

Parallel Programming HW3 REPORT

109062671 吳竣宇

1 Implementation

- a. Blocked Floyd Warshall algorithm

- b. Phase 2的切法:

依Spec中的pseudo code將pivot row和pivot column分別用不同的kernel計算並讓gpu中的block一一對應矩陣中的block。

Phase 3的切法:

依Spec中的pseudo code將pivot block的左上、左下、右上、右下分別用不同的kernel計算，並讓gpu中的block一一對應矩陣中的block。

- c. hw3-2和hw3-3的configuration相同:

Blocking factor: 64

- 為了盡可能maximum blocking factor, 並讓Spec的pseudo code中的每個phase都能將data複製到share memory中執行, 已知phase 3所需存取矩陣的blocks數量為3個(pivot column block, pivot row block, target block), 由於GeForce GTX 1080的share memory為49152bytes, 若我們將blocking factor設為64, 則能剛剛好將3blocks複製進share memory($3 * 64 * 64 * \text{sizeof(int)} = 49152\text{bytes}$)。

Number of block:

- 依照Spec的pseudo code中phase 2和phase 3在每個round中需要執行的矩陣block數, 決定每個kernel所需的gpu blocks的數量。

Number of threads: 1024

- 使用block中的threads數量的最大值，以達到好的優化結果。

- d. 使用Unified Memory建立一個Virtual address, 讓cpu和gpus去存取同一個address space。

- e. hw3-1:

- 參考Spec的pseudo code寫好sequential code後, 用pthread做implement, 首先將對index (i, j)做divide, 將需要做計算(update)的所有index(i, j)盡可能平分給不同的threads執行。

hw3-2:

- 依照Spec中的pseudo code將BFW分成三個phase, 其中phase 2再分成兩個kernel去計算一整個pivot row及pivot column, phase 3分成四個kernel去計算pivot block的左上、左下、右上、右下的blocks, kernel中和hw3-1相同, 將index (i,j)平分給不同的threads做計算(update)。

hw3-3:

- 利用cudaMallocManaged()開一個unified address space, 讓cpu和gpus可以直接存取matrix D, 並利用openmp加速呼叫不同的gpu執行kernel做計算, 理論上計算完應該要做memory peer async copy, 不過由於copy時間太長, 只能讓它page fault以盡可能多過一點測資。

2 Profiling Results(hw3-2)

Test data: /home/pp21/share/hw3-2/cases/p20k1

Kernel: BFW_Phase_3(int*, int, int, int, int)	Metric Description	Min	Max	Avg
achieved_occupancy	Achieved Occupancy	0.492374	0.937151	0.926301
sm_efficiency	Multiprocessor Activity	3.62%	99.95%	98.88%
shared_load_throughput	Shared Memory Load Throughput	101.23GB/s	3557.7GB/s	3276.3GB/s
shared_store_throughput	Shared Memory Store Throughput	8.4396GB/s	288.54GB/s	265.99GB/s
gld_throughput	Global Load Throughput	867.28MB/s	18.925GB/s	18.664GB/s
gst_throughput	Global Store Throughput	2.0642GB/s	71.991GB/s	66.403GB/s

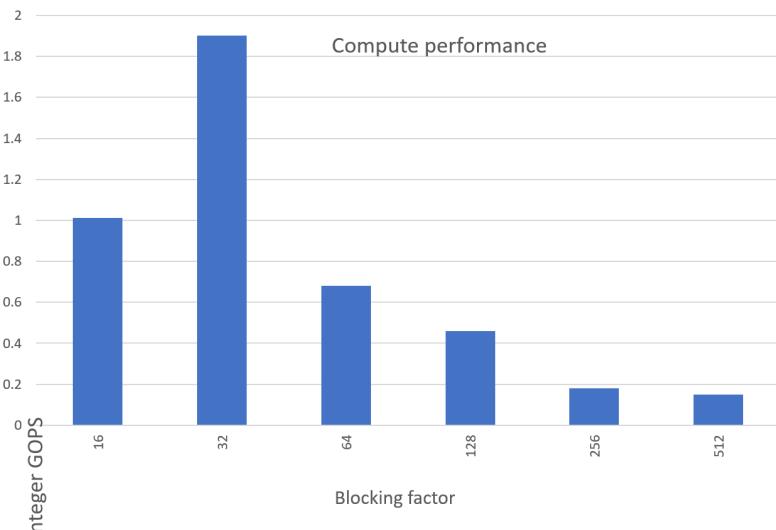
3 Experiment & Analysis

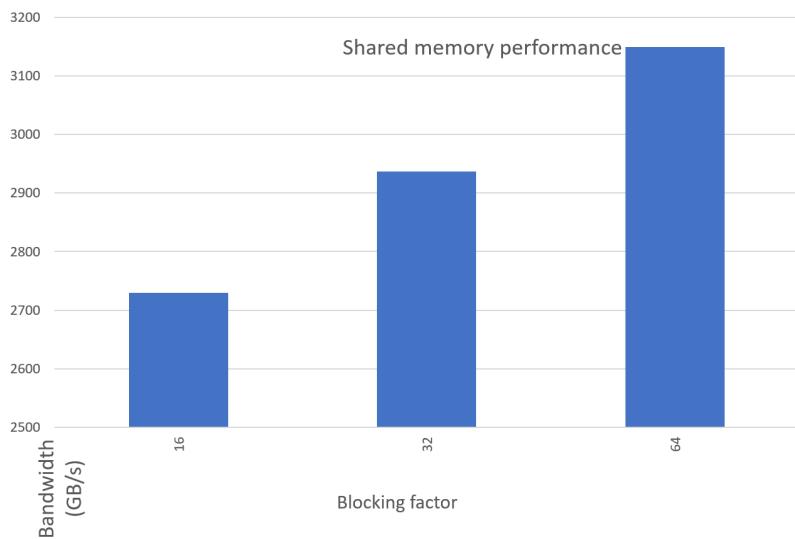
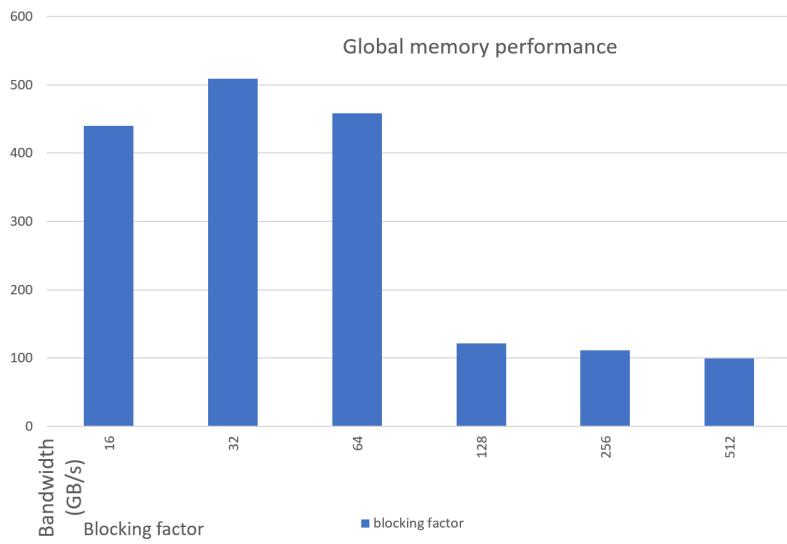
a. System Spec:

- All the experiments were executed in hades.

b. Blocking Factor:

- Test data: /home/pp21/share/hw3-2/cases/c20.1
- B = 16, 32, 64的数据皆在使用shared memory的前提下測integer GOPS, global memory performance則是在皆無使用Shared memory的前提下作實驗



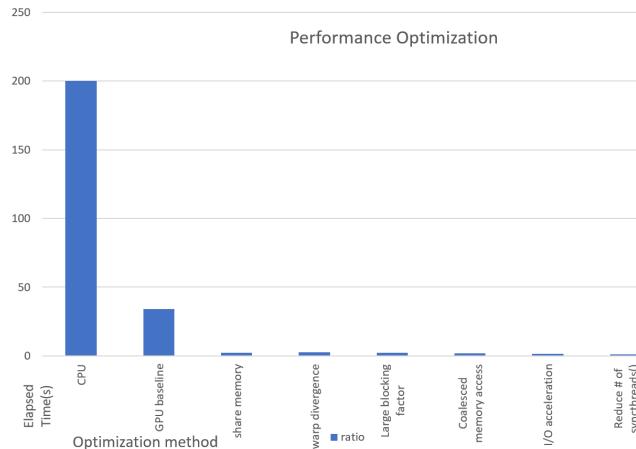


c. Optimization(hw3-2):

以下為我有用到的優化技巧:

- CPU (143.83s)
- GPU baseline (24.48s)
- Use share memory (1.72s)
- Reduce the warp divergence + unroll the loop (1.91s)
 - padding to the multiple of blocking factor
 - if else -> cuda api min()
- Large blocking factor (1.57s)
 - B = 32 -> B = 64
- Coalesced memory access + Handle bank conflict (1.32s)

- linear addressing
- I/O acceleration (1.12s)
- Reduce the number of calling syncthreads() (0.79s)
 - 由於phase 3的target block和pivot row(column block)不重疊, 因此可以將 syncthreads()拔掉

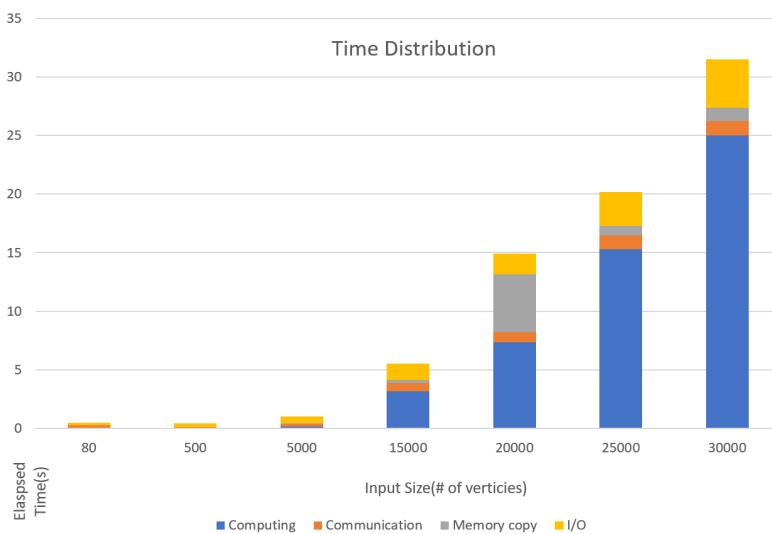


d. Weak scalability(hw3-3):

- Test data: /home/pp21/share/hw3-3/cases/c04.1
- 以gpu:1和gpu:2做比較, 後者是前者的0.15倍due to page fault.

e. Time distribution(hw3-2):

- Communication的計算方法是用nvprof的api 時間加總 - 所有kernel的計算時間
- CPU I/O time則是使用chrono的api量測
- 其餘的皆使用nvprof上的數字做計算加總



f. Others:

- hw3-2:

- 我在phase 3實驗，在share memory開一個target block存取update的值，最後再傳回share memory，和直接在迴圈中將update的值存回global memory中的matrix D相比，前者的速度會更快。

- hw3-3:

- 我分別嘗試過使用`cudamalloc`的p2p和`cudamallocmanaged`的p2p並使用`stream`做`async copy`以增加concurrency，最後結果皆沒有使用unified memory不做gpus之間的互相傳輸還快，也試過做兩個gpu的load balancing(將矩陣切成兩塊子矩陣分別讓不同gpu做phase 1~3的處理)，最後發現由於copy到其他gpu的時候要不重複blocks傳送的話一定得一行一行傳，這樣會造成呼叫api的次數過多，原本copy就很耗時了呼叫太多次api就會更慘，再來當copy到重複的矩陣block時會造成重複存取的問題，這也會使得copy的速度變慢，實驗結果就讓人不盡滿意。

4 Experience & Conclusion

(a). Experience:

- 要先仔細看實驗spec再來寫作業，否則沒照優化順序寫code最後寫報告時發現，回去重作會很花時間。

(b). Conclusion:

- 學到很多cuda的優化技巧。