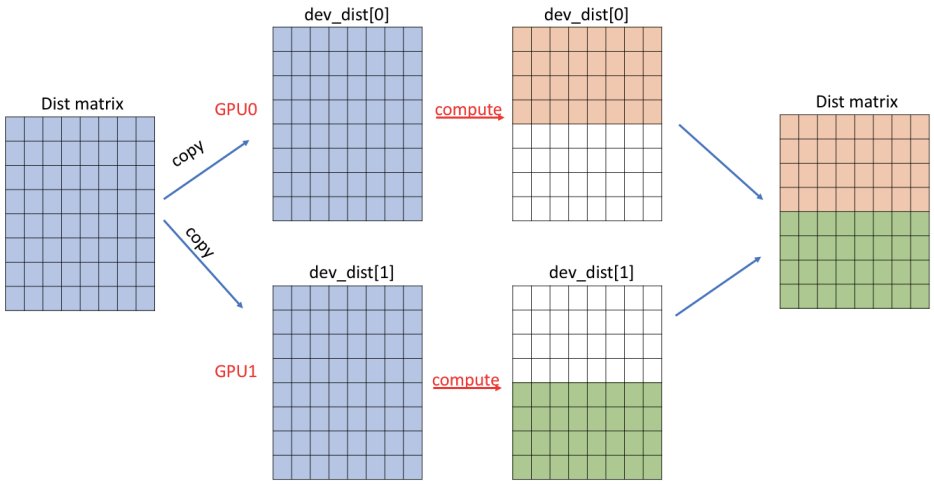
Parallel Programming hw4-2 ----Blocked Folyd-Warshall (Multi-GPU)

學號：109062639　姓名：葉哲欣

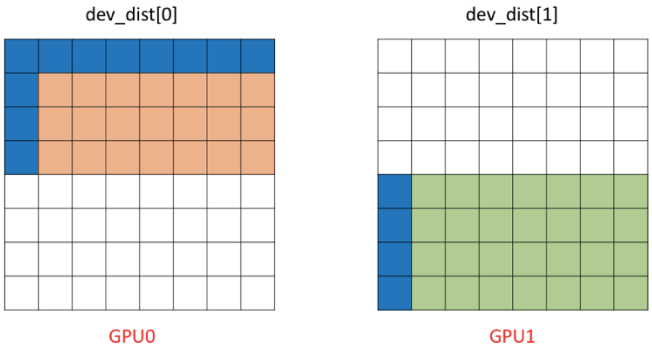
Implementation:

想法：將 data 分為 2 區塊 ，GPU 採取同時運算，確保自己負責的區塊資料正確性。
建立 2 個 data 大小的 matrix(dev_dist[0],dev_dist[1])並且複製 dist 的 data，每個 GPU 在自己對應 到的 dev_dist 做運算



由於 phase1,phase2 所使用到的 block 數量較少，較快執行完成,故讓每個 GPU 運算完整的 dev_dist

Phase3:



每個 round 各 GPU 需運算一個(ceil(round,num_gpu))*(round-1)個 block(藍色+紅色)
故 phase3 所需 thread 及 block：
dim3 ThreadPerBlock(k,k)
dim3 BlockPerGrid(ceil(round,num_gpu),round)

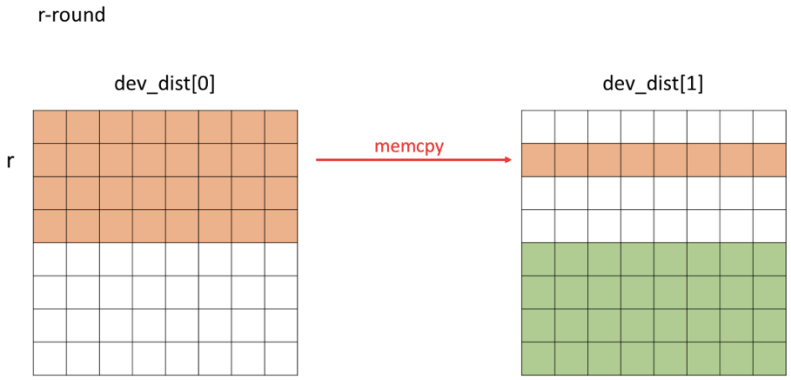
執行每一個 round,兩 GPU 各自執行 Phase1,Phase2,Phase3 但要如何確保資料正確性？

方法：

執行第 r-th round

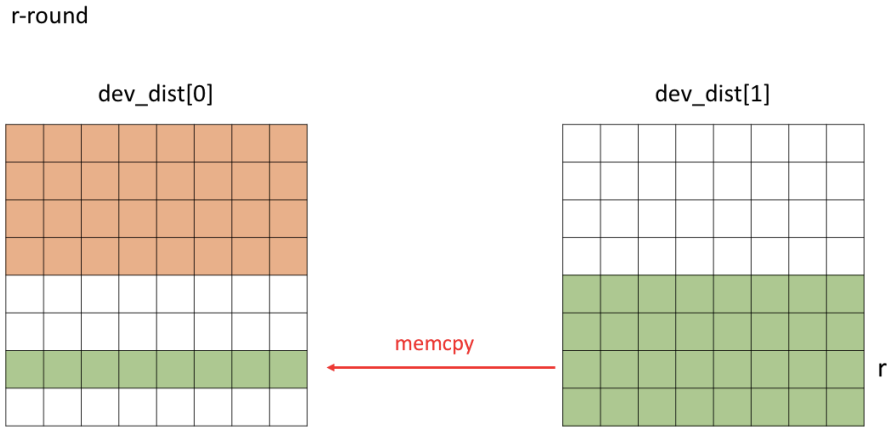
若 r < ceil(Round,num_gpu)時

將 dev_dist[0] 第 r -row 的所有 block memcopy 到 dev_dist[1]相對應位置



若 r >=ceil(Round,num_gpu)時

將 dev_dist[1] 第 r -row 的所有 block memcopy 到 dev_dist[0]相對應位置



Profiling Results

Occupancy

GPU1		GPU2	
kernel	**Achieved occupancy(avg.)**	**kernel**	**Achieved occupancy(avg.)**
Phase1	0.495939	Phase1	0.495970
Phase2	0.858472	Phase2	0.861369
Phase3	**0.887581**	**Phase3**	**0.889586**

sm efficiency

GPU1		GPU2	
kernel	**sm efficiency**	**kernel**	**sm efficiency**
Phase1	3.68%	Phase1	3.69%
Phase2	66.89%	Phase2	67.72%
Phase3	**91.91%**	**Phase3**	**92.03%s**

Share memory load/store throughput

GPU1			GPU2		
kernel	**Load throughput**	**Store throughput**	**kernel**	**Load throughput**	**Store throughput**
Phase1	52.744GB/s	17.948GB/s	Phase1	53.059GB/s	18.056GB/s
Phase2	1284.0GB/s	672.14GB/s	Phase2	1280.9GB/s	670.50GB/s
Phase3	**2503.0GB/s**	**104.29GB/s**	**Phase3**	**2482.4GB/s**	**108.74GB/s**

Global load/store throughput

GPU1			GPU2		
kernel	**Load throughput**	**Store throughput**	**kernel**	**Load throughput**	**Store throughput**
Phase1	659.33MB/s	659.33MB/s	Phase1	663.27MB/s	663.27MB/s
Phase2	47.552GB/s	23.581GB/s	Phase2	47.436GB/s	23.523GB/s
Phase3	**185.20GB/s**	**62.280GB/s**	**Phase3**	**186.87GB/s**	**61.765GB/s**

Experiment and Analysis

System Spec: hades server

Time distribution:

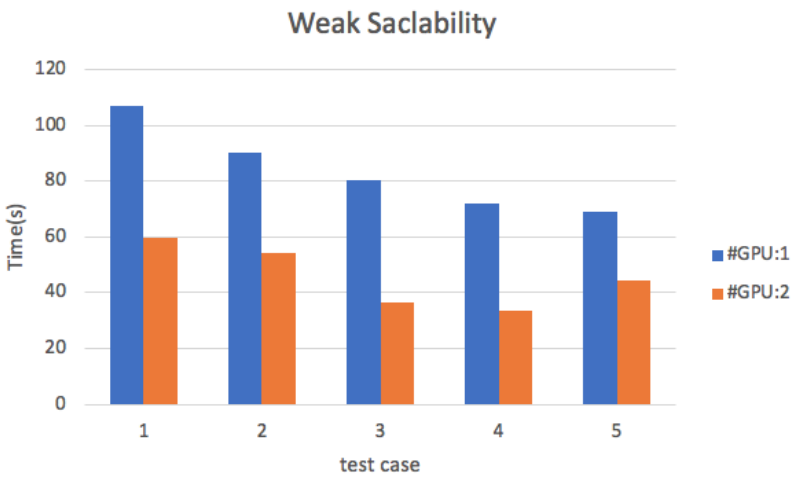
採用 nvprof 來查詢 kernel compute 以及 memry htd,dth time

使用 omp_get_wtime()來紀錄 I/O time

*nvpro 會測到兩個 gpu 的時間加總，故 compute and memory copy time 要除 2

(b)Weak Scalability

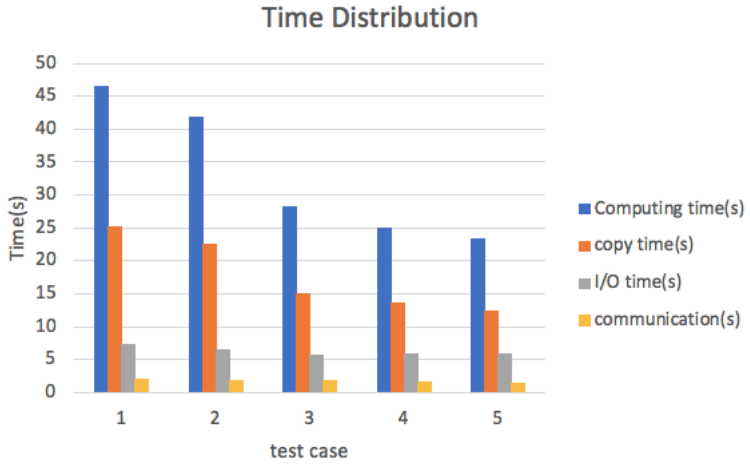
	#GPU:2	#GPU:1
Test case	**cost time(s)**	**cost time(s)**
p35k1	59.600828	106.844367
P34k1	54.333784	89.929763
P33k1	36.412536	80.288615
P32k1	33.388332	71.838691
P31k1	44.56218	69.112767



#GPU1:one gpu compute time
#GPU2:two gpu compute time

(c.)Time distribution

Test case	Computing time(s)	copy time(s)	I/O time(s)	communication(s)
p35k1	46.59345	25.2605	7.374005	2.094
P34k1	41.94735	22.70815	6.626022	1.9675
P33k1	28.35855	15.06925	5.783509	1.859
P32k1	25.08395	13.6616	5.887422	1.723
P31k1	23.391	12.4346	5.883014	1.528



Optimization :

1. memory copy2D：使用 memcpy2D 來加速存取時間

2. cudaHostalloc:使用 pinned memory 來減少 gpu 取得 host data 時間

Experiment and Analysis

從這次作業我學到如何運用 cuda 來做 multi-gpu 的運算，來加速程式運算。如何將 dist 拆成 2 塊平行化，並且利用不同的技巧來減少資料傳輸，複製的時間。本次作業困難點就是 data 該如何分割給 2 個 gpu 實作，並且只傳輸少部分的 data 給其他 gpu 平行運算，運算完後再合併 data。另外也學習到使用不同的 memcpy 和 allocate 機制來大幅改善效能