Korea Advanced Institute of Science and Technology

School of Electrical Engineering

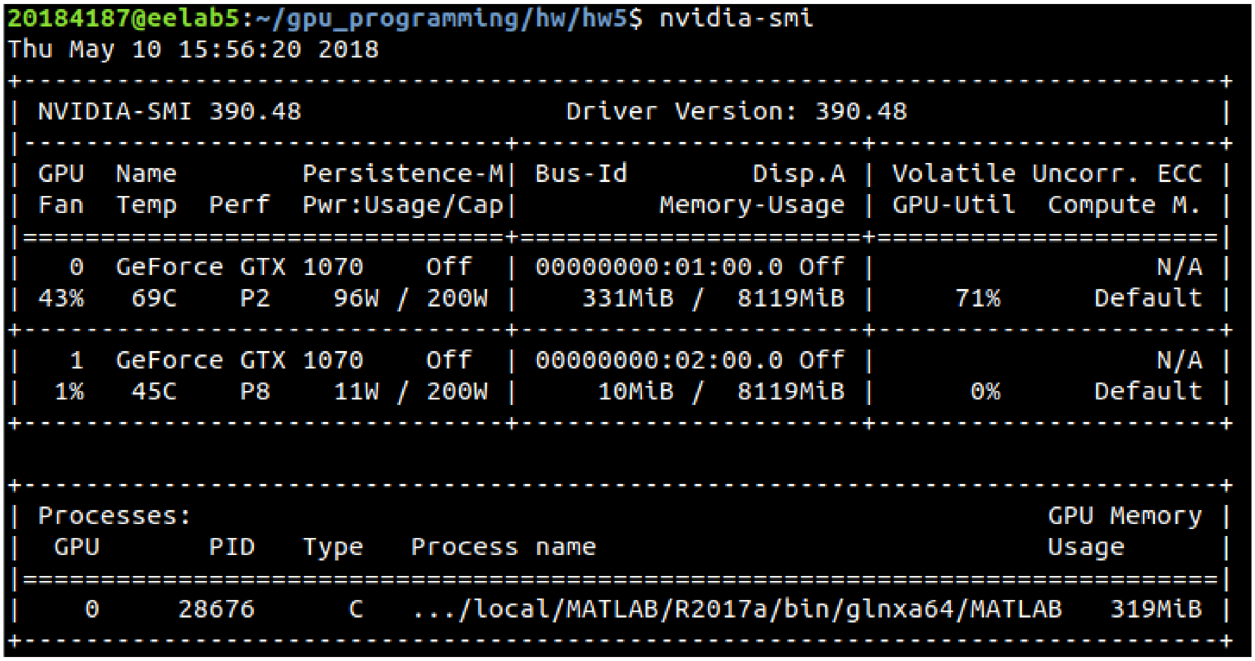
EE817 GPU Programming and Its Applications Spring 2018

Student’s Name: Dinh Vu

Student’s ID: 20184187

Homework 5

The computer, used in my homework 4, contains NVIDIA GeForce GT 1070 based on Pascal GP104 architecture.



```
20184187@eelab5:~/gpu_programming/hw/hw5$ nvidia-smi
Thu May 10 15:56:20 2018
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 390.48                 Driver Version: 390.48                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GeForce GTX 1070    Off  | 00000000:01:00.0 Off |                  N/A |
| 43%   69C    P2    96W / 200W |    331MiB /  8119MiB |     71%      Default |
+-------------------------------+----------------------+----------------------+
|   1  GeForce GTX 1070    Off  | 00000000:02:00.0 Off |                  N/A |
|  1%   45C    P8    11W / 200W |     10MiB /  8119MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0     28676      C   .../local/MATLAB/R2017a/bin/glnxa64/MATLAB   319MiB |
+-----------------------------------------------------------------------------+
```

Figure 1.1. Graphic card information

1. Without Stream

The source code for matrix multiplication using only global memory without stream is `matrixMulGmem.cu` file.

Each thread computes value of each element in matrix C. Each row of matrix A and the corresponding column of matrix B is read from global memory. Then each element in matrix C with thread index is calculated parallel. This programming strategy is presented in Figure 1.2.

Figure 1.3 shows the procedure of the program while Figure 1.4 displays the summary of the execution time.

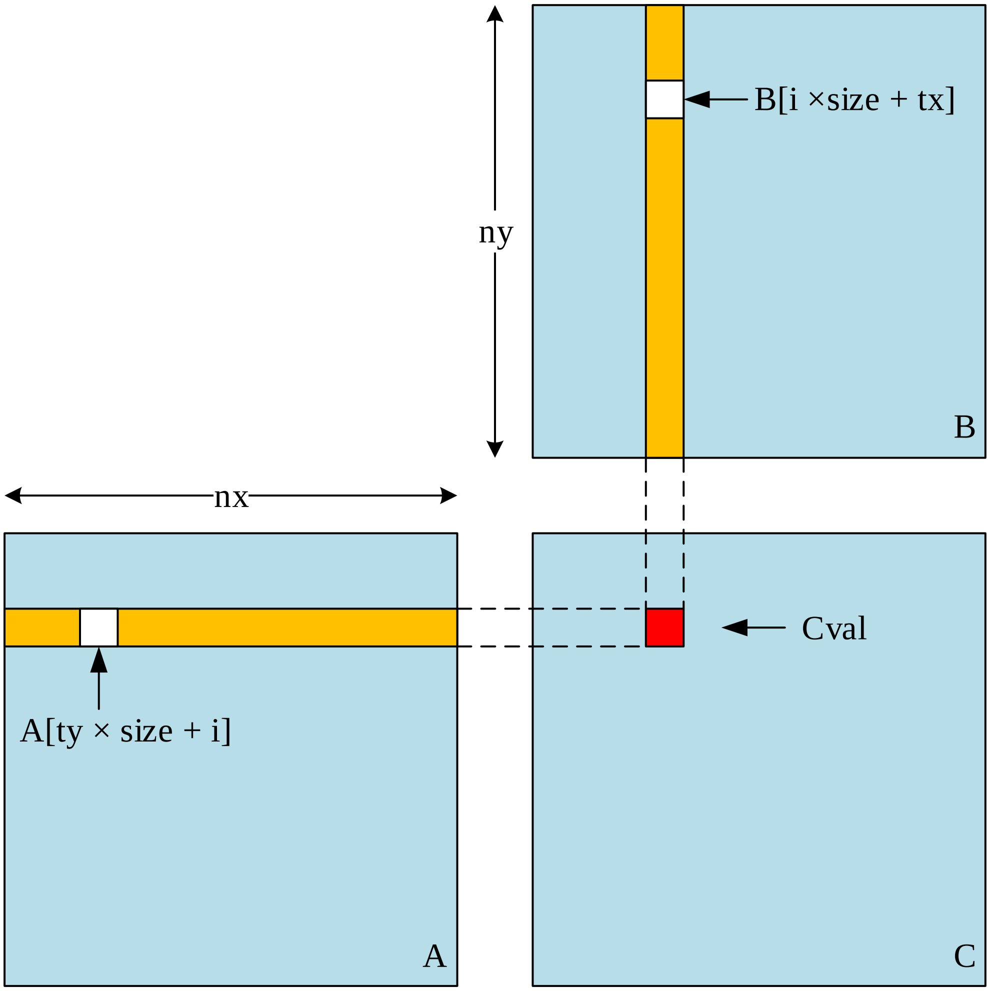
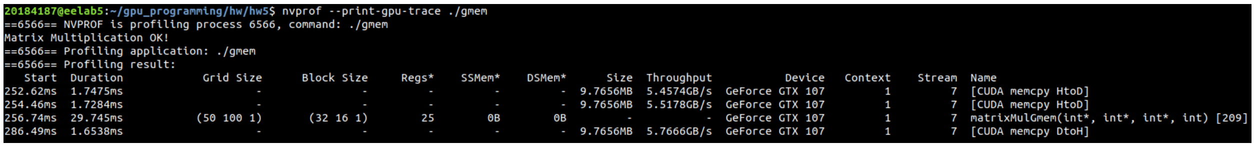
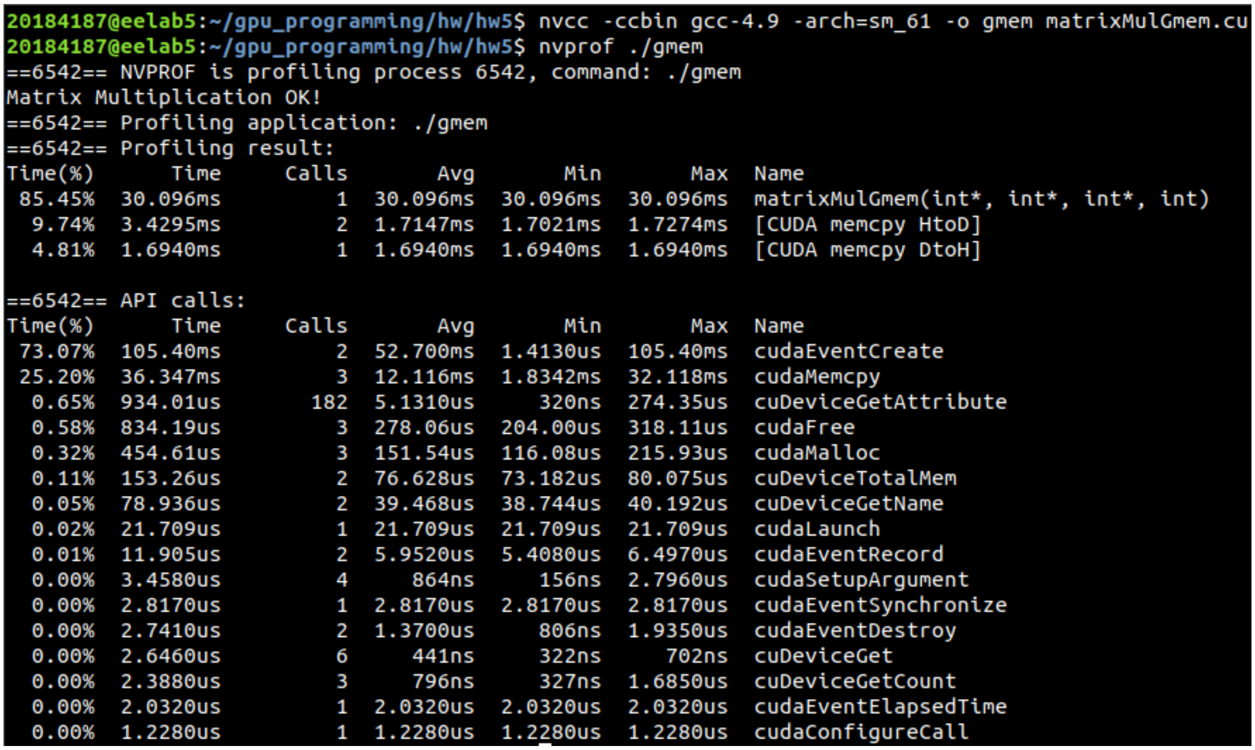


Figure 1.2. Matrix multiplication only using global memory



```
20184187@eelab5:~/gpu_programming/hw/hw5$ nvprof --print-gpu-trace ./gmem
==6566== NVPROF is profiling process 6566, command: ./gmem
Matrix Multiplication OK!
==6566== Profiling application: ./gmem
==6566== Profiling result:
   Start  Duration            Grid Size      Block Size     Regs*    SSMem*    DSMem*      Size  Throughput           Device   Context    Stream  Name
252.62ms  1.7475ms                    -               -         -         -         -  9.7656MB  5.4574GB/s  GeForce GTX 107         1         7  [CUDA memcpy HtoD]
254.46ms  1.7284ms                    -               -         -         -         -  9.7656MB  5.5178GB/s  GeForce GTX 107         1         7  [CUDA memcpy HtoD]
256.74ms  29.745ms           (50 100 1)       (32 16 1)        25        0B        0B         -           -  GeForce GTX 107         1         7  matrixMulGmem(int*, int*, int*, int) [209]
286.49ms  1.6538ms                    -               -         -         -         -  9.7656MB  5.7666GB/s  GeForce GTX 107         1         7  [CUDA memcpy DtoH]
```

Figure 1.3. The procedure of the program using only global memory



```
20184187@eelab5:~/gpu_programming/hw/hw5$ nvcc -ccbin gcc-4.9 -arch=sm_61 -o gmem matrixMulGmem.cu
20184187@eelab5:~/gpu_programming/hw/hw5$ nvprof ./gmem
==6542== NVPROF is profiling process 6542, command: ./gmem
Matrix Multiplication OK!
==6542== Profiling application: ./gmem
==6542== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 85.45%  30.096ms         1  30.096ms  30.096ms  30.096ms  matrixMulGmem(int*, int*, int*, int)
  9.74%  3.4295ms         2  1.7147ms  1.7021ms  1.7274ms  [CUDA memcpy HtoD]
  4.81%  1.6940ms         1  1.6940ms  1.6940ms  1.6940ms  [CUDA memcpy DtoH]

==6542== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 73.07%  105.40ms         2  52.700ms  1.4130us  105.40ms  cudaEventCreate
 25.20%  36.347ms         3  12.116ms  1.8342ms  32.118ms  cudaMemcpy
  0.65%  934.01us       182  5.1310us     320ns  274.35us  cuDeviceGetAttribute
  0.58%  834.19us         3  278.06us  204.00us  318.11us  cudaFree
  0.32%  454.61us         3  151.54us  116.08us  215.93us  cudaMalloc
  0.11%  153.26us         2  76.628us  73.182us  80.075us  cuDeviceTotalMem
  0.05%  78.936us         2  39.468us  38.744us  40.192us  cuDeviceGetName
  0.02%  21.709us         1  21.709us  21.709us  21.709us  cudaLaunch
  0.01%  11.905us         2  5.9520us  5.4080us  6.4970us  cudaEventRecord
  0.00%  3.4580us         4     864ns     156ns  2.7960us  cudaSetupArgument
  0.00%  2.8170us         1  2.8170us  2.8170us  2.8170us  cudaEventSynchronize
  0.00%  2.7410us         2  1.3700us     806ns  1.9350us  cudaEventDestroy
  0.00%  2.6460us         6     441ns     322ns     702ns  cuDeviceGet
  0.00%  2.3880us         3     796ns     327ns  1.6850us  cuDeviceGetCount
  0.00%  2.0320us         1  2.0320us  2.0320us  2.0320us  cudaEventElapsedTime
  0.00%  1.2280us         1  1.2280us  1.2280us  1.2280us  cudaConfigureCall
```

Figure 1.4. The execution time of the matrix multiplication using only global memory

2. Depth-first

The `matrixMulDepth.cu` is source code for matrix multiplication overlapping kernel execution with depth-first data transfer. Because I remote the computer in Haedong Lounge, **nvvp** cannot display the profile of my program. Therefore, instead using **nvvp**, the option `--print-gpu-trace` of **nvprof** command is used to display the sequence of streams.

Matrix B is still load entirely from global memory. Because the required number of streams is 8, matrix A is divided to 8 slides in order to split into each stream. Each stream will read each slide A[i] from global memory then calculate the output slide C[i], where i = 0, 1, 2,…, 8. Hence, each slide of the output matrix C also is computed following each stream as Figure 2.1.

All streams using the same kernel `matrixMulDepth`. The programming strategy of kernel `matrixMulDepth` is presented in Figure 2.2.

The order operation of the streams and the summary of execution time of each function are shown in Figure 2.3 and Figure 2.4, respectively.

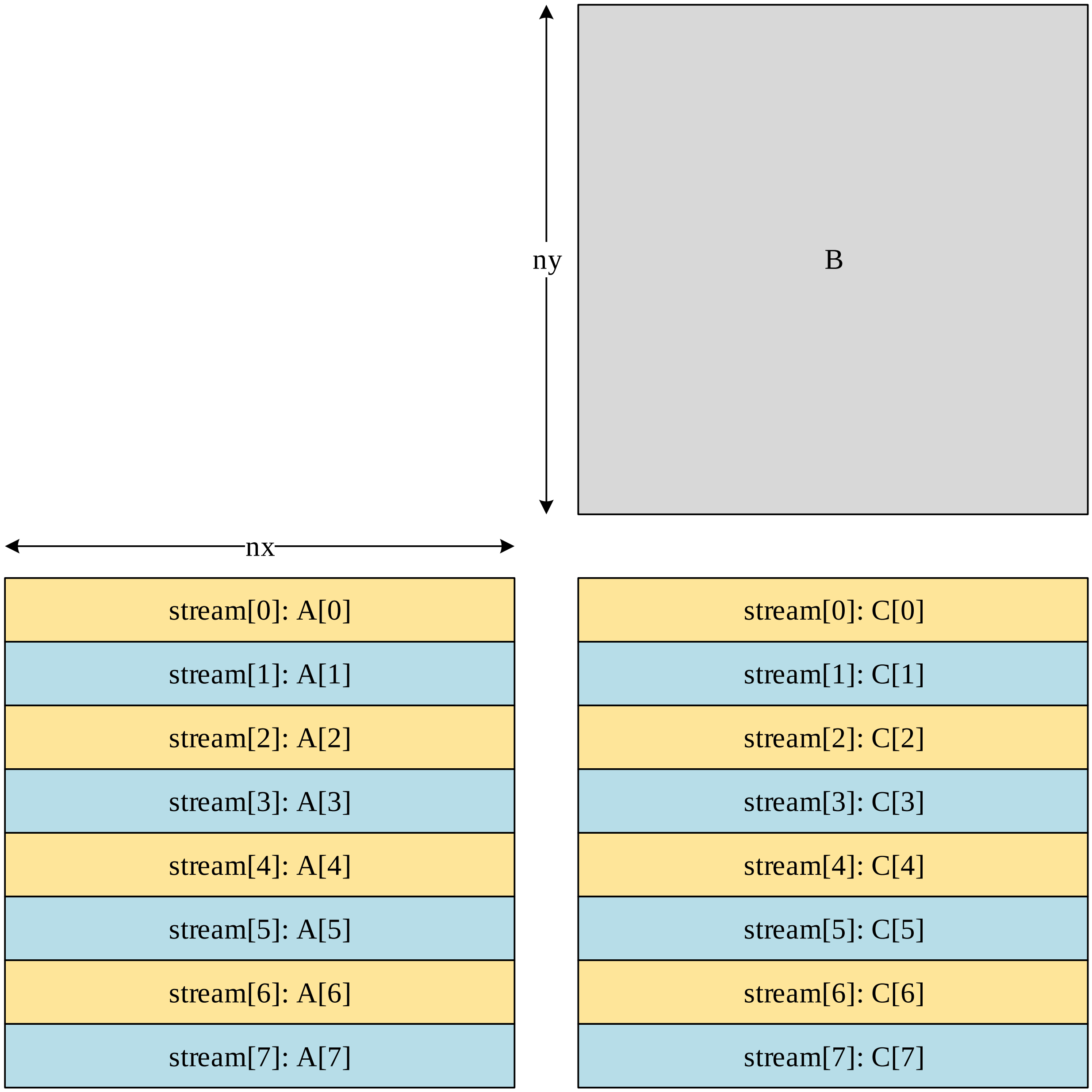


Figure 2.1. The programming strategy for matrix multiplication using streams

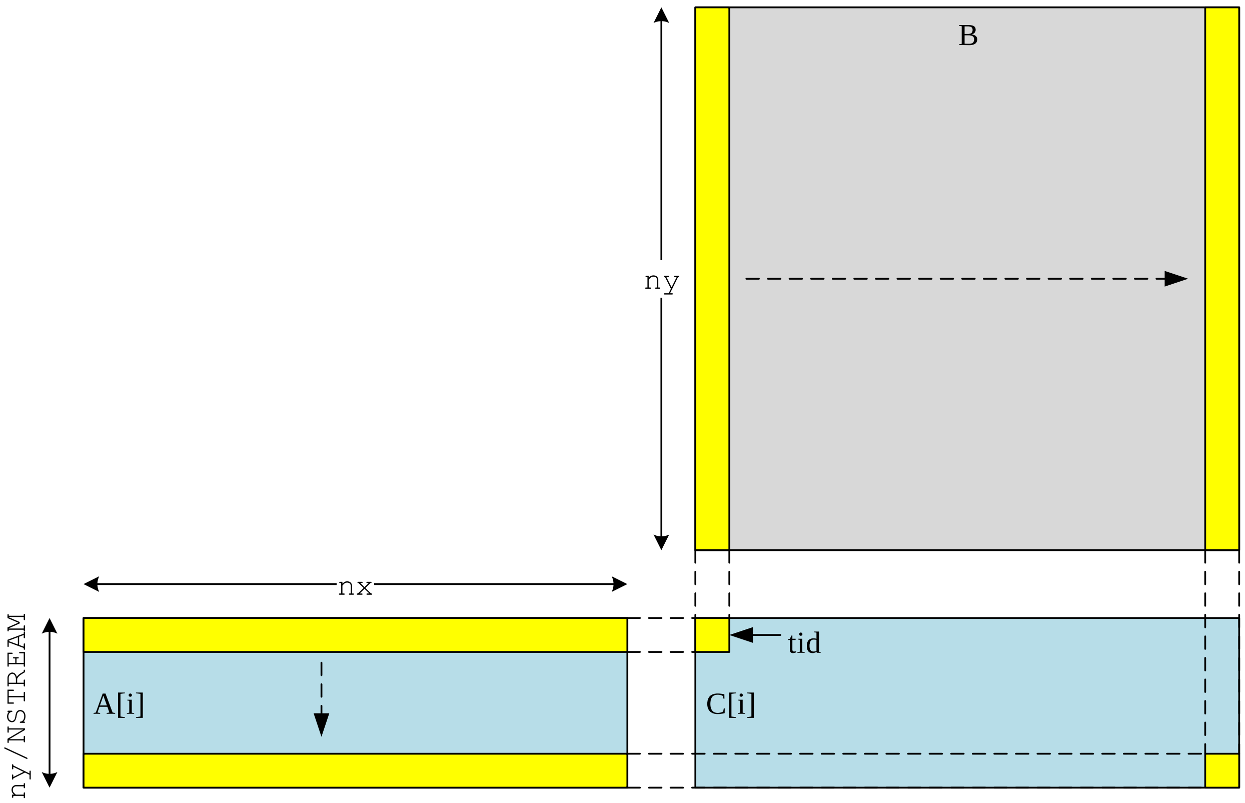
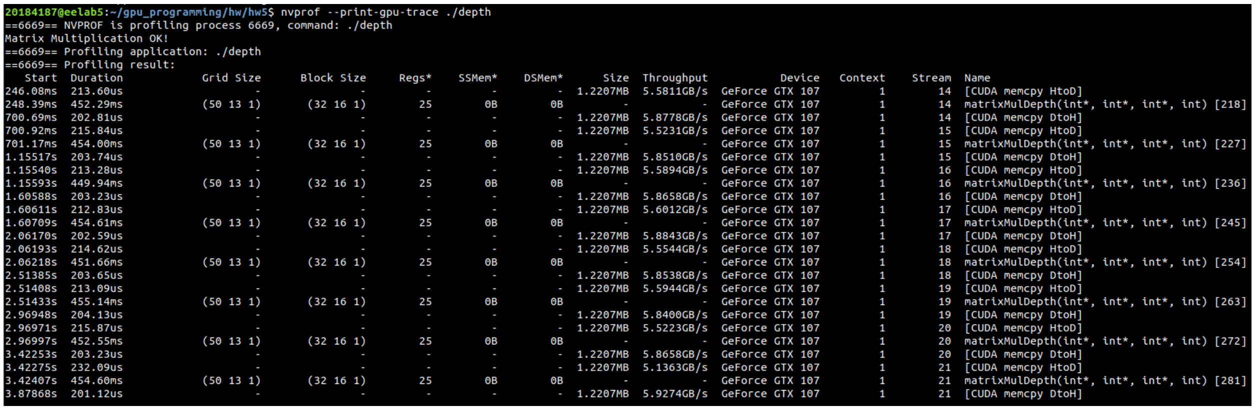
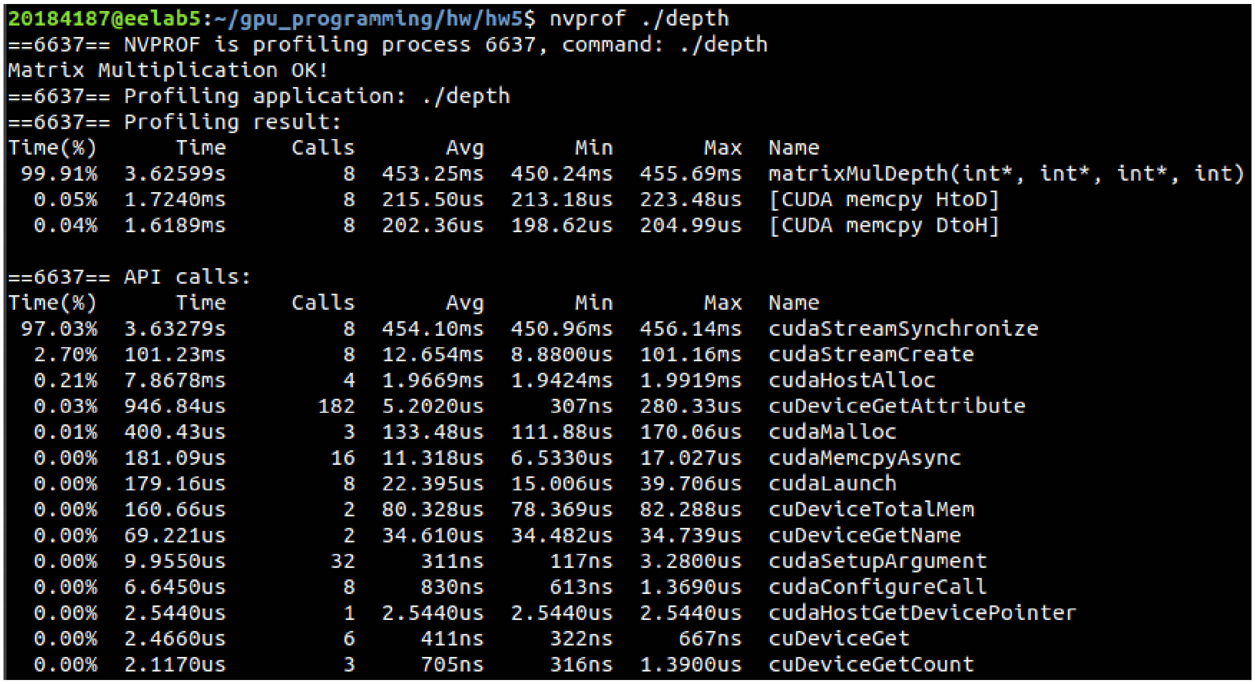


Figure 2.2. The operation of kernel `matrixMulDepth`



```
20184187@eelab5:~/gpu_programming/hw/hw5$ nvprof --print-gpu-trace ./depth
==6669== NVPROF is profiling process 6669, command: ./depth
Matrix Multiplication OK!
==6669== Profiling application: ./depth
==6669== Profiling result:
   Start  Duration            Grid Size      Block Size     Regs*    SSMem*    DSMem*      Size  Throughput           Device   Context    Stream  Name
246.08ms  213.60us                    -               -         -         -         -  1.2207MB  5.5811GB/s  GeForce GTX 107         1        14  [CUDA memcpy HtoD]
248.39ms  452.29ms            (50 13 1)       (32 16 1)        25        0B        0B         -           -  GeForce GTX 107         1        14  matrixMulDepth(int*, int*, int*, int) [218]
700.69ms  202.81us                    -               -         -         -         -  1.2207MB  5.8778GB/s  GeForce GTX 107         1        14  [CUDA memcpy DtoH]
700.92ms  215.84us                    -               -         -         -         -  1.2207MB  5.5231GB/s  GeForce GTX 107         1        15  [CUDA memcpy HtoD]
701.17ms  454.00ms            (50 13 1)       (32 16 1)        25        0B        0B         -           -  GeForce GTX 107         1        15  matrixMulDepth(int*, int*, int*, int) [227]
1.15517s  203.74us                    -               -         -         -         -  1.2207MB  5.8510GB/s  GeForce GTX 107         1        15  [CUDA memcpy DtoH]
1.15540s  213.28us                    -               -         -         -         -  1.2207MB  5.5894GB/s  GeForce GTX 107         1        16  [CUDA memcpy HtoD]
1.15593s  449.94ms            (50 13 1)       (32 16 1)        25        0B        0B         -           -  GeForce GTX 107         1        16  matrixMulDepth(int*, int*, int*, int) [236]
1.60588s  203.23us                    -               -         -         -         -  1.2207MB  5.8658GB/s  GeForce GTX 107         1        16  [CUDA memcpy DtoH]
1.60611s  212.83us                    -               -         -         -         -  1.2207MB  5.6012GB/s  GeForce GTX 107         1        17  [CUDA memcpy HtoD]
1.60709s  454.61ms            (50 13 1)       (32 16 1)        25        0B        0B         -           -  GeForce GTX 107         1        17  matrixMulDepth(int*, int*, int*, int) [245]
2.06170s  202.59us                    -               -         -         -         -  1.2207MB  5.8843GB/s  GeForce GTX 107         1        17  [CUDA memcpy DtoH]
2.06193s  214.62us                    -               -         -         -         -  1.2207MB  5.5544GB/s  GeForce GTX 107         1        18  [CUDA memcpy HtoD]
2.06218s  451.66ms            (50 13 1)       (32 16 1)        25        0B        0B         -           -  GeForce GTX 107         1        18  matrixMulDepth(int*, int*, int*, int) [254]
2.51385s  203.65us                    -               -         -         -         -  1.2207MB  5.8538GB/s  GeForce GTX 107         1        18  [CUDA memcpy DtoH]
2.51408s  213.09us                    -               -         -         -         -  1.2207MB  5.5944GB/s  GeForce GTX 107         1        19  [CUDA memcpy HtoD]
2.51433s  455.14ms            (50 13 1)       (32 16 1)        25        0B        0B         -           -  GeForce GTX 107         1        19  matrixMulDepth(int*, int*, int*, int) [263]
2.96948s  204.13us                    -               -         -         -         -  1.2207MB  5.8400GB/s  GeForce GTX 107         1        19  [CUDA memcpy DtoH]
2.96971s  215.87us                    -               -         -         -         -  1.2207MB  5.5223GB/s  GeForce GTX 107         1        20  [CUDA memcpy HtoD]
2.96997s  452.55ms            (50 13 1)       (32 16 1)        25        0B        0B         -           -  GeForce GTX 107         1        20  matrixMulDepth(int*, int*, int*, int) [272]
3.42253s  203.23us                    -               -         -         -         -  1.2207MB  5.8658GB/s  GeForce GTX 107         1        20  [CUDA memcpy DtoH]
3.42275s  232.09us                    -               -         -         -         -  1.2207MB  5.1363GB/s  GeForce GTX 107         1        21  [CUDA memcpy HtoD]
3.42407s  454.60ms            (50 13 1)       (32 16 1)        25        0B        0B         -           -  GeForce GTX 107         1        21  matrixMulDepth(int*, int*, int*, int) [281]
3.87868s  201.12us                    -               -         -         -         -  1.2207MB  5.9274GB/s  GeForce GTX 107         1        21  [CUDA memcpy DtoH]
```

Figure 2.3. The procedure of streams in matrix multiplication with depth-approach



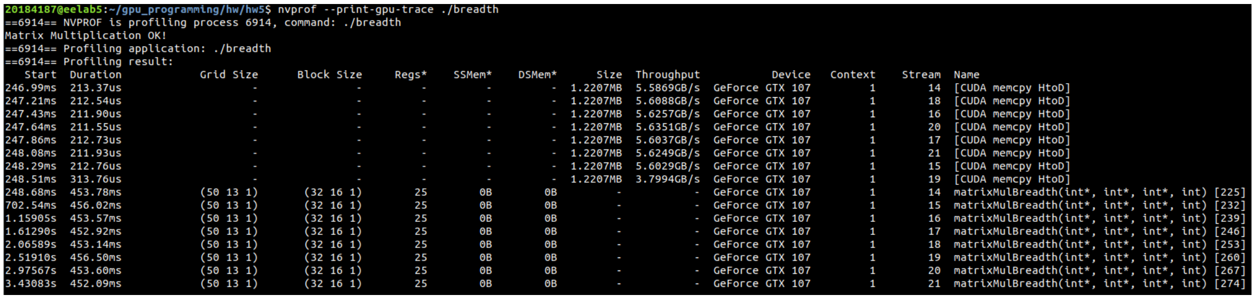
```
20184187@eelab5:~/gpu_programming/hw/hw5$ nvprof ./depth
==6637== NVPROF is profiling process 6637, command: ./depth
Matrix Multiplication OK!
==6637== Profiling application: ./depth
==6637== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 99.91%  3.62599s         8  453.25ms  450.24ms  455.69ms  matrixMulDepth(int*, int*, int*, int)
  0.05%  1.7240ms         8  215.50us  213.18us  223.48us  [CUDA memcpy HtoD]
  0.04%  1.6189ms         8  202.36us  198.62us  204.99us  [CUDA memcpy DtoH]

==6637== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 97.03%  3.63279s         8  454.10ms  450.96ms  456.14ms  cudaStreamSynchronize
  2.70%  101.23ms         8  12.654ms  8.8800us  101.16ms  cudaStreamCreate
  0.21%  7.8678ms         4  1.9669ms  1.9424ms  1.9919ms  cudaHostAlloc
  0.03%  946.84us       182  5.2020us     307ns  280.33us  cuDeviceGetAttribute
  0.01%  400.43us         3  133.48us  111.88us  170.06us  cudaMalloc
  0.00%  181.09us        16  11.318us  6.5330us  17.027us  cudaMemcpyAsync
  0.00%  179.16us         8  22.395us  15.006us  39.706us  cudaLaunch
  0.00%  160.66us         2  80.328us  78.369us  82.288us  cuDeviceTotalMem
  0.00%  69.221us         2  34.610us  34.482us  34.739us  cuDeviceGetName
  0.00%  9.9550us        32     311ns     117ns  3.2800us  cudaSetupArgument
  0.00%  6.6450us         8     830ns     613ns  1.3690us  cudaConfigureCall
  0.00%  2.5440us         1  2.5440us  2.5440us  2.5440us  cudaHostGetDevicePointer
  0.00%  2.4660us         6     411ns     322ns     667ns  cuDeviceGet
  0.00%  2.1170us         3     705ns     316ns  1.3900us  cuDeviceGetCount
```

Figure 2.4. Summary of the activities on GPU of matrix multiplication with depth-approach

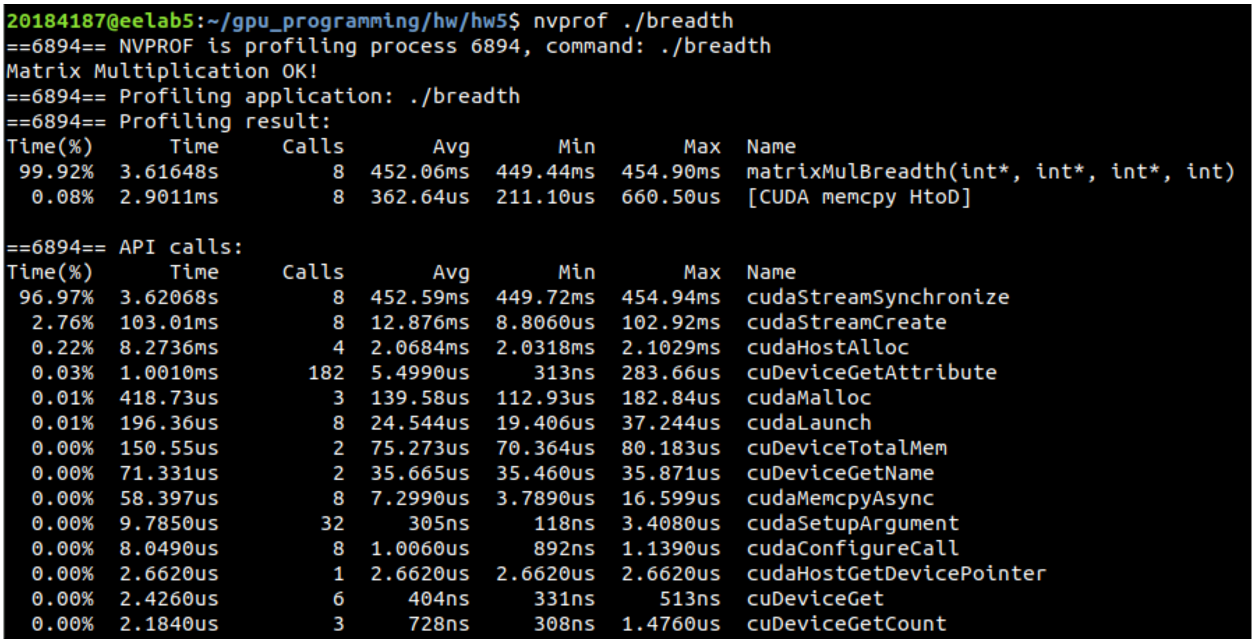
3. Breadth-first

The source code for matrix multiplication using breadth-first data transfer is matrixMulBreadth.cu. The programming strategy is similar to the depth-first. The only difference is that each activity copy data from host to device, kernel and copy result from device to host have each for-loop.



```
20184187@eelab5:~/gpu_programming/hw/hw5$ nvprof --print-gpu-trace ./breadth
==6914== NVPROF is profiling process 6914, command: ./breadth
Matrix Multiplication OK!
==6914== Profiling application: ./breadth
==6914== Profiling result:
   Start  Duration            Grid Size      Block Size     Regs*    SSMem*    DSMem*      Size  Throughput           Device   Context    Stream  Name
246.99ms  213.37us                    -               -         -         -         -  1.2207MB  5.5869GB/s  GeForce GTX 107         1        14  [CUDA memcpy HtoD]
247.21ms  212.54us                    -               -         -         -         -  1.2207MB  5.6088GB/s  GeForce GTX 107         1        18  [CUDA memcpy HtoD]
247.43ms  211.90us                    -               -         -         -         -  1.2207MB  5.6257GB/s  GeForce GTX 107         1        16  [CUDA memcpy HtoD]
247.64ms  211.55us                    -               -         -         -         -  1.2207MB  5.6351GB/s  GeForce GTX 107         1        20  [CUDA memcpy HtoD]
247.86ms  212.73us                    -               -         -         -         -  1.2207MB  5.6037GB/s  GeForce GTX 107         1        17  [CUDA memcpy HtoD]
248.08ms  211.93us                    -               -         -         -         -  1.2207MB  5.6249GB/s  GeForce GTX 107         1        21  [CUDA memcpy HtoD]
248.29ms  212.76us                    -               -         -         -         -  1.2207MB  5.6029GB/s  GeForce GTX 107         1        15  [CUDA memcpy HtoD]
248.51ms  313.76us                    -               -         -         -         -  1.2207MB  3.7994GB/s  GeForce GTX 107         1        19  [CUDA memcpy HtoD]
248.68ms  453.78ms          (50 13 1)       (32 16 1)        25        0B        0B         -           -  GeForce GTX 107         1        14  matrixMulBreadth(int*, int*, int*, int) [225]
702.54ms  456.02ms          (50 13 1)       (32 16 1)        25        0B        0B         -           -  GeForce GTX 107         1        15  matrixMulBreadth(int*, int*, int*, int) [232]
1.15905s  453.57ms          (50 13 1)       (32 16 1)        25        0B        0B         -           -  GeForce GTX 107         1        16  matrixMulBreadth(int*, int*, int*, int) [239]
1.61290s  452.92ms          (50 13 1)       (32 16 1)        25        0B        0B         -           -  GeForce GTX 107         1        17  matrixMulBreadth(int*, int*, int*, int) [246]
2.06589s  453.14ms          (50 13 1)       (32 16 1)        25        0B        0B         -           -  GeForce GTX 107         1        18  matrixMulBreadth(int*, int*, int*, int) [253]
2.51910s  456.50ms          (50 13 1)       (32 16 1)        25        0B        0B         -           -  GeForce GTX 107         1        19  matrixMulBreadth(int*, int*, int*, int) [260]
2.97567s  453.60ms          (50 13 1)       (32 16 1)        25        0B        0B         -           -  GeForce GTX 107         1        20  matrixMulBreadth(int*, int*, int*, int) [267]
3.43083s  452.09ms          (50 13 1)       (32 16 1)        25        0B        0B         -           -  GeForce GTX 107         1        21  matrixMulBreadth(int*, int*, int*, int) [274]
```

Figure 3.1. The procedure of streams in matrix multiplication with bread-approach



```
20184187@eelab5:~/gpu_programming/hw/hw5$ nvprof ./breadth
==6894== NVPROF is profiling process 6894, command: ./breadth
Matrix Multiplication OK!
==6894== Profiling application: ./breadth
==6894== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 99.92%  3.61648s         8  452.06ms  449.44ms  454.90ms  matrixMulBreadth(int*, int*, int*, int)
  0.08%  2.9011ms         8  362.64us  211.10us  660.50us  [CUDA memcpy HtoD]

==6894== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 96.97%  3.62068s         8  452.59ms  449.72ms  454.94ms  cudaStreamSynchronize
  2.76%  103.01ms         8  12.876ms  8.8060us  102.92ms  cudaStreamCreate
  0.22%  8.2736ms         4  2.0684ms  2.0318ms  2.1029ms  cudaHostAlloc
  0.03%  1.0010ms       182  5.4990us     313ns  283.66us  cuDeviceGetAttribute
  0.01%  418.73us         3  139.58us  112.93us  182.84us  cudaMalloc
  0.01%  196.36us         8  24.544us  19.406us  37.244us  cudaLaunch
  0.00%  150.55us         2  75.273us  70.364us  80.183us  cuDeviceTotalMem
  0.00%  71.331us         2  35.665us  35.460us  35.871us  cuDeviceGetName
  0.00%  58.397us         8  7.2990us  3.7890us  16.599us  cudaMemcpyAsync
  0.00%  9.7850us        32     305ns     118ns  3.4080us  cudaSetupArgument
  0.00%  8.0490us         8  1.0060us     892ns  1.1390us  cudaConfigureCall
  0.00%  2.6620us         1  2.6620us  2.6620us  2.6620us  cudaHostGetDevicePointer
  0.00%  2.4260us         6     404ns     331ns     513ns  cuDeviceGet
  0.00%  2.1840us         3     728ns     308ns  1.4760us  cuDeviceGetCount
```

Figure 3.2. Summary of execution time for each activity on GPU processing matrix multiplication with breadth-approach