Final project report

姓名: 王領崧 107062107

Topic

Parallized Maze Routing

Background & Motivation

1. What is maze routing ?

   在 Eda tool 的 global routing 和 detailed routing 中，maze routing 是很常被使用到的 routing 手法。簡單來說，maze routing 是要在 grid graph 中，找到連接所有 pins 的最小 cost 的 path。

   Maze routing 的實作十分容易，基本上尋找 shortest path 的 algorithm，像是 bfs, A* 都可以用，但其缺點就是非常 time-consuming。

2. Motivation

   基於 maze routing 非常 time-consuming，我希望能透過課堂所學的 OpenMP, CUDA 來加速 maze routing。剛好有一篇 paper **“GAMER: GPU Accelerated Maze Routing”** 就是利用 GPU 來加速 maze routing， 因此我的目標會先嘗試做出 paper 講述的方法，接著再看看有沒有可以改進的地方。

Problem Description

給定一個 weighted grid graph 和 k pins，找出能連接 k pins 的 minimum cost path。

- Input
  - H, W: Grid graph 的長 & 寬
  - Pins: K 個 pin 的 (x, y) 座標
  - Weights: vertical / horizontal
- Output

- Path: 列出所有在 shortest path 上面的 grid 的 (x, y) 座標

- Example

```
3 3
2
Pin 1 1
Pin 0 2
Vertical 24 26
Vertical 47 13
Vertical 70 2
Horizontal 33 68 51
Horizontal 80 24 77
```

Grid graph 大小: 3 * 3

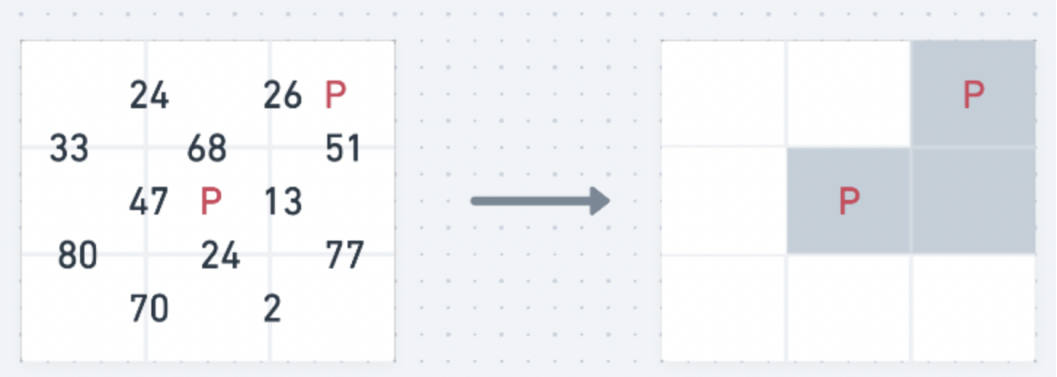
Pins: 總共有 2 個 pins。一個在 (1, 1)，一個在 (0, 2)

Weights: 可參考下圖

```
0 2
1 1
1 2
```

Output: shortest path 由 (0, 2) (1, 1) (1, 2) 的 grid 組成

- 圖像化



Implementation

1. Algorithm

a. Choose one pin as source and set its cost to 0

b. Relaxing the cost of other grids using sweep operations

c. Find out the unrouted pin with minimum cost

d. Retrace its path and set the grid cost belongs to the path to 0

e. Iteratively performing step 2-4 until all the pins are routed

2. Sweep Concept

- sweep 的核心概念是利用 dynamic programming 來更新 cost
- 一次的 sweep operation 是由 horizontal + vertical 來完成。而 horizontal 裡面還有從左到右 & 從右到左兩種方向，vertical 也是相同道理。
- 底下為進行 horizontal sweep 的圖例

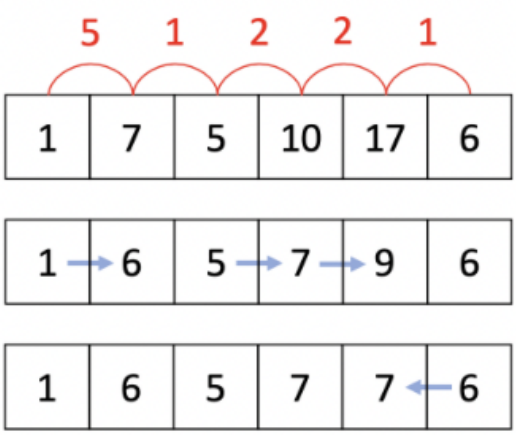


Fig. 3: Sweep

- Sweep operation

  每次 relax cost (Algorithm (b) 的部分) 會進行多次的 sweep operation (horizontal + vertical)，直到整個 grid graph 的 cost converge 為止。

3. Parallel sweep operation

此次有實作兩個版本，一個是 divide & conquer，另一個為 reformulation。

1. Divide & Conquer

   Sweep operation 每個 row, column 原本就都是 independent 的，因此可以平行化處理。

在 Row, column 內，我們可以透過 divide & conquer 來平行化。
以 horizontal sweep 為例，sweep 會被拆解成 $log_2 N$ 個回合，第一回合會是 2 個 2 個 element 為一個 group，右邊的 element 會去透過比較左邊 element 的 cost + weight 來更新自己的 cost。第二輪開始，group 的 element 會 x2 變為 4 個，一樣是右半邊的 group elements 會透過左半邊的 element 來更新 cost，不過因為左半邊 element 的 cost 都已更新完成，因此我們只需要拿左半邊最右邊的 element 來比較更新即可。重複執行到 $log_2 N$ 回合就完成 horizontal sweep。

CUDA 實作: 每個 block 負責一個 row, column 的運算，block 的 threads 負責 row, column 內部的 parallal sweep。在開始 sweep 前，會先將此 row, column 的 grid cost & weight 都 load 進 shared memory 中，來增加後續 memory access 的速度。並且如果 row element 的數量超過 thread 的數量，就會以 stride loop 的方式讓每個 thread 負責多個 elements。

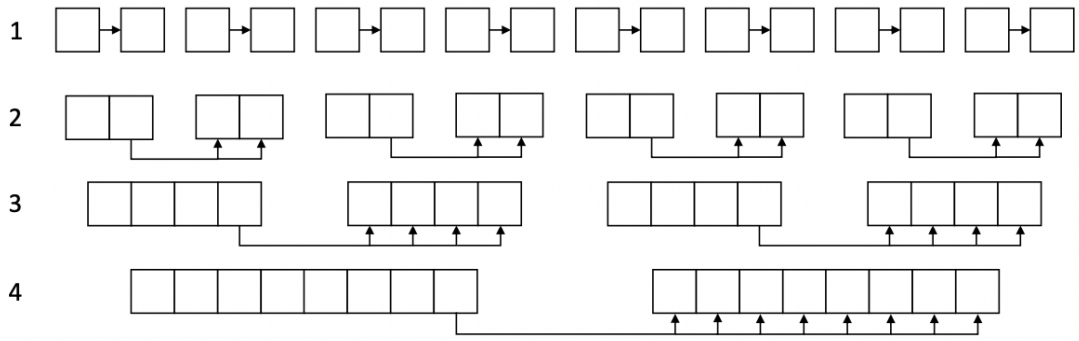


Fig. 5: Parallelizable Sweep

2. Reformulation

Sweep operation 如果表示成數學式，就會像是 (1) 所示。以 idx `i` 的 cost 為例，取決於從 idx `0 ~ i-1` 中，cost + weight 最小的那個。不過這邊可以注意的是，summation 的地方會被重複計算很多次，因此，如果我們可以先把 summation 計算起來 (`s(i)`)，那麼就可以將 sweep operation 表示成 (2)，也就會變成兩個 prefix 的問題 (prefix sum + prefix min)。

$$d(i) = \min_{0 \le j \le i} \left(d(j) + \sum_{k=j+1}^{i} c(k) \right) \quad (1)$$

$$d(i) - s(i) = \min_{0 \le j \le i} \left(d(j) - s(j) \right) \quad (2)$$

$$s(i) = \sum_{j=0}^{i} c(j)$$

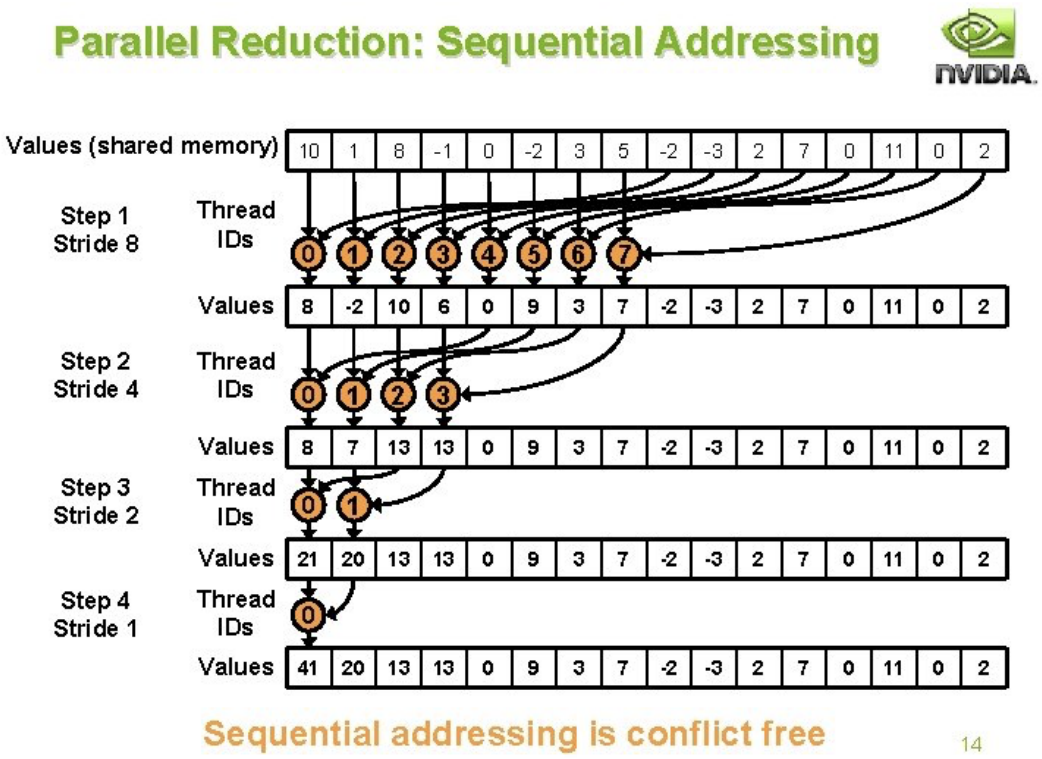
CUDA 實作: 同樣是讓每個 block 負責一個 row, column 的運算，block 內的 threads 會負責執行 reformulation。首先的 prefix sum，我是利用 nvidia 網站提供的 exclusive prefix 去平行化計算 prefix sum，而 prefix min 的部分，因為時間的關係，來不及平行化，因此這個部分並未完成 QQ

4. Parallel get minimum cost

   這邊是講解 algorithm (d) 的平行化方法。

   平行化方法源自於 parallel reduction 的 idea，兩兩 pins 互相比較，比較小的記錄下來，直到比完 $log_2 N$ round，最後 array[0] 即會儲存 minimum cost pin。

   CUDA 實作: 因為 pin 的數量都不會到太多，因此使用一個 block 的 shared memory 就可以完成了 (如果 pin 數量 > thread 數量，就會用 stride loop 來讓每個 thread 負責 `threadID + k * stride` idx 的工作)



Evaluation

OpenMP: run on Apollo (6 CPUs)

CUDA: run on Hades (1 GPU)

Speedup

- OpenMP: 3x
- CUDA: 36x

Testcase	W	H	Pins	bfs	sweep	omp	cuda	omp speedup	cuda speedup
1	256	256	4	0.0945	0.3579	0.1331	0.0289	2.69	12.38
2	256	256	8	0.2104	0.6984	0.2658	0.057	2.63	12.25
3	256	256	16	0.3708	1.4031	0.5179	0.1176	2.71	11.93
4	512	512	4	0.5953	6.0169	1.5856	0.2339	3.79	25.72
5	512	512	8	1.3021	11.2363	2.9808	0.3885	3.77	28.92
6	512	512	16	2.6955	24.3993	6.4435	0.8476	3.79	28.79
7	1024	1024	4	6.629	70.5602	20.0804	1.9302	3.51	**36.56**
8	1024	1024	8	11.0103	135.697	37.5864	3.7695	3.61	**36**
9	1024	1024	16	22.2053	292.951	79.7602	8.0401	3.67	**36.44**

底下為 pin 數量比較多的 experiments (grid 面積 : pin 數量大約是 1000 : 1)

Speedup

- OpenMP: 3x
- CUDA: 40x

Testcase	W	H	Pins	bfs	sweep	omp	cuda	omp speedup	cuda speedup
1	64	64	4	0.003	0.0075	0.0049	0.0024	1.53	3.13
2	100	100	10	0.023	0.0538	0.0236	0.0116	2.28	4.64
3	200	200	40	0.4144	1.1289	0.4281	0.1875	2.64	6.02
4	256	256	65	1.0675	4.2704	1.5903	0.3318	2.69	12.87
5	350	350	120	3.8009	24.6705	8.2991	1.4413	2.97	17.12
6	500	500	250	20.949	264.568	65.1752	6.5087	4.06	40.65
7	512	512	260	23.8469	271.663	72.8673	6.2595	3.73	43.4
8	750	750	560	117.231	> 10min	533.454	29.2559	x	x
9	1024	1024	700	442.846	> 10min	> 10min	109.943	x	x
10	1024	1024	450	278.178	> 10min	> 10min	84.3711	x	x

可以看到計算量越大，CUDA speedup 的效果會越好，而 OpenMP 加速就大概 3 倍，我覺得是因為 OpenMP 的平行只有在外圍的 for loop，也就是平行化 row, column 而已，並且 OpenMP 只有 6 個 CPU thread，因此加速效果有限；反觀 CUDA 的 block 數量很多，隨著 grid graph W, H 變大平行度也會變大，因此在上面的實驗中 speedup 有繼續提升 (但如果變得更大應該就不會提升了)。

Discussion

- Why bfs is faster than sweep ?

Testcase	W	H	Pins	bfs	sweep	omp
1	256	256	4	0.0945	0.3579	0.1331
2	256	256	8	0.2104	0.6984	0.2658
3	256	256	16	0.3708	1.4031	0.5179
4	512	512	4	0.5953	6.0169	1.5856
5	512	512	8	1.3021	11.2363	2.9808
6	512	512	16	2.6955	24.3993	6.4435
7	1024	1024	4	6.629	70.5602	20.0804
8	1024	1024	8	11.0103	135.697	37.5864
9	1024	1024	16	22.2053	292.951	79.7602

跑完實驗結果發現，sweep 比 bfs 還要慢很多，甚至連 openMP 的版本都比較慢。一開始以為只是因為 sweep 要跑很多 iteration 導致的，不過在 final presentation 時助教的提點，才發現 bfs 的作法是有問題的，因為我這種從 pin 出發的 bfs，就已經

將 shortest path 限制在經過最少的 grid 的條件，這樣不但時間複雜度與 sweep 不相同 (bfs: `O(W*H)` , sweep: `O(k*W*H)`)，計算出來的答案也不全然是正確的。

- Why needs so many sweep iterations to converge the grid graph ?

  因為 project 實作的為 simple 2D routing，所以當 grid graph size 很大時，就會需要經過很多次的 iteration 才會 converge grid graph。不過根據 paper 的描述，在實際 routing 的應用 (3D routing)，不會只有單純的 vertical / horizontal weight，還會有 via cost、其他的 constraint 等等，這會大大降低 iterations 的次數。

Summary

- 完成 sweep sequential version
- 完成 sweep with openMP version
  - speedup: 3x
- 完成 sweep with divide & conquer CUDA version
  - speedup: 36x
- sweep with reformulation 剩下 prefix min 的部分未完成
  - 無法提供 evaluation

Thanks

感謝老師和助教這學期的指導！