HW4-2 Blocked All-Pairs Shortest Path (Multi-cards)

106062322 江岷錡

Implementation

本次實作 `Floyd Warshall Algorithm` 的 Multi-cards cuda 版本，由於 cuda 3 個 Phases 執行細節與 `hw4-1` 完全相同，故以下著重描述 Multi-Cards 的 Optimization : `Data Split` & `Phase 3 Parallelization`。

Data split

首先，由於本次 spec 為 Single Node 上 Multiple GPU，我使用 OpenMP 開設兩個 thread ，並各自 assign 一個 GPU 供其操作。

```
#pragma omp parallel num_threads(2)
{
    const int cpuThreadId = omp_get_thread_num();
    cudaSetDevice(cpuThreadId);
    ...
}
```

接續，我個別在各自 Global Memory ，開設一個完整 `n*n` matrix ，並分散 load 不同部分的資料。

- 平分 `number of blocks (n/BLOCK_SIZE)` 給兩個 GPU 。
- 對於 GPU1 而言，他會得到上半部的 Matrix Data。
- 對於 GPU2 而言，他會得到下半部的 Matrix Data 。
- 若 `number of blocks` 無法整除 `2` ，則讓 `GPU2` 多補 `1` 個 Block。

透過此方式，可讓 Load Data 的部分平行化，降低 CUDA Memcpy 的時間。

Phase 3 Parallization

接續，便進入 `r round` 的 for-loop，這部分改寫的演算法為：

1. 每個 Round 開始前，負責該 `round row` 的 GPU ，先將資料放回 Host Memory ，供另一個 GPU 後續讀入。
2. 每個 Thread (GPU) ，再次將資料從 Host Memory 讀回 GPU Global Memory 。
3. 各自進行 3 個 Phase。特別的是，對於 GPU 1 而言，Phase 3 他只會計算上半部；對於 GPU 2 而言， Phase 3 他只會計算下半部。
4. 重複 r round 後即完成計算。

```
for(int r = 0; r < round; r++) {
  const size_t roundBlockOffset = r * BLOCK_SIZE * n;
  // Every thread has its own yOffset
  const int isInSelfRange = (r >= yOffset) && (r < (yOffset + roundPerThread));
  if (isInSelfRange) {
      cudaMemcpy(Dist + roundBlockOffset, dst[cpuThreadId] + roundBlockOffset, ro
wBlockSize, cudaMemcpyDeviceToHost);
  }
  #pragma omp barrier

  cudaMemcpy(dst[cpuThreadId] + r * BLOCK_SIZE * n, Dist + r * BLOCK_SIZE * n, ro
wBlockSize, cudaMemcpyHostToDevice);

  /* Phase 1*/
  Phase1 <<<1, block_dim>>>(dst[cpuThreadId], r, n);

  /* Phase 2*/
  Phase2 <<<blocks, block_dim>>>(dst[cpuThreadId], r, n);

  /* Phase 3*/
  Phase3 <<<grid_dim, block_dim>>>(dst[cpuThreadId], r, n, yOffset);
}

...

__global__ void Phase3(int *dist, int Round, int n, int yOffset) {
  const int j = blockIdx.x;
  const int i = blockIdx.y + yOffset;
  ...
}
```

Experiment & Analysis

System Spec

使用學校提供的 hades Cluster。

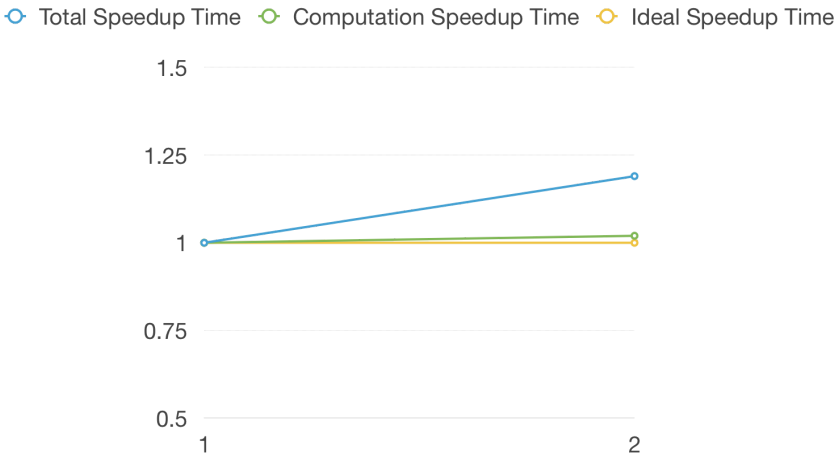
Weak Scalability

相較 Strong Scalability，Weak Scalability 需配置 **等量** 的資源，到不同的運算單位 (GPU \ Thread) 上。觀察目的為，確認資料是否平等的分配在不同資源上，最佳化整體計算效率。

同時，計算量是以 `V^3` 作為計算。因此，在 GPU 只有 1 Core 的配置上，我執行一個自行設計的 `22208 n` 測資。在 GPU 有 2 Core 的配置上，我執行自行設計的 `28032 n` 測資。兩個測資約差 2 倍計算量 (28032^3 / 22208^3 = 2.01)。

Weak Scalibility

	1	2
Total Speedup Time	1	1.19
Computation Speedup Time	1	1.02
Ideal Speedup Time	1	1



由數據中可見，我們的表現極度貼近理想值。Computation Speedup Time 與 Ideal Time 的偏差，我個人認為是因為我僅平行 Phase 3 ，Phase 1 及 Phase 2 皆無進行優化。但由數據中仍可見， 沒有優化 Phase 1 及 Phase 2 ，也不會太過影響整體時間，故優化的 CP 值不高。而 Computation Speedup Time 與 Total Speedup Time 的偏差，我個人認為原因在於 IO (CPU 讀資料、Memcpy ...)的時間，這類為定性固化、無法加速的工作，導致速度顯得更慢。

Time Distribution

為了分別計算不同 input size 下，cuda 執行的 performance，我分別執行 `p31k1`, `p32k1`, `p34k1`, `p35k1`, `p36k1` 的測資。

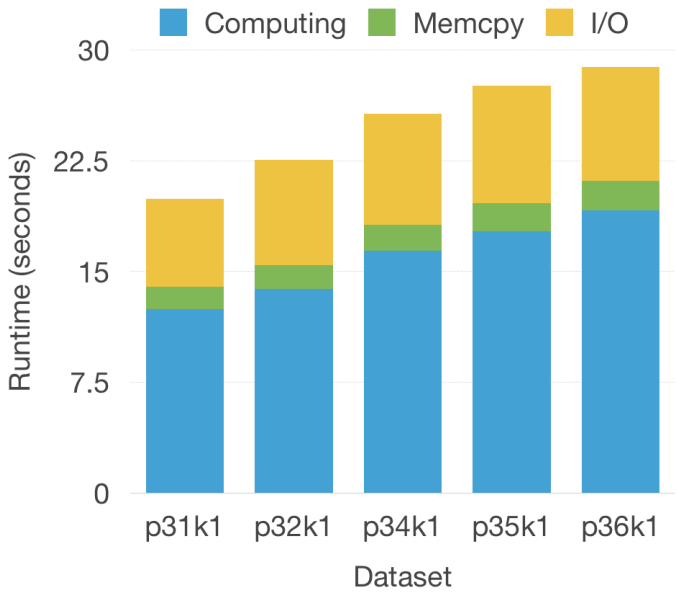
`computing` 與 `memory copy (H2D, D2H)` 的時間計算，我使用 `nvprof` 負責搜集資訊（若 nvprof 採樣的結果為兩個 GPU Device 的加總，在下方實驗中已重新針對單一 Device 計算時間）。

而對於 CPU 的 IO ，我使用 `clock_gettime(CLOCK_MONOTONIC, &start)` 進行時間採樣。

*註 `Expanded N` = 計算時 Matrix 的 `n`

Time Distribution

	p31k1	p32k1	p34k1	p35k1	p36k1
Expanded N	31040	32064	34048	34944	36032
Computing	12.47	13.842	16.405	17.722	19.133
Memcpy	1.48	1.58	1.78	1.88	2
I/O	5.95	7.127	7.468	7.968	7.71

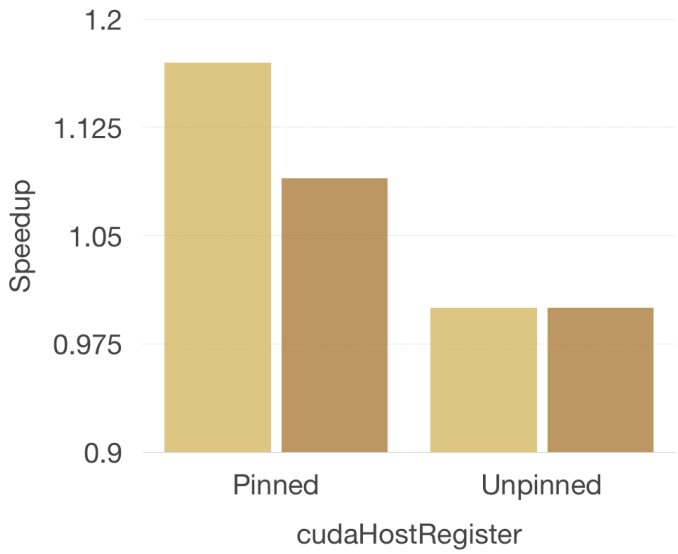


從數據中可見，當資料量逐步升級時，I/O Time 與 Computation Time 的成長速度皆較高，兩者佔據的時間也較多。然則，IO Time 是全部實作中唯一無法避免的，因此如何提升 CUDA Computation Time ，也是本次實作的其一目標。而 CUDA 的 Computation Time ，也通常與其 access data 的效率有絕對正比關係，因此如何提升 memory access ，也是可關注的焦點。

Other Experiements

Pin Memory 重要性

Pin Memory 可觸發 Direct Memory Access(DMA) ，使 Host ←→ Device Memory Access 效率更好。本次實作過程中，發現是否有 Pinned 與 Unpinned Host Memory ，相對於 HW4-1 ，對最終結果有較大影響。透過實驗，得出以下數據：



（深褐色為 Single GPU 情況下，淺褐色為 Multi GPU 情況下）

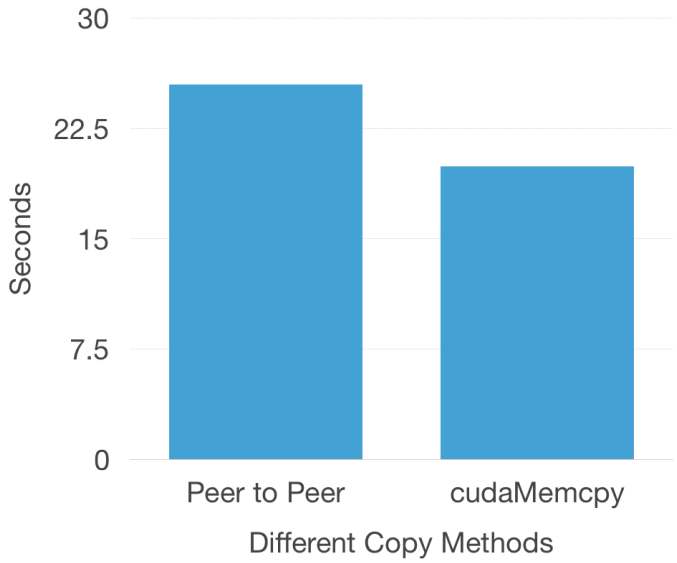
由圖中可見，當 Pin 住 Host Memory Data 時，不論 Single GPU 或 Multiple GPU 都能得出極好效果，但 Multiple GPU 情況下加快速度較高。個人推測當 Memory 被重複讀取時，可能先行 load 至較快的 cache 上，致使兩個 GPU 在讀取時效率較高。反觀單一 GPU 時提升效率有限。

CUDA Peer to Peer vs CUDA Memcpy

由於實作上，當每個 Round 結束時，都需進行跨 GPU Device 的資料 copy 與 syncing 。理論上，直接夠過 PeerToPeer 傳遞資料，理應會有較好的效率，但實作上卻得出相反結論，以下為實驗數據：

Performance

	Peer to Peer	cudaMemcpy
Total_time	25.472	19.90



針對 `p31k1` 測資的測試可發現，當我將每個 Round 的資料 Syncing ，由 `cudaMemcpy` 轉為 `PeerToPeer` 時，反而執行時間會提高。個人推測原因在於，本次跨 GPU 間並沒有 `NVLink` 的協助，導致 Copy 上仍需經過傳統 `PCI-e`，讓執行時間提高。而使用原生的 CudaMemcpy ，實作上可降低一次 Copy 時間， Copy 過程中或許也會 cache 在較快快取上，導致效率較好。

Experience & Conclusion

本次實作體驗到 Share Device 的操作，如何搭配 OpenMP 進行有效分散。事實上看了相當多的文章，嘗試過了 PeerToPeer \ UVA \ Direct Access Other GPU Global Memory \ Zero Copy 等作法，但都沒有獲得等量效益，反而傳統最直接的寫法，就能得到極好的 Performance 。

然後辛苦助教了，上次 12 點多 profiling 機器壞掉，都還是努力把機器修好，辛苦了!