Milestone 1 Report

Team: hybrid

Members:

Enyi Jiang(enyij2) rai_id: 5c78c48284318364bab96eaf

Xi Chen(xichen30) rai_id: 5c78c40d84318363f903701c

Yifan Chen(yifanc3) rai_id: 5c78c40e84318363f903701d

School Affiliation: On Campus

- Milestone 1

❏ Include a list of all kernels that collectively consume more than 90% of the program time.

CUDA memcpy HtoD

cudnn::detail::implicit_convolve_sgemm

volta_cgemm_64x32_tn

op_generic_tensor_kernel

fft2d_c2r_32x32

volta_sgemm_128x128_tn

void cudnn::detail::pooling_fw_4d_kernel

fft2d_r2c_32x32

❏ Include a list of all CUDA API calls that collectively consume more than 90% of the program time.

cudaStreamCreateWithFlag

cudaMemGetInfo

cudaFree

❏ Include an explanation of the difference between kernels and API calls

- ❏ Kernels are C functions which are flagged to be run on a GPU (or a device). It is a more low-level program that instructs the performance of each thread. Kernels have no APIs as they are not libraries.
- ❏ API calls are C function calls that executed by the host (CPU) to back up kernel codes, including transferring data, managing memory and so on. API calls certainly use some C libraries.

❏ Show output of rai running MXNet on the CPU

- ❏ EvalMetric: {'accuracy': 0.8236}

  9.29 user 3.73 system 0:05.39 elapsed 241%CPU (0avgtext+0avgdata 2471196maxresident)k

  0inputs+2824outputs (0major+668118minor)pagefaults 0swap

Screenshot of the Output:

```
Successfully installed mxnet
* Running /usr/bin/time python m1.1.py
Loading fashion-mnist data... done
Loading model... done
New Inference
EvalMetric: {'accuracy': 0.8236}
9.29user 3.73system 0:05.39elapsed 241%CPU (0avgtext+0avgdata 2471196maxresident)k
0inputs+2824outputs (0major+668118minor)pagefaults 0swap
s
```

❏ List program run time

- ❏ 9.29 - user 3.73 - system 0:05.39 elapsed 241%CPU

❏ Show output of rai running MXNet on the GPU

- ❏ Eval

```
* Running /usr/bin/time python m3.1.py 1000
Loading fashion-mnist data... done
Loading model... done
New Inference
Op Time: 0.000637
Op Time: 0.003675
Correctness: 0.852 Model: ece408
4.20user 3.33system 0:04.20elapsed 179%CPU (
0avgtext+0avgdata 2759512maxresident)k
0inputs+4584outputs (0major+618929minor)pagefaults 0swaps
```

  Metric: {'accuracy': 0.8236}

Screenshot of the output:

```
* Running /usr/bin/time python m1.2.py
Loading fashion-mnist data... done
Loading model... done
New Inference
EvalMetric: {'accuracy': 0.8236}
4.50user 3.43system 0:10.74elapsed 73%CPU (0avgtext+0avgdata 2843392maxresident)k
0inputs+4552outputs (0
major+660536minor)pagefaults 0swaps
```

```
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   46.51%  21.659ms        20  1.0829ms  1.0880us  20.991ms  [CUDA memcpy HtoD]
                   18.23%  8.4911ms         1  8.4911ms  8.4911ms  8.4911ms  void cudnn::detail::implicit_convolve_sgemm<float, float, int=1024, int=5, int=5, int=3, int=3, int=3, i
nt=1, bool=1, bool=0, bool=1>(int, int, int, float const *, int, float*, cudnn::detail::implicit_convolve_sgemm<float, float, int=1024, int=5, int=5, int=3, int=3, int=3, int=1, boo
l=1, bool=0, bool=1>*, kernel_conv_params, int, float, float, int, float, float, int, int)
                   10.68%  4.9752ms         1  4.9752ms  4.9752ms  4.9752ms  volta_cgemm_64x32_tn
                    6.29%  2.9295ms         2  1.4647ms  25.088us  2.9044ms  void op_generic_tensor_kernel<int=2, float, float, float, int=256, cudnnGenericOp_t=7, cudnnNanPropagati
on_t=0, cudnnDimOrder_t=0, int=1>(cudnnTensorStruct, float*, cudnnTensorStruct, float const *, cudnnTensorStruct, float const *, float, float, float, float, dimArray, reducedDivisor
Array)
                    5.06%  2.3587ms         1  2.3587ms  2.3587ms  2.3587ms  volta_sgemm_128x128_tn
                    5.06%  2.3565ms         1  2.3565ms  2.3565ms  2.3565ms  void fft2d_c2r_32x32<float, bool=0, bool=0, unsigned int=1, bool=0, bool=0>(float*, float2 const *, int,
 int, int, int, int, int, int, int, int, float, float, cudnn::reduced_divisor, bool, float*, float*, int2, int, int)
                    4.05%  1.8849ms         1  1.8849ms  1.8849ms  1.8849ms  void cudnn::detail::pooling_fw_4d_kernel<float, float, cudnn::detail::maxpooling_func<float, cudnnNanPro
pagation_t=0>, int=0, bool=0>(cudnnTensorStruct, float const *, cudnn::detail::pooling_fw_4d_kernel<float, float, cudnn::detail::maxpooling_func<float, cudnnNanPropagation_t=0>, int
=0, bool=0>, cudnnTensorStruct*, cudnnPoolingStruct, float, cudnnPoolingStruct, int, cudnn::reduced_divisor, float)
                    3.35%  1.5583ms         1  1.5583ms  1.5583ms  1.5583ms  void fft2d_r2c_32x32<float, bool=0, unsigned int=0, bool=0>(float2*, float const *, int, int, int, int,
 int, int, int, int, int, cudnn::reduced_divisor, bool, int2, int, int)
                    0.33%  154.91us         1  154.91us  154.91us  154.91us  void mshadow::cuda::MapPlanLargeKernel<mshadow::sv::saveto, int=8, int=1024, mshadow::expr::Plan<mshadow
::Tensor<mshadow::gpu, int=2, float>, float>, mshadow::expr::Plan<mshadow::expr::ScalarExp<float>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=2, int)
                    0.16%  76.159us         1  76.159us  76.159us  76.159us  void mshadow::cuda::SoftmaxKernel<int=8, float, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2,
 float>, float>, mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=2, float>, float>>(mshadow::gpu, int=2, unsigned int)
                    0.07%  30.367us        13  2.3350us  1.2160us  7.5840us  void mshadow::cuda::MapPlanKernel<mshadow::sv::saveto, int=8, mshadow::expr::Plan<mshadow::Tensor<mshado
w::gpu, int=2, float>, float>, mshadow::expr::Plan<mshadow::expr::ScalarExp<float>, float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=2)
                    0.06%  25.824us         1  25.824us  25.824us  25.824us  volta_sgemm_32x128_tn
                    0.05%  23.392us         2  11.696us  2.4960us  20.896us  void mshadow::cuda::MapPlanKernel<mshadow::sv::plusto, int=8, mshadow::expr::Plan<mshadow::Tensor<mshado
w::gpu, int=2, float>, float>, mshadow::expr::Plan<mshadow::expr::Broadcast1DExp<mshadow::Tensor<mshadow::gpu, int=1, float>, float, int=2, int=1>, float>>(mshadow::gpu, unsigned in
t, mshadow::Shape<int=2>, int=2)
                    0.04%  16.832us         1  16.832us  16.832us  16.832us  void fft2d_r2c_32x32<float, bool=0, unsigned int=1, bool=0>(float2*, float const *, int, int, int, int,
 int, int, int, int, int, cudnn::reduced_divisor, bool, int2, int, int)
                    0.02%  9.9840us         9  1.1090us     992ns  1.5360us  [CUDA memset]
    0.02%  7.7760us         1  7.7760us  7.7760us  7.7760us  [CUDA memcpy DtoH]
                    0.01%  4.7680us         1  4.7680us  4.7680us  4.7680us  void mshadow::cuda::MapPlanKernel<mshadow::sv::saveto, int=8, mshadow::expr::Plan<mshadow::Tensor<mshado
w::gpu, int=2, float>, float>, mshadow::expr::Plan<mshadow::expr::ReduceWithAxisExp<mshadow::red::maximum, mshadow::Tensor<mshadow::gpu, int=3, float>, float, int=3, bool=1, int=2>,
 float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=2)
```

```
      API calls:   47.29%  3.80438s        22  172.93ms  14.300us  1.97833s  cudaStreamCreateWithFlags
                   29.61%  2.38157s        24  99.232ms  78.559us  2.37638s  cudaMemGetInfo
                   20.05%  1.61322s        19  84.906ms  1.0110us  434.72ms  cudaFree
                    1.06%  85.180ms       216  394.35us     905ns  48.894ms  cudaEventCreateWithFlags
                    0.78%  62.434ms       912  68.458us     318ns  25.580ms  cudaFuncSetAttribute
                    0.55%  43.933ms         9  4.8814ms  47.899us  21.108ms  cudaMemcpy2DAsync
                    0.28%  22.323ms        29  769.75us  3.3850us  10.576ms  cudaStreamSynchronize
                    0.18%  14.253ms        68  209.60us  7.3620us  2.1641ms  cudaMalloc
                    0.06%  4.6346ms        12  386.21us  8.2800us  4.0283ms  cudaMemcpy
                    0.06%  4.6207ms         4  1.1552ms  409.61us  1.7090ms  cudaGetDeviceProperties
                    0.03%  2.3209ms       375  6.1890us     293ns  325.92us  cuDeviceGetAttribute
                    0.01%  987.73us         8  123.47us  14.321us  772.82us  cudaStreamCreateWithPriority
                    0.01%  812.76us         2  406.38us  51.531us  761.23us  cudaHostAlloc
                    0.01%  724.93us         9  80.548us  11.550us  488.79us  cudaMemsetAsync
                    0.01%  654.04us         4  163.51us  96.749us  282.29us  cuDeviceTotalMem
                    0.01%  633.87us        30  21.128us  8.3030us  96.220us  cudaLaunchKernel
                    0.01%  556.26us         4  139.07us  87.316us  214.50us  cudaStreamCreate
                    0.00%  322.49us       210  1.5350us     545ns  5.9370us  cudaDeviceGetAttribute
                    0.00%  272.82us         4  68.205us  48.044us  107.61us  cuDeviceGetName
                    0.00%  191.07us        32  5.9700us  1.0030us  44.003us  cudaSetDevice
                    0.00%  119.00us       564     210ns      82ns     610ns  cudaGetLastError
                    0.00%  66.913us         6  11.152us  1.9910us  41.214us  cudaEventCreate
                    0.00%  57.658us         6  9.6090us  1.3020us  39.701us  cudaEventRecord
                    0.00%  50.209us        18  2.7890us     720ns  4.8710us  cudaGetDevice
                    0.00%  27.096us         2  13.548us  4.8480us  22.248us  cudaHostGetDevicePointer
                    0.00%  14.600us         1  14.600us  14.600us  14.600us  cudaBindTexture
                    0.00%  6.6720us         2  3.3360us  2.2730us  4.3990us  cudaDeviceGetStreamPriorityRange
                    0.00%  6.5120us         3  2.1700us  1.4580us  2.8380us  cudaStreamWaitEvent
                    0.00%  6.2300us         6  1.0380us     569ns  1.8760us  cuDeviceGetCount
                    0.00%  5.0380us        18     279ns     114ns     685ns  cudaPeekAtLastError
                    0.00%  4.4180us         1  4.4180us  4.4180us  4.4180us  cuDeviceGetPCIBusId
                    0.00%  4.3120us         5     862ns     401ns  1.4530us  cuDeviceGet
                    0.00%  4.1910us         3  1.3970us     808ns  2.4080us  cuInit
                    0.00%  3.5660us         1  3.5660us  3.5660us  3
.5660us  cudaUnbindTexture
                    0.00%  3.0820us         4     770ns     356ns  1.2920us  cuDeviceGetUuid
                    0.00%  2.9870us         1  2.9870us  2.9870us  2.9870us  cudaEventQuery
                    0.00%  2.8870us         4     721ns     439ns  1.4220us  cudaGetDeviceCount
                    0.00%  2.0230us         3     674ns     334ns  1.1830us  cuDriverGetVersion
```

- ❏ List program run time
  - ❏ 4.50 - user 3.43 - system 0:10.74 elapsed 73%CPU

- Milestone 2

❏ List whole program execution time

- ❏ 100

  3.33 user 2.72 system 0:01.09 elapsed 556%CPU

- ❏ 1000

  4.34 user 2.77 system 0:02.01 elapsed 353%CPU

- ❏ 10000

  18.85 user 4.54 system 0:15.29 elapsed 153%CPU

❏ List Op Times

- ❏ 100
  Op Time: 0.034923
  Op Time: 0.075481
- ❏ 1000
  Op Time: 0.254188
  Op Time: 0.754450
- ❏ 10000
  Op Time: 2.840924
  Op Time: 10.858028

- Milestone 3

❏ Correctness and timing with 3 different dataset sizes

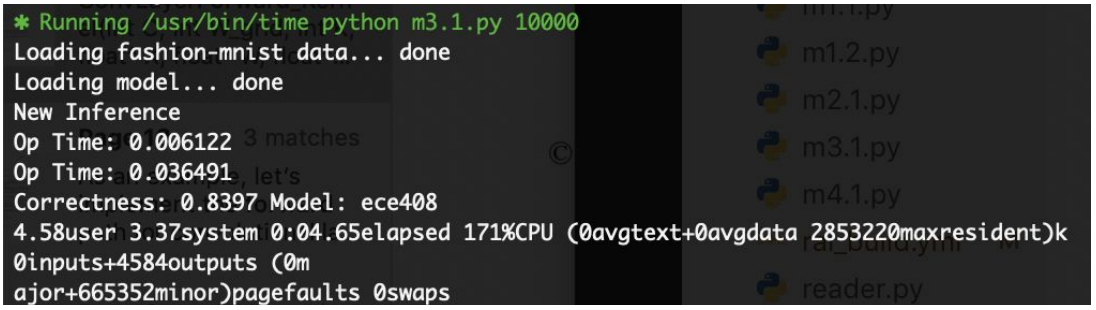
- ❏ 100

```
* Running /usr/bin/time python m3.1.py 100
Loading fashion-mnist data... done
Loading model... done
New Inference
Op Time: 0.000081
Op Time: 0.000390
Correctness: 0.84 Model: ece408
4.14user 3.14system 0:04.25elapsed 170%CPU (0avgtext+0avgdata 2750852maxresident
)k
0inputs+4584outputs (0major+617296minor)pagefaults 0swaps
```

- ❏ 1000

```
* Running /usr/bin/time python m3.1.py 1000
Loading fashion-mnist data... done
Loading model... done
New Inference
Op Time: 0.000637
Op Time: 0.003675
Correctness: 0.852 Model: ece408
4.20user 3.33system 0:04.20elapsed 179%CPU (
0avgtext+0avgdata 2759512maxresident)k
0inputs+4584outputs (0major+618929minor)pagefaults 0swaps
```

- 10000

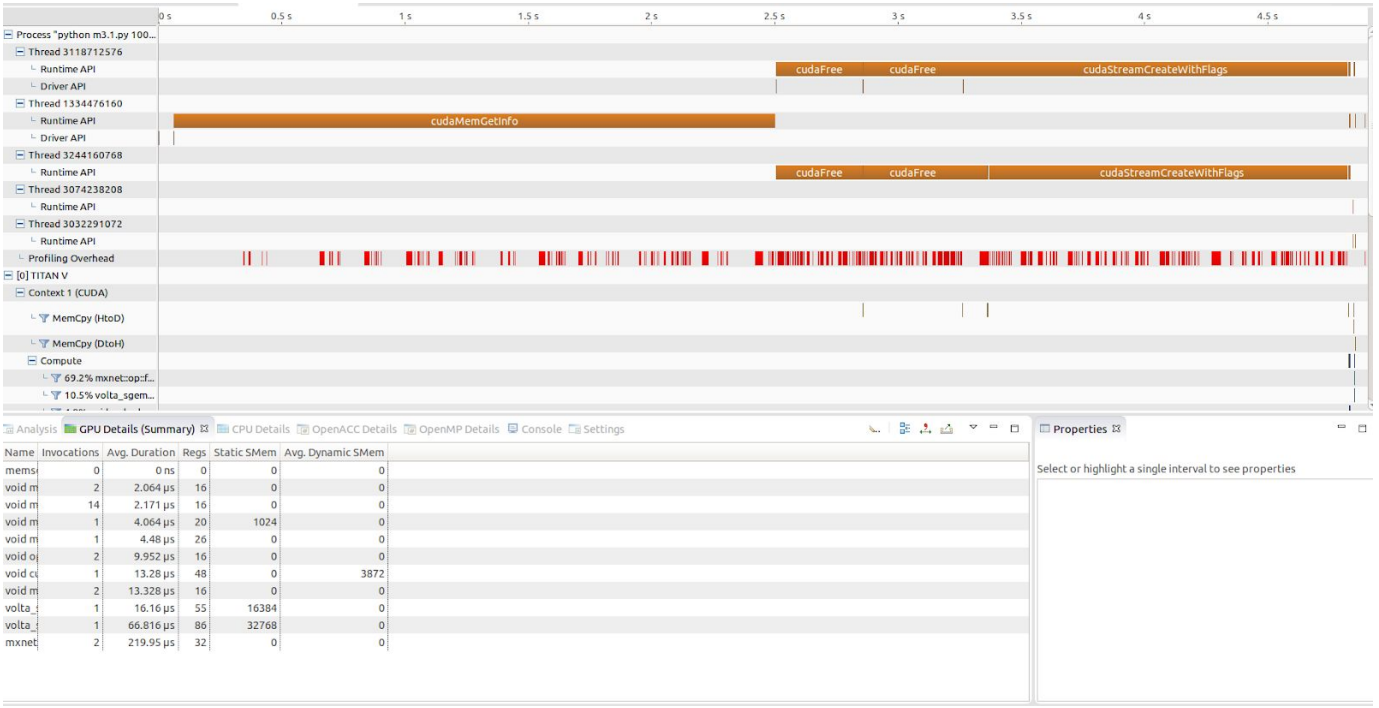


- Report: demonstrate nvprof profiling the execution

  The operation time for the GPU version is much smaller than the CPU version.

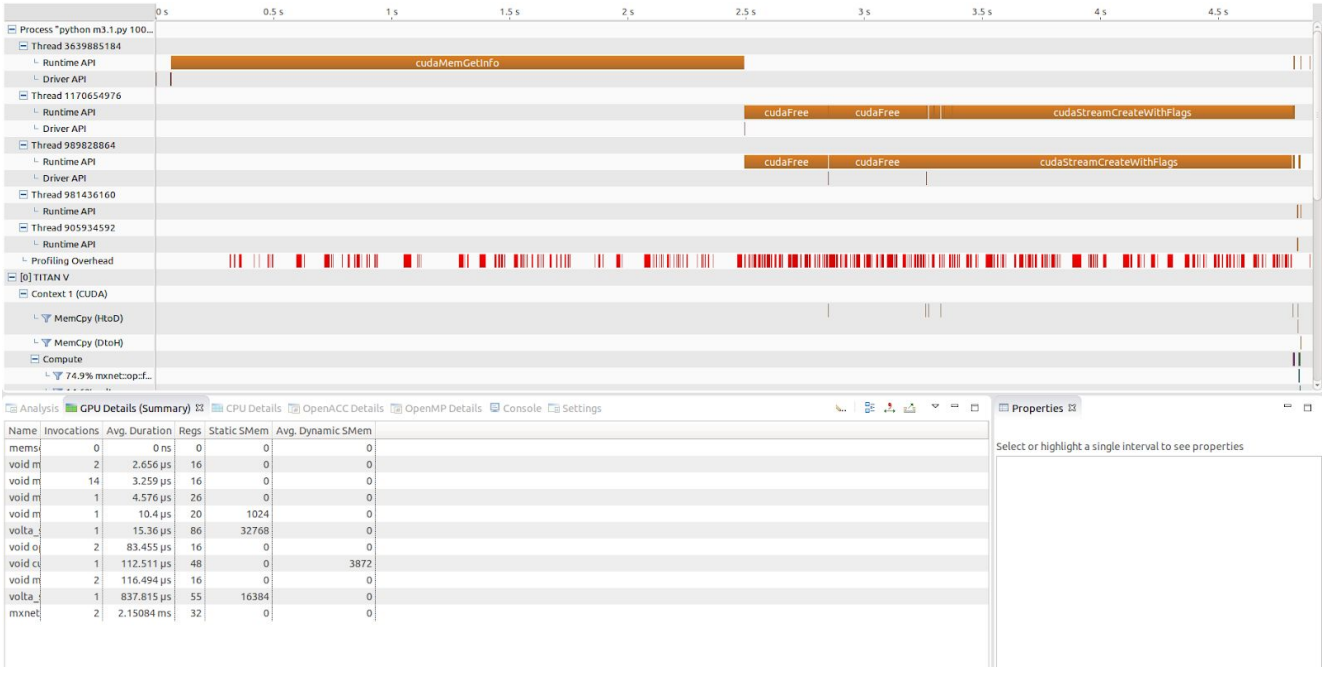
  - 100

```
* Running nvprof -f -o timeline.nvvp python m3.1.py 100
Loading fashion-mnist data... done
==369== NVPROF is profiling process 369, command: python m3.1.py 100
Loading model... done
New Inference
Op Time: 0.000089
Op Time: 0.000399
Correctness: 0.84 Model: ece408
==369== Generated result file: /build/timeline.nvvp
```



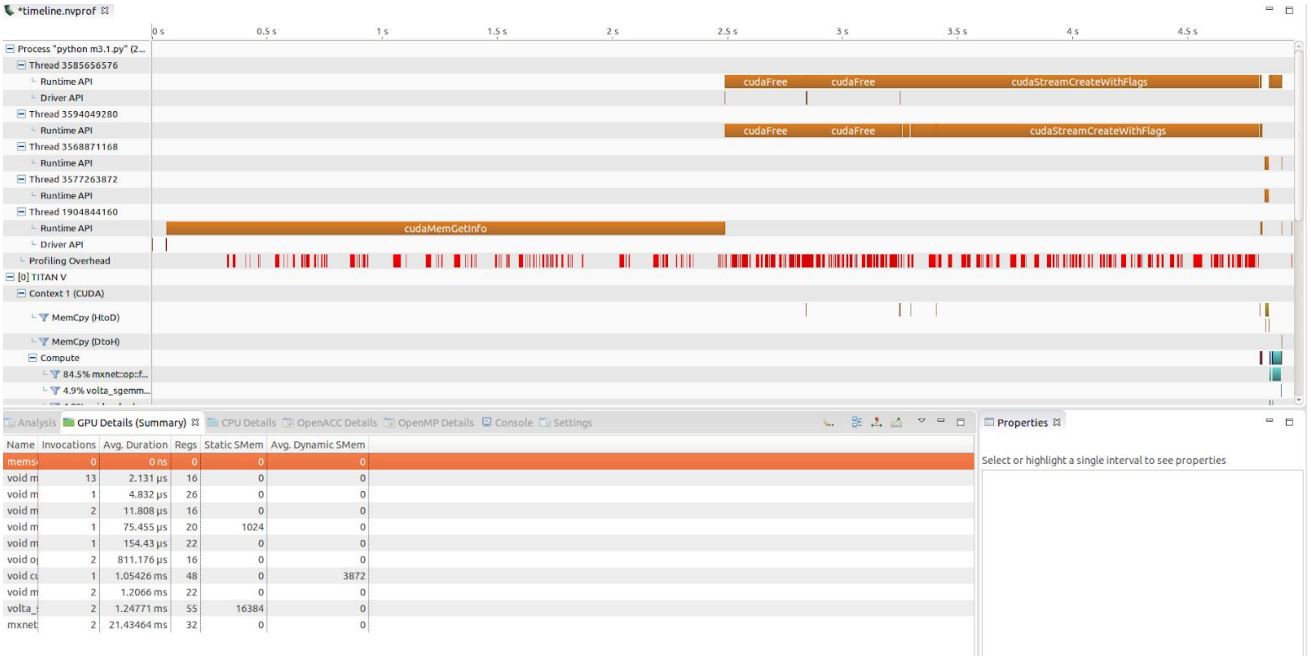
❑ 1000

```
* Running nvprof -f -o timeline.nvvp python m3.1.py 1000
Loading fashion-mnist data... done
==368== NVPROF is profiling process 368, command: python m3.1.py 1000
Loading model... done
New Inference
Op Time: 0.000655
Op Time: 0.003719
Correctness: 0.852 Model: ece408
==368== Generated result file: /build/timeline.nvvp
```



❏ 10000

```
* Running nvprof -f -o timeline.nvvp python m3.1.py
Loading fashion-mnist data... done
==369== NVPROF is profiling process 369, command: python m3.1.py
Loading model... done
New Inference
Op Time: 0.006215
Op Time: 0.036641
Correctness: 0.8397 Model: ece408
==369== Generated result file: /build/timeline.nvvp
```



- **Milestone 4**

❏ Three GPU Optimizations

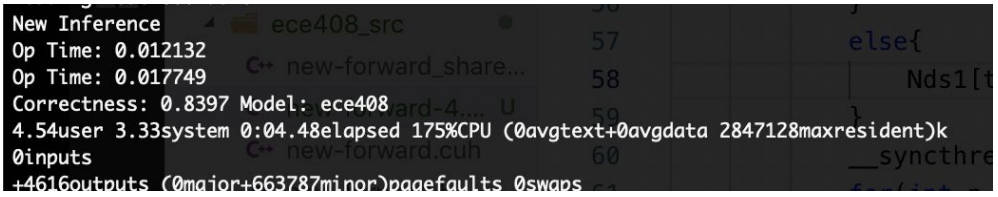
❏ Original kernel runtime:

```
Loading fashion-mnist data... done
Loading model... done
New Inference
Op Time: 0.006204
Op Time: 0.036518
Correctness: 0.8397 Model: ece408
```

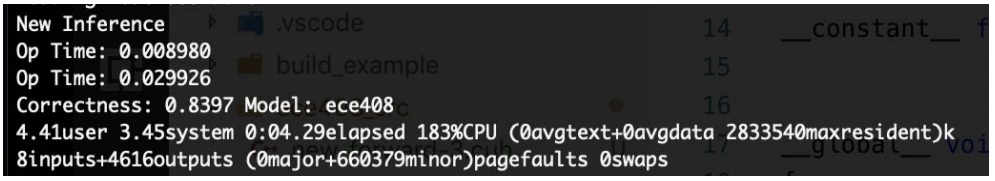
❏ 1. Shared Memory convolution

(kernel code is named forward_kernel_cov_shared in new-forward.cuh, the whole code is in new-forward-4.1.1.cuh)

- This optimization utilizes strategy 2 of doing convolution with shared memory. We first load the tile into shared memory and then do convolution using shared memory. The dimensions of grids are the same with the codes in milestone 3 that we add batch size and number of feature maps into it to avoid launching the kernel for multiple times. In the kernel code, we need to take two different input data sizes into consideration. I have tried two methods. The first one is to loop through channels once in which I load input data to shared memory and then do convolution for this channel. The other one is to loop through channels twice, with the first loop loading data and the second loop finish convolution. Comparing the operation time, I choose the second one.
- BLOCK_WIDTH = 24, forward 2 is more efficient



- BLOCK_WIDTH = 16, forward 1 reaches the most efficient stage, still slower than milstone 3



- 2. Weight matrix (kernel values) in constant memory

(kernel code is named forward_kernel_cons_kernel_1 and forward_kernel_cons_kernel_2 in new-forward.cuh, the whole code is in new-forward-4.1.2.cuh)

- This is a quite straightforward implementation: I have tried to print out the values of C, K, W for different layers and found out that there are two kinds of kernel values: 1*5*5*6 = 150 and 6*5*5*16 = 2400. Thus, I have defined these two constant memories in my code and write two new kernel functions by using different kernel constant memories. Besides, I add the cudaMemcpyToSymbol() in my host code.

```
Loading fashion-mnist data... done
Loading model... done
New Inference
Op Time: 0.005625
Op Time: 0.033060
Correctness: 0.8397 Model: ece408
```

- 3. Unrolling + Matrix multiplication

  (kernel code is named unroll, matrixMultiplyShared and forward_kernel_unroll_matmul in new-forward.cuh, the whole code is in new-forward-4.1.3.cuh)

  - Separate kernel: Implemented unrolling to reduce convolution to matrix multiplication. Convolution masks are loaded into constant memory. In order to avoid the overhead for repetitively invoke the kernel, we set up the grid dimension so that all inputs in the batch are processed at once (as opposed to looping over samples in the minibatch, which greatly worsens the runtime). Matrix multiplication kernel was similar to the one from MP that uses tiling, except that one of the input (convolution mask) can be directly obtained from constant memory so that we don't load them into shared memory.

    ```
    ==279== NVPROF is profiling process 279, command: python m4.1.py
    Loading model... done
    New Inference
    Op Time: 0.019081
    Op Time: 0.021847
    Correctness: 0.8397 Model: ece408
    ==279== Generated result file: /build/timeline.nvprof
    ```

  - With kernel fusion: Combined unrolling and matrix multiplication kernel to reduce number of memcpy and memory allocation on host side. Kernel for unrolling is removed and the unrolling process is "virtualized" during matrix multiplication. In the matrix multiplication kernel, we calculate for each "unrolled" coordinate and calculate its corresponding address in the original input. The matrix multiplication kernel uses same tiling strategy.

    ```
    Loading fashion-mnist data... done
    ==283== NVPROF is profiling process 283, command: python m4.1.py
    Loading model... done
    New Inference
    Op Time: 0.010344
    Op Time: 0.011646
    Correctness: 0.8397 Model: ece408
    ```

- demonstrate nvprof profiling the execution

- 1. Convolution in shared memory

Name	Invocatio	Avg. Duration	Regs	Static SMem	Avg. Dynamic SMem
memset (0)	0	0 ns	0	0	0
void mshadow::cuda::MapPlanKernel<mshadow::sv::saveto, int=8...	13	2.141 µs	16	0	0
void mshadow::cuda::MapPlanKernel<mshadow::sv::saveto, int=8...	1	4.864 µs	26	0	0
void mshadow::cuda::MapPlanKernel<mshadow::sv::plusto, int=8...	2	11.984 µs	16	0	0
void mshadow::cuda::SoftmaxKernel<int=8, float, mshadow::expr...	1	75.04 µs	20	1024	0
void mshadow::cuda::MapPlanLargeKernel<mshadow::sv::saveto,...	1	153.215 µs	22	0	0
void op_generic_tensor_kernel<int=2, float, float, float, int=256, c...	2	816.745 µs	16	0	0
void cudnn::detail::pooling_fw_4d_kernel<float, float, cudnn::deta...	1	1.05708 ms	48	0	3872
void mshadow::cuda::MapPlanLargeKernel<mshadow::sv::saveto,...	2	1.21445 ms	22	0	0
volta_sgemm_32x128_tn	2	1.2504 ms	55	16384	0
mxnet::op::forward_kernel(float*, float const *, float const *, int, in...	2	15.04183 ms	32	16128	0

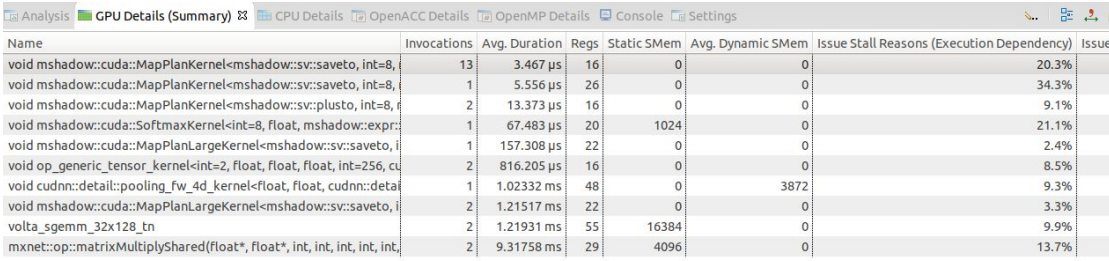
- 2. Weight matrix (kernel values) in constant memory

Name	Invocations	Avg. Duration	Regs	Static SMem	Avg. Dynamic SMem
mxnet::op::forward_kernel_cons_kernel_2(float*, float const *, float const *, int, int...	1	33.43621 ms	32	0	0
mxnet::op::forward_kernel_cons_kernel_1(float*, float const *, float const *, int, int...	1	5.83422 ms	32	0	0
volta_sgemm_32x128_tn	2	1.25452 ms	55	16384	0
void mshadow::cuda::MapPlanLargeKernel<mshadow::sv::saveto, int=8, int=102...	2	1.20967 ms	22	0	0
void cudnn::detail::pooling_fw_4d_kernel<float, float, cudnn::detail::maxpooling_f...	1	1.05183 ms	48	0	3872
void op_generic_tensor_kernel<int=2, float, float, float, int=256, cudnnGenericO...	2	811.927 µs	16	0	0
void mshadow::cuda::MapPlanLargeKernel<mshadow::sv::saveto, int=8, int=102...	1	155.679 µs	22	0	0
void mshadow::cuda::SoftmaxKernel<int=8, float, mshadow::expr::Plan<mshado...	1	75.775 µs	20	1024	0
void mshadow::cuda::MapPlanKernel<mshadow::sv::plusto, int=8, mshadow::exp...	2	11.984 µs	16	0	0
void mshadow::cuda::MapPlanKernel<mshadow::sv::saveto, int=8, mshadow::exp...	1	4.736 µs	26	0	0
void mshadow::cuda::MapPlanKernel<mshadow::sv::saveto, int=8, mshadow::exp...	13	2.136 µs	16	0	0
memset (0)	0	0 ns	0	0	0

- 3.
  - Separate kernel

Name	Invocations	Avg. Duration	Regs	Static SMem	Avg. Dynamic SMem
memset (0)	0	0 ns	0	0	0
void mshadow::cuda::MapPlanKernel<mshadow::sv::saveto, int=8, mshadow::expr	13	2.138 µs	16	0	0
void mshadow::cuda::MapPlanKernel<mshadow::sv::saveto, int=8, mshadow::expr	1	5.792 µs	26	0	0
void mshadow::cuda::MapPlanKernel<mshadow::sv::plusto, int=8, mshadow::expr	2	12.72 µs	16	0	0
void mshadow::cuda::SoftmaxKernel<int=8, float, mshadow::expr::Plan<mshadow	1	75.232 µs	20	1024	0
void mshadow::cuda::MapPlanLargeKernel<mshadow::sv::saveto, int=8, int=1024,	1	155.87 µs	22	0	0
void op_generic_tensor_kernel<int=2, float, float, float, int=256, cudnnGenericOp	2	861.768 µs	16	0	0
void cudnn::detail::pooling_fw_4d_kernel<float, float, cudnn::detail::maxpooling_	1	1.05638 ms	48	0	3872
void mshadow::cuda::MapPlanLargeKernel<mshadow::sv::saveto, int=8, int=1024,	2	1.22973 ms	22	0	0
volta_sgemm_32x128_tn	2	1.28981 ms	55	16384	0
mxnet::op::unroll(int, int, int, int, float*, float*)	2	6.39553 ms	32	0	0
mxnet::op::matrixMultiplyShared(float*, float*, int, int, int, int, int, int, int, int)	2	10.51397 ms	29	4096	0

  - Fusion



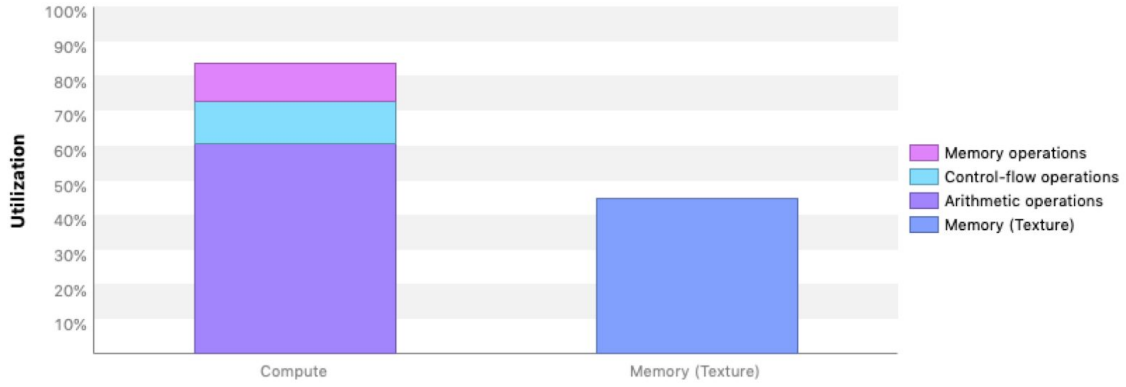
Name	Invocations	Avg. Duration	Regs	Static SMem	Avg. Dynamic SMem	Issue Stall Reasons (Execution Dependency)	Issue
void mshadow::cuda::MapPlanKernel<mshadow::sv::saveto, int=8,	13	3.467 µs	16	0	0	20.3%	
void mshadow::cuda::MapPlanKernel<mshadow::sv::saveto, int=8,	1	5.556 µs	26	0	0	34.3%	
void mshadow::cuda::MapPlanKernel<mshadow::sv::plusto, int=8,	2	13.373 µs	16	0	0	9.1%	
void mshadow::cuda::SoftmaxKernel<int=8, float, mshadow::expr::	1	67.483 µs	20	1024	0	21.1%	
void mshadow::cuda::MapPlanLargeKernel<mshadow::sv::saveto, i	1	157.308 µs	22	0	0	2.4%	
void op_generic_tensor_kernel<int=2, float, float, float, int=256, cu	2	816.205 µs	16	0	0	8.5%	
void cudnn::detail::pooling_fw_4d_kernel<float, float, cudnn::detai	1	1.02332 ms	48	0	3872	9.3%	
void mshadow::cuda::MapPlanLargeKernel<mshadow::sv::saveto, i	2	1.21517 ms	22	0	0	3.3%	
volta_sgemm_32x128_tn	2	1.21931 ms	55	16384	0	9.9%	
mxnet::op::matrixMultiplyShared(float*, float*, int, int, int, int, int,	2	9.31758 ms	29	4096	0	13.7%	

- use NVVP to analyze the optimization

❏ Original Kernel Info(forward1):

mxnet::op::forward_kernel_origin(float*, float const *,	
Queued	n/a
Submitted	n/a
Start	3.41757 s (...
End	3.42316 s (...
Duration	5.58269 ms...
Stream	Default
Grid Size	[10000,6,9]
Block Size	[16,16,1]
Registers/Thread	32
Shared Memory/Block	0 B
Launch Type	Normal
▼Efficiency	
Global Load Efficiency	⚠ 65.4%
Global Store Efficiency	78.6%
Shared Efficiency	n/a
Warp Execution Efficiency	99.8%
Not-Predicated-Off Warp E...	89.9%
▼Occupancy	
Achieved	87.8%
Theoretical	100%
▼Shared Memory Configuration	
Shared Memory Executed	0 B
Shared Memory Bank Size	4 B

❏ Kernel Performance Limiter:



Latency Analysis:

Occupancy Per SM				
Active Blocks		8	32	0 4 8 12 16 20 24 28 32
Active Warps	56.22	64	64	0 8 16 24 32 40 48 56 64
Active Threads		2048	2048	0 256 512 768 1024 1280 1536 1792 2048
Occupancy	87.8%	100%	100%	0% 25% 50% 75% 100%
Warps				
Threads/Block		256	1024	0 128 256 384 512 640 768 896 1024
Warps/Block		8	32	0 4 8 12 16 20 24 28 32
Block Limit		8	32	0 4 8 12 16 20 24 28 32
Registers				
Registers/Thread		32	65536	0 8192 16384 24576 32768 40960 49152 57344 65536
Registers/Block		8192	65536	0 16k 32k 48k 64k
Block Limit		8	32	0 4 8 12 16 20 24 28 32
Shared Memory				
Shared Memory/Block		0	98304	0 32k 64k 96k
Block Limit		0	32	0 4 8 12 16 20 24 28 32

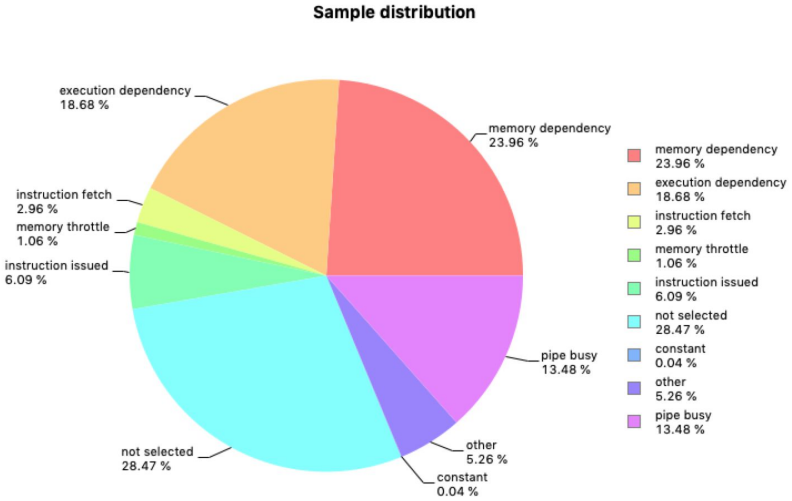
Memory Usage:

Shared Memory			
Shared Loads	0	0 B/s	Idle Low Medium High Max
Shared Stores	0	0 B/s	
Shared Total	0	0 B/s	
L2 Cache			
Reads	26949509	154.475 GB/s	Idle Low Medium High Max
Writes	18480264	105.929 GB/s	
Total	45429773	260.404 GB/s	
Unified Cache			
Local Loads	0	0 B/s	Idle Low Medium High Max
Local Stores	0	0 B/s	
Global Loads	626406236	3,590.565 GB/s	
Global Stores	18480000	105.928 GB/s	
Texture Reads	251459986	5,765.482 GB/s	
Unified Total	896346222	9,461.975 GB/s	
Device Memory			
Reads	33600742	192.6 GB/s	Idle Low Medium High Max
Writes	18506322	106.078 GB/s	
Total	52107064	298.678 GB/s	
System Memory [PCIe configuration: Gen3 x8, 8 Gbit/s]			
Reads	0	0 B/s	Idle Low Medium High Max
Writes	5	28.66 kB/s	Idle Low Medium High Max

Divergent branches:

Line / File	new-forward.cuh - /mxnet/src/operator/custom
49	Divergence = 30.6% [1320000 divergent executions out of 4320000 total executions]

Kernel Profile - PC Sampling



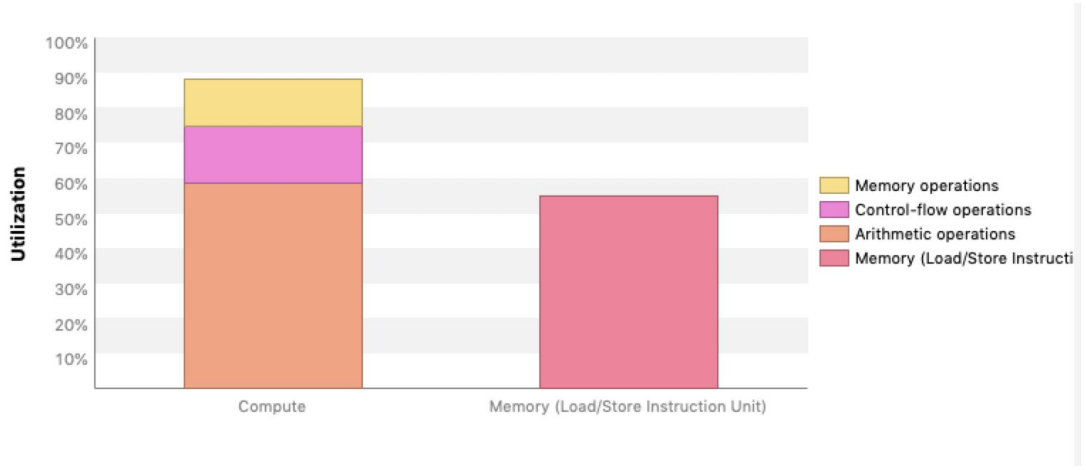
- 1. Convolution in shared memory improves operation time not that much as expected. The problem we find is mostly because the control divergence becomes larger comparing to the original kernel code. Thus, the time saved by reading shared memory conteracts with the latency due to divergence. Thus we need to improve the kernel code using other strategy that has less control divergence. Following is the results in detail.

forward1

forward_kernel information:

mxnet::op::forward_kernel(float*, float const *, float co...	
Queued	n/a
Submitted	n/a
Start	3.2523 s (3,252,296,835 ns)
End	3.2604 s (3,260,396,116 ns)
Duration	8.09928 ms (8,099,281 ns)
Stream	Default
Grid Size	[10000,6,16]
Block Size	[16,16,1]
Registers/Thread	32
Shared Memory/Block	7 KiB
Launch Type	Normal
▼Efficiency	
Global Load Efficiency	⚠ 25.5%
Global Store Efficiency	77.8%
Shared Efficiency	⚠ 73.3%
Warp Execution Efficiency	96.7%
Not-Predicated-Off Warp Ex...	80.8%
▼Occupancy	
Achieved	93.3%
Theoretical	100%
▼Shared Memory Configuration	
Shared Memory Executed	0 B
Shared Memory Bank Size	4 B

Utilization of Compute and Memory are both improved.



Divergence is high, which is the cons of strategy 2. In each block, we only use the left right square of the TILE_WIDTH rather than the whole block.

Line / File	new-forward.cuh - /mxnet/src/operator/custom
84	Divergence = 75% [28800000 divergent executions out of 38400000 total executions]
90	Divergence = 21.9% [1680000 divergent executions out of 7680000 total executions]

Memory Bandwidth and Utilization shows shared memory is in use, which is faster than global memory.

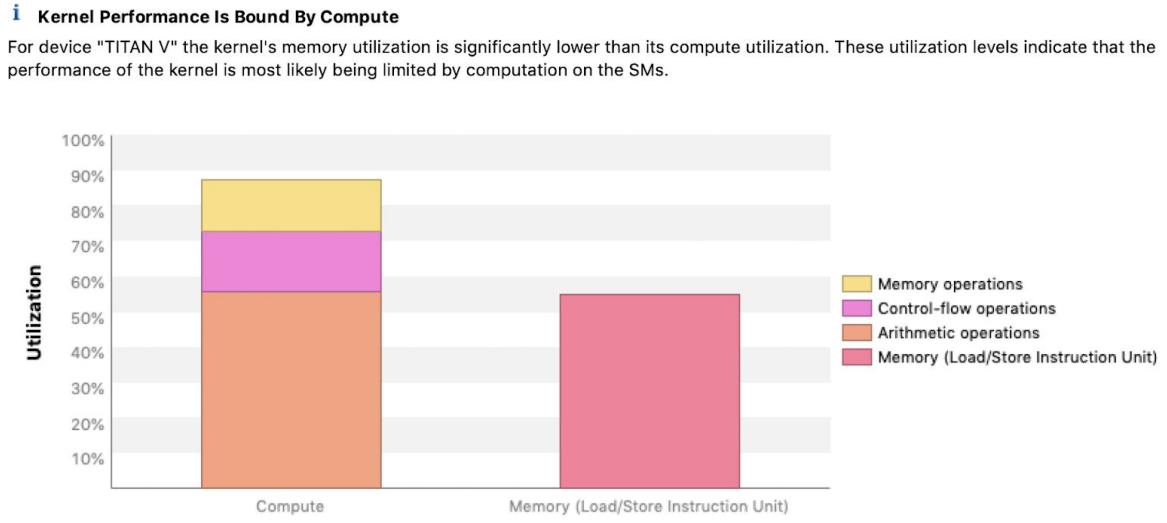
	Transactions	Bandwidth	Utilization
Shared Memory			
Shared Loads	150206671	2,373.847 GB/s	
Shared Stores	7680000	121.374 GB/s	
Shared Total	157886671	2,495.221 GB/s	Idle Low Medium High Max
L2 Cache			
Reads	32476710	128.314 GB/s	
Writes	30240016	119.477 GB/s	
Total	62716726	247.792 GB/s	Idle Low Medium High Max
Unified Cache			
Local Loads	0	0 B/s	
Local Stores	0	0 B/s	
Global Loads	176400000	696.951 GB/s	
Global Stores	30240000	119.477 GB/s	
Texture Reads	311583966	4,924.233 GB/s	
Unified Total	518223966	5,740.661 GB/s	Idle Low Medium High Max
Device Memory			
Reads	43200930	170.685 GB/s	
Writes	30284209	119.652 GB/s	
Total	73485139	290.337 GB/s	Idle Low Medium High Max
System Memory [PCIe configuration: Gen3 x8, 8 Gbit/s]			
Reads	0	0 B/s	Idle Low Medium High Max
Writes	5	19.754 kB/s	Idle Low Medium High Max

forward2

forward_kernel information:

Queued	n/a
Submitted	n/a
Start	3.32027 s (3,320,267,686 ns)
End	3.3363 s (3,336,296,943 ns)
Duration	16.02926 ms (16,029,257 ns)
Stream	Default
Grid Size	[10000,16,1]
Block Size	[24,24,1]
Registers/Thread	32
Shared Memory/Block	15.75 KiB
Launch Type	Normal
▼Efficiency	
Global Load Efficiency	⚠ 24.2%
Global Store Efficiency	81%
Shared Efficiency	83.6%
Warp Execution Efficiency	96.5%
Not-Predicated-Off Warp Ex...	81.5%
▼Occupancy	
Achieved	78.6%
Theoretical	84.4%
▼Shared Memory Configuration	
Shared Memory Executed	0 B
Shared Memory Bank Size	4 B

Utilization of Compute and Memory are both improved.



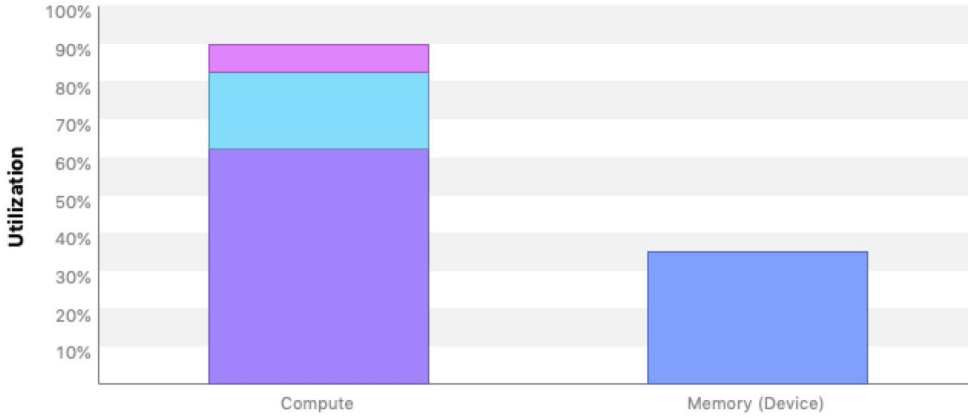
Divergence is still high.

▼Line / File	new-forward.cuh - /mxnet/src/operator/custom
67	Divergence = 83.3% [72000000 divergent executions out of 86400000 total executions]
90	Divergence = 77.8% [2240000 divergent executions out of 2880000 total executions]

Memory Bandwidth and Utilization shows that device memory is used less while shared memory utilization is highly improved.

	Transactions	Bandwidth	Utilization
Shared Memory			
Shared Loads	362154254	2,891.946 GB/s	
Shared Stores	17280000	137.988 GB/s	
Shared Total	379434254	3,029.934 GB/s	Idle Low Medium High Max
L2 Cache			
Reads	58741132	117.268 GB/s	
Writes	8000498	15.972 GB/s	
Total	66741630	133.24 GB/s	Idle Low Medium High Max
Unified Cache			
Local Loads	0	0 B/s	
Local Stores	0	0 B/s	
Global Loads	425985896	850.417 GB/s	
Global Stores	8000000	15.971 GB/s	
Texture Reads	744733468	5,946.993 GB/s	
Unified Total	1178719364	6,813.381 GB/s	Idle Low Medium High Max
Device Memory			
Reads	58081078	115.95 GB/s	
Writes	6604259	13.184 GB/s	
Total	64685337	129.135 GB/s	Idle Low Medium High Max
System Memory [PCIe configuration: Gen3 x8, 8 Gbit/s]			
Reads	0	0 B/s	Idle Low Medium High Max
Writes	5	9.981 kB/s	Idle Low Medium High Max

- 2. Using the constant memory for storing kernel matrices, we could access memory in constant ones instead of global ones. Thus, the time for accessing memory would decrease so we could get a slightly better runtime for this optimization.
  - The Compute utilization becomes higher than before and memory utilization becomes smaller. Besides, we have found that the usage of unified cache and device memory become smaller compared with the original kernel implementation. These happen because we have used constant memory to cut down access time to the global memory.



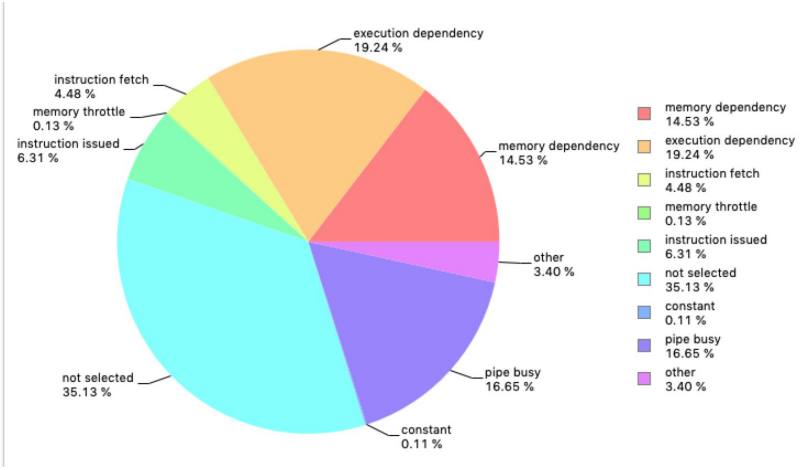
Unified Cache

Local Loads	0	0 B/s	
Local Stores	0	0 B/s	
Global Loads	1962185326	2,090.837 GB/s	
Global Stores	10080000	10.741 GB/s	
Texture Reads	574453429	2,448.471 GB/s	
Unified Total	2546718755	4,550.049 GB/s	Idle Low Medium High Max

Device Memory

Reads	133920724	142.701 GB/s	
Writes	9582680	10.211 GB/s	
Total	143503404	152.912 GB/s	Idle Low Medium High Max

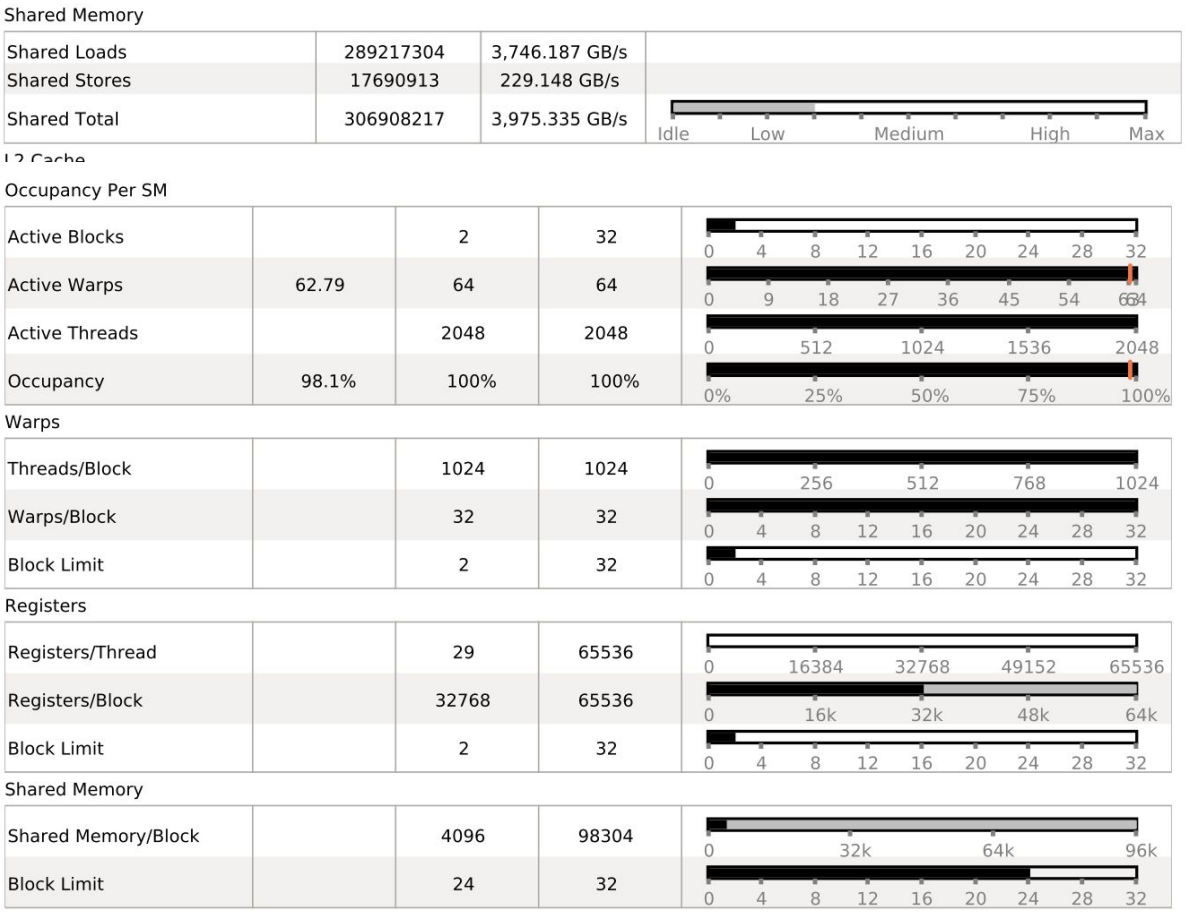
- I also found that we have cut down the percentage of memory dependency and increase the pipe busy ratio, which is a good sign for optimization since we have used more ratio of computing resources than before.



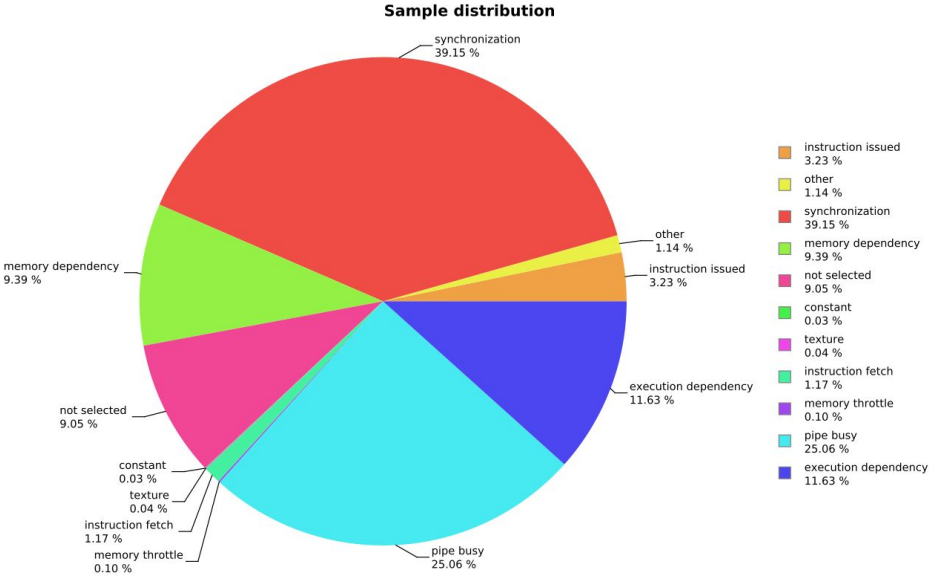
- 3. For the separate kernel approach, the op time for the first layer was slower than that without optimization, while the second op time is improved. nvvp analysis shows that the execution has low efficiency in global and shared memory usage. We believed that it results from the redundant memory read and writes required to performing the unrolling

step. Since the input to the first layer has a smaller size than the second layer, it's possible that the performance gain from reducing convolution to matmul is not large enough to compensate for the cost of excessive global memory access. Therefore, we performed further optimization to combine the two kernels and perform the address calculation in matrix multiply kernel instead of creating an unrolled input matrix. The optimization achieved a performance gain of ~50% compared to that with the separate kernel, but the op time for the first layer is still slower than the original. nvvp shows that we have low computed utilization. In order to further improve the performance, we may need to tune the grid/block dimensions. It might also be helpful to use a different implementation for smaller input size.

Compared with the original kernel, a key improvement is we achieved better occupancy by having more active warps, made use of more registers per block and shared memory.



For factors that caused latency, it has less memory dependency and execution dependency, but synchronization became a limiting factor.


Sample distribution
synchronization
39.15 %
other
1.14 %
instruction issued
3.23 %
memory dependency
9.39 %
execution dependency
11.63 %
not selected
9.05 %
constant
0.03 %
texture
0.04 %
instruction fetch
1.17 %
memory throttle
0.10 %
pipe busy
25.06 %
instruction issued
3.23 %
other
1.14 %
synchronization
39.15 %
memory dependency
9.39 %
not selected
9.05 %
constant
0.03 %
texture
0.04 %
instruction fetch
1.17 %
memory throttle
0.10 %
pipe busy
25.06 %
execution dependency
11.63 %