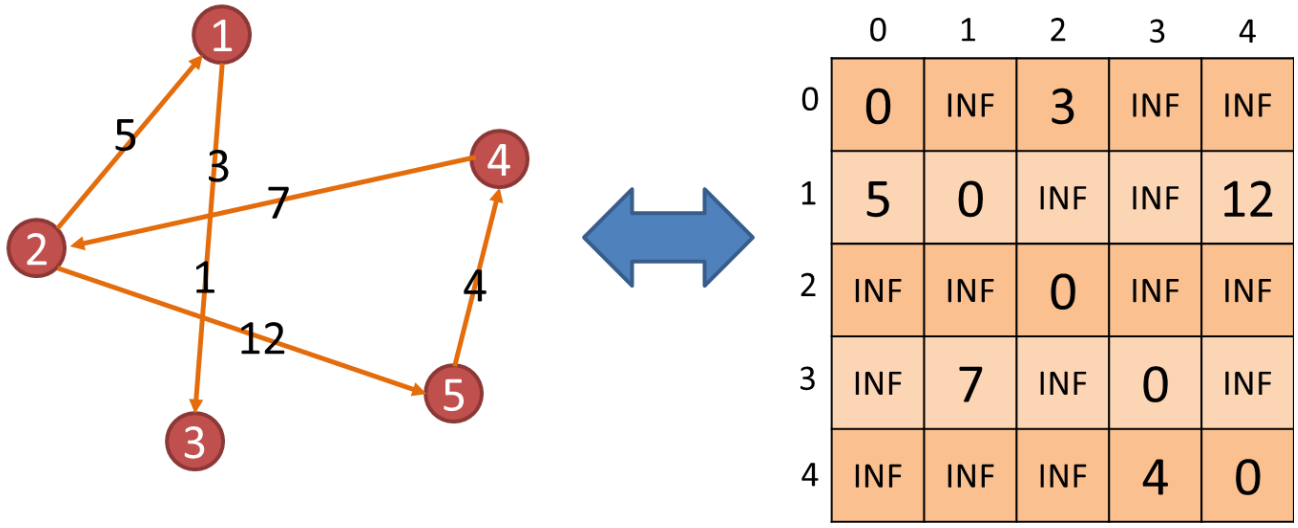
HW 4: Blocked All-Pairs Shortest Path

105062600 Yi-cheng,Chao 15:30, January 18, 2017

Design Concepts – Single-GPU

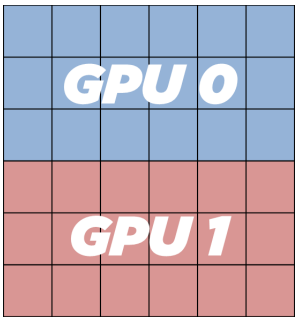
1. 資料儲存方式我使用 adjacency martix 記錄每條 directed path weighted，第一列的值為 source，第二列為 destination，第三行為 weighted，並依資訊存入矩陣中，此外在初始化 adjacency martix 時先預設如果 i==j 則給 0，其他則給 infinity (infinity 定義為 1e7)，因此產生完的矩陣如下圖所示：



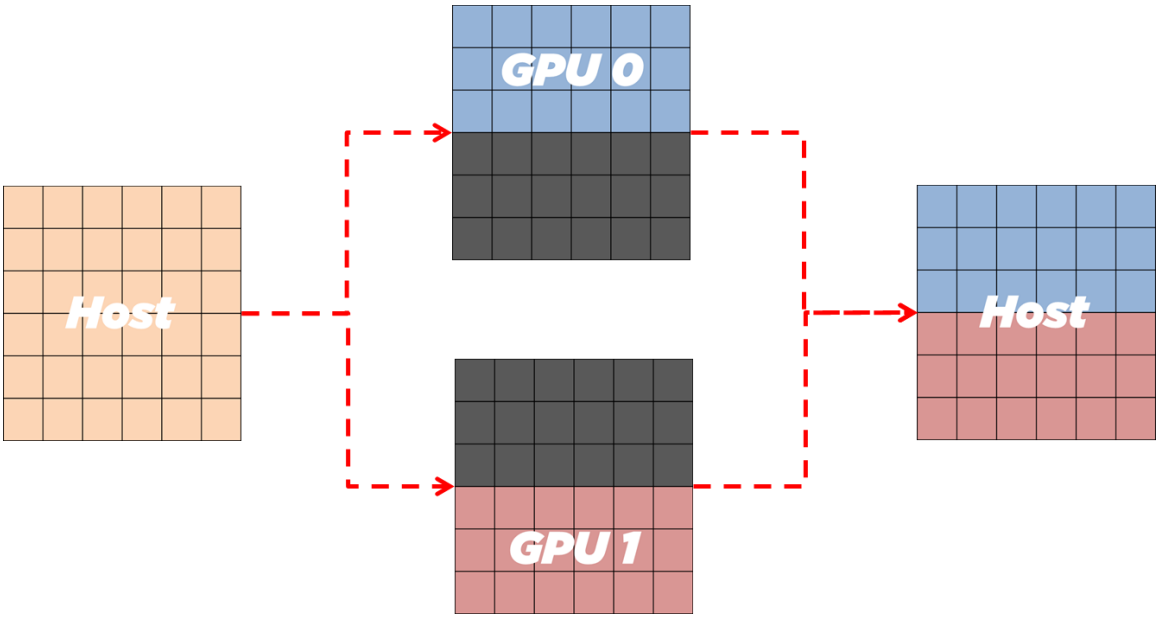
2. 此外我使用一維陣列來儲存二維的 adjacency matrix 內的 data，原本的 adjacency matrix 內的 data 位置使用 row major 方式轉換後儲存到一維陣列中。
3. Grid dimension 部分我使用 Blocked 為單位設定大小，phase1 的話為(1)，phase2 為(round – 1, 2)(round 為執行次數)，phase3 則為(round-1, round-1)來進行分配。
4. 因為 blocking factor 為自己設定，我使用 maxThreadPerBlock 得到的資訊為 1024，因此我不考慮 blocking factor 超過 32 的情況，僅考慮 blocking factor 小於 vertex number 的時候把 blocking factor 縮小為與 vertex number 相同，而在建立 threadperblock 時便使用 cuda 內建的型態 dim3，宣告 Block dimension 為(blocking factor, blocking factor)。
5. Launch kernal 後每個 thread 便可以依照自己的 threadIdx 來知道分配到計算該 block 的哪個位置，而有 Grid 有多個 block 時則使用 blockIdx 知道自己分配到計算哪個 block。
6. 此外會發生 vertex number 不能被 blocking factor 整除的狀況，因此我計算還要補上多少 blocking factor 才能整除 vertex number，之後 vertex number 補上這部份去產生我的 host memory 及 device memory，如此便能保證切割出來的 Block 可以不會有餘數。

Design Concepts – Multi-GPU-OpenMP

1. 大致上架構、資料儲存、Grid/Block 分割同 Single-GPU 版本，但多了兩張卡之間的溝通，因此在 cudaMalloc 時也要兩台機器都分配到 Memory，大小則也是完整的 adjacency matrix 空間。
2. Phase1&2 因為執行的時間影響不大，因此我讓各自的 GPU 都去計算自己的 phase1&phase2，而 phase3 是我去平行給 multi-GPU 的部分，我的分配方法是割成上下兩塊，若有無法整除的部分則交給第二塊 GPU 進行計算，如下圖所示：



3. 每個 iteration 會把每次計算完自己的部分後 cudaMemcpyDeviceToHost 回 host matrix 做集合，之後再由 cudaMemcpyHostToDevice 去分配直到結束，每個 iteration 資料分配的示意圖如下圖所示：



Design Concepts –Multi-GPU-MPI

1. MPI 的版本資料分割概念與 OpenMP 相似，每個 MPI Process 都會各自進行讀檔，僅原本從 OpenMP Thread ID 判斷的部分改為 MPI Rank，此外因為架構上是 Distributed Memory，以及資料間彼此溝通使用 MPI_Send/MPI_Recv，以及在最後資料蒐集部分使用 MPI_Sendrecv 來進行資料的彙整。

Profiling Results

Single-GPU

```
==28026== Profiling application: ./HW4_cuda.exe testcase/in5 out 16
==28026== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 98.50%  11.6311s       375  31.016ms  30.742ms  32.705ms  FW_phase3(int, int, int, int*, int)
  0.55%  64.846ms         1  64.846ms  64.846ms  64.846ms  [CUDA memcpy DtoH]
  0.52%  61.820ms       375  164.85us  157.51us  171.59us  FW_phase2(int, int, int, int*, int)
  0.39%  46.519ms         1  46.519ms  46.519ms  46.519ms  [CUDA memcpy HtoD]
  0.03%  3.5047ms       375  9.3450us  9.1240us  10.111us  FW_phase1(int, int, int, int*, int)

==28026== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 97.34%  11.6663s       750  15.555ms  155.02us  32.709ms  cudaDeviceSynchronize
  1.32%  157.75ms         2  78.873ms  1.7470us  157.74ms  cudaEventCreate
  1.20%  143.88ms         2  71.942ms  46.559ms  97.325ms  cudaMemcpy
  0.10%  12.282ms      1125  10.917us  6.2960us  90.934us  cudaLaunch
  0.01%  1.6239ms      5625     288ns     191ns  12.289us  cudaSetupArgument
  0.01%  672.10us      1125     597ns     290ns  20.992us  cudaConfigureCall
  0.00%  507.97us       182  2.7910us     187ns  103.36us  cuDeviceGetAttribute
  0.00%  418.29us         1  418.29us  418.29us  418.29us  cudaMalloc
  0.00%  418.14us         2  209.07us  207.82us  210.32us  cuDeviceTotalMem
  0.00%  305.78us         1  305.78us  305.78us  305.78us  cudaFree
  0.00%  223.42us         1  223.42us  223.42us  223.42us  cudaGetDeviceProperties
  0.00%  49.950us         2  24.975us  22.339us  27.611us  cuDeviceGetName
  0.00%  35.524us         1  35.524us  35.524us  35.524us  cudaFreeHost
  0.00%  20.689us         2  10.344us  10.309us  10.380us  cudaEventRecord
  0.00%  5.4400us         3  1.8130us     316ns  4.7540us  cuDeviceGetCount
  0.00%  4.2170us         2  2.1080us     528ns  3.6890us  cudaSetDevice
  0.00%  3.7260us         1  3.7260us  3.7260us  3.7260us  cudaEventSynchronize
  0.00%  3.3970us         1  3.3970us  3.3970us  3.3970us  cudaEventElapsedTime
  0.00%  2.8740us         1  2.8740us  2.8740us  2.8740us  cudaGetDeviceCount
  0.00%  1.7980us         6     299ns     196ns     423ns  cuDeviceGet
```

Multi-GPU-OpenMP

```
==30236== Profiling application: ./HW4_openmp.exe testcase/in5 out 32
==30236== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 95.39%  7.23568s       376  19.244ms  16.682ms  21.827ms  FW_phase3(int*, int, int, int, int, int)
  2.24%  170.19ms       376  452.63us  395.90us  511.57us  FW_phase2(int*, int, int, int, int)
  1.39%  105.47ms       378  279.02us  126.97us  22.822ms  [CUDA memcpy HtoD]
  0.88%  67.106ms       190  353.19us  119.36us  21.935ms  [CUDA memcpy DtoH]
  0.09%  7.1389ms       376  18.986us  17.344us  20.497us  FW_phase1(int*, int, int, int, int)

==30236== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 91.44%  5.80848s       568  10.226ms  140.61us  38.662ms  cudaMemcpy
  2.86%  181.37ms       752  241.18us  4.6980us  641.37us  cudaDeviceSynchronize
  1.89%  120.14ms         2  60.072ms  2.2120us  120.14ms  cudaEventCreate
  1.67%  106.05ms         2  53.027ms  579.14us  105.48ms  cudaMalloc
  1.09%  69.124ms         1  69.124ms  69.124ms  69.124ms  cudaMallocHost
  0.73%  46.440ms         1  46.440ms  46.440ms  46.440ms  cudaFreeHost
  0.21%  13.102ms      1128  11.614us  7.0160us  92.474us  cudaLaunch
  0.03%  2.1915ms         2  1.0957ms  1.0597ms  1.1318ms  cudaGetDeviceProperties
  0.03%  2.0653ms      6016     343ns     198ns  61.683us  cudaSetupArgument
  0.02%  992.70us         2  496.35us  386.77us  605.93us  cudaFree
  0.01%  822.97us       182  4.5210us     325ns  166.08us  cuDeviceGetAttribute
  0.01%  743.01us         2  371.50us  368.93us  374.08us  cuDeviceTotalMem
  0.01%  593.41us      1128     526ns     292ns  17.423us  cudaConfigureCall
  0.00%  88.659us         8  11.082us  5.2110us  26.492us  cudaSetDevice
  0.00%  79.496us         2  39.748us  35.022us  44.474us  cuDeviceGetName
  0.00%  41.152us         2  20.576us  10.572us  30.580us  cudaEventRecord
  0.00%  26.004us         1  26.004us  26.004us  26.004us  cudaEventSynchronize
  0.00%  6.6430us         3  2.2140us     512ns  5.3100us  cuDeviceGetCount
  0.00%  4.3950us         1  4.3950us  4.3950us  4.3950us  cudaEventElapsedTime
  0.00%  3.2020us         6     533ns     366ns     649ns  cuDeviceGet
  0.00%  1.5120us         1  1.5120us  1.5120us  1.5120us  cudaGetDeviceCount
```

Multi-GPU-MPI

```
[105062600@pp31 hw4]$ nvprof -i profile.out.0
======== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 95.34%  3.27042s       375  8.7211ms  8.5939ms  8.7584ms  FW_phase3(int*, int, int, int, int, int)
  2.04%  69.850ms       376  185.77us  122.40us  23.528ms  [CUDA memcpy HtoD]
  1.29%  44.386ms       188  236.10us  114.97us  22.360ms  [CUDA memcpy DtoH]
  1.28%  43.844ms       375  116.92us  114.83us  119.10us  FW_phase2(int*, int, int, int, int)
  0.05%  1.7232ms       375  4.5950us  4.5170us  4.8470us  FW_phase1(int*, int, int, int, int)

======== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 87.17%  1.76919s       564  3.1369ms  139.66us  31.232ms  cudaMemcpy
  6.55%  132.85ms         2  66.426ms  1.7870us  132.85ms  cudaEventCreate
  4.32%  87.577ms         1  87.577ms  87.577ms  87.577ms  cudaMallocHost
  1.15%  23.293ms         1  23.293ms  23.293ms  23.293ms  cudaFreeHost
  0.55%  11.136ms      1125  9.8980us  6.8520us  127.81us  cudaLaunch
  0.07%  1.4246ms       182  7.8270us     257ns  331.45us  cuDeviceGetAttribute
  0.07%  1.4216ms      6000     236ns     166ns  12.187us  cudaSetupArgument
  0.03%  644.88us         2  322.44us  312.52us  332.36us  cuDeviceTotalMem
  0.03%  544.15us         1  544.15us  544.15us  544.15us  cudaGetDeviceProperties
  0.02%  471.27us      1125     418ns     284ns  5.4990us  cudaConfigureCall
  0.02%  459.16us         1  459.16us  459.16us  459.16us  cudaMalloc
  0.01%  223.61us         2  111.80us  29.591us  194.02us  cuDeviceGetName
  0.01%  169.76us         1  169.76us  169.76us  169.76us  cudaFree
  0.00%  31.553us         2  15.776us  9.7240us  21.829us  cudaEventRecord
  0.00%  26.361us         1  26.361us  26.361us  26.361us  cudaEventSynchronize
  0.00%  8.2640us         1  8.2640us  8.2640us  8.2640us  cudaSetDevice
  0.00%  6.2760us         3  2.0920us     407ns  5.2700us  cuDeviceGetCount
  0.00%  2.8640us         6     477ns     281ns     682ns  cuDeviceGet
  0.00%  2.8250us         1  2.8250us  2.8250us  2.8250us  cudaEventElapsedTime
```

```
[105062600@pp31 hw4]$ nvprof -i profile.out.1
======== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 97.26%  4.24045s       375  11.308ms  11.205ms  11.347ms  FW_phase3(int*, int, int, int, int, int)
  1.29%  56.328ms       375  150.21us  148.08us  151.70us  FW_phase2(int*, int, int, int, int)
  0.85%  36.979ms       376  98.348us  64.191us  12.534ms  [CUDA memcpy HtoD]
  0.55%  24.191ms       189  127.99us  60.254us  12.243ms  [CUDA memcpy DtoH]
  0.05%  1.9718ms       375  5.2580us  5.1620us  5.5250us  FW_phase1(int*, int, int, int, int)

======== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 93.11%  4.32669s       565  7.6579ms  80.087us  23.729ms  cudaMemcpy
  3.74%  173.93ms         1  173.93ms  173.93ms  173.93ms  cudaMallocHost
  2.08%  96.625ms         2  48.312ms  1.8610us  96.623ms  cudaEventCreate
  0.51%  23.821ms         1  23.821ms  23.821ms  23.821ms  cudaFreeHost
  0.23%  10.701ms       182  58.793us     305ns  4.3072ms  cuDeviceGetAttribute
  0.22%  10.047ms      1125  8.9300us  6.2400us  57.777us  cudaLaunch
  0.04%  2.0803ms      6000     346ns     238ns  17.347us  cudaSetupArgument
  0.02%  787.15us         2  393.57us  360.42us  426.73us  cuDeviceTotalMem
  0.01%  523.32us         1  523.32us  523.32us  523.32us  cudaGetDeviceProperties
  0.01%  516.04us      1125     458ns     265ns  17.119us  cudaConfigureCall
  0.01%  369.03us         2  184.51us  67.348us  301.68us  cuDeviceGetName
  0.01%  355.55us         1  355.55us  355.55us  355.55us  cudaMalloc
  0.00%  222.98us         1  222.98us  222.98us  222.98us  cudaFree
  0.00%  17.306us         2  8.6530us  7.3200us  9.9860us  cudaEventRecord
  0.00%  8.6720us         1  8.6720us  8.6720us  8.6720us  cudaSetDevice
  0.00%  7.2290us         3  2.4090us     517ns  5.7720us  cuDeviceGetCount
  0.00%  5.9830us         1  5.9830us  5.9830us  5.9830us  cudaEventSynchronize
  0.00%  4.2510us         6     708ns     323ns  1.2980us  cuDeviceGet
  0.00%  1.5770us         1  1.5770us  1.5770us  1.5770us  cudaEventElapsedTime
```

Experiment & Analysis

System Environment

執行程式使用課程提供的 GPU Cluster (140.114.91.176)。

Time Measurement

使用 cuda 內建的 cudaEven 來進行時間紀錄。

Performance Measurement

測資隨我的分析方法而改變，因此列在每個測資的下方。

System Spec

GPU/CPU 詳細資訊及使用指令如下，CPU/GPU 僅顯示其中一組，如下所示：

[105062600@pp31] nvidia-smi

```
Tue Jan 17 09:26:41 2017
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 367.48                 Driver Version: 367.48                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla C2070         On   | 0000:03:00.0     Off |                  Off |
| 30%   71C    P8    N/A /  N/A |      0MiB /  6066MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla M2090         On   | 0000:06:00.0     Off |                  Off |
| N/A   N/A    P0    82W /  N/A |    230MiB /  6066MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
```

[105062600@pp31] cat /proc/cpuinfo

```
model name      : Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz
stepping        : 7
microcode       : 0x710
cpu MHz         : 1199.859
cache size      : 20480 KB
physical id     : 1
siblings        : 8
core id         : 6
cpu cores       : 8
apicid          : 44
initial apicid  : 44
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx f
xsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopolog
y nonstop_tsc aperfmperf eagerfpu pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 cx16 xtpr pdcm pcid
dca sse4_1 sse4_2 x2apic popcnt tsc_deadline_timer aes xsave avx lahf_lm ida arat epb pln pts dtherm tpr_shadow
vnmi flexpriority ept vpid xsaveopt
bogomips        : 5192.71
clflush size    : 64
cache_alignment : 64
address sizes   : 46 bits physical, 48 bits virtual
power management:
```

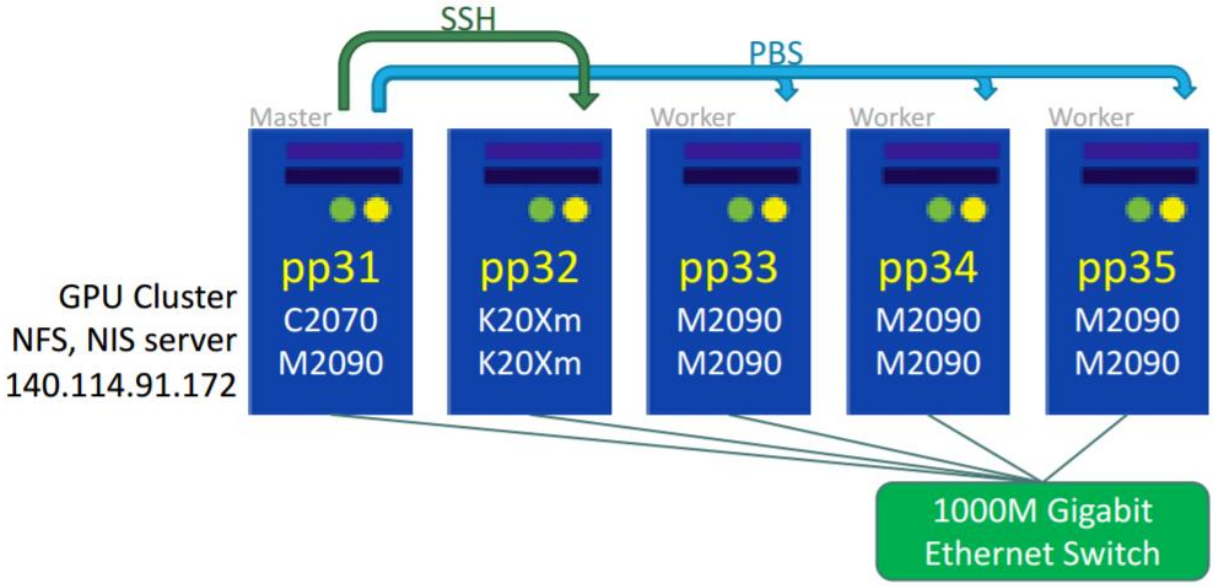
課程提供之 Tesla M2090 詳細規格如下：

Table 1. Tesla M2090 Board Configuration

Specification	Description
Generic SKU reference	•900-21030-0040-100 (ships without bracket) •900-21030-0045-100 (ships with bracket)
Chip	T20A GPU
Package size	42.5 mm × 42.5 mm
Processor clock	1.3 GHz
Memory clock	1.85 GHz
Memory I/O	384-bit GDDR5
Memory configuration	24 pcs 128M × 16 GDDR5 SDRAM
External connectors	None
Internal connectors and headers	•8-pin PCI Express power connector •6-pin PCI Express power connector
Board power	<= 225 W
Thermal cooling solution	Passive heat sink

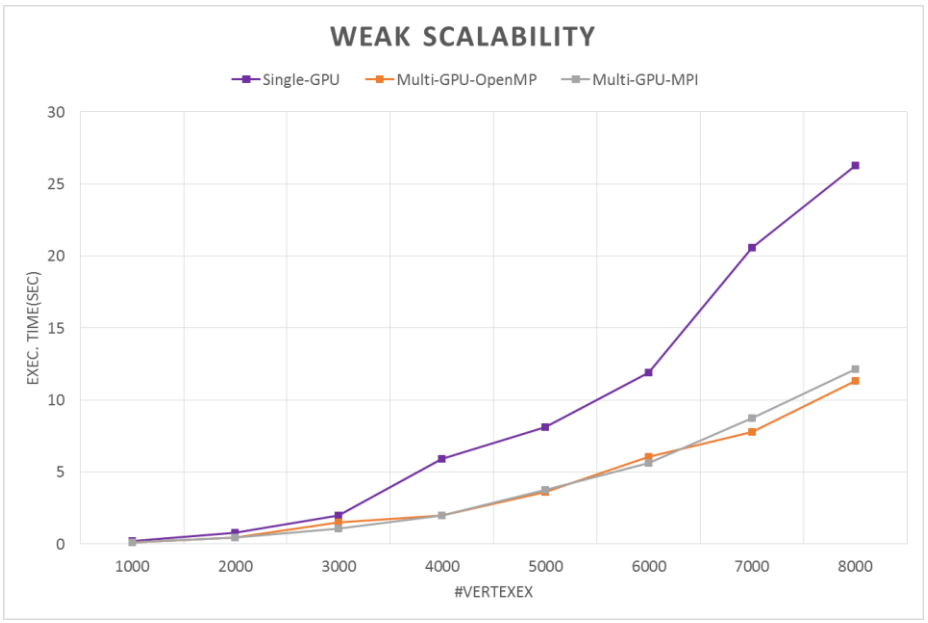
(圖片引用自 Nvidia-Tesla M2090 開發板規格文件)

課程提供之 GPU 使用網路為 Ethernet，如下圖所示：



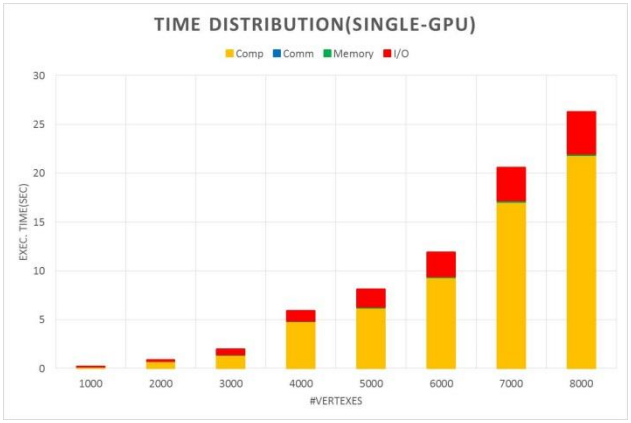
(圖片引用自課堂講義 PP2016_Lab2_tutorial)

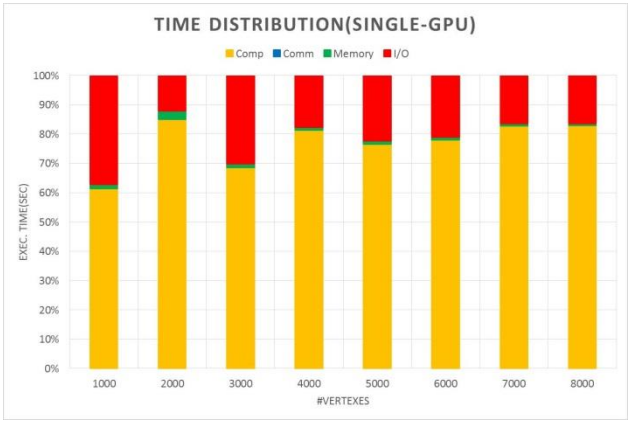
Weak Scalability Analysis
(x-axis: # of Vertexes; y-axis: Execution time (s))
Input parameter: #vertexes 1000~8000



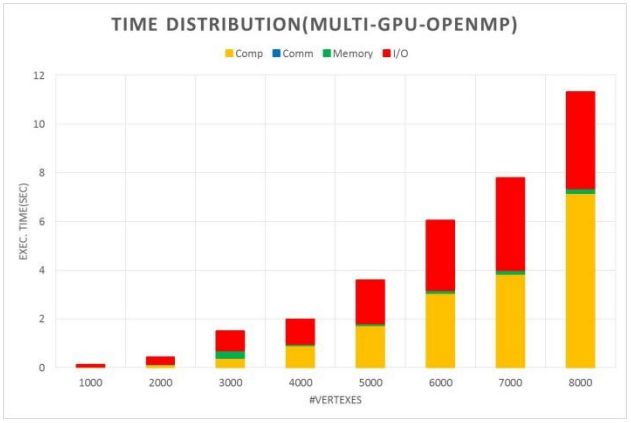
由圖中可以看到雙卡的加速效果約可以縮短一半的執行時間，而因為 OpenMP 版本的記憶體是使用 global shared memory，比起使用 distributed memory 且還要進行 Send/Recv 的 MPI 版本所花費的整體時間在小一點。

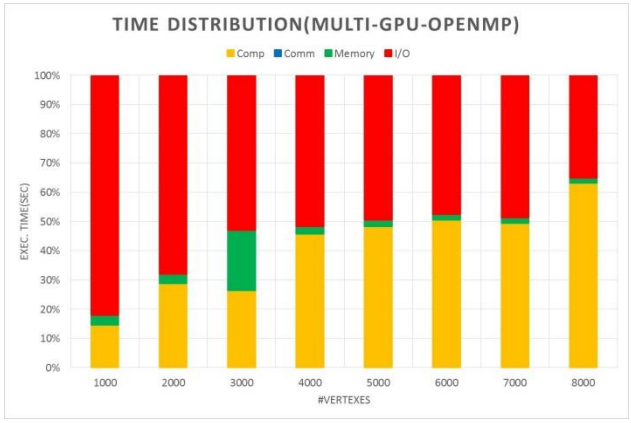
Time Distribution Analysis
(x-axis: # of Vertexes; y-axis: Execution time (s))
Input parameter: #vertexes 1000~8000



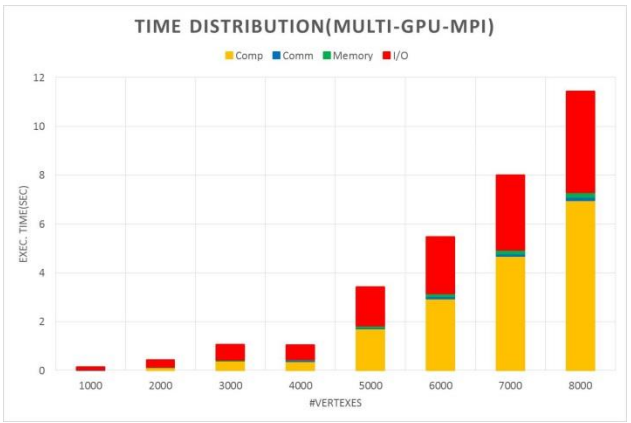


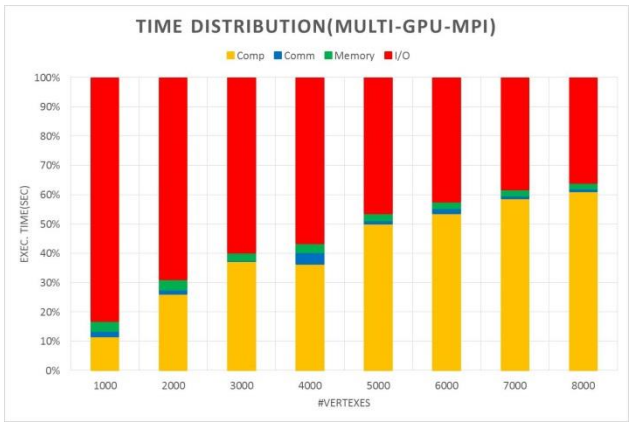
Single GPU 主要的 Bottleneck 可以很清楚的看到都是由 computation time 所 dominate，但因為隨著頂點數的增加，I/O time 的 cost 亦會增加，因此兩者所佔的比例從圖中僅看出隨著頂點數的增加 computation 所佔執行時間逐漸上升，但這個情況會在頂點數越大的時候影響越微小。





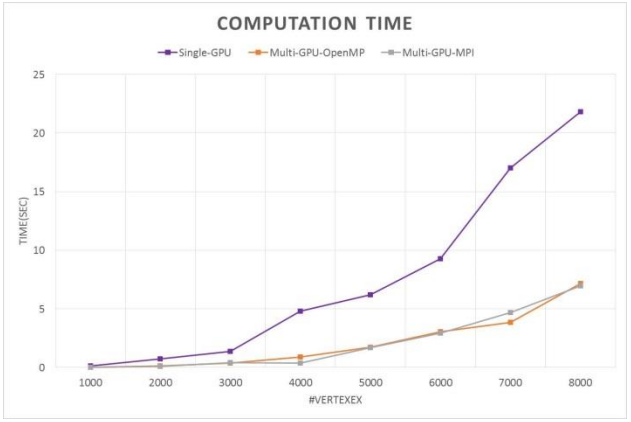
Multi-GPU 有效的降低了執行時間，但由於 I/O time 是不可能去 speed up 的部分，加速主要節省的都是 computation time 的部分，雖然有額外的 memory time cost，但加速的效益下仍可以看到執行時間的減少，此外因為 OpenMP 使用 global shared memory 直接進行溝通，因此並沒有任何的 communication cost。

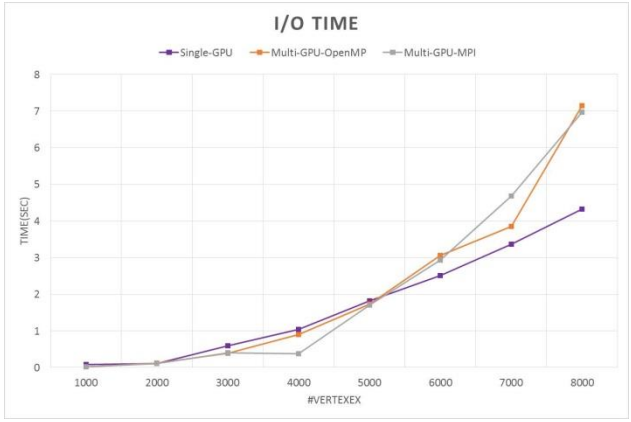




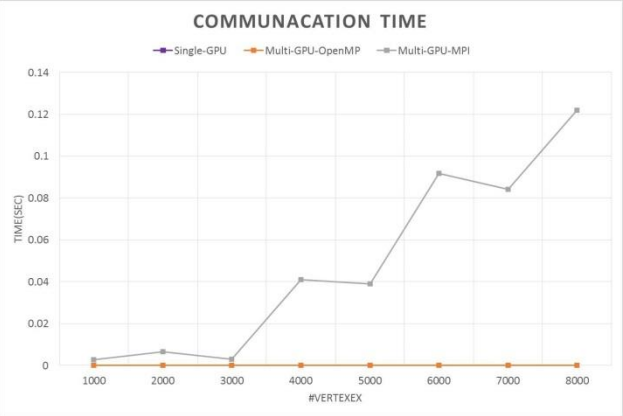
Multi-GPU-MPI 版本多了 communication time 的額外負擔，使得執行時間上略高於 OpenMP 版本，除此之外大部分的特性與 OpenMP 版本相同，都是大幅度的加速了 computation time 的部分，而 I/O time 的 cost 不太有變動。

Weak Scalability Time Distribution Analysis
(x-axis: # of Vertexes; y-axis: Execution time (s))
Input parameter: #vertexes 1000~8000



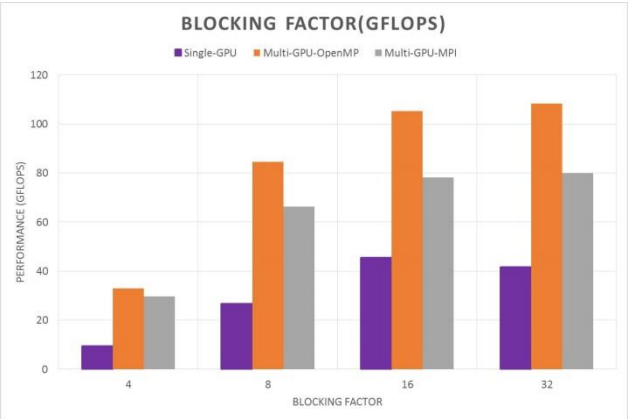


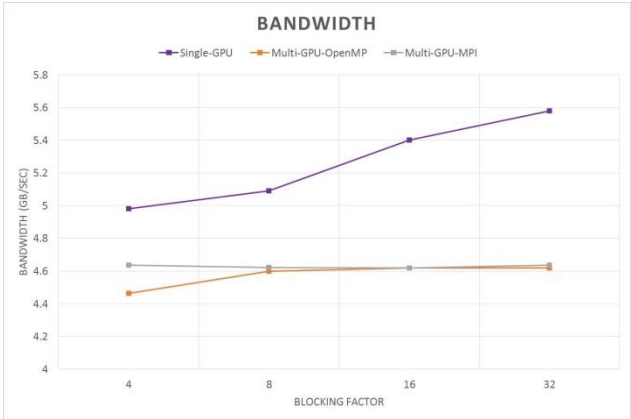
獨立抓出各個時間來分析，我們可以看到 computation time 在 single-GPU 和 multi-GPU 的影響下所節省的時間，然而因為分割後計算的切割都差不多，因此我們可以看到兩種版本的 computation time 差距都是在誤差內的結果，而 I/O time 則是因為每次 GPU 使用狀況不一樣，因此 I/O time 的浮動值會略大，但可以看出三者都有差不多的 I/O time，正是因為讀檔的#vertexes 與 #edges 都相同的結果。



而因為 multi-GPU 版本需要多次無論是 Host to Device 或者 Device to Host 的 memory copy，比起 single-GPU 僅需各一次的 memory copy，導致 memory copy time 的 cost 相對大，而因為 OpenMP 版本使用 global shared memory 需負擔更多次的 memory copy，而這部分在 MPI 版本中則是由 Send/Recv 代替，因此可以看到 OpenMP 所需要花費的 memory copy time 是略大於 MPI 版本的；而在 communication time 的部分則為 MPI 版本獨有的額外支出，之所以呈現梯狀我的猜測是因為每次的 iteration 後都要 Send/Recv 來進行資料的溝通，然而每次的 computation 的結束時間不一，對於特定情況會導致互相建立 Blocking 的 acknowledge 時的等待時間較少導致 cost 會略低形成上圖的結果。

Blocking Factor-Gflops/Bandwidth
Input parameter: testcase/in5

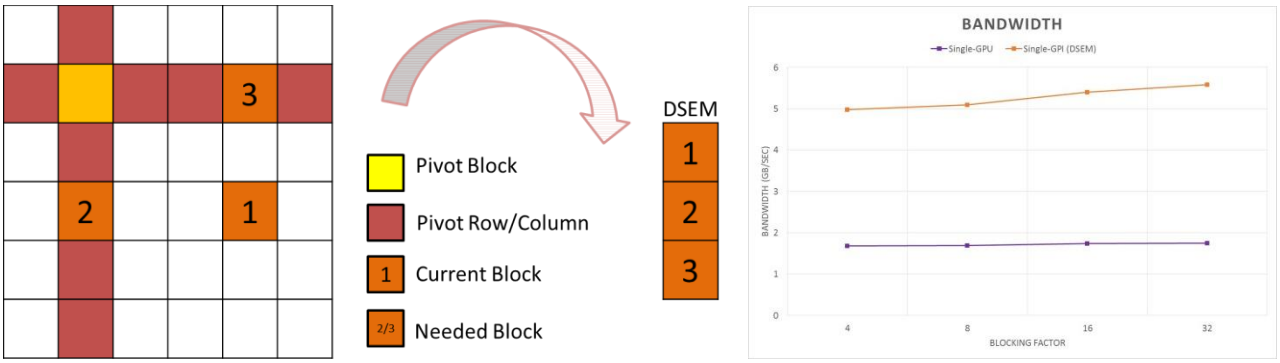




由於 cuda 中每個 thread 是以 32 個為一綑的 wrap 來執行，因此 block size 要為 32 的倍數會是比較恰當的分配方法，由圖可判別三種版本分別最佳 performance 在 blocking factor16/32/16 (分別對應 single/OpenMP/MPI)，而 Bandwidth 其實 multi-GPU 版本其中一個的會是 6.10GB/s 上下，而另一個則只有 3.10GB/s 左右，推測是因為 band conflict 的問題導致在 bandwidth 會受到影響所導致。

Optimization

因為 blocking factor 由輸入而定，因此我使用 dynamic shared momory 來優化我的程式，我的方法為使用三個等同 block 大小的一維 shared memory 分別儲存該計算位置所要存取到的資料，如下圖所示：



由於使用 shared memory 可以大幅增加 Bandwidth 如上圖所示，因此能夠在提高 Performance 使得執行時間更來的快。

Difficulty Encounter

這次使用的 Blocked Floyd-Warshall 演算法在理解提供之 Sequential 版本變傷透了腦筋，最後與同學討論後才理解 phase2&3 要去分開四個部分來計算，過程十分挫折，也在 debug 途中遇到了不少想好久才發現只是小 bug 的問題，像是 blocking factor 如果大於整體的 vertex number 的處理，以及最後要使用 cudaFree 把使用的 GPU memory 歸還，或者是在 MPI 版本的最後寫檔時不能兩個 rank 都進行開檔否則會因為負責寫入的 processor 只有一個，導致檔案被開啟後馬上關閉之後就進行 judge，這些種種都導致在送入 hw4judge 的時候出現一些零零散散的錯誤，導致在程式撰寫過程中有時候都會鬱悶到無法開電腦，尤其可遲交的時間不像之前充裕，而我理解力又比較差一點，總之能開始寫報告真的是很開心的。

Conclusion & Experience

Blocked Floyd-Warshall 有點類似原本的 Floyd-Warshall Algorithm，但像是原本的每個步驟都放大到 Blocked index 去執行，GPU 計算的 performance 非常快，六千點的 APSP 也能壓在十幾秒內執行完畢，相比作業 3 的 SSSP 十分能體驗到其強大的運算能力以及魅力，美中不足大概就對 Cuda Optimazation 的了解不夠透徹，程式撰寫期間很多時候都在 Trial & Error 試方法，也知道自己程式能有很多改進的地方，會使用 Cuda 的同學一定能把 performance 飆到極限，讓我十分感覺自嘆不如，原本當初希望成績能亮眼最後只能淪落到只求及格，過程演化讓我每次都深深感覺到自己知識/能力上的不足，希望這是最後一次我拿大學部是材料系做藉口吧 XD！