

Parallel Programming

Blocked All Pairs Shortest Path

Peter Huang

1 、 Analysis

By draw one example, I can get following ideas and conclusions for the homework.

- A. Inter-Phase has strong dependence like Phase 2 needs to calculate before Phase1 is done and Phase 3 needs to calculate before Phase2 is done.
- B. Each cal (calculation action) in Phase 1 or 3 are independent so that we can parallelized any calculation actions within Phase1-3.
- C. Phase 3 dominated all calculation action, because it has the most amount of blocks between Phase1 – Phase3
- D. All the calculation actions in Blocked APSP is equal with classic Floyd-Warshall algorithm. Just sequence of calculation action is different only.

Block Factor 22

	0 1	2 3	4 5
0	1 2	3 4	5 6
1	7 8	9 10	11 12
2	13 14	15 16	17 18
3	19 20	21 22	23 24
4	25 26	27 28	29 30
5	31 32	33 34	35 36

Block Factor 22

phase 2 $k=0, 1$

Row

order in block

① $k=0$

② $k=1$

③ $k=0$

④ $k=1$

⑤ $k=0$

⑥ $k=1$

⑦ $k=0$

⑧ $k=1$

⑨ $k=0$

⑩ $k=1$

⑪ $k=0$

⑫ $k=1$

⑬ $k=0$

⑭ $k=1$

⑮ $k=0$

⑯ $k=1$

⑰ $k=0$

⑱ $k=1$

⑲ $k=0$

⑳ $k=1$

㉑ $k=0$

㉒ $k=1$

㉓ $k=0$

㉔ $k=1$

㉕ $k=0$

㉖ $k=1$

㉗ $k=0$

㉘ $k=1$

㉙ $k=0$

㉚ $k=1$

㉛ $k=0$

㉜ $k=1$

㉝ $k=0$

㉞ $k=1$

㉟ $k=0$

㊱ $k=1$

㊲ $k=0$

㊳ $k=1$

㊴ $k=0$

㊵ $k=1$

㊶ $k=0$

㊷ $k=1$

㊸ $k=0$

㊹ $k=1$

㊺ $k=0$

㊻ $k=1$

㊼ $k=0$

㊽ $k=1$

㊾ $k=0$

㊿ $k=1$

phase 3

① $k=0$

② $k=1$

③ $k=0$

④ $k=1$

⑤ $k=0$

⑥ $k=1$

⑦ $k=0$

⑧ $k=1$

⑨ $k=0$

⑩ $k=1$

⑪ $k=0$

⑫ $k=1$

⑬ $k=0$

⑭ $k=1$

⑮ $k=0$

⑯ $k=1$

⑰ $k=0$

⑱ $k=1$

⑲ $k=0$

⑳ $k=1$

㉑ $k=0$

㉒ $k=1$

㉓ $k=0$

㉔ $k=1$

㉕ $k=0$

㉖ $k=1$

㉗ $k=0$

㉘ $k=1$

㉙ $k=0$

㉚ $k=1$

㉛ $k=0$

㉜ $k=1$

㉝ $k=0$

㉞ $k=1$

㉟ $k=0$

㊱ $k=1$

㊲ $k=0$

㊳ $k=1$

㊴ $k=0$

㊵ $k=1$

㊶ $k=0$

㊷ $k=1$

㊸ $k=0$

㊹ $k=1$

㊺ $k=0$

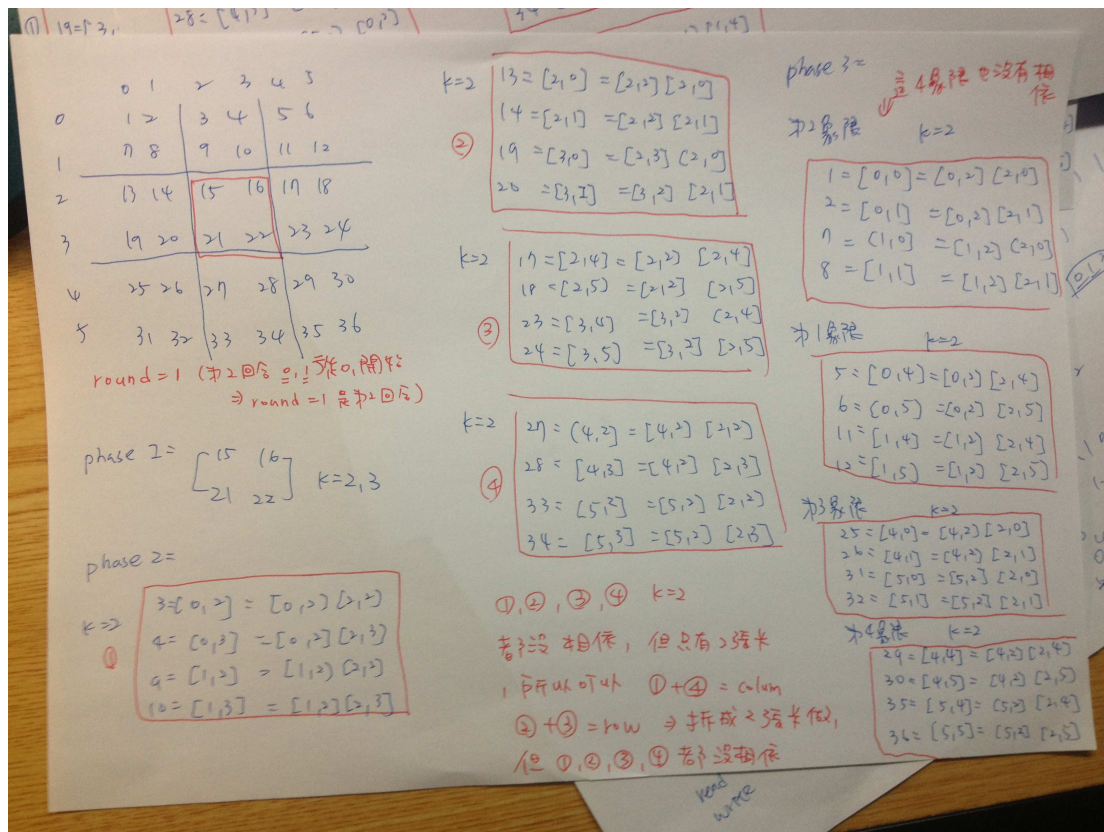
㊻ $k=1$

㊼ $k=0$

㊽ $k=1$

㊾ $k=0$

㊿ $k=1$



2、 Implementation

Single GPU:

A. How to divide data:

Handle out data by blocked APSP with 3 Phase

B. How to distribute data and configuration setting:

GridDim(Block_Height, Block_Width, 1)

BlockDim(Blocking_Factor, 1),

Optimal Blocking Factor:128

Because the GPU card in one SM which has 8 blocks, there are up to 1024 threads running. $1024 / 8 = 128$ threads in each block, which can maximally utilize GPU card's performance.

C. How to do blocked All Pairs Shortest Path in GPU kernel:

In kernel functions, because my blockDim is one dimension, I need a for loop in kernel function to simulate the two dimension actions to calculate all points. However to know which Block in GPU do calculate action, I use X, Y coordinate as offset to do mapping. In block (0,0) will always handle out data of (X,Y) block like:

```
i = blockIdx.x+x,  
j = blockIdx.y+y;
```

Multi GPU:

A. How to divide data:

Parallelized Phase3 for two GPU cards

Before divide data for parallelized computation, I analyze the computation time between phase1-3. I discover that phase3 dominated all computation action. In single GPU, Blocking Factor: 128 , All computation time is 14.341 sec when using in5 as input file. However, the phase 3 take 11.790 sec.

B. How to distribute data and configuration setting:

The setting of Multi GPU follows as Single GPU setting.

C. Multi GPU - implementation:

Using bit/char vector to records which value is be modified by GPU card and the bit/char vector size is N*N. When data has been modified when set 1 to the value. Then, when I need to map the data, the program scan the bit/char vector once, and outdate the value by the modified bit/char.

3、 Profiling Result

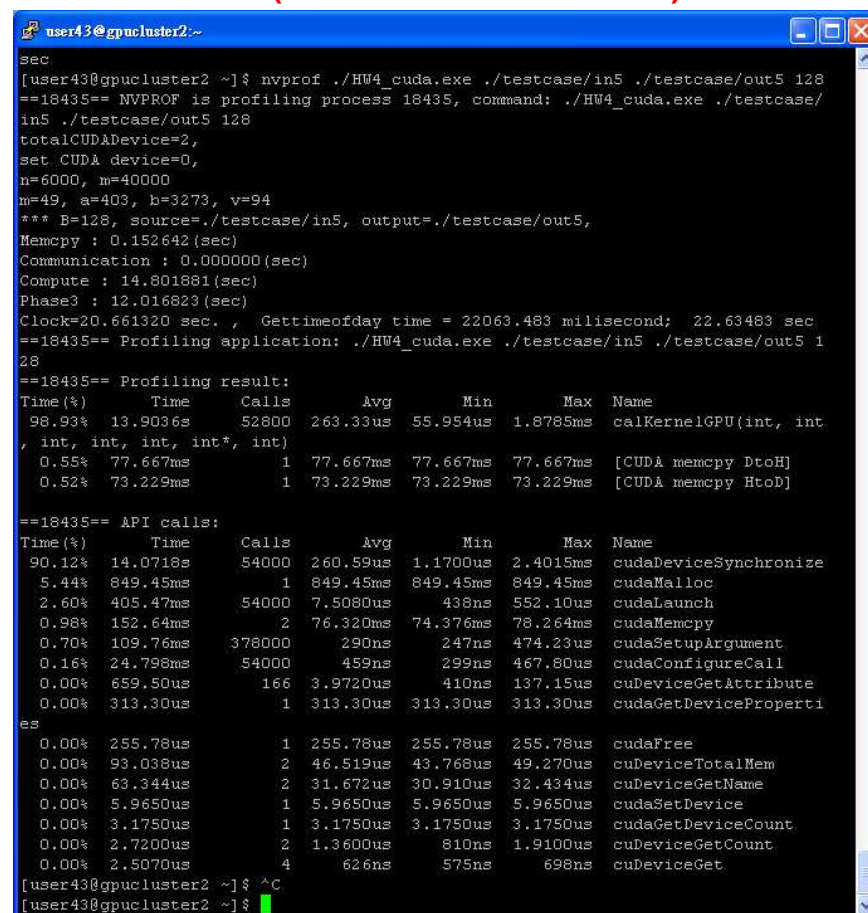
Why Blocking Factor=128 gets best performance:

BlockDim(Blocking_Factor, 1)

Because the GPU card in one SM which has 8 blocks, there are up to 1024 threads running. $1024 / 8 = 128$ threads in each block, which can maximally utilize GPU card's performance. If I use more threads Like Blocking Factor=256. For on SM, there are $256 * 8 = 2048$ threads in One SM, which means there are 1024 threads will idle and needs to consuming context switch time. (Ps. because running threading available at the same time is up to 1024 threads.)

Profiling in gpubluster2 server using Main device:0.

Block factor:128 (Best Performance in GPU)



```
user43@gpubluster2:~$ nvprof ./HW4_cuda.exe ./testcase/in5 ./testcase/out5 128
==18435== NVPROF is profiling process 18435, command: ./HW4_cuda.exe ./testcase/
in5 ./testcase/out5 128
totalCUDADevice=2,
set CUDA device=0,
n=6000, m=40000
m=49, a=403, b=3273, v=94
*** B=128, source=./testcase/in5, output=./testcase/out5,
Memcpy : 0.152642(sec)
Communication : 0.000000(sec)
Compute : 14.801881(sec)
Phase3 : 12.016823(sec)
Clock=20.661320 sec., Gettimeofday time = 22063.483 millisecond; 22.63483 sec
==18435== Profiling application: ./HW4_cuda.exe ./testcase/in5 ./testcase/out5 1
28
==18435== Profiling result:
Time(%) Time Calls Avg Min Max Name
98.93% 13.9036s 52800 263.33us 55.954us 1.8785ms calKernelGPU(int, int
, int, int, int, int*, int)
0.55% 77.667ms 1 77.667ms 77.667ms 77.667ms [CUDA memcpy DtoH]
0.52% 73.229ms 1 73.229ms 73.229ms 73.229ms [CUDA memcpy HtoD]
==18435== API calls:
Time(%) Time Calls Avg Min Max Name
90.12% 14.0718s 54000 260.59us 1.1700us 2.4015ms cudaDeviceSynchronize
5.44% 849.45ms 1 849.45ms 849.45ms 849.45ms cudaMalloc
2.60% 405.47ms 54000 7.5080us 438ns 552.10us cudaLaunch
0.98% 152.64ms 2 76.320ms 74.376ms 78.264ms cudaMemcpy
0.70% 109.76ms 378000 290ns 247ns 474.23us cudaSetupArgument
0.16% 24.798ms 54000 459ns 299ns 467.80us cudaConfigureCall
0.00% 659.50us 166 3.9720us 410ns 137.15us cuDeviceGetAttribute
0.00% 313.30us 1 313.30us 313.30us 313.30us cudaGetDeviceProperti
es
0.00% 255.78us 1 255.78us 255.78us 255.78us cudaFree
0.00% 93.038us 2 46.519us 43.768us 49.270us cuDeviceTotalMem
0.00% 63.344us 2 31.672us 30.910us 32.434us cuDeviceGetName
0.00% 5.9650us 1 5.9650us 5.9650us 5.9650us cudaSetDevice
0.00% 3.1750us 1 3.1750us 3.1750us 3.1750us cudaGetDeviceCount
0.00% 2.7200us 2 1.3600us 810ns 1.9100us cuDeviceGetCount
0.00% 2.5070us 4 626ns 575ns 698ns cuDeviceGet
[user43@gpubluster2 ~]$ ^C
[user43@gpubluster2 ~]$
```

Block factor: 32

```

user43@gpucluster2:~
[user43@gpucluster2 ~]$ nvprof ./HW4_cuda.exe ./testcase/in5 ./testcase/out5 32
==18445== NVPROF is profiling process 18445, command: ./HW4_cuda.exe ./testcase/in5
./testcase/out5 32
totalCUDADevice=2,
set CUDA device=0,
n=6000, m=40000
m=49, a=403, b=3273, v=94
*** B=32, source=./testcase/in5, output=./testcase/out5,
Memcpy : 0.156990(sec)
Communication : 0.000000(sec)
Compute : 38.291800(sec)
Phase3 : 37.121888(sec)
Clock=43.615791 sec., Gettimeofday time = 44457.725 millisecond; 44.457725 sec
==18445== Profiling application: ./HW4_cuda.exe ./testcase/in5 ./testcase/out5 32
==18445== Profiling result:
Time(%)   Time      Calls      Avg      Min      Max      Name
99.59%    37.3764s    53760    695.25us  10.622us  6.1859ms  calKernelGPU(int, int, 1
nt, int, int, int*, int)
0.21%    78.089ms     1    78.089ms  78.089ms  78.089ms  [CUDA memcpy DtoH]
0.21%    77.638ms     1    77.638ms  77.638ms  77.638ms  [CUDA memcpy HtoD]

==18445== API calls:
Time(%)   Time      Calls      Avg      Min      Max      Name
97.68%    37.5382s    54000    695.15us  1.2910us  6.1895ms  cudaDeviceSynchronize
1.10%    423.97ms    54000    7.8510us  511ns    509.89us  cudaLaunch
0.45%    174.26ms     1    174.26ms  174.26ms  174.26ms  cudaMalloc
0.41%    156.99ms     2    78.495ms  78.291ms  78.698ms  cudaMemcpy
0.28%    108.59ms    378000    287ns    247ns    496.85us  cudaSetupArgument
0.07%    25.393ms    54000    470ns    329ns    10.793us  cudaConfigureCall
0.00%    686.75us    166    4.1370us  415ns    162.67us  cuDeviceGetAttribute
0.00%    308.17us     1    308.17us  308.17us  308.17us  cudaGetDeviceProperties
0.00%    259.76us     1    259.76us  259.76us  259.76us  cudaFree
0.00%    84.438us     2    42.219us  40.773us  43.665us  cuDeviceTotalMem
0.00%    67.655us     2    33.827us  30.885us  36.770us  cuDeviceGetName
0.00%    12.422us     1    12.422us  12.422us  12.422us  cudaSetDevice
0.00%    2.6680us     4    667ns    578ns    790ns    cuDeviceGet
0.00%    2.5880us     2    1.2940us  633ns    1.9550us  cuDeviceGetCount
0.00%    2.1050us     1    2.1050us  2.1050us  2.1050us  cudaGetDeviceCount.
[user43@gpucluster2 ~]$

```

Block factor: 64

```

user43@gpucluster2:~
[user43@gpucluster2 ~]$ nvprof ./HW4_cuda.exe ./testcase/in5 ./testcase/out5 64
==18455== NVPROF is profiling process 18455, command: ./HW4_cuda.exe ./testcase/in5
./testcase/out5 64
totalCUDADevice=2,
set CUDA device=0,
n=6000, m=40000
m=49, a=403, b=3273, v=94
*** B=64, source=./testcase/in5, output=./testcase/out5,
Memcpy : 0.153282(sec)
Communication : 0.000000(sec)
Compute : 21.113639(sec)
Phase3 : 19.499131(sec)
Clock=26.863216 sec., Gettimeofday time = 27209.137 millisecond; 27.209137 sec
==18455== Profiling application: ./HW4_cuda.exe ./testcase/in5 ./testcase/out5 64
==18455== Profiling result:
Time(%)   Time      Calls      Avg      Min      Max      Name
99.26%    20.2147s    53440    378.27us  25.309us  3.2370ms  calKernelGPU(int, int, 1
nt, int, int, int*, int)
0.38%    78.063ms     1    78.063ms  78.063ms  78.063ms  [CUDA memcpy DtoH]
0.36%    73.477ms     1    73.477ms  73.477ms  73.477ms  [CUDA memcpy HtoD]

==18455== API calls:
Time(%)   Time      Calls      Avg      Min      Max      Name
96.10%    20.3848s    54000    377.50us  1.2860us  3.2404ms  cudaDeviceSynchronize
1.93%    409.38ms    54000    7.5810us  440ns    497.14us  cudaLaunch
0.72%    153.27ms     2    76.633ms  74.602ms  78.665ms  cudaMemcpy
0.62%    132.00ms     1    132.00ms  132.00ms  132.00ms  cudaMalloc
0.51%    108.20ms    378000    286ns    244ns    474.98us  cudaSetupArgument
0.11%    23.514ms    54000    435ns    332ns    470.60us  cudaConfigureCall
0.00%    654.57us    166    3.9430us  410ns    136.92us  cuDeviceGetAttribute
0.00%    315.32us     1    315.32us  315.32us  315.32us  cudaGetDeviceProperties
0.00%    264.64us     1    264.64us  264.64us  264.64us  cudaFree
0.00%    82.145us     2    41.072us  39.195us  42.950us  cuDeviceTotalMem
0.00%    65.653us     2    32.826us  30.210us  35.443us  cuDeviceGetName
0.00%    5.7400us     1    5.7400us  5.7400us  5.7400us  cudaSetDevice
0.00%    2.9530us     1    2.9530us  2.9530us  2.9530us  cudaGetDeviceCount
0.00%    2.6370us     2    1.3180us  602ns    2.0350us  cuDeviceGetCount
0.00%    2.3690us     4    592ns    453ns    663ns    cuDeviceGet
[user43@gpucluster2 ~]$

```

Blocking factor:256

```
user43@gpucluster2:~  
[user43@gpucluster2 ~]$ nvprof ./HW4_cuda.exe ./testcase/in5 ./testcase/out5 256  
==18473== NVPROF is profiling process 18473, command: ./HW4_cuda.exe ./testcase/in5  
./testcase/out5 256  
totalCUDADevice=2,  
set CUDA device=0,  
n=6000, m=40000  
m=49, a=403, b=3273, v=94  
*** B=256, source=./testcase/in5, output=./testcase/out5,  
Memcpy : 0.151990(sec)  
Communication : 0.000000(sec)  
Compute : 16.758840(sec)  
Phase3 : 11.653000(sec)  
Clock=22.561277 sec. , Gettimeofday time = 23003.281 millisecond; 23.3281 sec  
==18473== Profiling application: ./HW4_cuda.exe ./testcase/in5 ./testcase/out5 256  
==18473== Profiling result:  
Time(%) Time Calls Avg Min Max Name  
99.06% 15.8786s 52160 304.42us 71.492us 1.6712ms calKernelGPU(int, int, 1  
nt, int, int, int*, int)  
0.48% 77.514ms 1 77.514ms 77.514ms 77.514ms [CUDA memcpy DtoH]  
0.46% 73.382ms 1 73.382ms 73.382ms 73.382ms [CUDA memcpy HtoD]  
  
==18473== API calls:  
Time(%) Time Calls Avg Min Max Name  
94.47% 16.0449s 54000 297.13us 1.2880us 2.5393ms cudaDeviceSynchronize  
2.33% 395.37ms 54000 7.3210us 449ns 538.74us cudaLaunch  
1.51% 256.13ms 1 256.13ms 256.13ms 256.13ms cudaMalloc  
0.89% 151.99ms 2 75.995ms 73.872ms 78.117ms cudaMemcpy  
0.64% 108.84ms 378000 287ns 246ns 12.081us cudaSetupArgument  
0.15% 25.170ms 54000 466ns 353ns 12.280us cudaConfigureCall  
0.00% 661.85us 166 3.9870us 410ns 144.20us cuDeviceGetAttribute  
0.00% 319.45us 1 319.45us 319.45us 319.45us cudaGetDeviceProperties  
0.00% 258.51us 1 258.51us 258.51us 258.51us cudaFree  
0.00% 89.480us 2 44.740us 43.820us 45.660us cuDeviceTotalMem  
0.00% 69.798us 2 34.899us 30.995us 38.803us cuDeviceGetName  
0.00% 6.4000us 1 6.4000us 6.4000us 6.4000us cudaSetDevice  
0.00% 3.3200us 1 3.3200us 3.3200us 3.3200us cudaGetDeviceCount  
0.00% 2.9500us 2 1.4750us 813ns 2.1370us cuDeviceGetCount  
0.00% 2.5540us 4 638ns 500ns 787ns cuDeviceGet  
[user43@gpucluster2 ~]$
```

Blocking factor:512

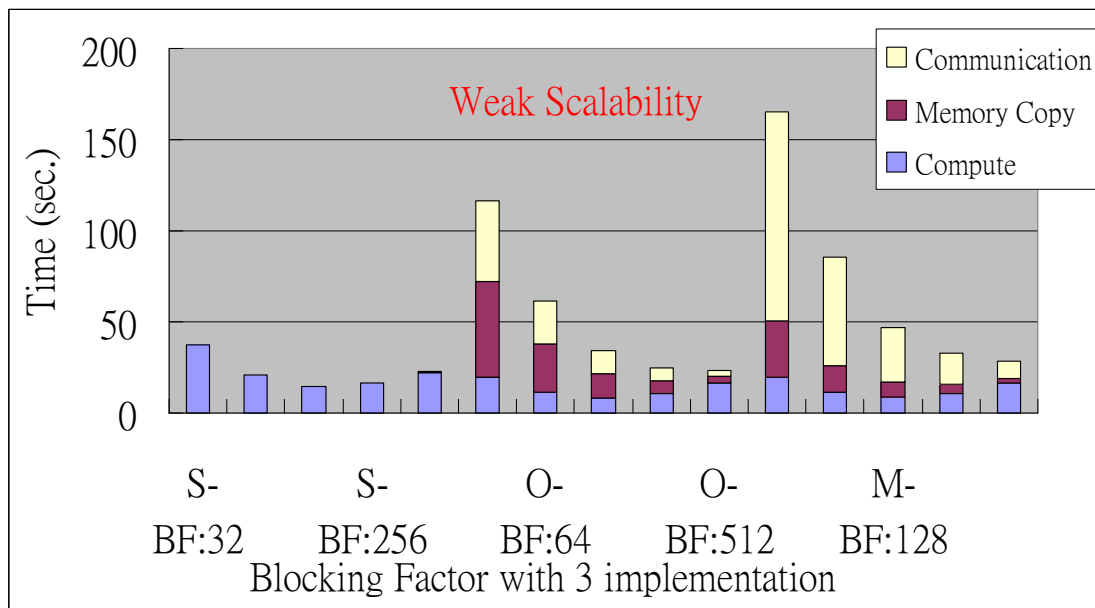
```
user43@gpucluster2:~  
[user43@gpucluster2 ~]$ nvprof ./HW4_cuda.exe ./testcase/in5 ./testcase/out5 512  
==18483== NVPROF is profiling process 18483, command: ./HW4_cuda.exe ./testcase/in5  
./testcase/out5 512  
totalCUDADevice=2,  
set CUDA device=0,  
n=6000, m=40000  
m=49, a=403, b=3273, v=94  
*** B=512, source=./testcase/in5, output=./testcase/out5,  
Memcpy : 0.152903(sec)  
Communication : 0.000000(sec)  
Compute : 22.803197(sec)  
Phase3 : 12.702482(sec)  
Clock=28.703159 sec. , Gettimeofday time = 29507.717 millisecond; 29.507717 sec  
==18483== Profiling application: ./HW4_cuda.exe ./testcase/in5 ./testcase/out5 512  
==18483== Profiling result:  
Time(%) Time Calls Avg Min Max Name  
99.32% 21.9302s 49600 442.14us 213.02us 1.6185ms calKernelGPU(int, int, 1  
nt, int, int, int*, int)  
0.35% 77.747ms 1 77.747ms 77.747ms 77.747ms [CUDA memcpy DtoH]  
0.33% 73.391ms 1 73.391ms 73.391ms 73.391ms [CUDA memcpy HtoD]  
  
==18483== API calls:  
Time(%) Time Calls Avg Min Max Name  
95.30% 22.0946s 54000 409.16us 1.1990us 2.0216ms cudaDeviceSynchronize  
1.78% 413.84ms 1 413.84ms 413.84ms 413.84ms cudaMalloc  
1.67% 387.48ms 54000 7.1750us 418ns 546.30us cudaLaunch  
0.66% 152.89ms 2 76.447ms 74.542ms 78.352ms cudaMemcpy  
0.48% 110.31ms 378000 291ns 244ns 480.67us cudaSetupArgument  
0.10% 24.283ms 54000 449ns 320ns 479.18us cudaConfigureCall  
0.00% 660.88us 166 3.9810us 407ns 142.38us cuDeviceGetAttribute  
0.00% 316.50us 1 316.50us 316.50us 316.50us cudaGetDeviceProperties  
0.00% 261.17us 1 261.17us 261.17us 261.17us cudaFree  
0.00% 87.188us 2 43.594us 39.795us 47.393us cuDeviceTotalMem  
0.00% 66.730us 2 33.365us 30.743us 35.987us cuDeviceGetName  
0.00% 5.5350us 1 5.5350us 5.5350us 5.5350us cudaSetDevice  
0.00% 3.1070us 1 3.1070us 3.1070us 3.1070us cudaGetDeviceCount  
0.00% 2.6850us 2 1.3420us 582ns 2.1030us cuDeviceGetCount  
0.00% 2.5150us 4 628ns 498ns 737ns cuDeviceGet  
[user43@gpucluster2 ~]$
```

4、 Experiment And Analysis

1、 System Spec

Running in **gpucluster2** with all cases by using **in5** file.

2、 Weak scalability & Time distribution



Single GPU (S)					
BF=Blocking Factor	BF:32	BF:64	BF:128	BF:256	BF:512
Memory Copy	0.139685	0.139703	0.139994	0.283836	0.139745
Communication	0	0	0	0	0
Compute	37.482757	20.601837	14.341786	16.361533	22.38672
phase3	36.558651	19.223927	11.790387	11.461537	12.514091

Multi GPU- OpenMP (O)					
BF=Blocking Factor	BF:32	BF:64	BF:128	BF:256	BF:512
Memcpy	52.6527	26.639823	13.210143	6.934426	3.748701
Communication	44.215727	23.427156	12.749118	7.141067	3.508327
Compute	19.539619	11.20195	8.528045	10.623591	16.411127
Total	116.408	61.26893	34.48731	24.69908	23.66816

Mulit GPU- MPI (M)					
BF=Blocking Factor	BF:32	BF:64	BF:128	BF:256	BF:512
Memcpy	31.185486	14.956198	8.425116	4.76195	2.315235
Communication	114.608579	59.551292	30.047409	17.700295	9.605734
Compute	19.488218	11.186668	8.599019	10.744805	16.392509
Total	165.2823	85.694158	47.071544	33.20705	28.313478

Analysis:

In single GPU

The blocking factor is 128 gets best performance as we predicted previous section. And the phase 3 are dominated on blocked all pairs shortest path.

GridDim(Block_Height, Block_Width, 1)
BlockDim(Blocking_Factor, 1) ,
Optimal Blocking Factor:128

Because the GPU card in one SM which has 8 blocks, there are up to 1024 threads running. $1024 / 8 = 128$ threads in each block, which can maximally utilize GPU card's performance.

In Multi-GPU

A. In OpenMP & MPI there are the almost same computation time in GPU, because the calculated pointed in the programs are the same and with the same parallelized action in phase 3.

B. Total Time = Compute Time + (2 cards' communication time, data separation and merger action)

In OpenMP there are almost double memory copy time, because in OpenMP I records all memory copy action which happens in One processor. However, in MPI there are two processors, so I just records memory copy in rank=0's processor.

This is why we get almost two double memory copy time in openMP.

In communication time the MPI take a lot time than OpenMP, because MPI mechanism needs to use send and receive to communicate with another GPU card and OpenMP is shared memory, so all the communication happens between processors only. Hence, the MPI take a lot of communication time.

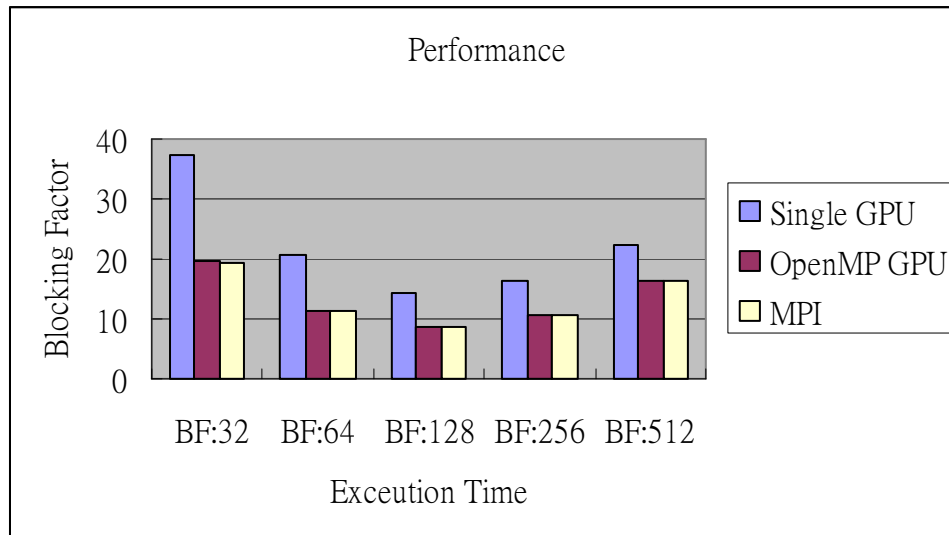
However, based on the result, if we want to get better performance, we need to trade off Compute Time and 2 cards' communication time, data separation and merger action. Although blocking factor has best performance in computation in GPU card, it take too much on data's separated, merged and communicated action. So In multi-GPU card if we need to decide how to divide our data by these evaluation. Finally, we can see if we use BF=512, it take a little time on data handling and also its computation time in GPU are acceptable. And, then we get best performance in my experiment between BF=32 ~ BF=512.

5 、 Blocking Factor

Test file :in5 n=6000

Because blocked APSP's calculated action is equal with classic floyd-warshall, we can evaluate that calculation action in kernel mode is the same as class floyd-warshall. $O(N^3)$

A.Execution Time Performance



Excution time (sec.)	BF:32	BF:64	BF:128	BF:256	BF:512
Single GPU	37.482757	20.601837	14.341786	16.361533	22.38672
OpenMP GPU	19.539619	11.20195	8.528045	10.623591	16.411127
MPI	19.488218	11.186668	8.599019	10.744805	16.392509

Analysis

In single GPU there are bad performance in computation, because we only use one card to calculate all data. However, in multiple GPU we always get better performance on computation because there are more cards to parallelize to calculate data.

B.B.Performance (GFLOPs)

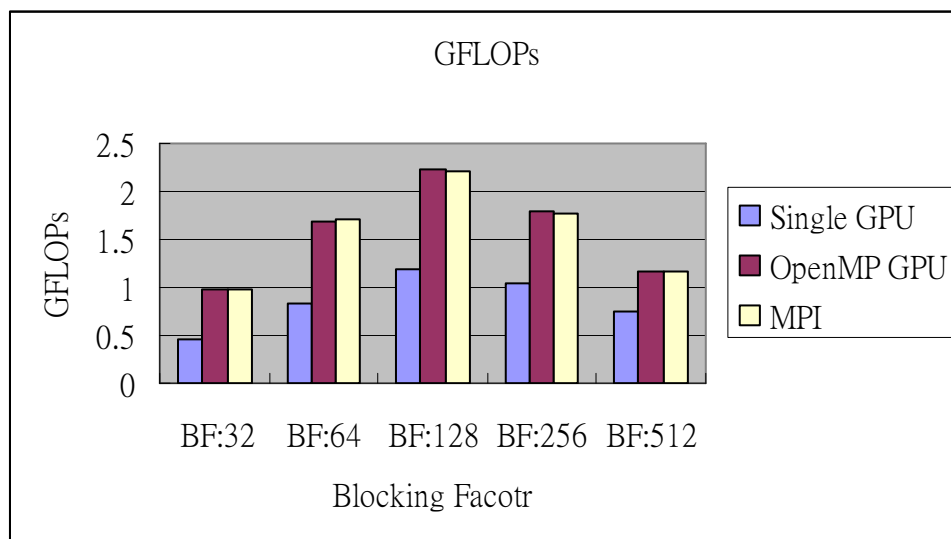
Single GPU GFlops= $O(n^3 \cdot \text{operation}) = O(n^3 \cdot 17) = 6000 \cdot 6000 \cdot 6000 \cdot 17$ (GPU kernel with 17 operation)

Multi GPU GFlops= $O(n^3 \cdot \text{operation}) = 6000 \cdot 6000 \cdot 6000 \cdot 19$

1data needs how much instruction	BF:32	BF:64	BF:128	BF:256	BF:512
Single GPU	17/37.48	17/20.60	17/14.34	17/16.36	17/22.38
OpenMP GPU	19/19.53	19/11.20	19/8.52	19/10.62	19/16.41

MPI	19/19.48	19/11.18	19/8.59	19/10.74	19/16.39
-----	----------	----------	---------	----------	----------

Gflops	BF:32	BF:64	BF:128	BF:256	BF:512
Single GPU	0.45357524	0.825242718	1.185495119	1.039119804	0.759606792
OpenMP GPU	0.972862263	1.696428571	2.230046948	1.789077213	1.157830591
MPI	0.975359343	1.699463327	2.211874272	1.769087523	1.159243441



C. Bandwidth [GB/sec]

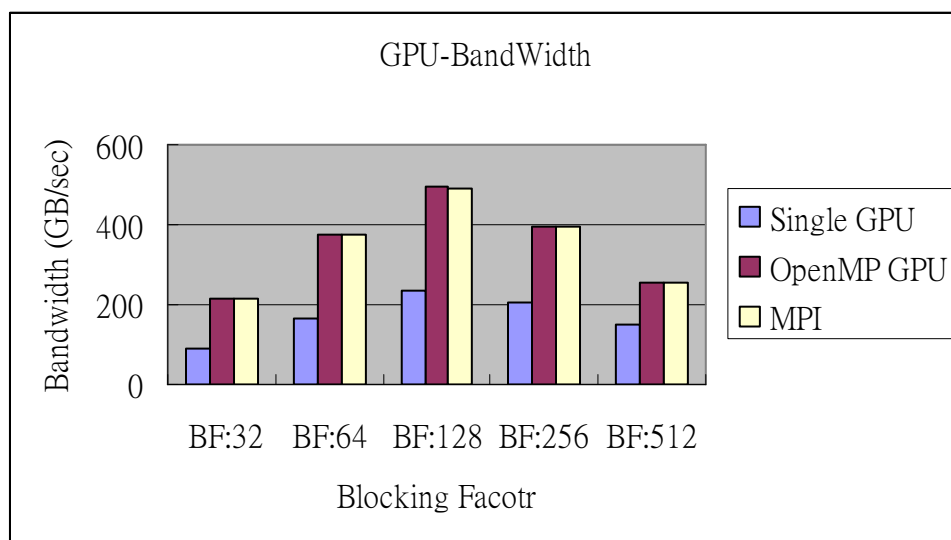
Single GPU bandwidth = $O(n^3 \cdot 4)$ (Ps. 3 input data and 1 write back data) = $6000 \cdot 6000 \cdot 6000 \cdot 4 \cdot 4$ byte (PS. int = 4 byte) = 3375 GB / execution time

Single GPU	BF:32	BF:64	BF:128	BF:256	BF:512
GB/sec	3375	3375	3375	3375	3375
GPU execution time	37.48	20.60	14.34	16.36	22.38
Result	90.04	163.82	235.32	206.27	150.75

Multi GPU bandwidth = $O(n^3 \cdot 5)$ (Ps. 3 input data and 2 write back data) = $6000 \cdot 6000 \cdot 6000 \cdot 5 \cdot 4$ byte = 4219 GB / execution time

OpenMP GPU	BF:32	BF:64	BF:128	BF:256	BF:512
GB/sec	4219	4219	4219	4219	4219
GPU execution time	19.53	11.20	8.52	10.62	16.41
Result	215.92	376.63	494.72	397.13	257.08

MPI	BF:32	BF:64	BF:128	BF:256	BF:512
GB/sec	4219	4219	4219	4219	4219
GPU execution time	19.48	11.18	8.59	10.74	16.39
Result	216.48	377.14	490.63	392.65	257.37



GB/sec	BF:32	BF:64	BF:128	BF:256	BF:512
Single GPU	90.04	163.82	235.32	206.27	150.75
OpenMP GPU	215.92	376.63	494.72	397.13	257.08
MPI	216.48	377.14	490.63	392.65	257.37

Analysis

Based on Gflops and bandwidth, we can know all programs are limited by calculation capability or bandwidth capability. When the program ups to the limit of bandwidth, I know I need to improve the access of memory to achieve better performance. However, if

we find that our program cannot maximally utilize our calculation, which means the GPU card's performance is not be utilized, because the amount of data is not large enough to utilized all our GPU card performance.

6 、 Compare 3 implementation

Sequential Code with Blocking Factor=128 takes 947.07 sec in in5.

Single Cuda with Blocking Factor=128 takes 14.481 sec in in5.

OpenMP with Blocking Factor=128 takes 34.487 sec in in5 and it takes 26 sec in Memory Copy and Communication.

MPI with Blocking Factor=128 takes 47.071 sec in in5 and it takes 38 sec in Memory Copy and Communication.

Speedup

Single cuda:	$947.07 / 14.481 = 65.3$
OpenMp:	$947.07 / 34.487 = 27.4$
MPI:	$947.07 / 47.071 = 20.1$

Performance: Single cuda > OpenMP > MPI

Analysis:

OpenMP: the computation gets better in 2 GPU cards than in 1card. If we only compare the communication time which is reduced from 14.341 to 8.528. However, because there are two cards, my program get more cost in memory copy and communication, because my programs use D2H and H2D not D2D. And, also I only copy memory for main programs. Finally, I get more time in Memory copy and communication.

MPI: the computation also gets better in 2 GPU cards than in 1 card. If we only compare the communication time which is reduced from 14.341 to 8.599. However, there are more communication time between processors and MPI's communication take most of time, which is why MPI get poor performance, when compared with OpenMP and single cuda.

Improvement:

In OpenMp the 2 cards' communications uses D2H and then call H2D in another device. If we can D2D in our 2 GPU card' devices, then the time for Memory Copy and Communication can be reduced. Then, get better performance.

7 、 Experience/Conclusion

a 、 What I have learned from this assignment

- 1 、 How to divide data and assign data into SIMD (single instruction and multiple data). For example: when there are $n \times n$ data, in cuda we can parallelized the $n \times n$ action in kernel function, and these $n \times n$ data at least take n^2 computation in sequential code. However, in cuda we only need take one time in kernel function with different threads, which largely saves computation time.
- 2 、 Learn some cuda's conception and analysis data's dependence and learn how design GridDim and BlockDim to cuda device. And how to get correct computation in cuda kernel function.

b 、 What difficulty did I encounter when

implementing this assignment

- 1 、 Before programming cuda program, I get many bugs in dividing data. And don't know how to parallelized my codes. However, finally, I try to simulate cuda action with OpenMP. Try to validate my programs in CPU version and it is good way to validate all my idea before implementing cuda code.
- 2 、 When bugging in cuda, I use printf and compared the right data in sequential code with different phase to find out which part my calculation is wrong or which part makes the bugs when I try to communication data in two cuda device.