CS542200 Parallel Programming

Homework 4: Blocked All-Pairs Shortest Path

104062703 曾若淳

Implementation

對於一個有N個點的圖， 我使用 blocked Floyd-Warshall來算出所有path 的最短路徑。基本上切data 的方法就是讓cuda 的block 大小就等於 blocked Floyd-Warshall演算法中的block 大小。

Single-GPU: Cuda

需要計算的NxN個路徑，第一維和第二維各會被分成$round = \frac{N}{B}$個block，演算法在每個iteration 必須按照順序計算三個階段，但每個階段內不同block 間可以平行計算。我讓每個 block 內每個thread 計算 B個 $dist[a][b] = min(dist[a][b], dist[a][k] + dist[k][b])$ 的最短路徑的更新。

假如直接從global memory存取的話，會需要存取 $3B$ 個位置，整個block 會同時存取$3B^3$次，很慢而且不同thread 可能會存取重複的位置發生衝突。所以我使用了shared memory，事先將這個block 的所有位置dist[a][b] 以及所有dist[a][k], dist[k][b] 這三個block的資料放到shared memory裡，這部份可以平行做，讓每個thread 存取 $O(1 + 2B)$ 個位置即可。

當這個block 所有會用到的資料都放到shared memory後，就可以非常快速的計算最短路徑，算完每個thread在平行的將它負責的那個最短路徑dist[a][b]放回global memory裡。

Multi-GPU: OpenMP

有n 張GPU，我會開n 個OpenMP thread，每個控制一張GPU，而每個GPU負責計算 $\frac{round}{n} \times round$個block，GPU內計算方式和上面的single-GPU版本相同，差別在於每個iteration的三個階段間，必須將GPU內的資料做同步。我實做資料同步的方法為，在host 上pinned 一塊memory，GPU 會先將自己更新的部份放到host 上對應的位置，這部份使用openmp 的barrier來確保每張GPU都已經放到對應位置，之後GPU才將其他GPU的更新從host拿回device上。

Multi-GPU: MPI

基本上和OpenMP實做非常相似，有 n 張GPU，我會開的n 個MPI process。每個iteration 內每個階段間平行計算路徑的部份也跟single-GPU版本相同，而階段間的資料同步，我使用MPI_Barrier來確保 GPU 都已經將負責部份的更新到 host 對應的memory，然後GPU才會load 其他GPU負責的部份。

Performance Analysis

實驗環境為課程提供的GPU cluster，有兩張Tesla K20的node。

Profiling Results

我使用nvvp，以下為三個版本在problem size=500, block 大小=8,16,32的情況：

Single-GPU

B=8

```
cal(int, int, int**, int, int, int, int, int)
Maximum instruction execution count in assembly: 12021
Average instruction execution count in assembly: 3476
Instructions executed for the kernel: 1147254
Thread instructions executed for the kernel: 36712128
Non-predicated thread instructions executed for the kernel: 32220672

Warp non-predicated execution efficiency of the kernel: 87.8%
Warp execution efficiency of the kernel: 100.0%
```

L1/Shared Memory

Local Loads	0	0 B/s
Local Stores	0	0 B/s
Shared Loads	118563	111.447 GB/s
Shared Stores	112740	105.973 GB/s
Global Loads	131220	15.761 GB/s
Global Stores	11664	1.37 GB/s
Atomic	0	0 B/s
L1/Shared Total	374187	234.552 GB/s

L2 Cache

L1 Reads	134136	15.761 GB/s
L1 Writes	11664	1.37 GB/s

B=16

```
cal(int, int, int**, int, int, int, int, int)
Maximum instruction execution count in assembly: 34160
Average instruction execution count in assembly: 7249
Instructions executed for the kernel: 2392190
Thread instructions executed for the kernel: 76547040
Non-predicated thread instructions executed for the kernel: 66118192

Warp non-predicated execution efficiency of the kernel: 86.4%
Warp execution efficiency of the kernel: 100.0%
```

L1/Shared Memory

Local Loads	0	0 B/s
Local Stores	0	0 B/s
Shared Loads	551725	458.779 GB/s
Shared Stores	514408	427.749 GB/s
Global Loads	556920	59.022 GB/s
Global Stores	29120	3.027 GB/s
Atomic	0	0 B/s
L1/Shared Total	1652173	948.577 GB/s

L2 Cache

L1 Reads	567840	59.022 GB/s
L1 Writes	29120	3.027 GB/s

B=32

```
cal(int, int, int**, int, int, int, int, int)
```

Maximum instruction execution count in assembly: 45471
Average instruction execution count in assembly: 10004

Instructions executed for the kernel: 3301326
Thread instructions executed for the kernel: 105639080
Non-predicated thread instructions executed for the kernel: 90933588

Warp non-predicated execution efficiency of the kernel: 86.1%
Warp execution efficiency of the kernel: 100.0%

Local Loads	0	0 B/s
Local Stores	0	0 B/s
Shared Loads	1728929	471.052 GB/s
Shared Stores	1474683	401.782 GB/s
Global Loads	1528640	52.616 GB/s
Global Stores	43520	1.482 GB/s
Atomic	0	0 B/s
L1/Shared Total	4775772	926.932 GB/s

L2 Cache

L1 Reads	1544960	52.616 GB/s
L1 Writes	43520	1.482 GB/s

Multi-GPU: OpenMP

B=8

```
cal(int, int, int, int**, int, int, int, int, int)
```

Maximum instruction execution count in assembly: 9643
Average instruction execution count in assembly: 2765

Instructions executed for the kernel: 909774
Thread instructions executed for the kernel: 22864128
Non-predicated thread instructions executed for the kernel: 20021376

Warp non-predicated execution efficiency of the kernel: 68.8%
Warp execution efficiency of the kernel: 78.5%

L1/Shared Memory

Local Loads	0	0 B/s
Local Stores	0	0 B/s
Shared Loads	101424	103.93 GB/s
Shared Stores	101382	103.887 GB/s
Global Loads	124416	16.822 GB/s
Global Stores	9216	1.18 GB/s
Atomic	0	0 B/s
L1/Shared Total	336438	225.819 GB/s

L2 Cache

L1 Reads	131328	16.822 GB/s
L1 Writes	9216	1.18 GB/s

B=16

```
cal(int, int, int, int**, int, int, int, int, int)
Maximum instruction execution count in assembly: 4096
Average instruction execution count in assembly: 429
Instructions executed for the kernel: 141396
Thread instructions executed for the kernel: 4379568
Non-predicated thread instructions executed for the kernel: 3983984

Warp non-predicated execution efficiency of the kernel: 88.1%
Warp execution efficiency of the kernel: 96.8%
```

L1/Shared Memory

Local Loads	0	0 B/s
Local Stores	0	0 B/s
Shared Loads	34285	175.88 GB/s
Shared Stores	26463	135.754 GB/s
Global Loads	21908	15.154 GB/s
Global Stores	1952	1.252 GB/s
Atomic	0	0 B/s
L1/Shared Total	84608	328.04 GB/s

L2 Cache

L1 Reads	23632	15.154 GB/s
L1 Writes	2036	1.306 GB/s

B=32

```
cal(int, int, int, int**, int, int, int, int, int)
Maximum instruction execution count in assembly: 95718
Average instruction execution count in assembly: 18134
Instructions executed for the kernel: 5966161
Thread instructions executed for the kernel: 156113392
Non-predicated thread instructions executed for the kernel: 133974776

Warp non-predicated execution efficiency of the kernel: 70.2%
Warp execution efficiency of the kernel: 81.8%
```

Local Loads	0	0 B/s
Local Stores	0	107.403 MB/s
Shared Loads	3488157	532.157 GB/s
Shared Stores	3028032	461.96 GB/s
Global Loads	3170816	66.268 GB/s
Global Stores	90112	1.718 GB/s
Atomic	0	0 B/s
L1/Shared Total	9777117	1,062.21 GB/s

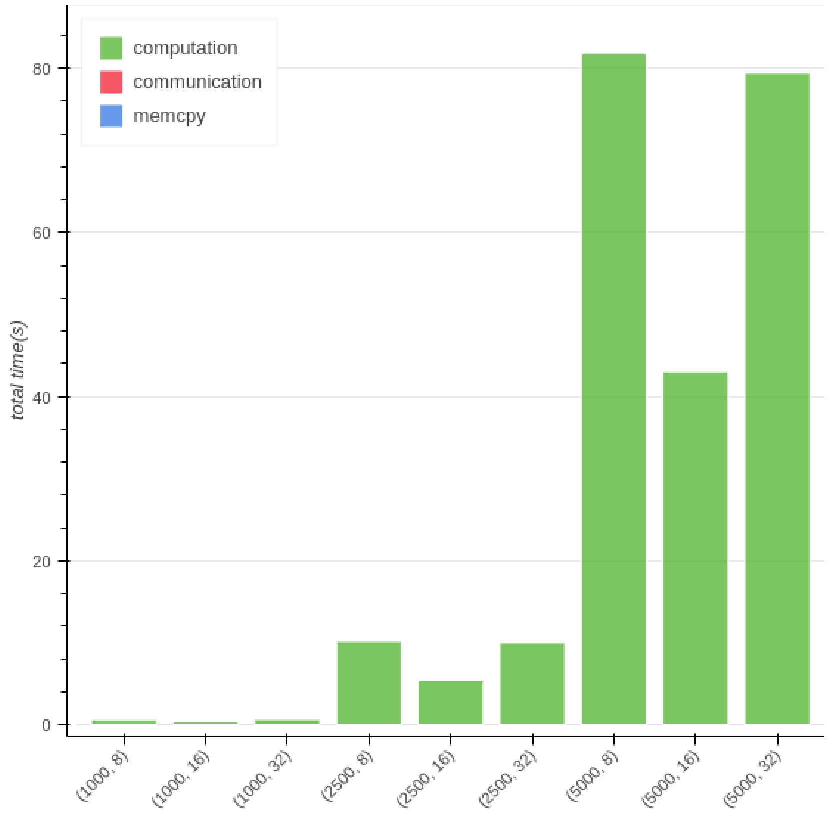
L2 Cache

L1 Reads	3486112	66.481 GB/s
L1 Writes	95766	1.826 GB/s

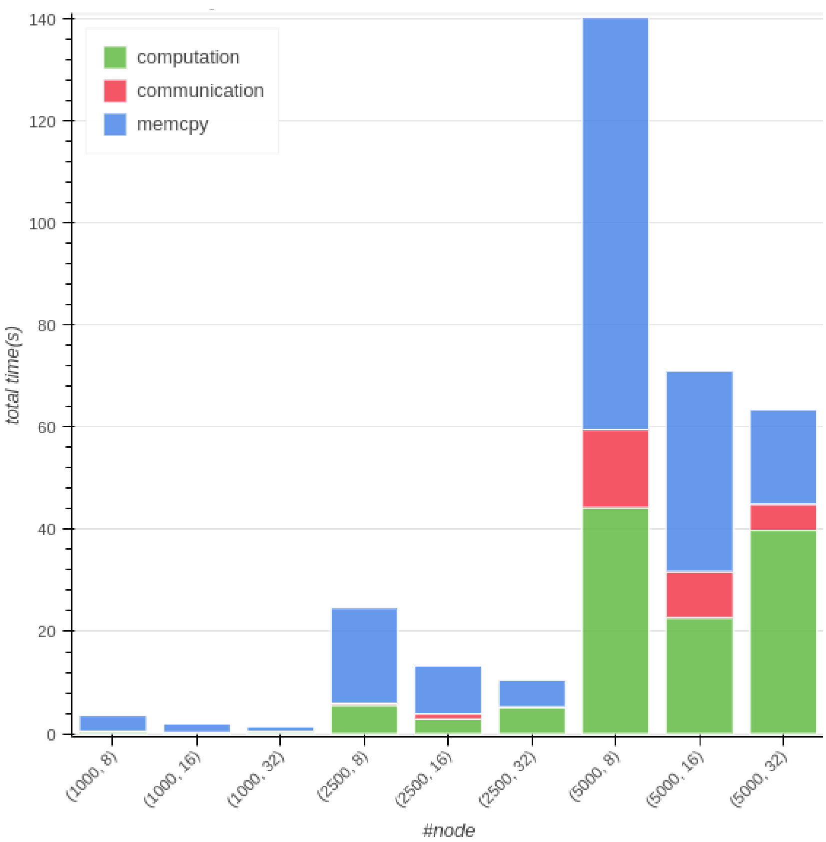
Weak Scalability

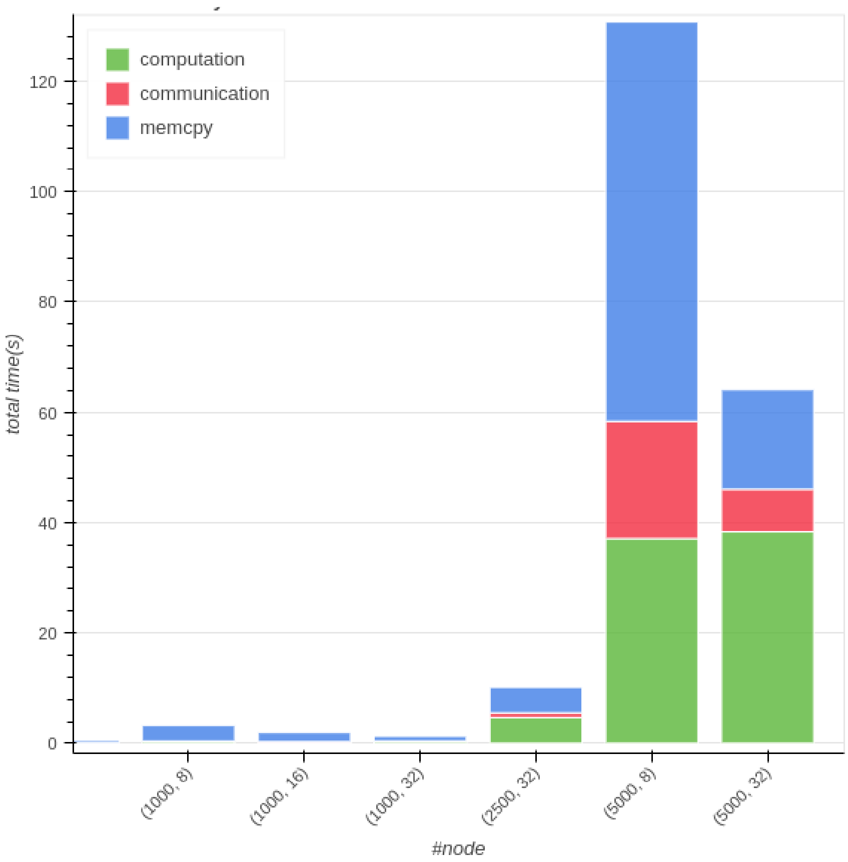
single-GPU版本，時間幾乎全花在計算上，而在不同problem size=node數下，可以看到使用大小為16的block 表現都最好。

（I/O的部份，音讀寫檔案的時間是固定的，故不列入比較）



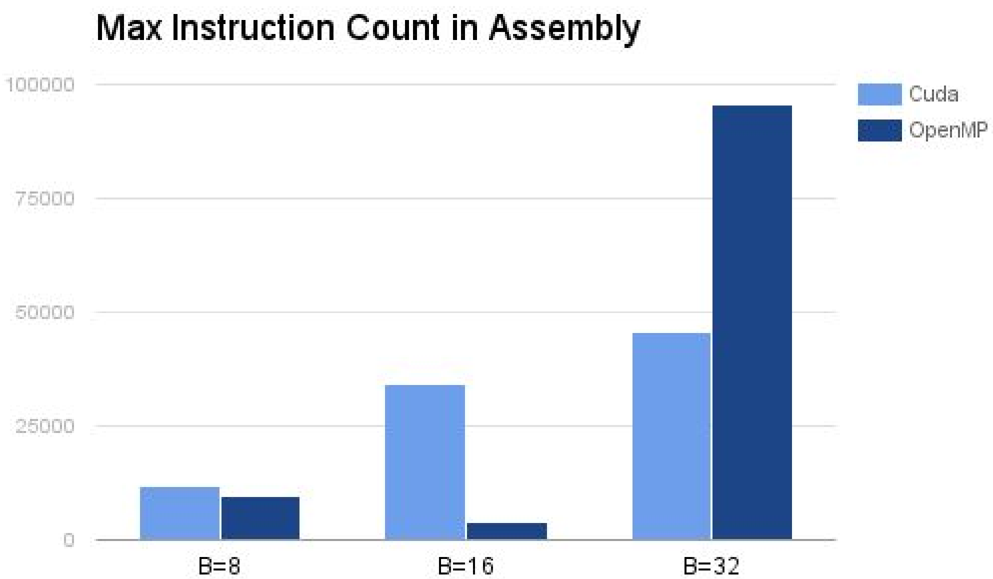
Multi-GPU的OpenMP版本，可以觀察到計算時間，由於使用兩張GPU大致上時間減半，但是花在溝通時間、複製資料的時間變長，整體時間主要由複製資料的部份dominate。而block 大小對於執行時間的影響，仍可看到在大小為16時能計算的最快，但是由於複製資料跟block大小成反比，所以在block最大=32時，整體執行時間最低。



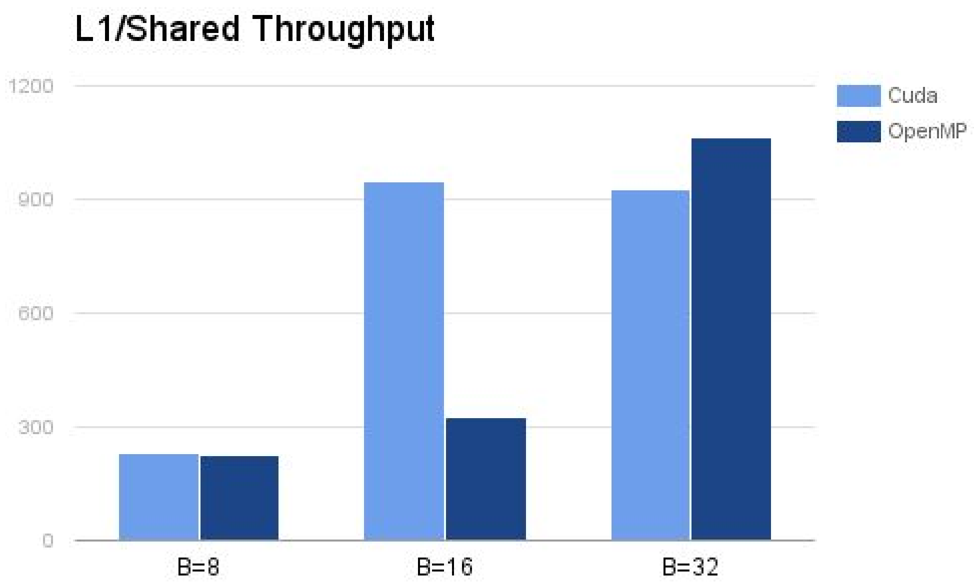
<table>
<tr>
<td>Multi-GPU的MPI版本，由於實做方式和OpenMP版本非常類似，weak scalability趨勢很類似，最好的block 大小也是32。</td>
<td>
</td>
</tr>
</table>

Blocking Factor

GFLOPS

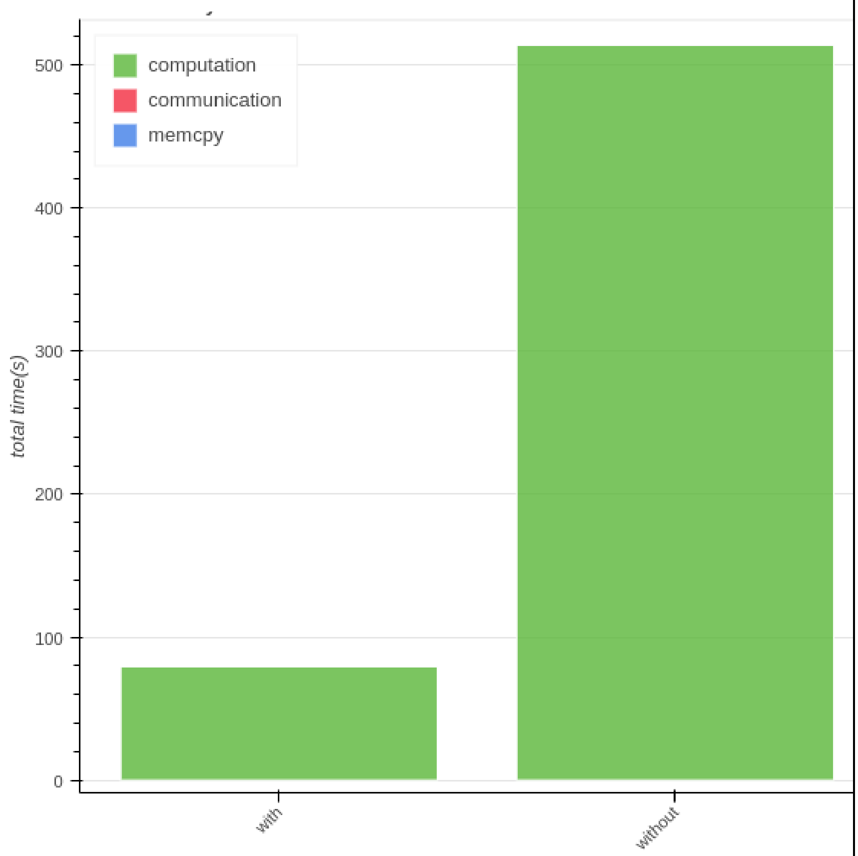


Device/Shared Memory Bandwidth

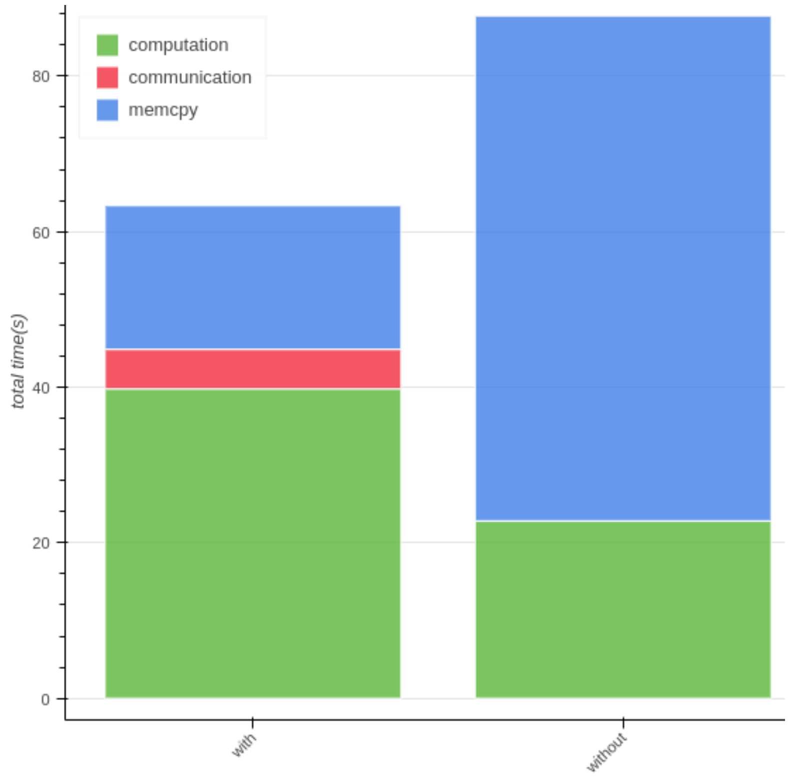


Optimization Techniques

以下實驗皆為problem size=5000個node， block大小為32時的測量結果

shared memory 在未使用shared-memory， 每次計算最短路徑都從global memory存取時所需的時間大約是有使用shared-memory版本的5倍。	

streaming
multi-GPU 版本在不同階段間必須做資料交換，右圖為資料交換使用streaming前後的比較圖，沒使用streaming平行存取data需要的複製資料時間是有用多個streaming的3倍。



Conclusion

對於計算All-Pair Shortest Path問題，使用單一GPU可以省去資料同步的時間，但每個階段間卻無法完全利用data-parallelism的特性，使用多張GPU可以盡量將資料平行處理，但是卻花了更多時間在資料同步上。我想多張GPU更適合不需要這麼頻繁的做資料同步，或者能容許延遲同步的問題上（比如neural network的training，晚幾個回合做更新，可能對最終performance沒有影響）。