Title: All-Pairs Shortest Path

Name: 曹寓恆

Student ID: 110521167

Implementation

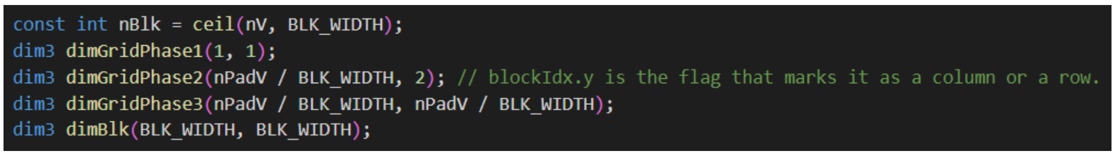
1. Which algorithm do you choose in hw3-1?

I tried running n-times Dijkstra's algorithm and Floyd-Washall algorithm to solve this problem. Although Floyd-Washall algorithm has a higher time complexity of is higher than the other, it still performs better. I think the reason is the simpler the simpler operation, so it became my final solution.

2. How do you divide your data in hw3-2, hw3-3? What's your configuration in hw3-2, hw3-3? And why? (e.g. blocking factor, #blocks, #threads)

(1) #Blocking factor = 32 (because of the maximum number of threads in a block is 1024)

(2) #blocks = ceil(#Vertex / Blocking factor) ^ 2

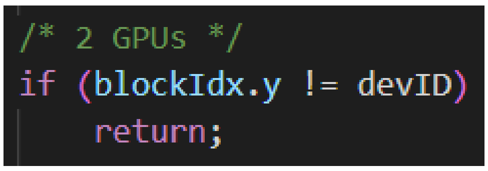


```
const int nBlk = ceil(nV, BLK_WIDTH);
dim3 dimGridPhase1(1, 1);
dim3 dimGridPhase2(nPadV / BLK_WIDTH, 2); // blockIdx.y is the flag that marks it as a column or a row.
dim3 dimGridPhase3(nPadV / BLK_WIDTH, nPadV / BLK_WIDTH);
dim3 dimBlk(BLK_WIDTH, BLK_WIDTH);
```

Configuration in hw3-2

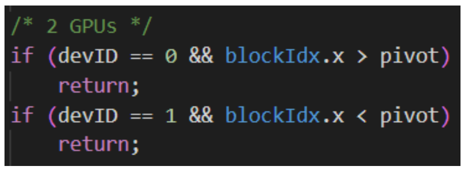
3. How do you implement the communication in hw3-3? Briefly describe your implementations in diagrams, figures or sentences.

After experiment, I figure out that there is no data dependency between blocks in phase 3. This would be a good entry point to parallelize. Then, I can call cudaMallocHost() to pinned host memory as a bridge connecting two GPUs, and call cudaHostGetDevicePointer() to access. In phase 2, one device handles pivot row blocks and the other one handles pivot column blocks. In phase 3, one device handles the left half and the other one handles right half. The conditions are shown below.



```
/* 2 GPUs */
if (blockIdx.y != devID)
    return;
```

Phase 2 condition



```
/* 2 GPUs */
if (devID == 0 && blockIdx.x > pivot)
    return;
if (devID == 1 && blockIdx.x < pivot)
    return;
```

Phase 3 conditions

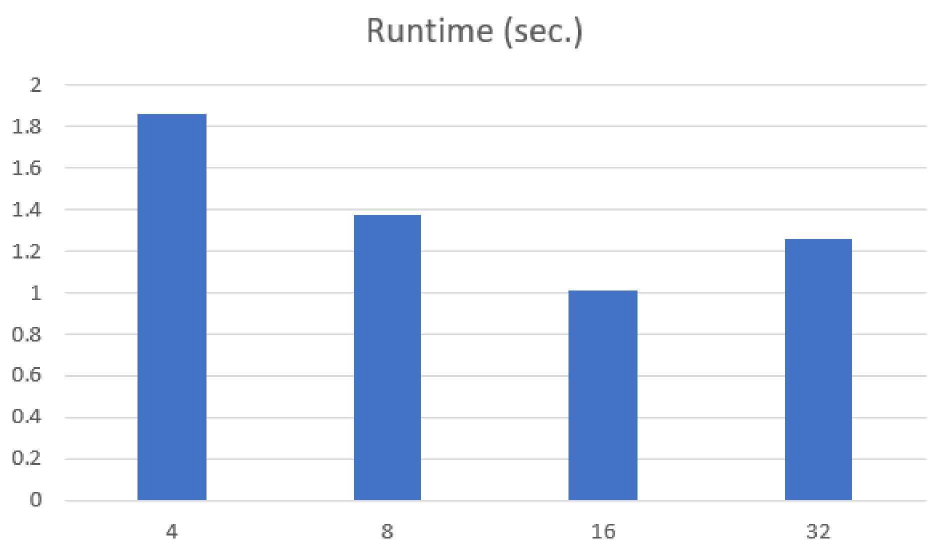
Profiling Results (hw3-2)

Experiment & Analysis

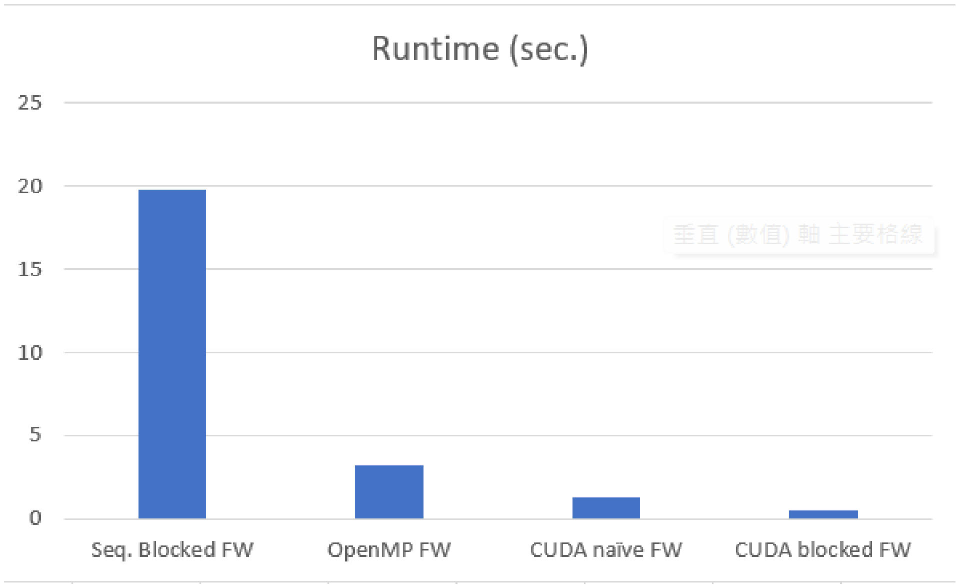
1. System spec

(1) 19 nodes for this course (apollo31 - 48, 50)
(2) Intel X5670 2x6 cores @ 2.93GHz (Hyper threading disabled)
(3) 96GB RAM (each node)
(4) 5.5TB shared RAID5 disk
(5) QDR Infiniband (40 Gb/s)

2. Blocking Factor (hw3-2) (experiment with c18.1)



Several values of blocking factor and runtime(c18.1)
the larger, almost the better



Several parallel methods and their running time (c18.1)

3. Optimization (hw3-2)

(1) Using shared memory as a buffer to accelerated in phase 1, 2 and 3. The size of the buffer is 32*32, only a tenth of this system. Perhaps increasing blocking factor will give better results.

(2) Padding elements to fit the width of the block, which avoid some branches in the kernel function.

(3) I tried to launch the kernel with a kernel to reduce the communication between the CPU and GPU, and in the end, the CPU calls __global__ BLOCK_FW() once and that's it. However, making BLOCK_FW() a normal function and setting phases 1, 2 and 3 as kernel functions would have better performance.

4. Discussion

When there are only two devices it is easy to use a branch to divide the task attribution, but when there are more devices more branches are introduced to divide the task, which is not good for the GPU.

Experiences / Conclusion

In this task, I learned about some basic CUDA applications. The topic is a bit difficult, but I can feel that SIMD is very powerful. I hope I will be familiar with this in the future.