metric Description         Min         Max         Avg
Device "NVIDIA GeForce GTX 1080 (0)"
    Kernel: flash_attention_kernel(float*, float*, float*, float*, int, int, int, int, int, int, int)
          1                        achieved_occupancy                        Achieved Occupancy    0.499988    0.499988    0.499988
          1                             sm_efficiency                   Multiprocessor Activity      98.46%      98.46%      98.46%
          1                     shared_load_throughput              Shared Memory Load Throughput  1355.1GB/s  1355.1GB/s  1355.1GB/s
          1                    shared_store_throughput             Shared Memory Store Throughput  21.173GB/s  21.173GB/s  21.173GB/s
          1                              gld_throughput                    Global Load Throughput  21.255GB/s  21.255GB/s  21.255GB/s
          1                              gst_throughput                   Global Store Throughput  84.691MB/s  84.691MB/s  84.691MB/s
```

2.1 Occupancy 49%

- 這個 kernel 每個 block 1024 threads、每個 thread 用 **54** 個 **register**，所以一個 block 會吃掉大約 55k registers。GTX 1080 每個 SM 只有 64k registers → 一個 **SM** 只能塞 **1** 個 **block**（**1024 threads**），理論上上限 2048 threads，因此實際 occupancy ≈ 50%。

- **Shared memory** 每個 **block** 只用 **16KB**（< 48KB/SM），不是主要瓶頸；真正限制 occupancy 的是 **register usage**，不是 shared memory。

2.2 SM Efficiency 98.46%

- SM efficiency 98.46% 代表 有工作可以執行的時間占比很高，大部分 cycle 都有 warps 在執行，而不是閒著。

- 雖然 occupancy 只有一半，但從 SM efficiency 看起來，現有 **warps** 已經足以把大部分 **memory latency** 藏掉，沒有明顯因為 memory stall 讓 SM 閒置。後續再看其他指標結果也可以看到pipeline並沒有因為memory access有太多stall的狀況

2.3 Shared Memory Throughput

- shared_load_throughput ≈ **1355 GB/s**、shared_store_throughput ≈ **21 GB/s**，讀取量遠大於寫入，符合「tile 到 shared 後被大量讀取」的設計。

- 如果有嚴重 bank conflict，shared throughput 會掉下來而且 SM efficiency 會變差；但現在 shared load throughput 很高、SM efficiency 也接近 100%，根據後續的指標也顯示沒有 **bank conflict**，shared memory 使用良好。

2.4 Global Memory Throughput

- gld_throughput ≈ **21.3 GB/s**、gst_throughput ≈ **0.085 GB/s**，相對 GTX 1080 理論 ~320 GB/s 算中低，代表 kernel 不是被 **DRAM** 帶寬卡住，而是主要在 shared / 計算端工作。

- 讀寫 throughput 雖然不高，但沒有看到極低或 SM efficiency 掉很慘的情況，配合我用 float4 + 連續排 Q/K/V 的寫法，可以說 **global memory** 有配合**threads**去連續存取一整塊**memory**，也在實驗中發現**coalescing**有很大的效能提升

---

3. Experiment & Analysis

3.a System Spec

我是利用apollo-gpu 所以沒有特別放上spec

3.b Optimization（含文字 + 圖表）

（每個點至少附一段說明 + 程式碼片段）

Profile instruction

```
srun -p nvidia nvprof --metrics
achieved_occupancy,sm_efficiency,shared_load_throughput,shared_store_th
roughput,gld_throughput,gst_throughput     ./hw4 ./testcases/t30 ./out
```

統一選用t28的profile結果做優化

Baseline（time = 7.83)

```
Invocations                               Event Name         Min         Max         Avg       Total
Device "NVIDIA GeForce GTX 1080 (0)"
    Kernel: attention_kernel(float*, float*, float*, float*, int, int)
          1                  shared_ld_bank_conflict           0           0           0           0
          1                  shared_st_bank_conflict           0           0           0           0

==2299618== Metric result:
Invocations                                 Metric Name                        Metric Description         Min         Max         Avg
Device "NVIDIA GeForce GTX 1080 (0)"
    Kernel: attention_kernel(float*, float*, float*, float*, int, int)
          1                        achieved_occupancy                             Achieved Occupancy    0.746151    0.746151    0.746151
          1                                       ipc                                    Executed IPC    0.169899    0.169899    0.169899
          1                              inst_integer                          Integer Instructions  1.8041e+11  1.8041e+11  1.8041e+11
          1                             inst_executed                        Instructions Executed  2.1276e+10  2.1276e+10  2.1276e+10
          1                               gld_throughput                       Global Load Throughput  35.980GB/s  35.980GB/s  35.980GB/s
          1                               gst_throughput                      Global Store Throughput  35.945MB/s  35.945MB/s  35.945MB/s
          1                        dram_read_throughput                  Device Memory Read Throughput  140.68GB/s  140.68GB/s  140.68GB/s
          1                       dram_write_throughput                 Device Memory Write Throughput  70.073GB/s  70.073GB/s  70.073GB/s
          1                          l2_read_throughput                          L2 Throughput (Reads)  148.52GB/s  148.52GB/s  148.52GB/s
          1                         l2_write_throughput                         L2 Throughput (Writes)  71.934GB/s  71.934GB/s  71.934GB/s
          1                           shared_efficiency                    Shared Memory Efficiency       0.00%       0.00%       0.00%
          1                  eligible_warps_per_cycle                 Eligible Warps Per Active Cycle    0.159367    0.159367    0.159367
          1                 warp_execution_efficiency                       Warp Execution Efficiency     100.00%     100.00%     100.00%
          1                    issue_slot_utilization                          Issue Slot Utilization       3.56%       3.56%       3.56%
          1                          stall_inst_fetch      Issue Stall Reasons (Instructions Fetch)       0.24%       0.24%       0.24%
          1                     stall_exec_dependency    Issue Stall Reasons (Execution Dependency)       4.33%       4.33%       4.33%
          1                   stall_memory_dependency            Issue Stall Reasons (Data Request)      92.79%      92.79%      92.79%
          1           stall_constant_memory_dependency      Issue Stall Reasons (Immediate constant)       0.00%       0.00%       0.00%
          1                             stall_texture                 Issue Stall Reasons (Texture)       0.80%       0.80%       0.80%
          1                                stall_sync         Issue Stall Reasons (Synchronization)       0.00%       0.00%       0.00%
          1                          stall_pipe_busy               Issue Stall Reasons (Pipe Busy)       0.01%       0.01%       0.01%
          1                              stall_other                   Issue Stall Reasons (Other)       1.81%       1.81%       1.81%
```

從profile結果來看當前最大的問題是memory完全跟不上計算的速度，因為完全使用速度很慢的global memory

shared memory coalescing:(time = 2.97)

改善策略：

- 因為每個tile內的Q都會用到同一組 K V tile，這是在計算 QK內積時最常被讀到的兩塊，所以先把它load進shared memory可以大大提升資料讀取速度，讓整體運算更貼近compute bound
- 先前讀取資料時也沒有符合coalescing，所以資料讀取時改成連續讀取進入shared memory，讀入後因為shared memory的特性只要不要bank conflict就能更靈活的存取資料

profile 結果：

```
Device "NVIDIA GeForce GTX 1080 (0)"
    Kernel: flash_attention_kernel(float*, float*, float*, float*, int, int, int, int, int, int, int)
          1                  shared_ld_bank_conflict           0           0           0           0
          1                  shared_st_bank_conflict           0           0           0           0

==2299896== Metric result:
Invocations                                 Metric Name                        Metric Description         Min         Max         Avg
Device "NVIDIA GeForce GTX 1080 (0)"
    Kernel: flash_attention_kernel(float*, float*, float*, float*, int, int, int, int, int, int, int)
          1                        achieved_occupancy                             Achieved Occupancy    0.093011    0.093011    0.093011
          1                                       ipc                                    Executed IPC    0.263329    0.263329    0.263329
          1                              inst_integer                          Integer Instructions  3.1699e+11  3.1699e+11  3.1699e+11
          1                             inst_executed                        Instructions Executed  2.6258e+10  2.6258e+10  2.6258e+10
          1                               gld_throughput                       Global Load Throughput  732.77GB/s  732.77GB/s  732.77GB/s
          1                               gst_throughput                      Global Store Throughput  22.038GB/s  22.038GB/s  22.038GB/s
          1                        dram_read_throughput                  Device Memory Read Throughput  8.5414GB/s  8.5414GB/s  8.5414GB/s
          1                       dram_write_throughput                 Device Memory Write Throughput  4.4364GB/s  4.4364GB/s  4.4364GB/s
          1                         l2_write_throughput                         L2 Throughput (Writes)  24.793GB/s  24.793GB/s  24.793GB/s
          1                           shared_efficiency                    Shared Memory Efficiency       6.06%       6.06%       6.06%
          1                  eligible_warps_per_cycle                 Eligible Warps Per Active Cycle    0.223833    0.223833    0.223833
          1                 warp_execution_efficiency                       Warp Execution Efficiency     100.00%     100.00%     100.00%
          1                    issue_slot_utilization                          Issue Slot Utilization       5.50%       5.50%       5.50%
          1                          stall_inst_fetch      Issue Stall Reasons (Instructions Fetch)       3.19%       3.19%       3.19%
          1                     stall_exec_dependency    Issue Stall Reasons (Execution Dependency)      15.18%      15.18%      15.18%
          1                   stall_memory_dependency            Issue Stall Reasons (Data Request)      79.13%      79.13%      79.13%
          1           stall_constant_memory_dependency      Issue Stall Reasons (Immediate constant)       0.01%       0.01%       0.01%
          1                             stall_texture                 Issue Stall Reasons (Texture)       0.00%       0.00%       0.00%
          1                                stall_sync         Issue Stall Reasons (Synchronization)       0.03%       0.03%       0.03%
          1                          stall_pipe_busy               Issue Stall Reasons (Pipe Busy)       0.02%       0.02%       0.02%
          1                              stall_other                   Issue Stall Reasons (Other)       2.38%       2.38%       2.38%
```

從profile結果看到，改變了讀取patten讓global memory的讀取更加順暢，bandwidth也大大提升，加入了shared memory後因為資料可以大量的到較快的記憶體存取也降低了因為記憶體存取的stall，但我的occupancy非常的低因為我只遇一個block去處理一整個tile，並沒有榨乾sm的完整計算資源，但就算如此光是改進記憶體存取就大大提升了run time

1 token per warp(Time = 2.52)

改善策略

因為前個版本sm的利用率過低，所以我選擇提告block中的warp數，並讓一個warp負責一個Q row也就是 tile x dim 筆資料，以此提升occupancy

Profile 結果：

```
Invocations                               Event Name         Min         Max         Avg       Total
Device "NVIDIA GeForce GTX 1080 (0)"
    Kernel: flash_attention_kernel(float*, float*, float*, float*, int, int, int, int, int, int, int)
          1                    shared_ld_bank_conflict           0           0           0           0
          1                    shared_st_bank_conflict           0           0           0           0

==529377== Metric result:
Invocations                               Metric Name                         Metric Description         Min         Max         Avg
Device "NVIDIA GeForce GTX 1080 (0)"
    Kernel: flash_attention_kernel(float*, float*, float*, float*, int, int, int, int, int, int, int)
          1                         achieved_occupancy                             Achieved Occupancy    0.998956    0.998956    0.998956
          1                                        ipc                                    Executed IPC    1.571038    1.571038    1.571038
          1                              inst_integer                          Integer Instructions  1.7592e+12  1.7592e+12  1.7592e+12
          1                             inst_executed                        Instructions Executed  1.2678e+11  1.2678e+11  1.2678e+11
          1                            gld_throughput                       Global Load Throughput  131.07GB/s  131.07GB/s  131.07GB/s
          1                            gst_throughput                      Global Store Throughput  7.2817MB/s  7.2817MB/s  7.2817MB/s
          1                       dram_read_throughput                 Device Memory Read Throughput  81.937GB/s  81.937GB/s  81.937GB/s
          1                      dram_write_throughput                Device Memory Write Throughput  105.20GB/s  105.20GB/s  105.20GB/s
          1                         l2_read_throughput                          L2 Throughput (Reads)  371.37GB/s  371.37GB/s  371.37GB/s
          1                        l2_write_throughput                         L2 Throughput (Writes)  240.31GB/s  240.31GB/s  240.31GB/s
          1                          shared_efficiency                    Shared Memory Efficiency     100.00%     100.00%     100.00%
          1                  eligible_warps_per_cycle               Eligible Warps Per Active Cycle    2.306972    2.306972    2.306972
          1                 warp_execution_efficiency                     Warp Execution Efficiency     100.00%     100.00%     100.00%
          1                    issue_slot_utilization                        Issue Slot Utilization      35.87%      35.87%      35.87%
          1                         stall_inst_fetch     Issue Stall Reasons (Instructions Fetch)       4.12%       4.12%       4.12%
          1                      stall_exec_dependency   Issue Stall Reasons (Execution Dependency)      10.79%      10.79%      10.79%
          1                    stall_memory_dependency           Issue Stall Reasons (Data Request)      69.05%      69.05%      69.05%
          1          stall_constant_memory_dependency     Issue Stall Reasons (Immediate constant)       0.00%       0.00%       0.00%
          1                             stall_texture                Issue Stall Reasons (Texture)       0.00%       0.00%       0.00%
          1                                stall_sync        Issue Stall Reasons (Synchronization)       7.37%       7.37%       7.37%
          1                          stall_pipe_busy              Issue Stall Reasons (Pipe Busy)       0.50%       0.50%       0.50%
          1                             stall_other                  Issue Stall Reasons (Other)       6.73%       6.73%       6.73%
```

Parallel bc in with different warp（time = 1.92)

改善策略：

原本我一個warp要seq的負責 tile x dim的資料量，如果tile size = 64 dim = 64那就是 4096次運算，但我每個Q都要跟K算dot product其實這之間是獨立的所以可以進一步平行

Profile結果

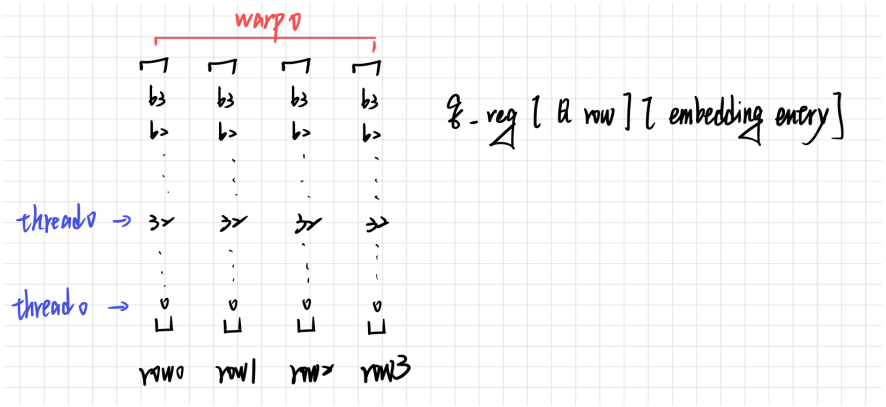
```
Invocations                               Event Name         Min         Max         Avg       Total
Device "NVIDIA GeForce GTX 1080 (0)"
    Kernel: flash_attention_kernel(float*, float*, float*, float*, int, int, int, int, int, int, int)
          1                    shared_ld_bank_conflict           0           0           0           0
          1                    shared_st_bank_conflict           0           0           0           0

==1564244== Metric result:
Invocations                               Metric Name                         Metric Description         Min         Max         Avg
Device "NVIDIA GeForce GTX 1080 (0)"
    Kernel: flash_attention_kernel(float*, float*, float*, float*, int, int, int, int, int, int, int)
          1                         achieved_occupancy                         Achieved Occupancy    0.998837    0.998837    0.998837
          1                                        ipc                               Executed IPC    1.738956    1.738956    1.738956
          1                              inst_integer                      Integer Instructions  1.7592e+12  1.7592e+12  1.7592e+12
          1                             inst_executed                     Instructions Executed  1.2678e+11  1.2678e+11  1.2678e+11
          1                           gld_throughput                   Global Load Throughput  129.15GB/s  129.15GB/s  129.15GB/s
          1                           gst_throughput                  Global Store Throughput  7.1750MB/s  7.1750MB/s  7.1750MB/s
          1                      dram_read_throughput             Device Memory Read Throughput  82.184GB/s  82.184GB/s  82.184GB/s
          1                     dram_write_throughput            Device Memory Write Throughput  104.36GB/s  104.36GB/s  104.36GB/s
          1                        l2_read_throughput                    L2 Throughput (Reads)  365.93GB/s  365.93GB/s  365.93GB/s
          1                       l2_write_throughput                   L2 Throughput (Writes)  236.79GB/s  236.79GB/s  236.79GB/s
          1                         shared_efficiency                  Shared Memory Efficiency      100.00%     100.00%     100.00%
          1                   eligible_warps_per_cycle             Eligible Warps Per Active Cycle    2.509320    2.509320    2.509320
          1                  warp_execution_efficiency                  Warp Execution Efficiency      100.00%     100.00%     100.00%
          1                     issue_slot_utilization                     Issue Slot Utilization      39.68%      39.68%      39.68%
          1                          stall_inst_fetch    Issue Stall Reasons (Instructions Fetch)       4.52%       4.52%       4.52%
          1                      stall_exec_dependency  Issue Stall Reasons (Execution Dependency)      11.81%      11.81%      11.81%
          1                    stall_memory_dependency          Issue Stall Reasons (Data Request)      68.17%      68.17%      68.17%
          1            stall_constant_memory_dependency    Issue Stall Reasons (Immediate constant)       0.00%       0.00%       0.00%
          1                             stall_texture             Issue Stall Reasons (Texture)       0.00%       0.00%       0.00%
          1                                stall_sync       Issue Stall Reasons (Synchronization)       5.92%       5.92%       5.92%
          1                          stall_pipe_busy             Issue Stall Reasons (Pipe Busy)       0.55%       0.55%       0.55%
          1                             stall_other                Issue Stall Reasons (Other)       7.44%       7.44%       7.44%
```

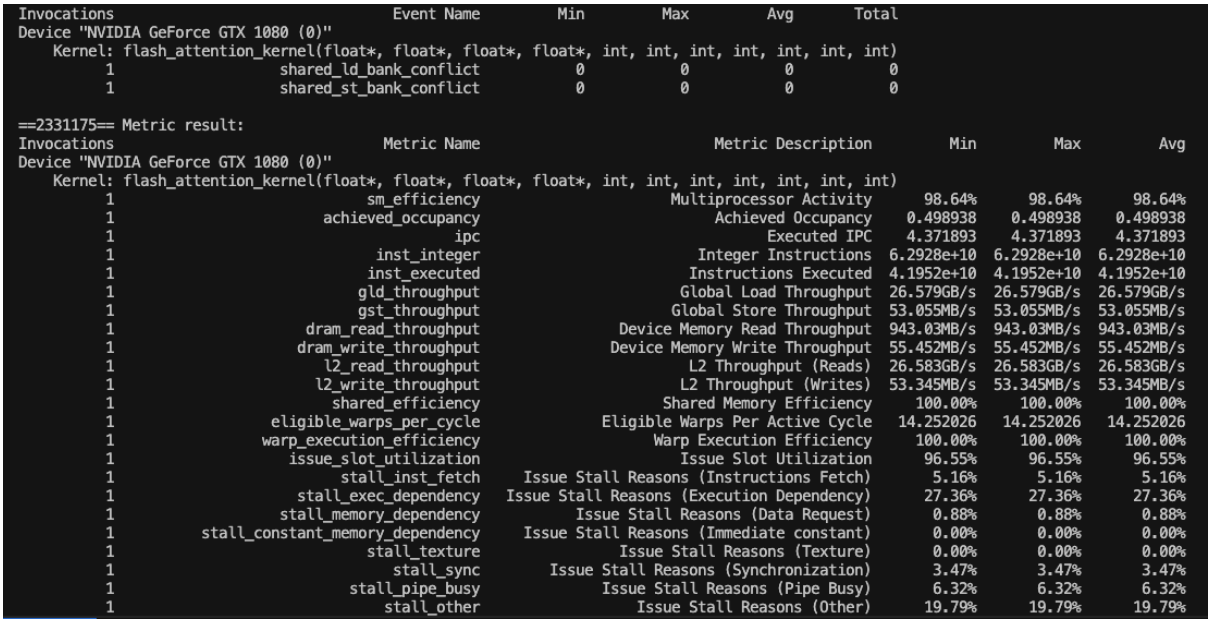
有稍微提升IPC和slot utilization所以代表這改善方向是正確的，並且也有反應在runtime的下降但還沒有到極限，且記憶體存取還是瓶頸因為還是存在69%的Issue Stall Reasons (Data Request)

coalescing load Q into shared memory（Time = 0.57）

原本想說Q只會讀取一次就沒有管他的讀取patten結果發現改善了K V 讀取方式後Issue Stall Reasons (Data Request) 還是偏高，所以嘗試改變Q的讀取方式看能不能得到效能提升



這樣的mapping方式我每個warp中的threads都讀取連續的記憶體位置，這樣的改進也造成效能的大進步

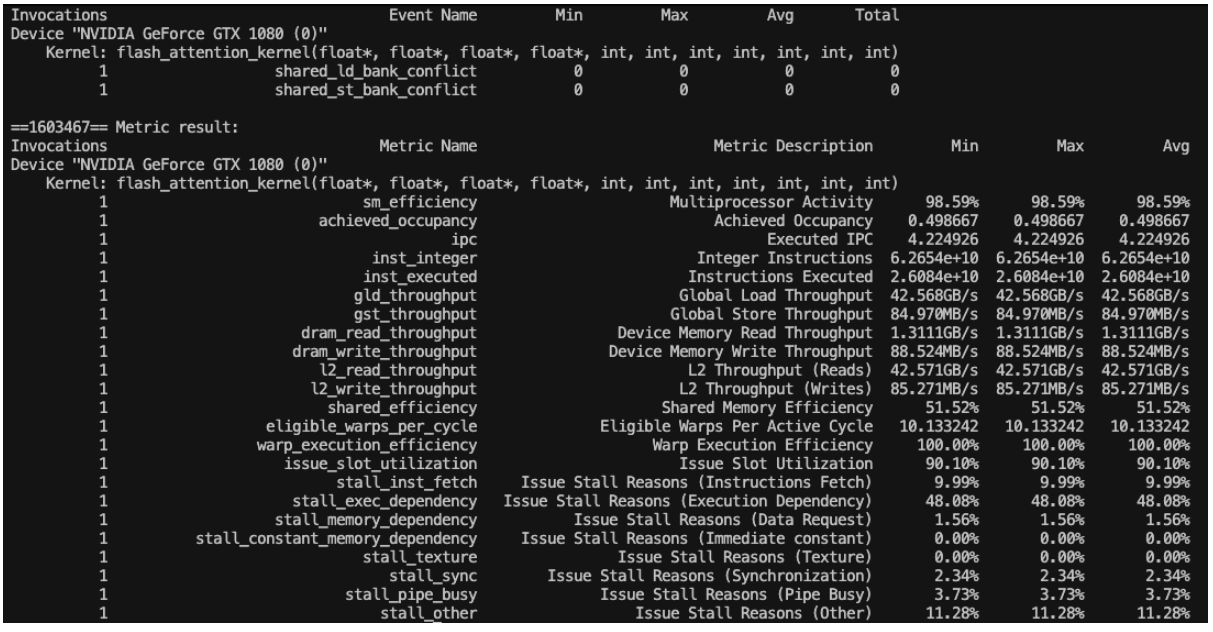


```
Invocations                               Event Name         Min         Max         Avg       Total
Device "NVIDIA GeForce GTX 1080 (0)"
    Kernel: flash_attention_kernel(float*, float*, float*, float*, int, int, int, int, int, int, int)
          1                    shared_ld_bank_conflict           0           0           0           0
          1                    shared_st_bank_conflict           0           0           0           0

==2331175== Metric result:
Invocations                                 Metric Name                            Metric Description         Min         Max         Avg
Device "NVIDIA GeForce GTX 1080 (0)"
    Kernel: flash_attention_kernel(float*, float*, float*, float*, int, int, int, int, int, int, int)
          1                             sm_efficiency                       Multiprocessor Activity      98.64%      98.64%      98.64%
          1                        achieved_occupancy                            Achieved Occupancy    0.498938    0.498938    0.498938
          1                                       ipc                                  Executed IPC    4.371893    4.371893    4.371893
          1                              inst_integer                     Integer Instructions  6.2928e+10  6.2928e+10  6.2928e+10
          1                             inst_executed                      Instructions Executed  4.1952e+10  4.1952e+10  4.1952e+10
          1                               gld_throughput                        Global Load Throughput  26.579GB/s  26.579GB/s  26.579GB/s
          1                               gst_throughput                       Global Store Throughput  53.055MB/s  53.055MB/s  53.055MB/s
          1                           dram_read_throughput                 Device Memory Read Throughput  943.03MB/s  943.03MB/s  943.03MB/s
          1                          dram_write_throughput                Device Memory Write Throughput  55.452MB/s  55.452MB/s  55.452MB/s
          1                             l2_read_throughput                       L2 Throughput (Reads)  26.583GB/s  26.583GB/s  26.583GB/s
          1                            l2_write_throughput                      L2 Throughput (Writes)  53.345MB/s  53.345MB/s  53.345MB/s
          1                              shared_efficiency                    Shared Memory Efficiency      100.00%      100.00%      100.00%
          1                     eligible_warps_per_cycle                 Eligible Warps Per Active Cycle  14.252026  14.252026  14.252026
          1                    warp_execution_efficiency                       Warp Execution Efficiency      100.00%      100.00%      100.00%
          1                       issue_slot_utilization                          Issue Slot Utilization       96.55%       96.55%       96.55%
          1                           stall_inst_fetch      Issue Stall Reasons (Instructions Fetch)       5.16%       5.16%       5.16%
          1                         stall_exec_dependency    Issue Stall Reasons (Execution Dependency)      27.36%      27.36%      27.36%
          1                       stall_memory_dependency           Issue Stall Reasons (Data Request)       0.88%       0.88%       0.88%
          1               stall_constant_memory_dependency      Issue Stall Reasons (Immediate constant)       0.00%       0.00%       0.00%
          1                                 stall_texture                Issue Stall Reasons (Texture)       0.00%       0.00%       0.00%
          1                                    stall_sync          Issue Stall Reasons (Synchronization)       3.47%       3.47%       3.47%
          1                              stall_pipe_busy                Issue Stall Reasons (Pipe Busy)       6.32%       6.32%       6.32%
          1                                   stall_other                     Issue Stall Reasons (Other)      19.79%      19.79%      19.79%
```

我的Issue Stall Reasons (Data Request) 大幅減少到0.8%，整個讓我各方面數值都上升目前看到的改善空間pipeline bubble可以再減少

use vectorized load Q K V（Time = 0.52）



```
Invocations                               Event Name         Min         Max         Avg       Total
Device "NVIDIA GeForce GTX 1080 (0)"
    Kernel: flash_attention_kernel(float*, float*, float*, float*, int, int, int, int, int, int, int)
          1                    shared_ld_bank_conflict           0           0           0           0
          1                    shared_st_bank_conflict           0           0           0           0

==1603467== Metric result:
Invocations                                 Metric Name                            Metric Description         Min         Max         Avg
Device "NVIDIA GeForce GTX 1080 (0)"
    Kernel: flash_attention_kernel(float*, float*, float*, float*, int, int, int, int, int, int, int)
          1                             sm_efficiency                       Multiprocessor Activity      98.59%      98.59%      98.59%
          1                        achieved_occupancy                            Achieved Occupancy    0.498667    0.498667    0.498667
          1                                       ipc                                  Executed IPC    4.224926    4.224926    4.224926
          1                              inst_integer                     Integer Instructions  6.2654e+10  6.2654e+10  6.2654e+10
          1                             inst_executed                      Instructions Executed  2.6084e+10  2.6084e+10  2.6084e+10
          1                               gld_throughput                        Global Load Throughput  42.568GB/s  42.568GB/s  42.568GB/s
          1                               gst_throughput                       Global Store Throughput  84.970MB/s  84.970MB/s  84.970MB/s
          1                           dram_read_throughput                 Device Memory Read Throughput  1.3111GB/s  1.3111GB/s  1.3111GB/s
          1                          dram_write_throughput                Device Memory Write Throughput  88.524MB/s  88.524MB/s  88.524MB/s
          1                             l2_read_throughput                       L2 Throughput (Reads)  42.571GB/s  42.571GB/s  42.571GB/s
          1                            l2_write_throughput                      L2 Throughput (Writes)  85.271MB/s  85.271MB/s  85.271MB/s
          1                              shared_efficiency                    Shared Memory Efficiency      51.52%      51.52%      51.52%
          1                     eligible_warps_per_cycle                 Eligible Warps Per Active Cycle  10.133242  10.133242  10.133242
          1                    warp_execution_efficiency                       Warp Execution Efficiency      100.00%      100.00%      100.00%
          1                       issue_slot_utilization                          Issue Slot Utilization       90.10%       90.10%       90.10%
          1                           stall_inst_fetch      Issue Stall Reasons (Instructions Fetch)       9.99%       9.99%       9.99%
          1                         stall_exec_dependency    Issue Stall Reasons (Execution Dependency)      48.08%      48.08%      48.08%
          1                       stall_memory_dependency           Issue Stall Reasons (Data Request)       1.56%       1.56%       1.56%
          1               stall_constant_memory_dependency      Issue Stall Reasons (Immediate constant)       0.00%       0.00%       0.00%
          1                                 stall_texture                Issue Stall Reasons (Texture)       0.00%       0.00%       0.00%
          1                                    stall_sync          Issue Stall Reasons (Synchronization)       2.34%       2.34%       2.34%
          1                              stall_pipe_busy                Issue Stall Reasons (Pipe Busy)       3.73%       3.73%       3.73%
          1                                   stall_other                     Issue Stall Reasons (Other)      11.28%      11.28%      11.28%
```

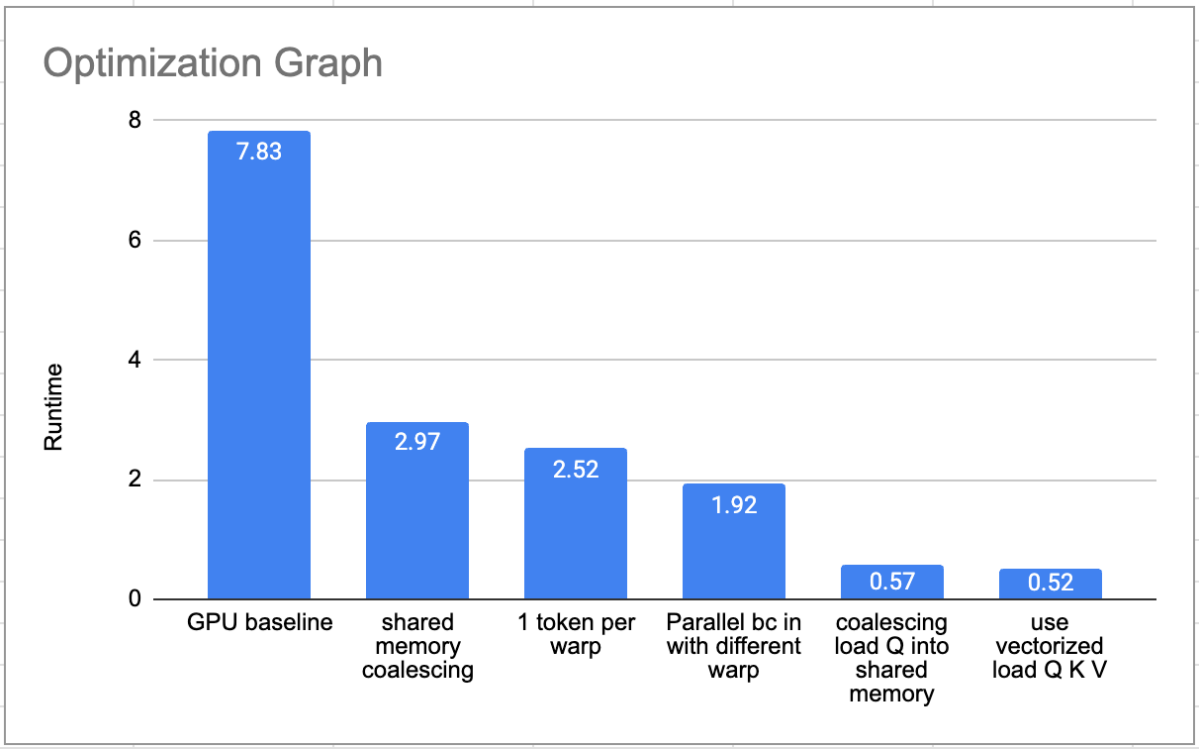
利用了vectorized load 1次將4個data 載入可以減少很多load instruction，所以也近一步的提升了整體運行效率，但因為我載入資料變快所以運行速度沒有提升的話就會顯示出Issue Stall Reasons (Execution Dependency)　48.08% 上升

最終取得rank1的runtime表現 (pp25s121)

- AC | TLE | TLE+ | NA
- Each testcase will run for 10s+time limit. If the program runs 10s more than tl
- NA is not accepted. It can be wrong answer, segmentation fault or runtime er
- The rank is based on Time + Penality time

User	Rank	Passed	Time	Penalty	t01	t02	t03	t0
pp25s121	1	30	17.58		0.21	0.21	0.26	0.
pp25s006	2	30	17.88		0.21	0.21	0.21	0.
pp25s070	3	30	17.93		0.26	0.26	0.26	0.
pp25s029	4	30	18.08		0.26	0.26	0.21	0.
pp25s023	5	30	18.08		0.26	0.31	0.21	0.
pp25s061	6	30	18.23		0.21	0.31	0.21	0.
pp25s072	7	30	18.38		0.21	0.21	0.21	0.
pp25s063	8	30	18.98		0.26	0.21	0.21	0.

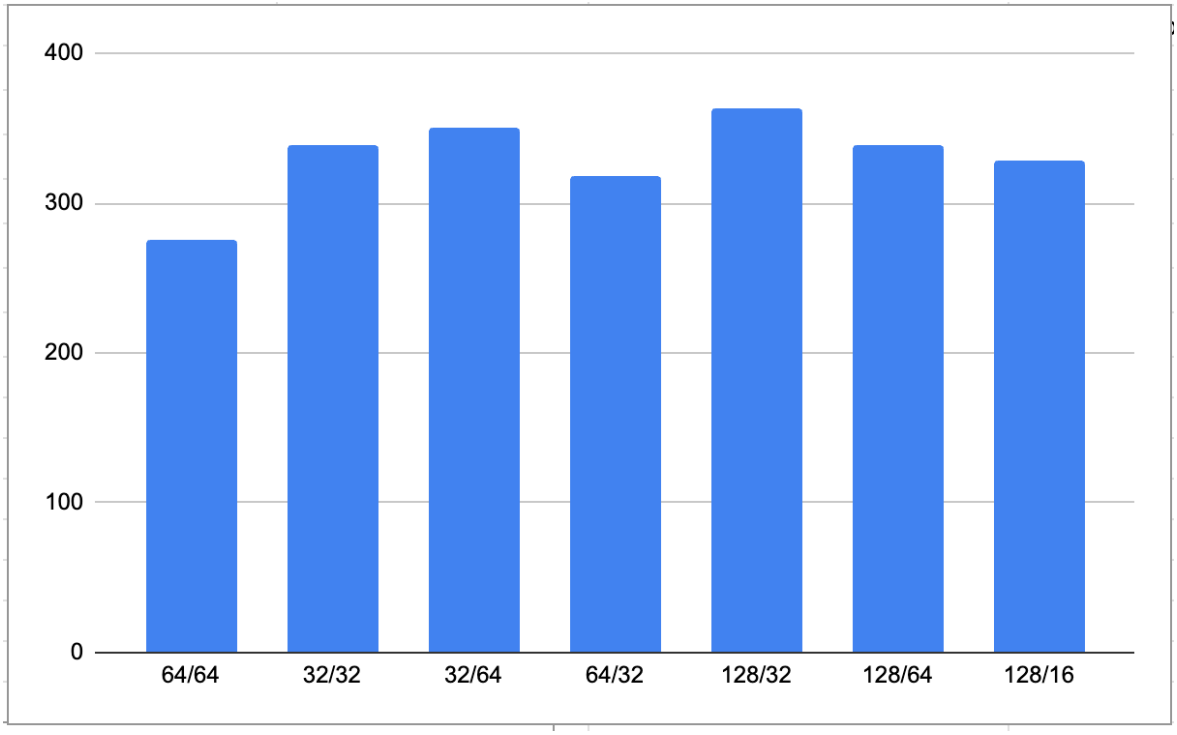
optimize graph



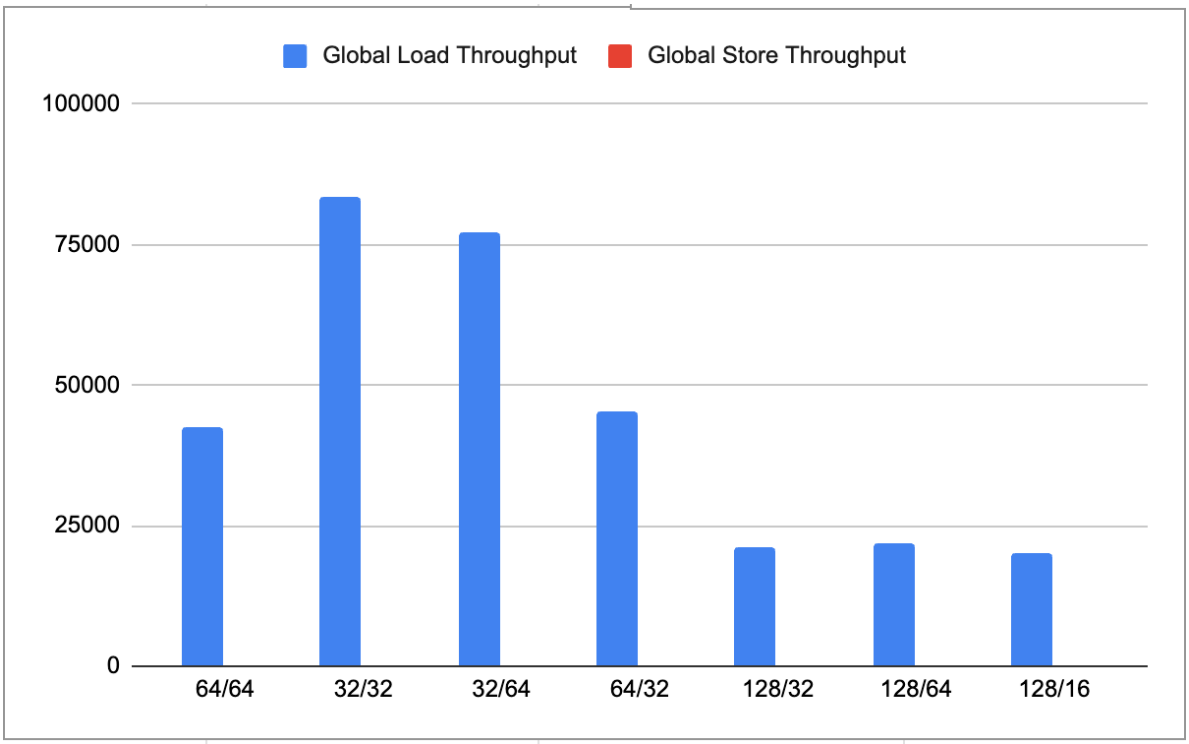
3.c Others

blocking factor:

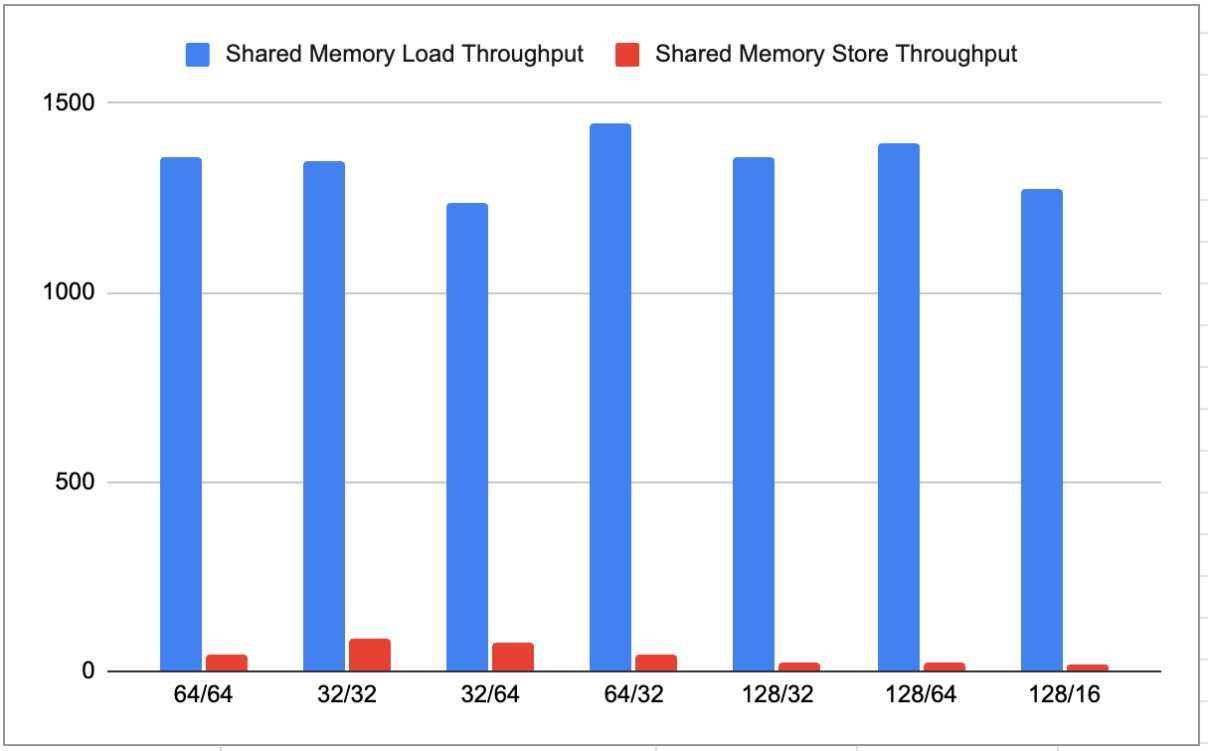
GLOPS



Global memory performance



Shared memory performance



Runtime performance

br	bc	Time
64	64	1.87
32	32	1.52
64	32	1.47
32	64	1.62
128	32	1.42
128	64	1.52
128	16	1.57

實驗結果顯示 br = 128 bc = 32 是最好的blocking factor

---

4. AMD GPU Porting & Analysis

4.a Implementation

- CUDA → HIP 移植流程

  Header Replacement

  #include <hip/hip_runtime.h>

  API Prefix Replacement

CUDA	HIP
cudaMalloc	hipMalloc
cudaMemcpy	hipMemcpy
cudaDeviceSynchronize	hipDeviceSynchronize
cudaFree	hipFree

  Warp Shuffle 函式替換

CUDA	HIP
__shfl_down_sync(mask, val, delta)	__shfl_down(val, delta, width)
__shfl_sync(mask, val, lane)	__shfl(val, lane, width)

NVIDIA 的 warp 大小 = 32　　AMD 的 wavefront 大小 = 64

因為程式碼中有寫 warpId = tx / 32 所以要替換成 warpId = tx / 64

- CUDA API 與 HIP API 差異

CUDA	HIP
cudaMalloc	hipMalloc
cudaFree	hipFree
cudaMemcpy	hipMemcpy
cudaMemcpyHostToDevice	hipMemcpyHostToDevice
cudaMemcpyDeviceToHost	hipMemcpyDeviceToHost
cudaDeviceSynchronize	hipDeviceSynchronize
cudaMallocHost	hipHostMalloc
cudaFreeHost	hipHostFree

- block/thread mapping 差異

概念	NVIDIA (GTX 1080, CUDA)	AMD (MI210, ROCm/HIP)
thread	thread	work-item / thread
block	block / CTA	work-group
warp 單位	**warp = 32 threads**	**wavefront = 64 threads**
SM / CU	SM (Streaming Multiprocessor)	CU (Compute Unit)

warp 調度單位	32-thread warp	64-thread wavefront

4.b Experiment & Analysis

所有profile結果都是利用nvprof rocprof做出來的

Optimize for amd

改善目標:
改善整體thread的mapping繞他符合amd的硬體架構, 也提高shared memory中的資料量, 看能不能有跟NV有相同的效能結果或是甚至更好

改善前

metrics	GTX 1080 MB/s	MI210 MB/s
shared memory load	1359.2	628.4
shared memory store	21.238	25.323
Gloable memory load	21321	12943.2
Global memory store	84.953	143.6

可以看到在記憶體access上GTX 1080的效能都比MI210好, 因為我程式碼中設計的讀取pattern 不完全對齊 warp = 64 threads的 amd gpu, 導致可能將近有一半的threads在idle, 間接影響了讀取的效率

改善後

metrics	GTX 1080 MB/s	MI210 MB/s
shared memory load	1359.2	2032.4
shared memory store	21.238	27.319
Gloable memory load	21321	27323.2
Global memory store	84.953	193.6

改善了**mapping pattern**後明顯看到效能逐漸對齊了**GTX 1080**, 甚至因為**shared memory**比較大也有更好的效能結果

---

5. Experience & Conclusion

5.a What I Learned

- 如何用 occupancy / memory throughput 思維優化

  這次實作讓我學到如何用 *occupancy* 與 *memory throughput* 的角度來思考 CUDA Kernel 的效能瓶頸。首先，透過 achieved_occupancy 可以快速判斷 kernel 是否受限於 register 或 shared memory；當 occupancy 只有約 50% 時，就代表 Thread 數雖多，但真正能同時在 SM 上活躍的 warps 不夠，這往往是因為 register 過多或 shared memory 分配偏大所導致。其次，memory throughput（global / shared）能直接反映 memory pipeline 是否運作順暢。像 shared memory load 可達到上千 GB/s，而 global load 只有 20 GB/s 左右，就能看出計算路徑已成功被封裝在 shared memory 中，global memory 帶寬不是主要瓶頸；反之若 gld throughput 過低，就需要檢查 coalescing 是否良好。透過這些 profiling 指標，我能更有系統地找出 kernel 的瓶頸，並以此決定優化方向，例如減少 register 使用、壓低 shared 使用量、移除 bank conflict、或重構 memory access pattern，以提升整體 GPU 使用效率。