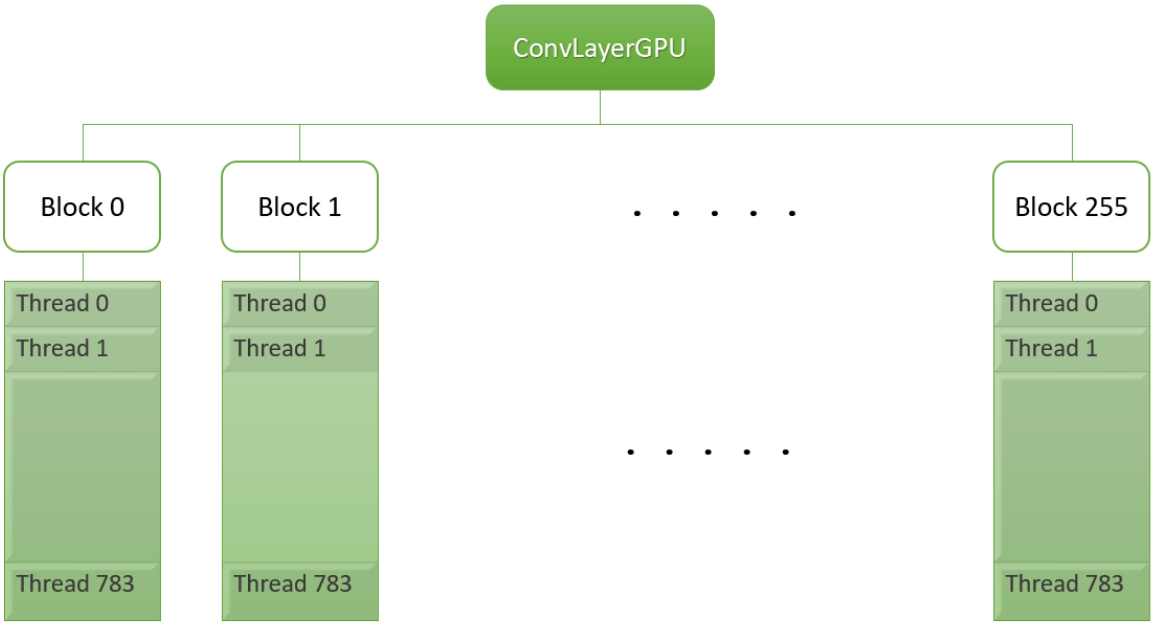
Computer Architecture (2017) Final Project Part3

Student1 ID：0410137 , Name：劉家麟 、 Student2 ID：0410110 , Name：林容安

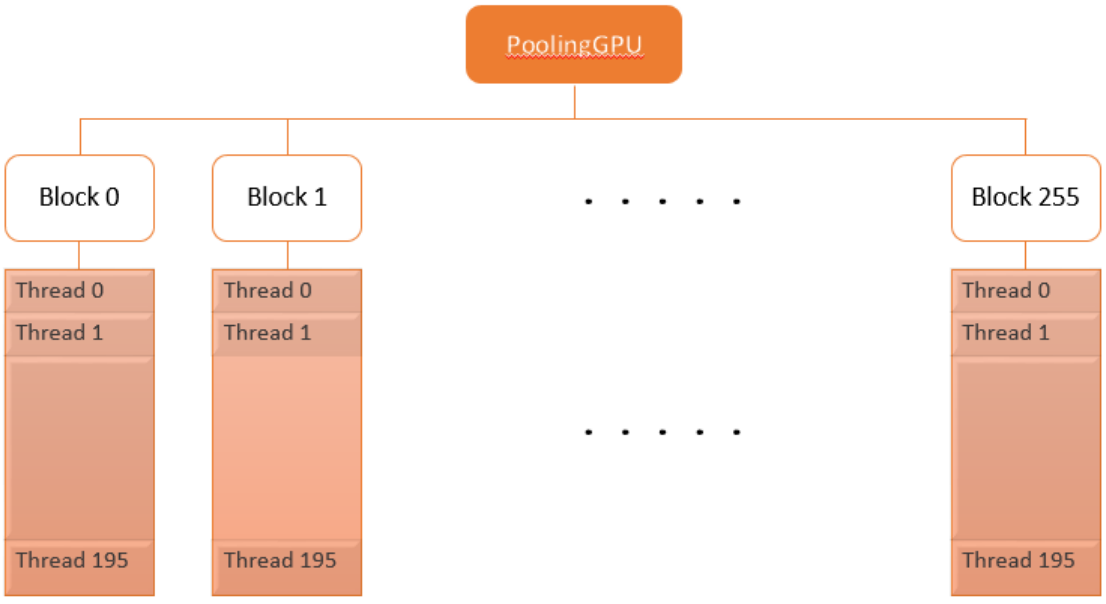
1. Algorithm：(10%)

ConvLayer_GPU:



Each block cope with a filter, so the number of blocks is set to (FILTNUM = 256). The threads take care of the elements in the frame, so we use 2D threads, each dimension of size FMSIZE 28 (total 28*28 threads) to implement the process.

PoolingGPU:



In PoolingGPU, we implement max pooling with 2*2 window size, so the threads we use become 2D, size 14 (FMSIZE/2) in each dimension (14*14 threads).

2. How do you (10%)
- increase data reuse
- reduce branch divergence or increase memory coalescing
- implement other optimization

There are several version, and we implemented different ways.

In the version 1, we increase memory coalescing with __shared__ variable in convLayerGPU(), so that different threads can access the same data array. By this way, the sum array can be divided into 192 (FMDEPTH) threads and sum up the array after synchronizing all these threads. Because the array is indexed by sli (FMDEPTH) and there will be some synchronization problem when putting both fmx, fmy and sli in same level of thread partition (__syncthreads() is to synchronize threads in a block), we cut FMSIZE with number of blocks, which is our version 2.

However, in version 2, it didn't perform as well as we expected. It only 2ms less than version 1. And we think its bad performance is owing to parallelism driven by GPU, which is thread-level not block-level. Besides, the way dividing total sum into an array needs more branch to detect when to sum up. Thus, we gave up this method and design our version 3.

In version 3, we cut as many threads as we can, so that we cut it into 256 (FILTNUM) blocks with 28*28 (FMSIZE*FMSIZE). In this way, not only can we have more parallelism with threads but also reduce branches. And the result of version 3 does much better than the other 2.

version 1

```
ca57@Taipei:~/CA2017FP-Part3$ make run
./CNNConvLayer
CPU time for executing a typical convolutional layer = 2060ms
GPU time for executing a typical convolutional layer = 107.893ms
Congratulations! You pass the check.
Speedup: 19.0945
```

version 2

```
ca57@Taipei:~/CA2017FP-Part3$ make run
./CNNConvLayer
CPU time for executing a typical convolutional layer = 2064ms
GPU time for executing a typical convolutional layer = 105.492ms
Congratulations! You pass the check.
Speedup: 19.5717
```

version 3

```
ca57@Taipei:~/CA2017FP-Part3$ make run
./CNNConvLayer
CPU time for executing a typical convolutional layer = 2063ms
GPU time for executing a typical convolutional layer = 61.903ms
Congratulations! You pass the check.
Speedup: 33.3367
```

3. Comparing part 3 with part 2 , do you get speedup? why or why not?(10%)

Compared to the result of part 2, we do gain a speedup. In every part of the project, we find a relatively large portion of execution time spent on memcpy; in part 3, the time of memcpy isn't included as we measure the resulting speedup, so we get about 6 times better performance than part 2.

As only for execution time, we don't really get a speedup. Last time, we made an effort and spent a lot of time on designing and adjusting the COO format algorithm. However, this time TA change the all input size so that we failed to apply FP2 directly on FP3, and it is already the end of semester which means there are lots of finals and final projects waiting for us, so we don't have such sufficient time to adjust COO format algorithm for the new input size. As a consequence, we don't get a speedup comparing to COO one.

4. Show how you use NVVP to help you find and solve perf(5%)

version1 nvprof

```
ca57@Taipei:~/CA2017FP-Part3$ nvprof ./CNNConvLayer
CPU time for executing a typical convolutional layer = 2071ms
==20443== NVPROF is profiling process 20443, command: ./CNNConvLayer
GPU time for executing a typical convolutional layer = 107.291ms
Congratulations! You pass the check.
Speedup: 19.3033
==20443== Profiling application: ./CNNConvLayer
==20443== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   99.26%  106.74ms         1  106.74ms  106.74ms  106.74ms  convGPU_v1(int*, int*, int*)
                    0.67%  720.74us         2  360.37us  183.68us  537.06us  [CUDA memcpy HtoD]
                    0.05%  58.528us         1  58.528us  58.528us  58.528us  [CUDA memcpy DtoH]
                    0.01%  12.480us         1  12.480us  12.480us  12.480us  poolGPU(int*, int*)
      API calls:   60.14%  165.23ms         4  41.308ms  148.34us  164.77ms  cudaMalloc
                   38.86%  106.76ms         2  53.380ms  15.358us  106.74ms  cudaDeviceSynchronize
                    0.43%  1.1681ms       188  6.2130us     250ns  249.82us  cuDeviceGetAttribute
                    0.25%  681.95us         3  227.32us  102.74us  475.02us  cudaMemcpy
                    0.19%  523.65us         2  261.83us  14.502us  509.15us  cudaLaunch
                    0.09%  239.11us         2  119.56us  71.614us  167.50us  cuDeviceTotalMem
                    0.04%  114.10us         2  57.051us  51.176us  62.927us  cuDeviceGetName
                    0.01%  13.769us         4  3.4420us     529ns  11.941us  cudaFree
                    0.00%  2.8970us         3     965ns     447ns  1.7950us  cuDeviceGetCount
                    0.00%  2.7520us         4     688ns     298ns  1.1040us  cuDeviceGet
                    0.00%  1.6760us         2     838ns     480ns  1.1960us  cudaConfigureCall
                    0.00%  1.6490us         5     329ns     130ns     719ns  cudaSetupArgument
```

We can tell that convGPU spent the most time in this picture, so we try to improve the function with cut it into more blocks so that it doesn't need to run so many for loops.

version2 nvprof

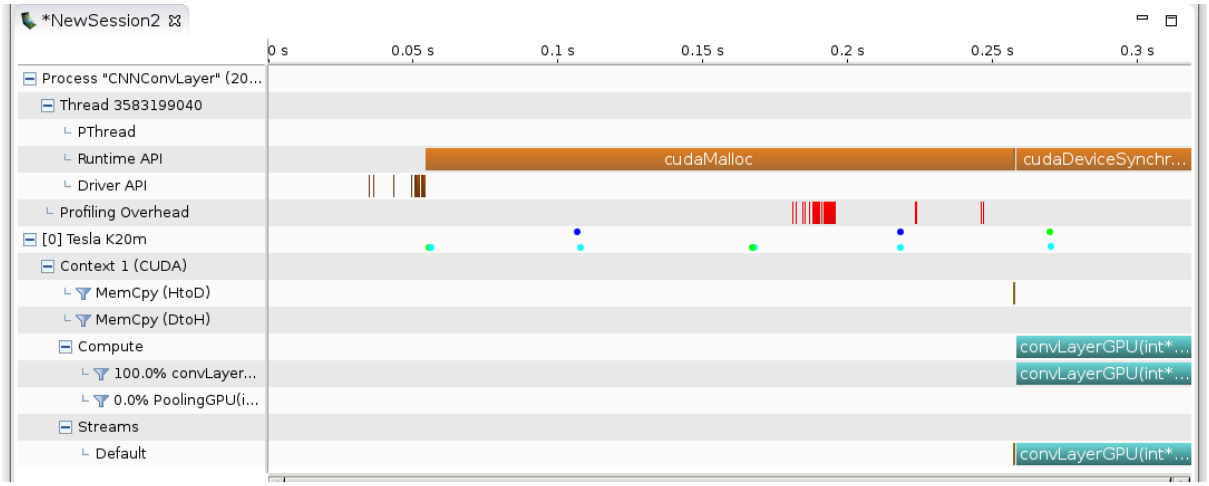
```
ca57@Taipei:~/CA2017FP-Part3$ nvprof ./CNNConvLayer
CPU time for executing a typical convolutional layer = 2062ms
==20502== NVPROF is profiling process 20502, command: ./CNNConvLayer
GPU time for executing a typical convolutional layer = 105.115ms
Congratulations! You pass the check.
Speedup: 19.6248
==20502== Profiling application: ./CNNConvLayer
==20502== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   99.25%  104.56ms         1  104.56ms  104.56ms  104.56ms  convGPU(int*, int*, int*)
                    0.68%  720.68us         2  360.34us  183.62us  537.06us  [CUDA memcpy HtoD]
                    0.06%  59.456us         1  59.456us  59.456us  59.456us  [CUDA memcpy DtoH]
                    0.01%  12.737us         1  12.737us  12.737us  12.737us  poolGPU(int*, int*)
      API calls:   58.11%  148.87ms         4  37.217ms  134.04us  148.45ms  cudaMalloc
                   40.83%  104.59ms         2  52.293ms  17.094us  104.57ms  cudaDeviceSynchronize
                    0.45%  1.1410ms       188  6.0690us     208ns  271.43us  cuDeviceGetAttribute
                    0.26%  656.83us         3  218.94us  100.75us  451.92us  cudaMemcpy
                    0.20%  520.94us         2  260.47us  13.747us  507.19us  cudaLaunch
                    0.08%  213.69us         2  106.85us  65.139us  148.55us  cuDeviceTotalMem
                    0.07%  178.09us         2  89.043us  47.610us  130.48us  cuDeviceGetName
                    0.00%  9.2190us         4  2.3040us     808ns  4.6540us  cudaFree
                    0.00%  2.1720us         3     724ns     233ns  1.5870us  cuDeviceGetCount
                    0.00%  1.9730us         4     493ns     217ns     859ns  cuDeviceGet
                    0.00%  1.6340us         2     817ns     466ns  1.1680us  cudaConfigureCall
                    0.00%  1.5060us         5     301ns     130ns     593ns  cudaSetupArgument
```

It didn't be improved so much, so we keep dividing the for loops.

version3 nvprof

```
ca57@Taipei:~/CA2017FP-Part3$ nvprof ./CNNConvLayer
CPU time for executing a typical convolutional layer = 2072ms
==20560== NVPROF is profiling process 20560, command: ./CNNConvLayer
GPU time for executing a typical convolutional layer = 61.577ms
Congratulations! You pass the check.
Speedup: 33.6539
==20560== Profiling application: ./CNNConvLayer
==20560== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   98.72%  61.018ms         1  61.018ms  61.018ms  61.018ms  convLayerGPU(int*, int*, int*)
                    1.17%  720.93us         2  360.47us  183.84us  537.09us  [CUDA memcpy HtoD]
                    0.10%  59.520us         1  59.520us  59.520us  59.520us  [CUDA memcpy DtoH]
                    0.02%  12.608us         1  12.608us  12.608us  12.608us  PoolingGPU(int*, int*)
      API calls:   70.63%  153.23ms         4  38.308ms  141.85us  152.80ms  cudaMalloc
                   28.14%  61.040ms         2  30.520ms  16.804us  61.023ms  cudaDeviceSynchronize
                    0.52%  1.1304ms       188  6.0120us     228ns  241.30us  cuDeviceGetAttribute
                    0.30%  658.73us         3  219.58us  99.963us  454.27us  cudaMemcpy
                    0.24%  529.19us         2  264.59us  14.727us  514.46us  cudaLaunch
                    0.10%  222.70us         2  111.35us  67.430us  155.27us  cuDeviceTotalMem
                    0.05%  110.13us         2  55.067us  48.962us  61.172us  cuDeviceGetName
                    0.00%  5.2460us         4  1.3110us     496ns  3.7290us  cudaFree
                    0.00%  2.5650us         3     855ns     293ns  1.6790us  cuDeviceGetCount
                    0.00%  2.5530us         4     638ns     286ns     967ns  cuDeviceGet
                    0.00%  1.6220us         2     811ns     418ns  1.2040us  cudaConfigureCall
                    0.00%  1.6070us         5     321ns     123ns     611ns  cudaSetupArgument
```

version3 mvvp



Function Name	Time on Critical Path (%)	Time on Critical Path	Waiting time	
cudaMalloc	63.42 %	202.76692 ms	0 ns	
convLayerGPU(int*, int*, int*)	19.10 %	61.05726 ms	0 ns	
<Other>	16.06 %	51.34306 ms	0 ns	
cuDeviceGetAttribute	0.97 %	3.11642 ms	0 ns	
[CUDA memcpy HtoD]	0.23 %	720.678 µs	0 ns	
cuDeviceTotalMem_v2	0.13 %	414.92 µs	0 ns	
cuDeviceGetName	0.06 %	194.556 µs	0 ns	
[CUDA memcpy DtoH]	0.02 %	59.489 µs	0 ns	
PoolingGPU(int*, int*)	0.00 %	12.448 µs	0 ns	
cuDeviceGetPCIBusId	0.00 %	8.191 µs	0 ns	
cudaFree	0.00 %	6.494 µs	0 ns	
cuDeviceGetCount	0.00 %	4.41 µs	0 ns	
cuDeviceGet	0.00 %	3.855 µs	0 ns	
cudaConfigureCall	0.00 %	479 ns	0 ns	
cudaSetupArgument	0.00 %	433 ns	0 ns	
pthread_enter	0.00 %	0 ns	0 ns	
pthread_exit	0.00 %	0 ns	0 ns	
cudaMemcpy	0.00 %	0 ns	59.489 µs	
cudaLaunch	0.00 %	0 ns	0 ns	
cudaDeviceSynchronize	0.00 %	0 ns	61.0697 ms	

For the picture, we can tell that convLayerGPU has been reduced from over 100ms to only about 61ms.

5. Feedback(5%)

In the beginning of the semester, we knew little about CUDA and parallel computing, not to mention dynamic programming under CUDA GPU. After these three parts of project, we get to be more familiar with the concepts of parallelism and how powerful the GPU is at enhancing the performance. Although we sometimes get really concerned over the new input type of each part, and sometimes cannot find the best solution to deal with different inputs, we still learned a lot from making efforts to make our program better.

Last but not least, we want to show our gratitude to the TAs who has been helping us patiently throughout the semester. Without your help, we might not make it to completing the project. And also we are sorry about that we always broke your station before FP deadlines.