# Parallel Programming
# Homework 4: Blocked All-Pairs Shortest Path

王鈺鎔 106062600

## Implementation

### APSP

I firstly expand data to meet the size of blocks by:

```
int expand_dim = n + block_size - (n % block_size);
```

To simplify computation, I store the data from the file in 1 dimension, and transfer to 2 dimension array by:

```
cudaMallocPitch(&device_s, &pitch, width, height);
cudaMemcpy2D(device_s, pitch, Dist, width, width, height,
cudaMemcpyHostToDevice);
```

In each iteration, the process compute the diagonal elements in phase1, compute the elements in k row and k column in phase2, and compute other elements in phase3. The relax process is shown in fig.1.

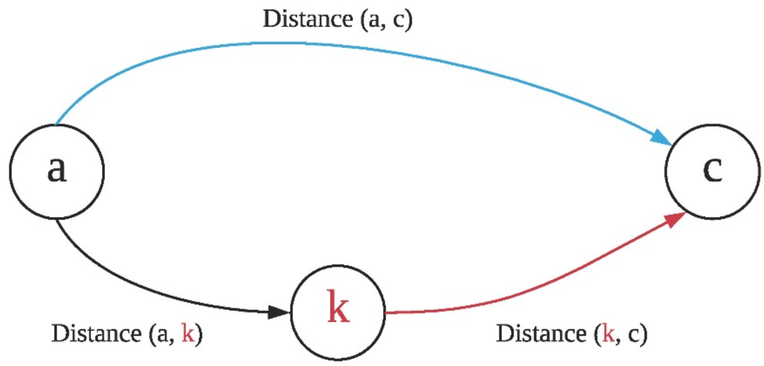


Figure. 1. For example, the red line which is the updated distance from k to c. If the original distance from a to c (blue line) is larger than the distance from a to k (black line) plus the distance from k to c (red line), then update the distance blue line to the black plus the red line.

### Multi-GPU

The methods to read and store the data from files is same to APSP. However, The data is divided in half by height. Thus, each device handles a half of data. In the end of each iteration, each device copy data to the host to combine data, and load to device again to update data in each device.

### Multi-Node

I use MPI and Cuda to implement this algorithm. The methods to divide and handle data is similar. The difference is that the data is separately stored in two nodes, not two GPUs. Moreover, I use MPI send & receive to communicate between two nodes.

(a). *How do you divide your data?*

**APSP**

I firstly expand data to meet the size of blocks by `int expand_dim = n + block_size - (n % block_size);` All the data is stored in one device.

**Multi-GPU**

I firstly expand data to meet the size of blocks by `int expand_dim = n + block_size - (n % block_size);` The process cuts data in half by height. Each device handle half of expanded data, and the remaining is stored in the last device.

**Multi-Node**

I firstly expand data to meet the size of blocks by `int expand_dim = n + block_size - (n % block_size);` The process cuts data in half by height. Each node handle half of expanded data, and the remaining is stored in the last node.

(b). *How do you implement the communication? (in multi-GPU versions)*

The data is firstly copied from host to device. In each iteration, the data in each device will be copied from device to host after computing phase1, 2 and 3. The data from each device then is combined, and the combined is copied to each device again.

(c). *What's your configuration? And why? (e.g. blocking factor, #blocks, #threads)*

- `Block factor = 32`
  - since the block size will be 1G, which can fit the size of general GPUs.
- `#blocks = expand_dim / block factor`
  - Since there is no block containing unused space.
- `#threads = 32`
  - Since only 1 thread is needed to relax edge per element.

# Profiling Results

## APSP

```
==18548== Profiling application: ./apsp testcases/5.in 5_test.out
==18548== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   94.58%  17.3491s       626  27.714ms  27.285ms  27.948ms  ApspPhase3(int, unsigned long, int*)
                    2.94%  538.91ms         2  269.45ms  269.37ms  269.54ms  [CUDA memcpy HtoD]
                    1.69%  310.64ms         1  310.64ms  310.64ms  310.64ms  [CUDA memcpy DtoH]
                    0.66%  121.20ms       626  193.61us  190.75us  196.83us  ApspPhase2(int, unsigned long, int*)
                    0.13%  24.012ms       626  38.357us  37.985us  38.625us  ApspPhase1(int, unsigned long, int*)
      API calls:   54.26%  10.3935s         3  3.46451s  269.45ms  9.85461s  cudaMemcpy2D
                   41.50%  7.94974s      1878  4.2331ms  3.7390us  27.927ms  cudaLaunch
                    4.23%  810.75ms         2  405.38ms  1.0773ms  809.68ms  cudaMallocPitch
                    0.00%  672.61us      5634     119ns      91ns  6.2340us  cudaSetupArgument
                    0.00%  424.21us        94  4.5120us     118ns  192.76us  cuDeviceGetAttribute
                    0.00%  298.66us      1878     159ns     125ns  14.506us  cudaConfigureCall
                    0.00%  124.77us         1  124.77us  124.77us  124.77us  cuDeviceTotalMem
                    0.00%  43.773us         1  43.773us  43.773us  43.773us  cuDeviceGetName
                    0.00%  2.5600us         3     853ns     127ns  2.2360us  cuDeviceGetCount
                    0.00%     675ns         2     337ns     131ns     544ns  cuDeviceGet
```

## Multi-GPU

```
==18849== Profiling application: ./multi_gpu testcases/5.in 5_test.out
==18849== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   95.97%  25.5500s       626  40.815ms  27.597ms  52.837ms  ApspPhase3(int, unsigned long, int*)
                    2.03%  540.34ms         2  270.17ms  270.16ms  270.19ms  [CUDA memcpy HtoD]
                    1.16%  309.93ms         1  309.93ms  309.93ms  309.93ms  [CUDA memcpy DtoH]
                    0.69%  184.24ms       626  294.31us  192.51us  2.5460ms  ApspPhase2(int, unsigned long, int*)
                    0.14%  37.364ms       626  59.686us  38.081us  2.2891ms  ApspPhase1(int, unsigned long, int*)
      API calls:   54.19%  14.5752s         3  4.85841s  270.08ms  14.0349s  cudaMemcpy2D
                   44.80%  12.0507s      1878  6.4168ms  3.8540us  50.825ms  cudaLaunch
                    1.00%  270.01ms         2  135.00ms  1.2343ms  268.77ms  cudaMallocPitch
                    0.00%  843.86us        94  8.9770us     172ns  379.88us  cuDeviceGetAttribute
                    0.00%  639.72us      5634     113ns      87ns  6.4140us  cudaSetupArgument
                    0.00%  332.23us      1878     176ns     135ns  15.899us  cudaConfigureCall
                    0.00%  154.22us         1  154.22us  154.22us  154.22us  cuDeviceTotalMem
                    0.00%  73.348us         1  73.348us  73.348us  73.348us  cuDeviceGetName
                    0.00%  2.8710us         3     957ns     209ns  2.3960us  cuDeviceGetCount
                    0.00%     892ns         2     446ns     182ns     710ns  cuDeviceGet
```

## Multi-Node

### Node1:

```
==11528== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   93.05%  12.7672s       625  20.428ms  20.244ms  23.290ms  FW_phase3(int*, int, int, int, int, int)
                    3.71%  509.09ms       626  813.24us  423.27us  184.80ms  [CUDA memcpy HtoD]
                    2.28%  312.31ms       314  994.62us  390.79us  181.31ms  [CUDA memcpy DtoH]
                    0.94%  128.29ms       625  205.27us  202.43us  235.49us  FW_phase2(int*, int, int, int, int)
                    0.03%  3.6446ms       625  5.8310us  5.6960us  10.272us  FW_phase1(int*, int, int, int, int)
      API calls:   93.39%  7.21356s       940  7.6740ms  325.62us  201.85ms  cudaMemcpy
                    3.41%  263.40ms      1250  210.72us  71.421us  825.26us  cudaDeviceSynchronize
                    2.91%  224.69ms         1  224.69ms  224.69ms  224.69ms  cudaMalloc
                    0.21%  16.391ms      1875  8.7410us  4.3360us  56.915us  cudaLaunch
                    0.02%  1.4336ms     10000     143ns      91ns  14.646us  cudaSetupArgument
                    0.02%  1.3709ms       188  7.2920us     110ns  344.06us  cuDeviceGetAttribute
                    0.01%  949.96us         1  949.96us  949.96us  949.96us  cudaFree
                    0.01%  772.60us         1  772.60us  772.60us  772.60us  cudaGetDeviceProperties
                    0.01%  568.90us      1875     303ns     141ns  14.937us  cudaConfigureCall
                    0.00%  314.17us         2  157.08us  148.19us  165.98us  cuDeviceTotalMem
                    0.00%  247.39us         2  123.69us  106.45us  140.94us  cuDeviceGetName
                    0.00%  10.278us         1  10.278us  10.278us  10.278us  cudaFreeHost
```

### Node2:

```
==11527== Profiling result:
                    0.00%  4.6070us         1  4.6070us  4.6070us  4.6070us  cudaSetDevice
                    0.00%  2.2010us         3     733ns     160ns  1.8480us  cuDeviceGetCount
                    0.00%     802ns         4     200ns     146ns     343ns  cuDeviceGet
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   93.67%  12.7997s       625  20.479ms  20.240ms  23.203ms  FW_phase3(int*, int, int, int, int, int)
                    3.12%  426.68ms       626  681.59us  414.47us  148.68ms  [CUDA memcpy HtoD]
                    2.23%  305.02ms       313  974.50us  390.76us  176.73ms  [CUDA memcpy DtoH]
                    0.95%  129.42ms       625  207.08us  202.98us  237.31us  FW_phase2(int*, int, int, int, int)
                    0.03%  3.6908ms       625  5.9050us  5.6320us  12.097us  FW_phase1(int*, int, int, int, int)
      API calls:   93.31%  7.11932s       939  7.5818ms  294.45us  197.57ms  cudaMemcpy
                    3.27%  249.28ms      1250  199.43us  7.6050us  267.90us  cudaDeviceSynchronize
                    3.08%  234.98ms         1  234.98ms  234.98ms  234.98ms  cudaMalloc
                    0.27%  20.478ms      1875  10.921us  4.0810us  82.073us  cudaLaunch
                    0.02%  1.5661ms       188  8.3300us     115ns  413.37us  cuDeviceGetAttribute
                    0.02%  1.5263ms     10000     152ns      91ns  16.674us  cudaSetupArgument
                    0.01%  979.20us         1  979.20us  979.20us  979.20us  cudaFree
                    0.01%  772.82us         1  772.82us  772.82us  772.82us  cudaGetDeviceProperties
                    0.01%  702.53us      1875     374ns     136ns  14.460us  cudaConfigureCall
                    0.00%  244.18us         2  122.09us  105.04us  139.14us  cuDeviceTotalMem
                    0.00%  89.790us         2  44.895us  32.086us  57.704us  cuDeviceGetName
                    0.00%  8.7830us         1  8.7830us  8.7830us  8.7830us  cudaFreeHost
                    0.00%  6.1670us         1  6.1670us  6.1670us  6.1670us  cudaSetDevice
                    0.00%  2.5530us         3     851ns     157ns  2.2220us  cuDeviceGetCount
                    0.00%  1.0500us         4     262ns     122ns     575ns  cuDeviceGet
```

# Experiment & Analysis

## Methodology

### (a). System Spec

Using the provided cluster to measure.

## Plots

### (a). Weak Scalability & Time Distribution

Data Size: 1. graph with 2500 vertex and 1250000 edges. 2. graph with 5000 vertex and 2500000 edges.

To have suitable data size to do the experiment, I generate edge weight with random number. Fig. 3 shows the execution time and time profile with two different input data and different number of GPU (left is run in 1 GPU and right is run in 2 GPU.) We can observe that the total time and computing time decrease but communication time increases.

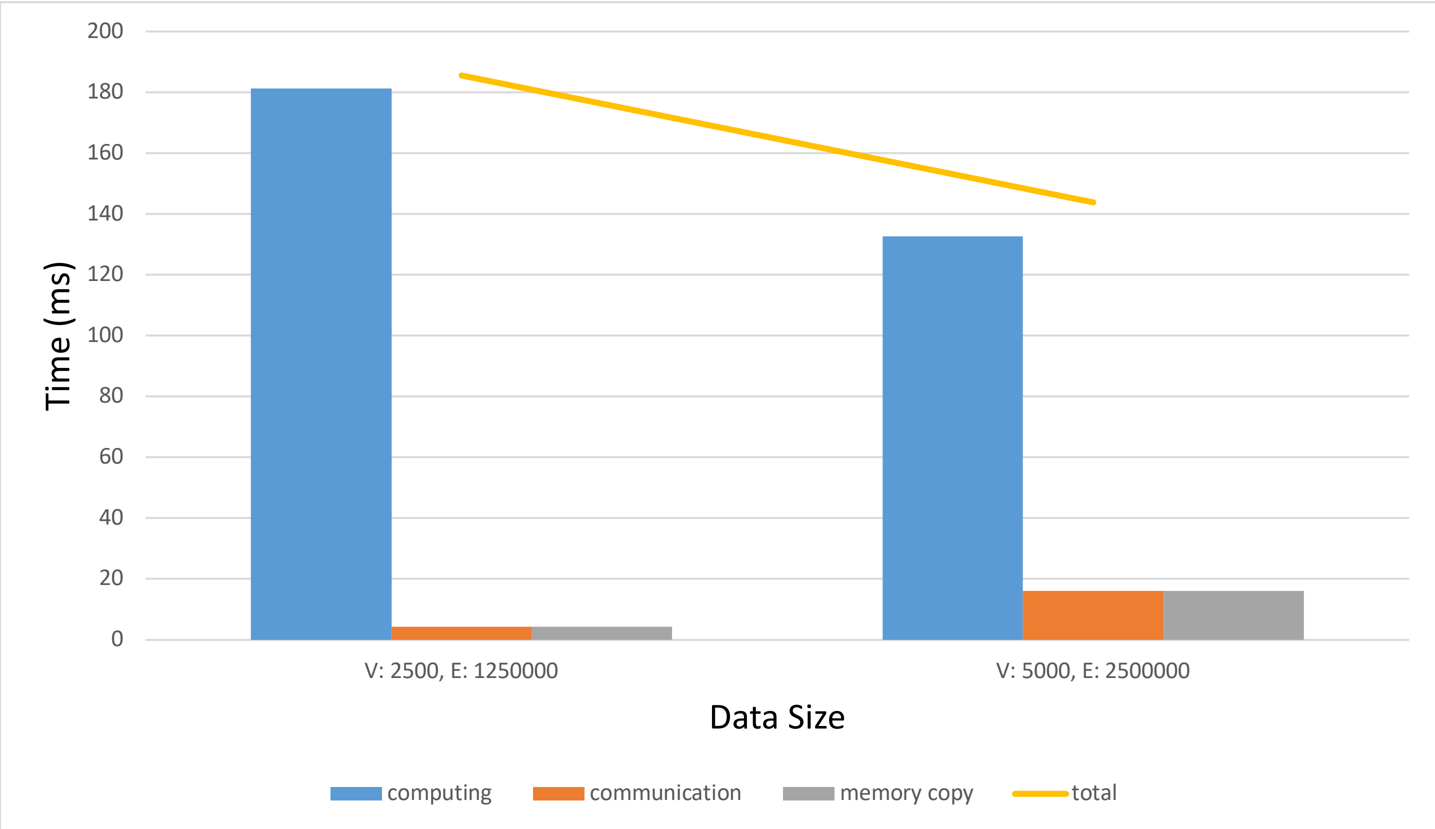


Figure 2. Weak scalability of Multi-Node

| | Total | #1 | #2 |
|---|---|---|---|
| improved_APSP | 439.33 (ms) | cudaMemcpy2D (64.76%) | cudaMallocPitch (34.94%) |
| APSP | 13.3779 (s) | cudaDeviceSynchronize (97.76%) | cudaMemcpy (1.11%) |

Figure 3. ASPS vs ASPS with CUDA 2D alignment.

**(b). Optimization**

CUDA 2D alignment and Shared memory

To optimize the algorithms, I store data in 2D instead of in 1D. Fig. 3 shows the the execution time of APSP and improved APSP and the API calls consuming first and second most time. We found that the time `cudaDeviceSynchronize` consumes occupies almost 98%. However, in improved APSP, 2D alignment API calls are the most consuming time.

## Experiences / Conclusion

In this homework, the most difficult assignment is how to implement Multi-Node since the implementation of synchronization is much more difficult than those of the other two algorithms.