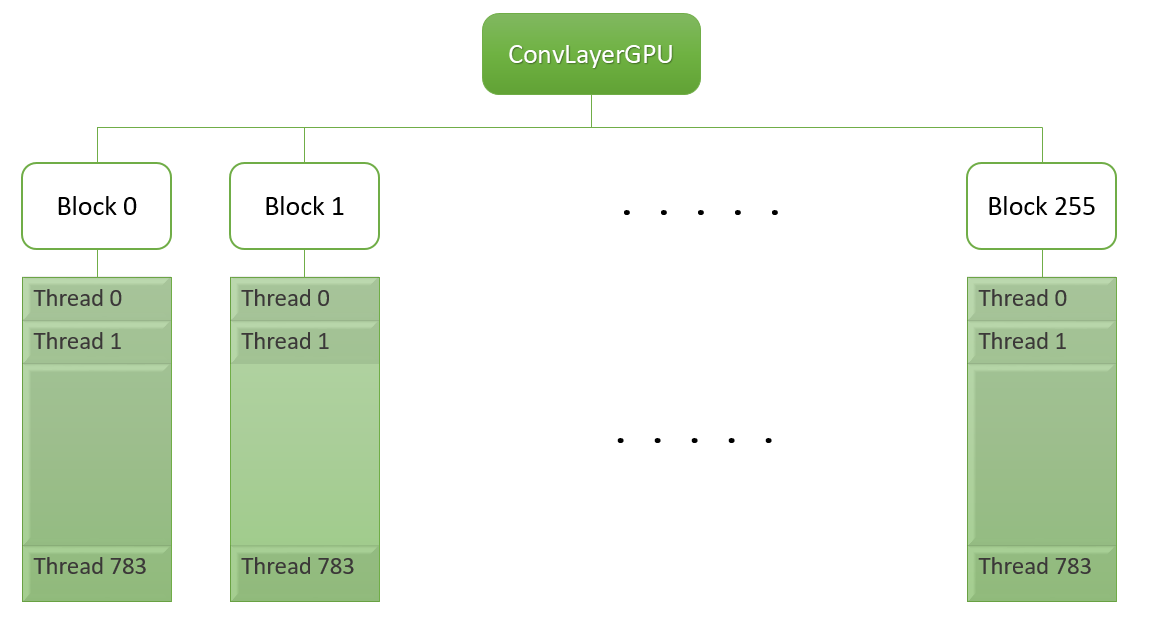
**Computer Architecture (2017) Final Project Part3**

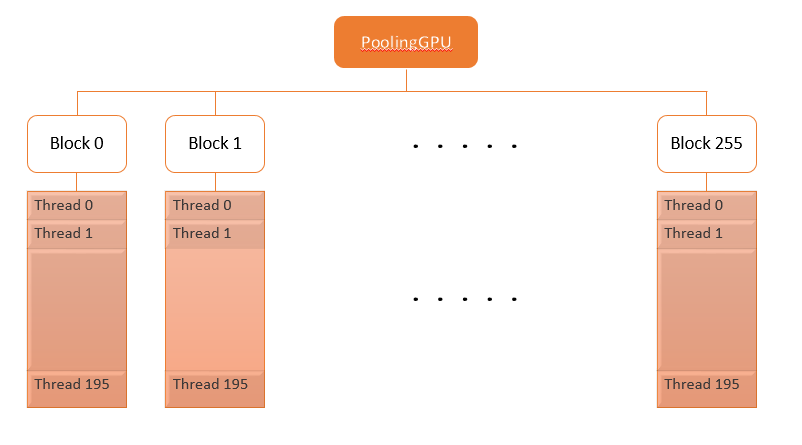
Student1 ID：0410137 , Name：劉家麟 、 Student2 ID：0410110 , Name：林容安

**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**

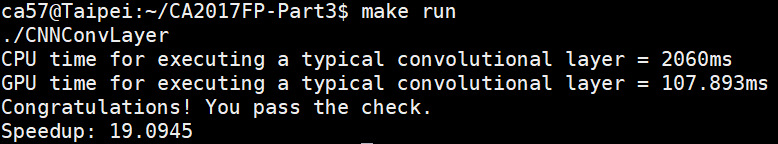
There are several version, and we implemented different ways.

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.

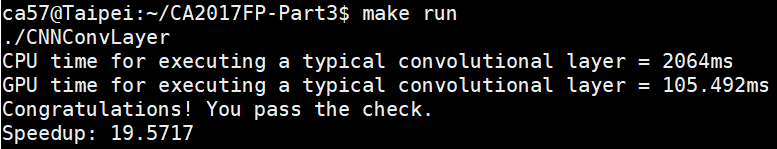
However, in version 2, it didn’t perform as well as we expected. It only 2ms less than version 1. And we think its bad performance is owing to parallelism driven by GPU, which is thread-level not block-level. Besides, the way dividing total sum into an array needs more branch to detect when to sum up.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

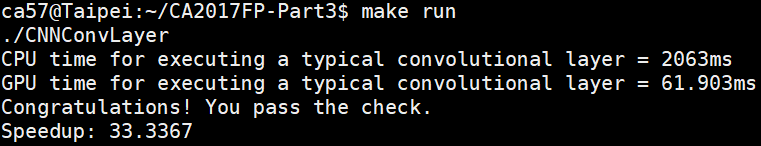
*version 1*

**

*version 2*

**

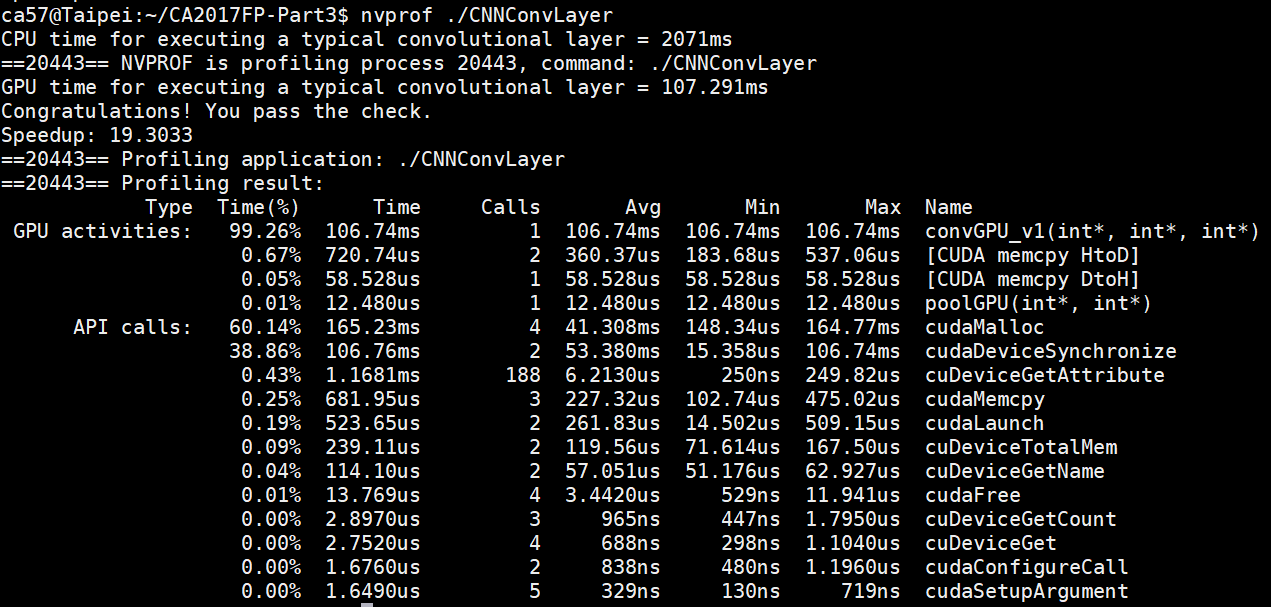
*version 3*

**

**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.

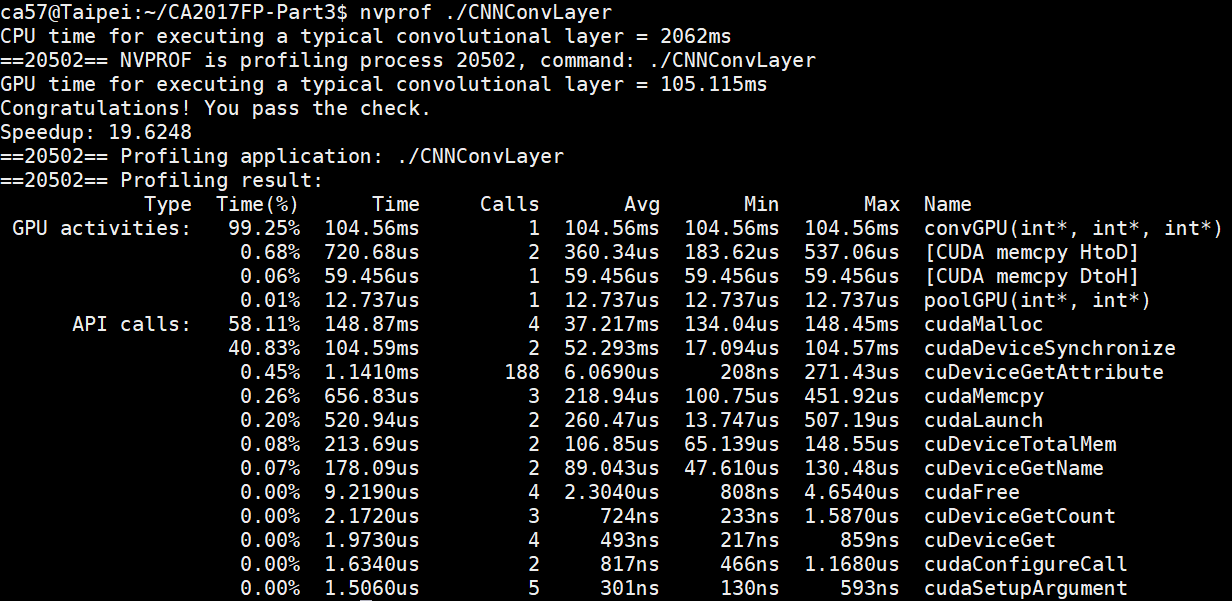
**4.** **Show how you use NVVP to help you find and solve perf(5%)**

*version1 nvprof*



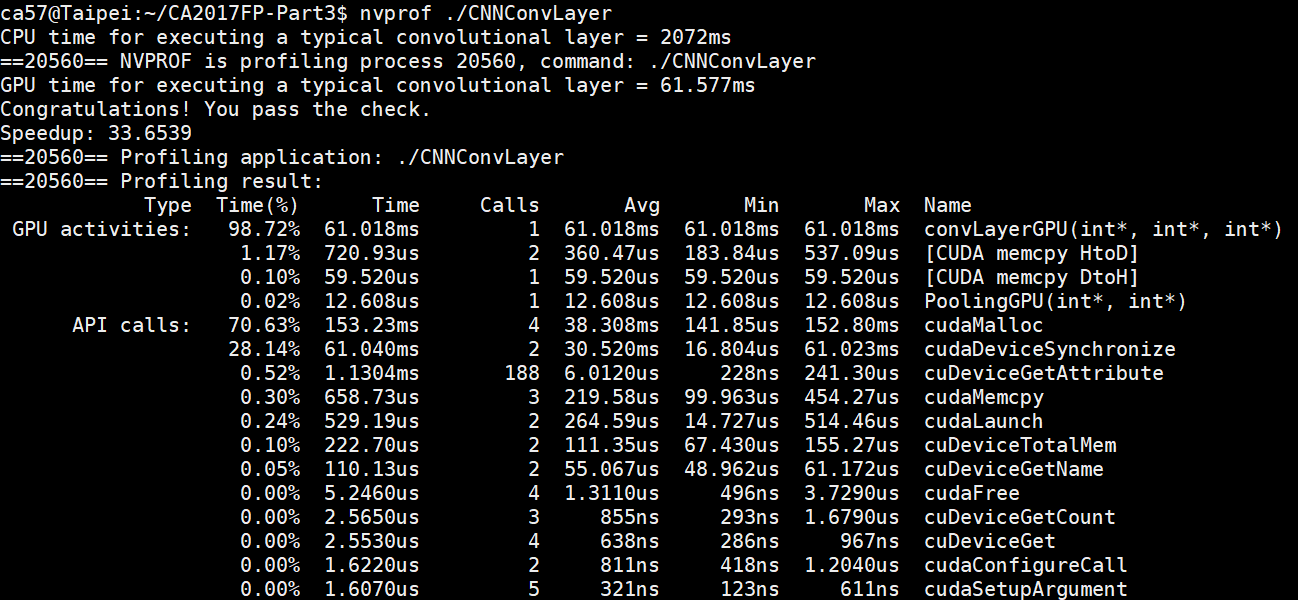
We can tell that convGPU spent the most time in this picture, so we try to improve the function with cut it into more blocks so that it doesn’t need to run so many for loops.

*version2 nvprof*

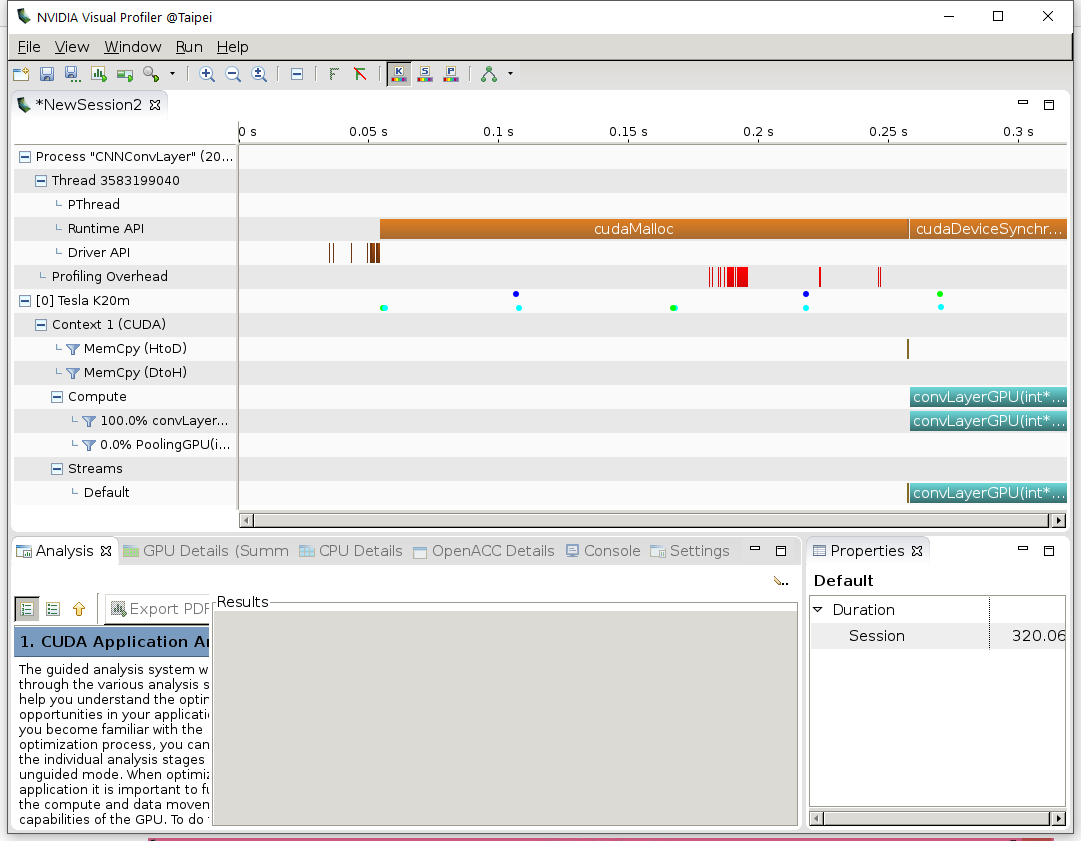
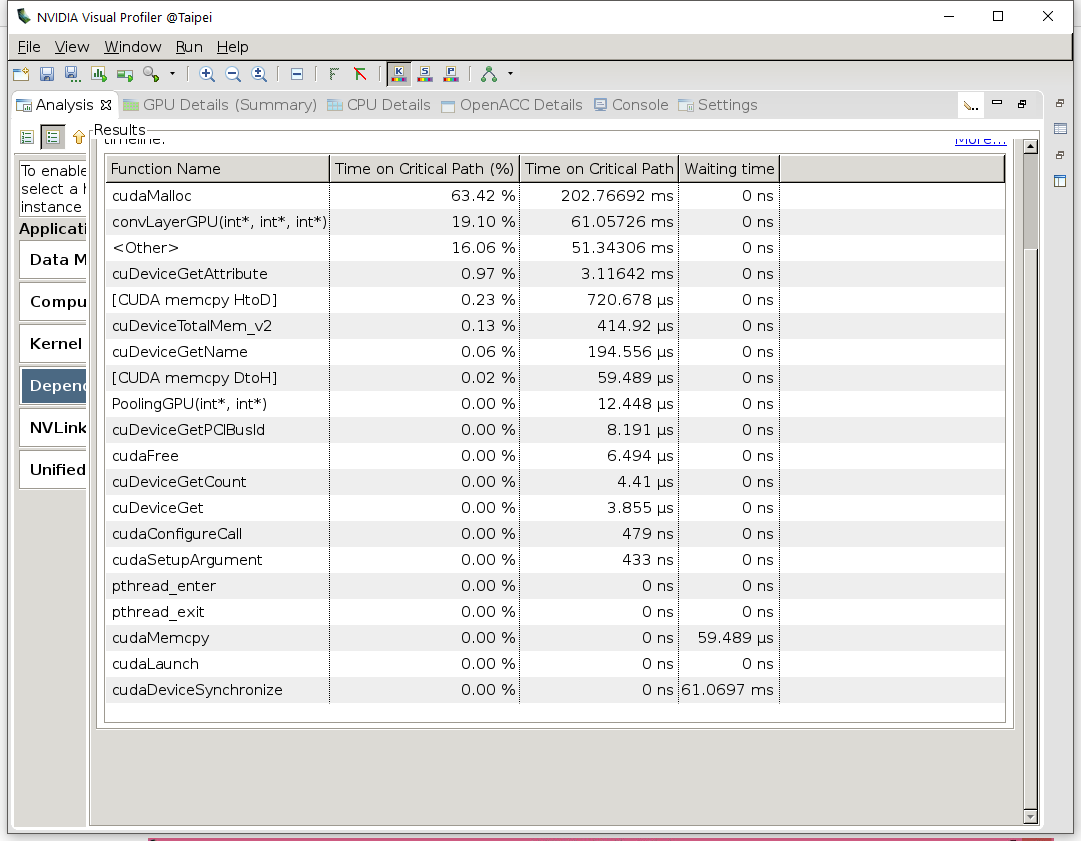


It didn’t be improved so much, so we keep dividing the for loops.

*version3 nvprof*

**

*version3 mvvp*

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.