Parallelized maze routing

王領崧

Outline

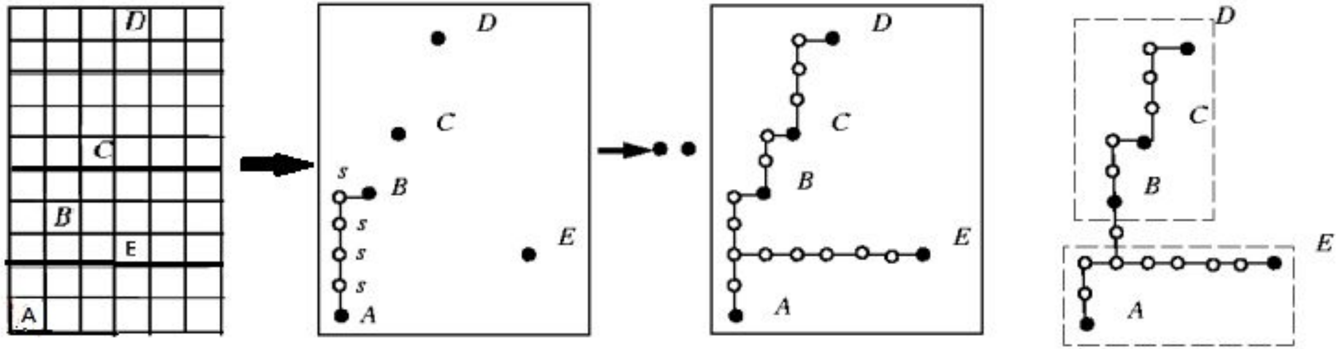
- Background
- Problem Description
- Implementation
- Evaluation

Outline

- Background
- Problem Description
- Implementation
- Evaluation

What is maze routing ?

- Commonly used in global routing or detailed routing in EDA tool
- Briefly speaking, find the minimum cost bewteen pins in grid graph
- Easy to implement but time-consuming



This project

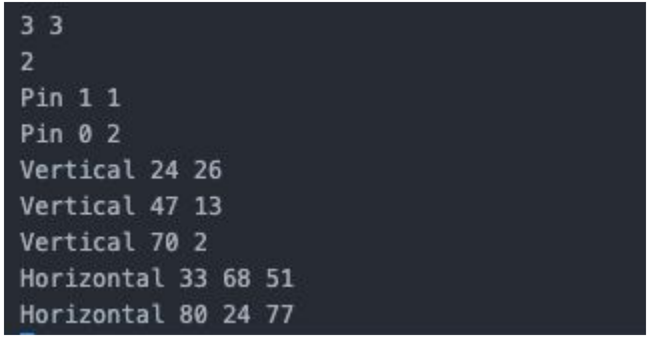
- Using CUDA and openMP to accelerate maze routing.
- Implement the idea from the paper “GAMER: GPU Accelerated Maze Routing”

Outline

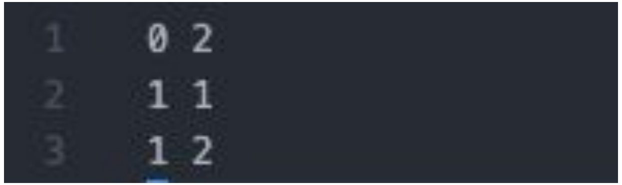
- Background
- Problem Description
- Implementation
- Evaluation

Problem Description

- On a given weighted graph, given k pins, find a shortest path connecting k pins.
- Input
  - H: grid graph height
  - W: grid graph width
  - Pins: pin’s (x, y) coordinate
  - Weights: vertical / horizontal
- Output : A k-pin shortest path
  - Path: all (x, y) coordinate

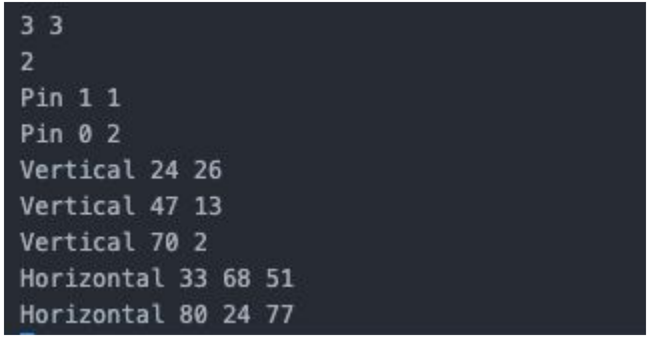


```
3 3
2
Pin 1 1
Pin 0 2
Vertical 24 26
Vertical 47 13
Vertical 70 2
Horizontal 33 68 51
Horizontal 80 24 77
```

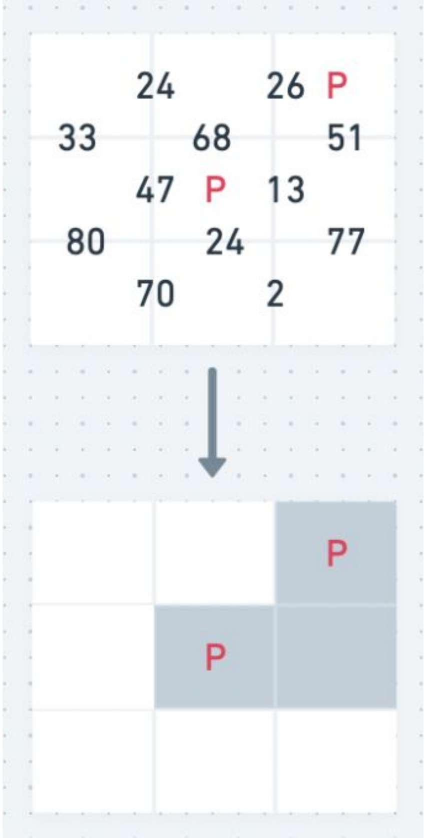


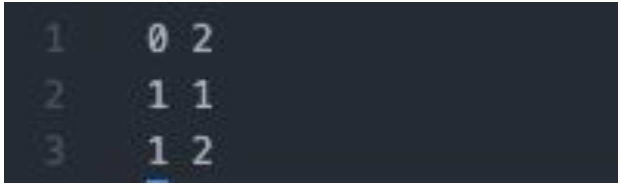
```
0 2
1 1
1 2
```

Problem Description



```
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
```





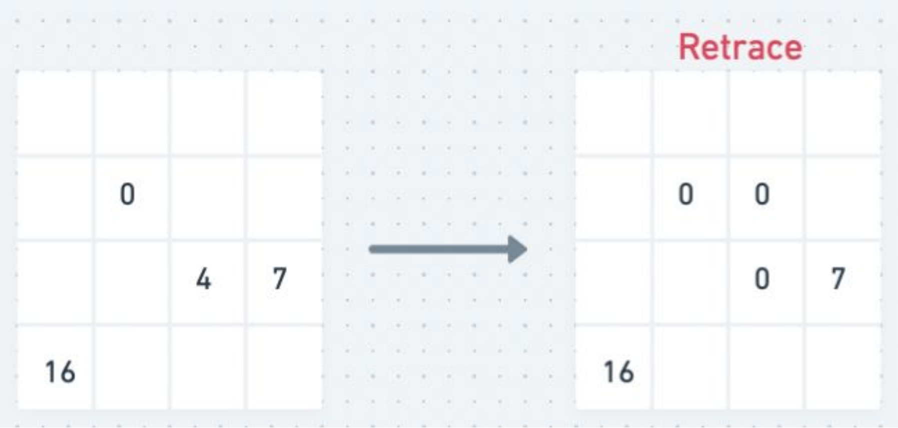
```
0 2
1 1
1 2
```

Outline

- Background
- Problem Description
- Implementation
- Evaluation

Algorithm

1. Choose one pin as source and set its cost to 0
2. Relaxing the cost of other grids using sweep operations
3. Find out the unrouted pin with minimum cost
4. Retrace its path and set the cost to 0
5. Iteratively performing step 2-4 until all the pins are routed

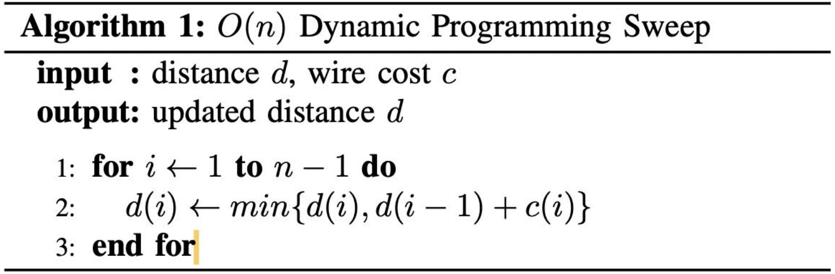


	0		
		4	7
16			

	0	0	
		0	7
16			

Sweep

- Dynamic programming
- Horizontal / Vertical with two directions



Algorithm 1: $O(n)$ Dynamic Programming Sweep

input : distance d, wire cost c

output: updated distance d

```
for $i \leftarrow 1$ to $n - 1$ do
    $d(i) \leftarrow min\{d(i), d(i-1) + c(i)\}$
end for
```

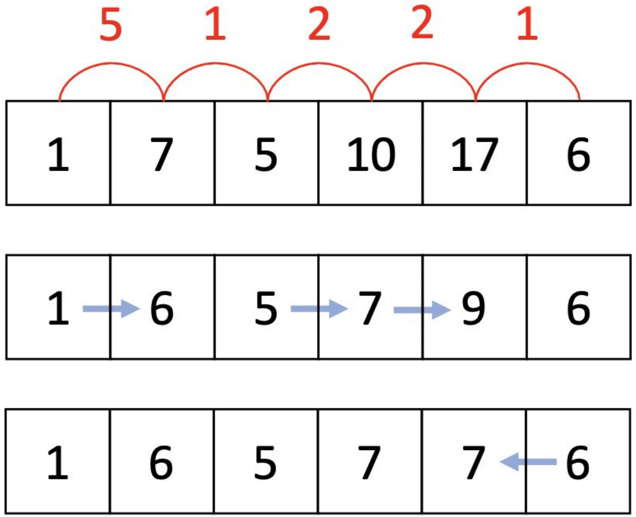
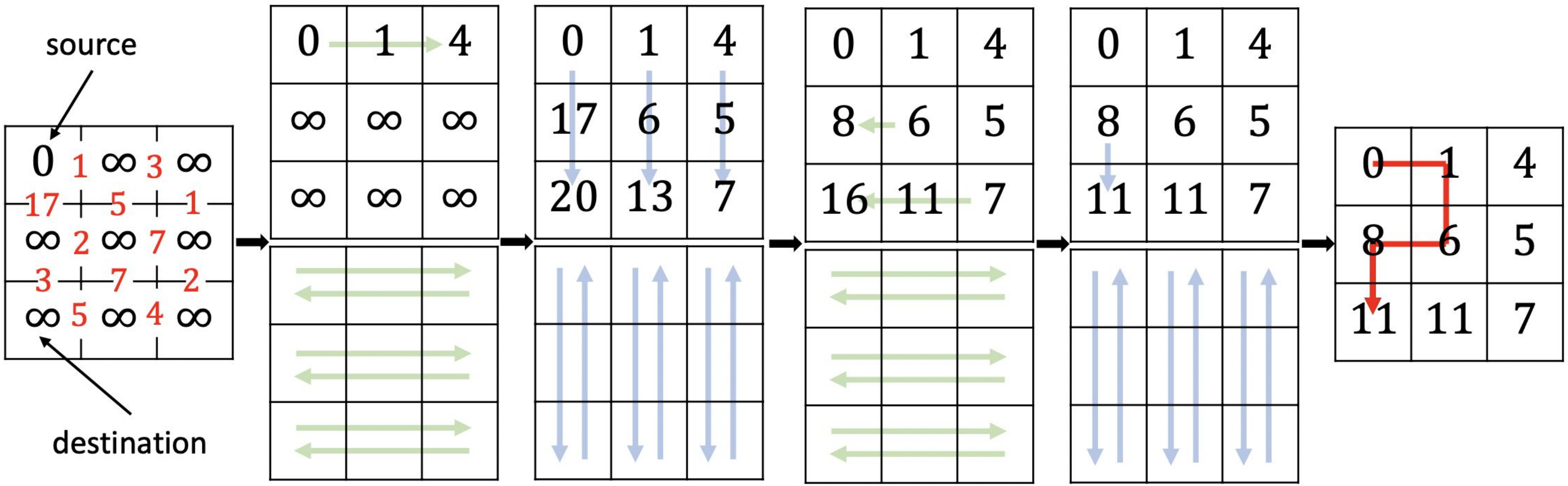


Fig. 3: Sweep

Sweep

- Iteratively performing vertical / horizontal sweeps with direction change.
- Until the grid cost are all minimal.



0	1	4
∞	∞	∞
∞	∞	∞

0	1	4
17	6	5
20	13	7

0	1	4
8	6	5
16	11	7

0	1	4
8	6	5
11	11	7

0	1	4
8	6	5
11	11	7

Sweep with divide and conquer (completed)

- Different rows and columns can work in parallel
- Parallelize inside the row and column (Groups are independent as well)

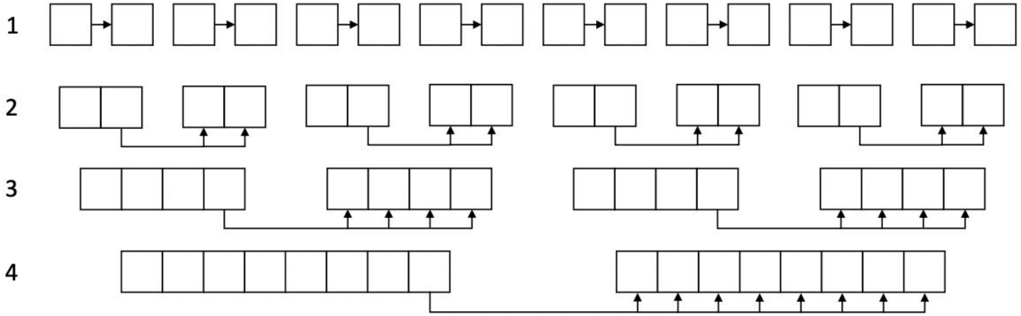


Fig. 5: Parallelizable Sweep

Sweep with reformulation (not completed)

- d(i): cost, c(i): weight
- Turn into two prefix problems

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

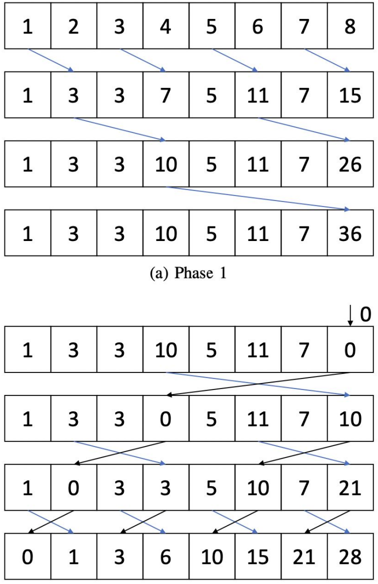
$$d(i) = \min_{0 \leq j \leq i} \left(d(j) + \sum_{k=j+1}^{i} c(k) \right) \qquad (1)$$

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

Sweep with reformulation

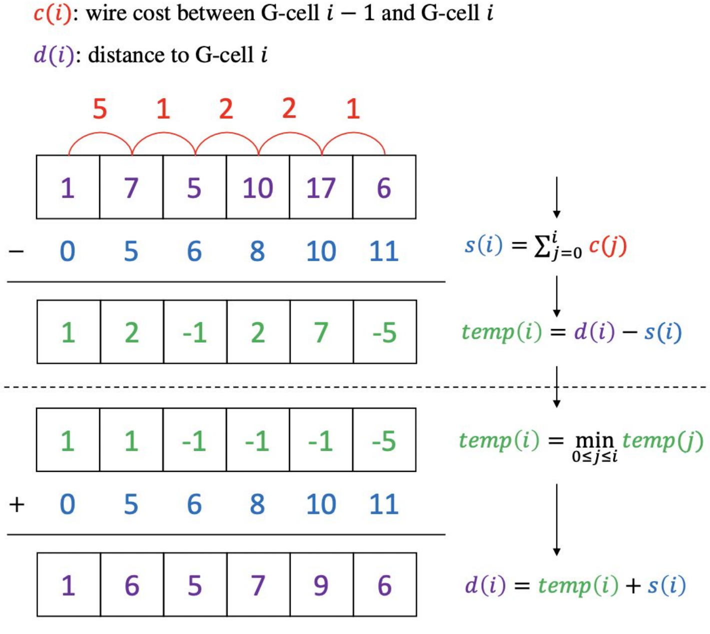
$$d(i) - s(i) = \min_{0 \le j \le i} (d(j) - s(j)) \qquad (2)$$

- d(i): cost, c(i): weight, s(i): prefix sum weight
- Prefix sum can be calculated by parallel exclusive



(a) Phase 1

(b) Phase 2

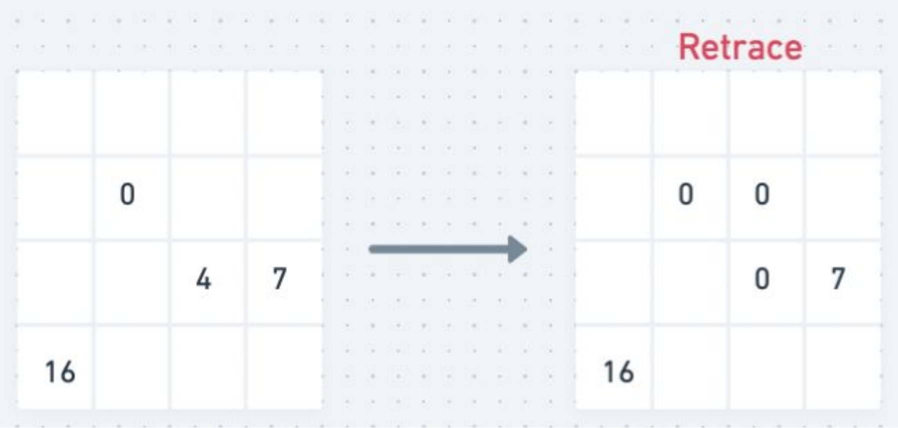

$c(i)$: wire cost between G-cell $i - 1$ and G-cell i

$d(i)$: distance to G-cell i

Fig. 6: Sweep by Prefix Sum and Prefix Min

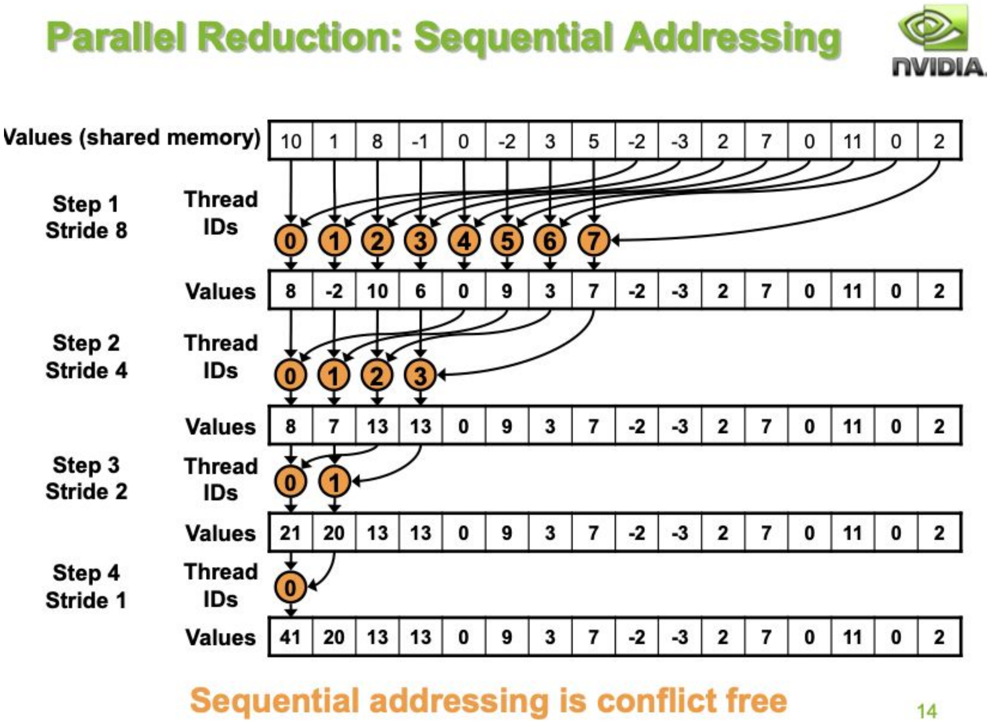
Algorithm

1. Choose one pin as source and set its cost to 0
2. Relaxing the cost of other grids using sweep operations
3. Find out the unrouted pin with minimum cost
4. Retrace its path and set the cost to 0
5. Iteratively performing step 2-4 until all the pins are routed



Get minimum cost pin

- Reduction



Parallel Reduction: Sequential Addressing

Values (shared memory)		10	1	8	-1	0	-2	3	5	-2	-3	2	7	0	11	0	2
Step 1 Stride 8	Thread IDs	0	1	2	3	4	5	6	7								
	Values	8	-2	10	6	0	9	3	7	-2	-3	2	7	0	11	0	2
Step 2 Stride 4	Thread IDs	0	1	2	3												
	Values	8	7	13	13	0	9	3	7	-2	-3	2	7	0	11	0	2
Step 3 Stride 2	Thread IDs	0	1														
	Values	21	20	13	13	0	9	3	7	-2	-3	2	7	0	11	0	2
Step 4 Stride 1	Thread IDs	0															
	Values	41	20	13	13	0	9	3	7	-2	-3	2	7	0	11	0	2

Sequential addressing is conflict free

14

Outline

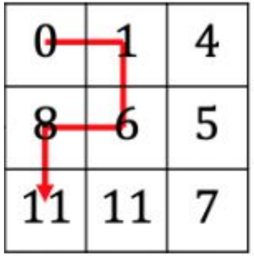
- Background
- Problem Description
- Implementation
- Evaluation

Evaluation

- Omp: Apollo (6 CPUs)
- Cuda: Hades (1 GPU)
- Omp speedup: 3x
- Cuda speedup: 36x

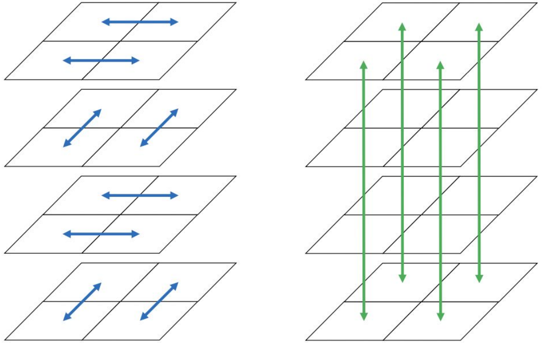
Testcase	W	H	Pins	bfs	sweep	omp	cuda	omp speedup	cuda speedup
1	256	256	4	0.0945	0.3579	0.1331	0.0289	2.69	12.38
2	256	256	8	0.2104	0.6984	0.2658	0.057	2.63	12.25
3	256	256	16	0.3708	1.4031	0.5179	0.1176	2.71	11.93
4	512	512	4	0.5953	6.0169	1.5856	0.2339	3.79	25.72
5	512	512	8	1.3021	11.2363	2.9808	0.3885	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**

Why is sweep much slower than bfs ?



- In simple 2D routing, sweep iterations will be too much
- In practical, via cost (green lines) will reduce the iterations significantly.

Testcase	W	H	Pins	bfs	sweep	omp	cuda
1	256	256	4	0.0945	0.3579	0.1331	0.0289
2	256	256	8	0.2104	0.6984	0.2658	0.057
3	256	256	16	0.3708	1.4031	0.5179	0.1176
4	512	512	4	0.5953	6.0169	1.5856	0.2339
5	512	512	8	1.3021	11.2363	2.9808	0.3885
6	512	512	16	2.6955	24.3993	6.4435	0.8476
7	1024	1024	4	6.629	70.5602	20.0804	1.9302
8	1024	1024	8	11.0103	135.697	37.5864	3.7695
9	1024	1024	16	22.2053	292.951	79.7602	8.0401



Summary

- Complete sweep divide & conquer version
    - speedup: 36x