

# **ECE 408 Report**

**Team Name:** tiledtitans

**Members:**

- Jackson DeDobbelaeere (dedobbe2)
- Matthew Grossfeld (grossfe2)
- Xinbo Wu (xinbowu2)

\*\*\*All on campus students\*\*\*

# MILESTONE 1

## List of all kernels that collectively consume more than 90% of the program time:

- [CUDA memcpy HtoD]
- void cudnn::detail::implicit\_convolve\_sgemm<float, float, int=1024, int=5, int=5, int=3, int=3, int=3, int=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=1, bool=1, bool=0, bool=1>\*, kernel\_conv\_params, int, float, float, int, float, float, int, int)
- Volta\_cgemm\_64x32\_tn
- void op\_generic\_tensor\_kernel<int=2, float, float, float, int=256, cudnnGenericOp\_t=7, cudnnNanPropagation\_t=0, cudnnDimOrder\_t=0, int=1>(cudnnTensorStruct, float\*, cudnnTensorStruct, float const \*, cudnnTensorStruct, float const \*, float, float, float, float, dimArray, reducedDivisorArray)
- Volta\_sgemm\_128x128\_tn
- 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, float, float, cudnn::reduced\_divisor, bool, float\*, float\*, int2, int, int)
- void cudnn::detail::pooling\_fw\_4d\_kernel<float, float, cudnn::detail::maxpooling\_func<float, cudnnNanPropagation\_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)

## List of all CUDA API calls that collectively consume more than 90% of the program time:

- cudaStreamCreateWithFlags
- cudaMemGetInfo
- cudaFree

## Explanation of the difference between kernels and API calls:

API calls are done with the CPU and execute from the time the call is made to the time it is returned. Kernels are ran on the GPU and are measured by the total time that the kernel is executing and running instructions. cudaLaunchKernel is an API function called from the CPU

that can launch kernels from the GPU. The profiling done on the API accounts for all of the launch overhead while the kernel timing only contains a small portion of it.

### Output of rai running MXNet on the CPU:

```
1. bash
om requests<2.19.0,>=2.18.4->mxnet==1.3.1) (2.6)
Requirement already satisfied: urllib3<1.23,>=1.21.1 in /root/.local/lib/python2.7/site-packages (from requests<2.19.0,>=2.18.4->mxnet==1.3.1) (1.22)
Requirement already satisfied: certifi>=2017.4.17 in /root/.local/lib/python2.7/site-packages (from requests<2.19.0,>=2.18.4->mxnet==1.3.1) (2018.11.29)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /root/.local/lib/python2.7/site-packages (from requests<2.19.0,>=2.18.4->mxnet==1.3.1) (3.0.4)
Installing collected packages: mxnet
  Running setup.py develop for mxnet
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}
9.14user 3.50system 0:05.28elapsed 239%CPU (0avgtext+0avgdata 2472816maxresident)k
0inputs+2824outputs (0
major+666007minor)pagefaults 0swaps
* The build folder has been uploaded to http://s3.amazonaws.com/files.rai-project.com/userdata/build-5c7c6526c63bea0ecd2adc99.tar.gz. The data will be present for only a short duration off time.
* Server has ended your request.
wirelessprv-10-194-40-73:CS 483 jacksondedobbelaeere$
```

### Program run time of MXNet on the CPU:

- User: 9.14
- System: 3.50
- Elapsed: 0:05.28

## Output of rai running MXNet on the GPU:

```
1. bash
om requests<2.19.0,>=2.18.4->mxnet==1.3.1) (2.6)
Requirement already satisfied: urllib3<1.23,>=1.21.1 in /root/.local/lib/python2.7/site-packages (from requests<2.19.0,>=2.18.4->mxnet==1.3.1) (1.22)
Requirement already satisfied: certifi>=2017.4.17 in /root/.local/lib/python2.7/site-packages (from requests<2.19.0,>=2.18.4->mxnet==1.3.1) (2018.11.29)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /root/.local/lib/python2.7/site-packages (from requests<2.19.0,>=2.18.4->mxnet==1.3.1) (3.0.4)
Installing collected packages: mxnet
  Running setup.py develop for mxnet
Successfully installed mxnet
* Running /usr/bin/time python m1.2.py
Loading fashion-mnist data... done
Loading model... done
New Inference
EvalMetric: {'accuracy': 0.8236}
4.23user 3.38system 0:04.17elapsed 182%CPU (0avgtext+
0avgdata 2846888maxresident)k
0inputs+4552outputs (0major+662021minor)pagefaults 0swaps
* The build folder has been uploaded to http://s3.amazonaws.com/files.rai-project.com/userdata/build-5c7c65e4c63bea0eebf3e091.tar.gz. The data will be present for only a short duration off time.
* Server has ended your request.
wirelessprv-10-194-40-73:CS 483 jacksondedobbelaere$
```

## Program run time of MXNet on the GPU:

- User: 4.23
- System: 3.38
- Elapsed: 0:04.17

# MILESTONE 2

## 100 Images:

- **Whole Program Execution Time:**
  - User: 2.77
  - System: 2.65
  - Elapsed: 0:01.03
- **Layer 1 Op Time:** 0.034247
- **Layer 2 Op Time:** 0.074316

## 1,000 Images:

- **Whole Program Execution Time:**
  - User: 4.24
  - System: 3.15
  - Elapsed: 0:01.95
- **Layer 1 Op Time:** 0.238204
- **Layer 2 Op Time:** 0.743339

## 10,000 Images:

- **Whole Program Execution Time:**
  - User: 14.97
  - System: 4.47
  - Elapsed: 0:11.37
- **Layer 1 Op Time:** 2.427247
- **Layer 2 Op Time:** 7.383249

# MILESTONE 3

## 100 Images:

- **Correctness:** 0.84
- **Whole Program Execution Time:**
  - User: 4.36
  - System: 3.27
  - Elapsed: 0:04.34
- **Layer 1 Op Time:** 0.000102
- **Layer 2 Op Time:** 0.000236

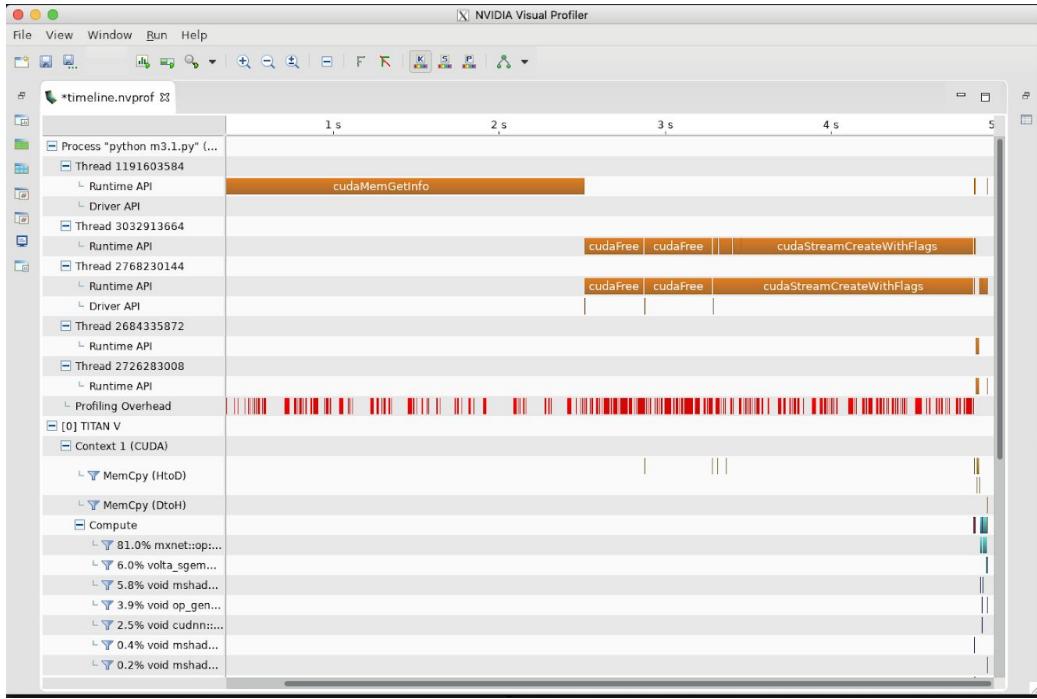
## 1,000 Images:

- **Correctness:** 0.852
- **Whole Program Execution Time:**
  - User: 4.42
  - System: 3.39
  - Elapsed: 0:04.22
- **Layer 1 Op Time:** 0.000908
- **Layer 2 Op Time:** 0.002467

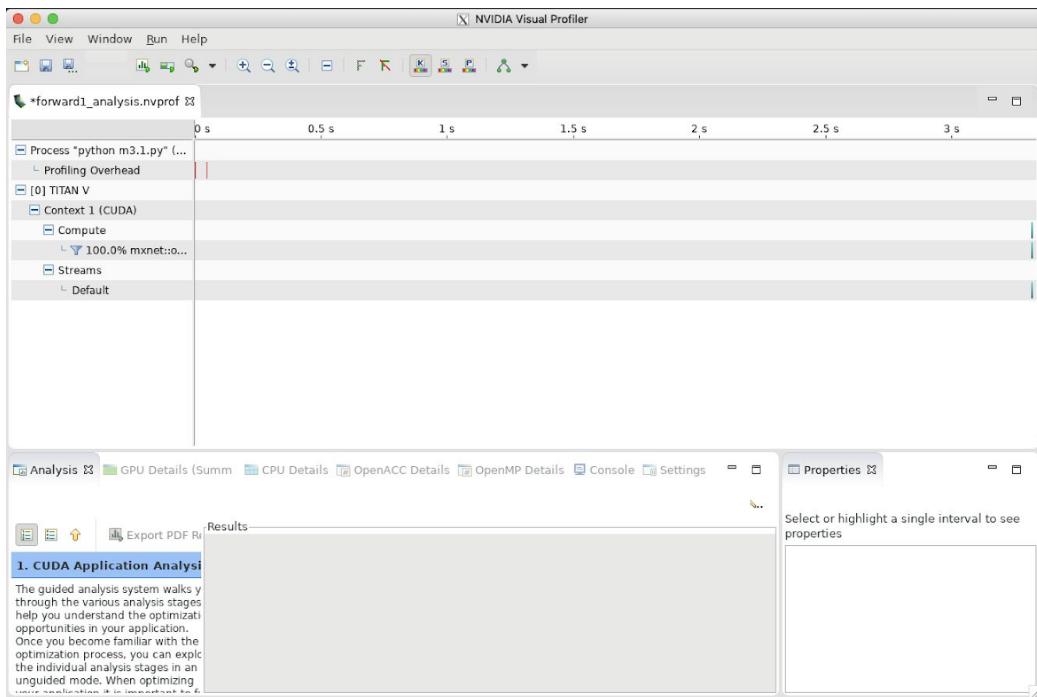
## 10,000 Images:

- **Correctness:** 0.8397
- **Whole Program Execution Time:**
  - User: 4.33
  - System: 3.31
  - Elapsed: 0:04.34
- **Layer 1 Op Time:** 0.009238
- **Layer 2 Op Time:** 0.024392

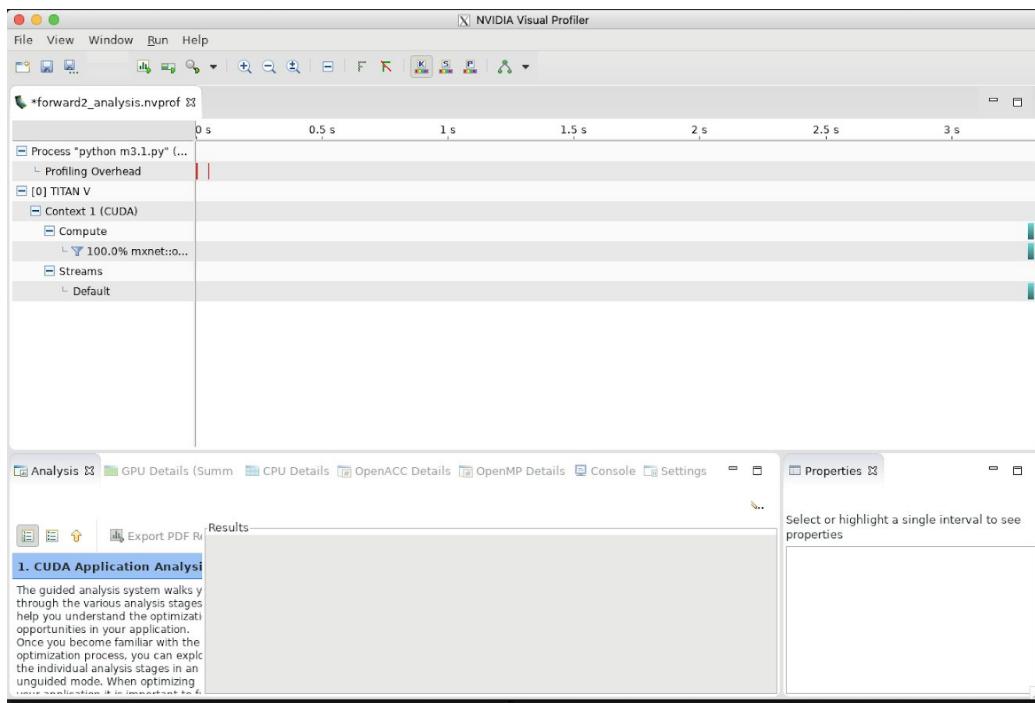
## NVVP output of timeline.nvprof:



## NVVP output of forward1\_analysis.nvprof:



## NVVP output of forward2\_analysis.nvprof:



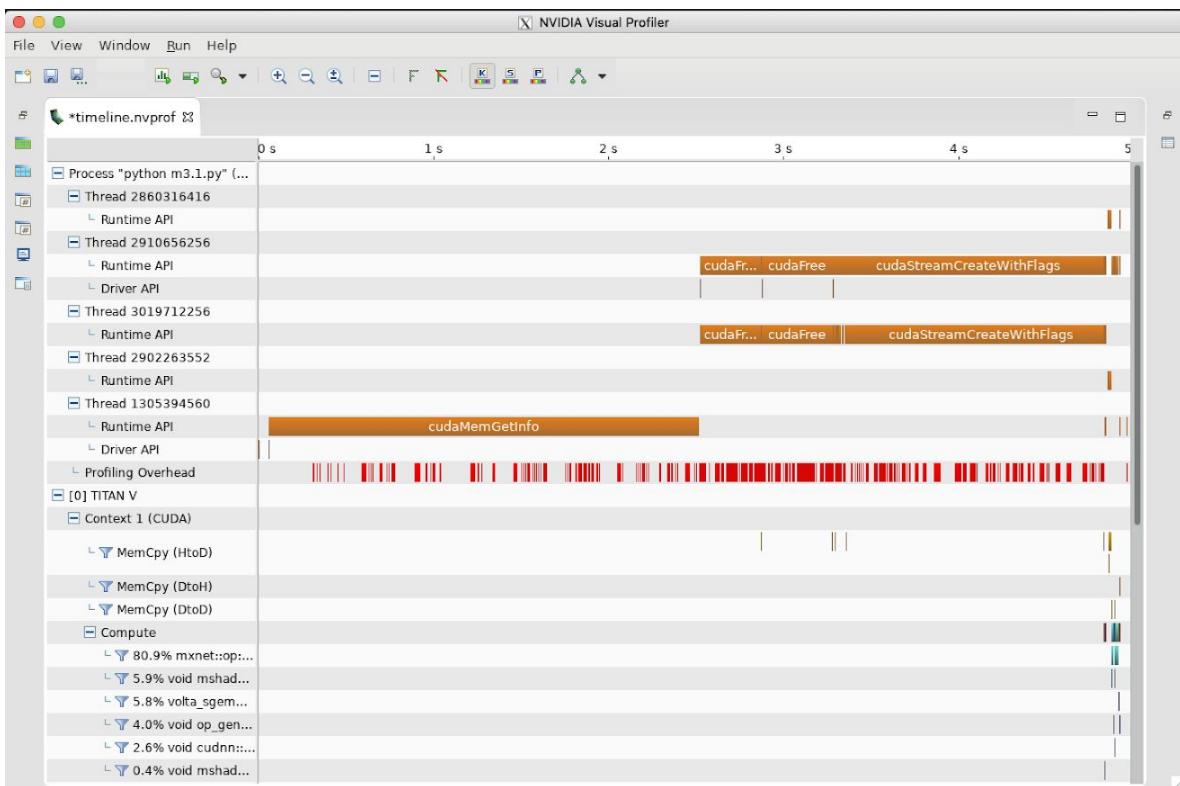
# MILESTONE 4

## Optimization #1: Kernel Values in Constant Memory

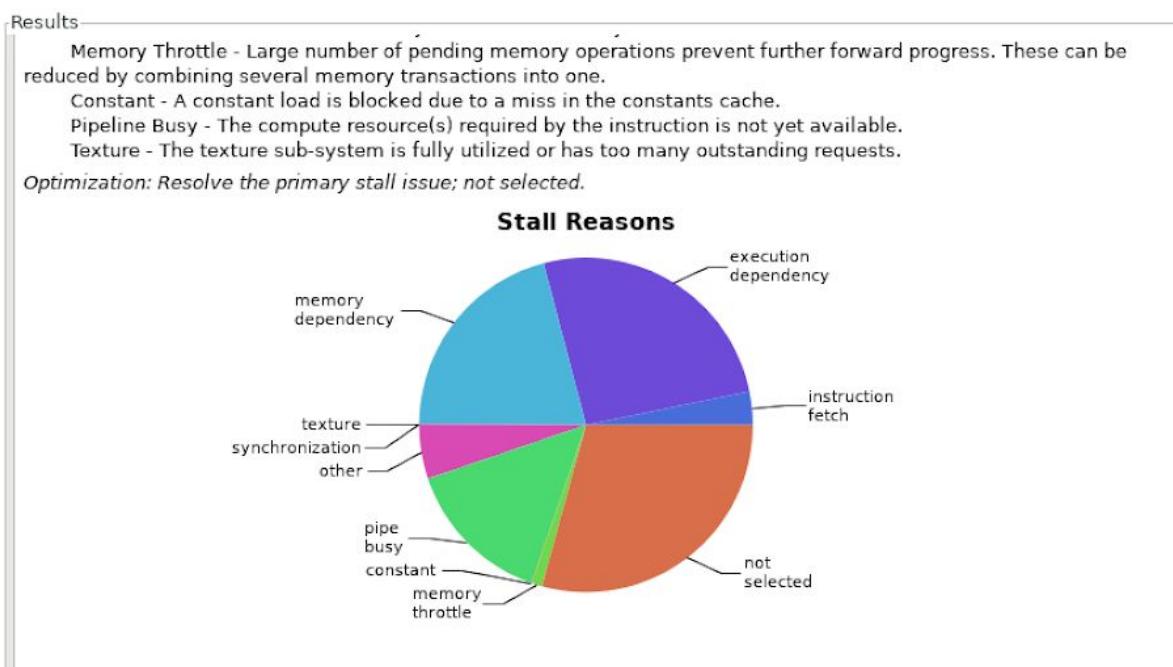
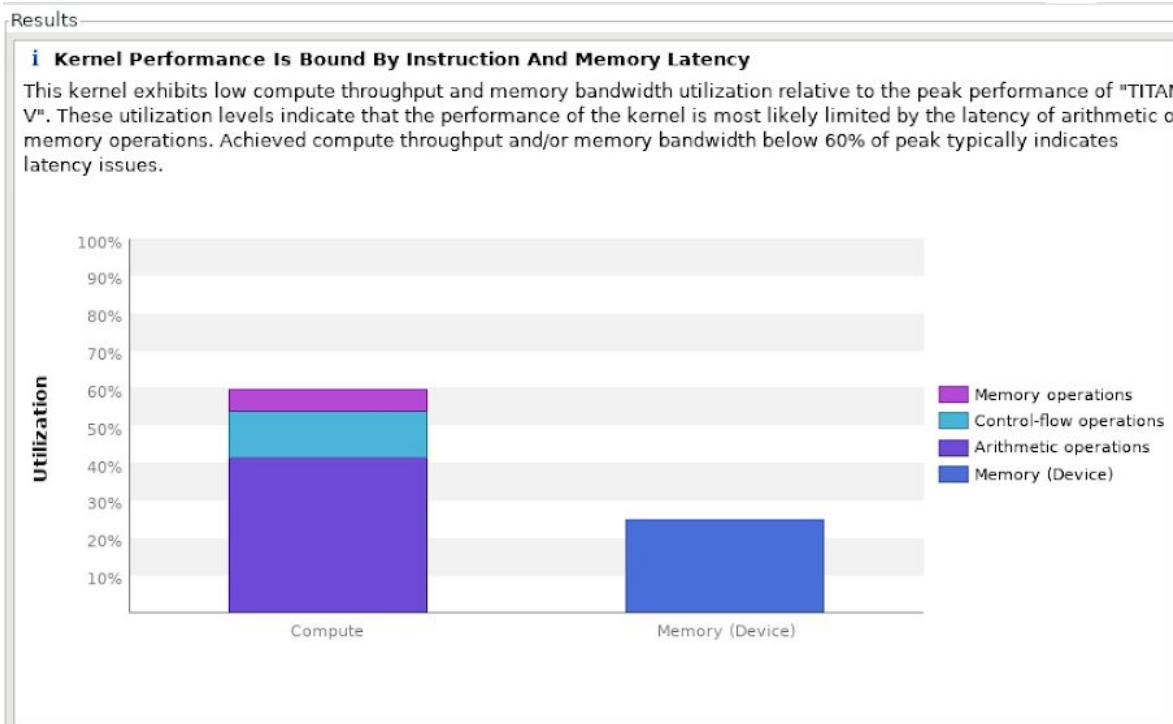
- **Description**

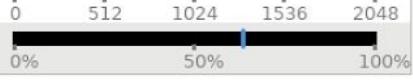
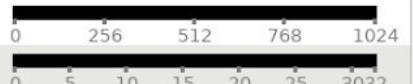
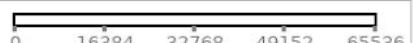
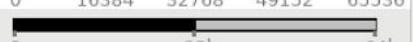
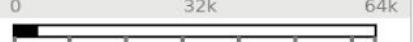
We create an array in constant memory on our device of size  $K \times K \times M \times C$  for storing our weights. This is easily done by adding the CUDA API call `cudaMemcpyToSymbol` before calling our kernel. This allows us to only access our weights from cached memory within the device as opposed to using the global memory. If we need to access all the weights in our kernel, we will have avoided  $K \times K \times M \times C$  global memory accesses.

- **Timeline Overview**

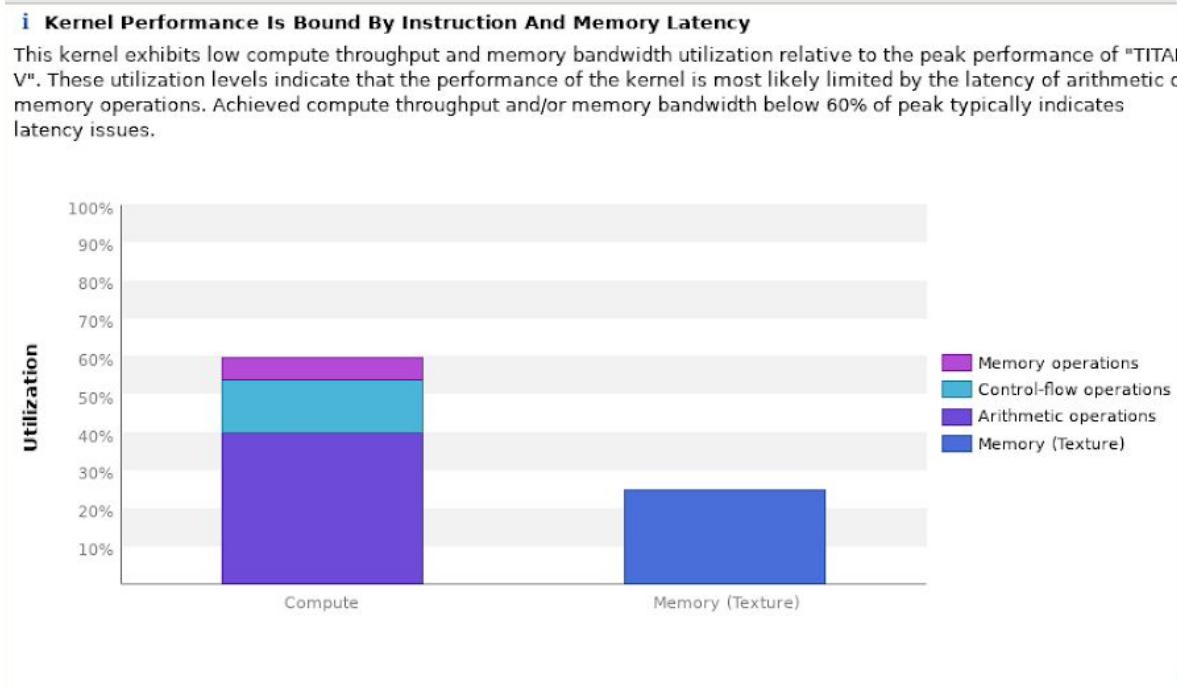


- Forward Layer 1



Results				
Active Warps	40.15	64	64	
Active Threads		2048	2048	
Occupancy	62.7%	100%	100%	
Warp Statistics				
Threads/Block		1024	1024	
Warps/Block		32	32	
Block Limit		2	32	
Register Statistics				
Registers/Thread		32	65536	
Registers/Block		32768	65536	
Block Limit		2	32	
Shared Memory Statistics				
Shared Memory/Block		0	98304	
Block Limit		0	32	

## • Forward Layer 2



## Results

Execution Dependency - An input required by the instruction is not yet available. Execution dependency stalls can potentially be reduced by increasing instruction-level parallelism.

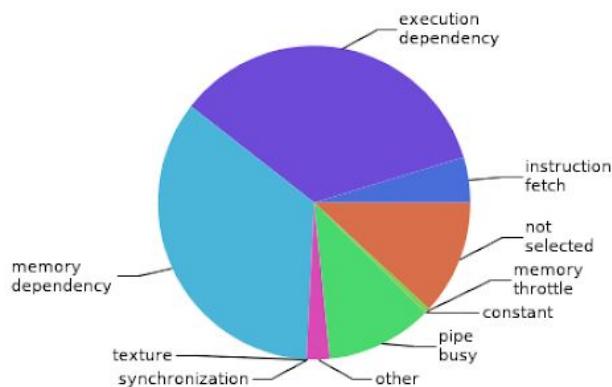
Texture - The texture sub-system is fully utilized or has too many outstanding requests.

Synchronization - The warp is blocked at a `__syncthreads()` call.

Constant - A constant load is blocked due to a miss in the constants cache.

Memory Dependency - A load/store cannot be made because the required resources are not available or are fully utilized, or too many requests of a given type are outstanding. Data request stalls can potentially be reduced by optimizing memory alignment and access patterns.

**Stall Reasons**



## Results

### Occupancy per SM

		2	32	
Active Blocks		2	32	
Active Warps	28.8	64	64	
Active Threads		2048	2048	
Occupancy	45%	100%	100%	

### Warp

		1024	1024	
Threads/Block		1024	1024	
Warps/Block		32	32	
Block Limit		2	32	

### Registers

		32	65536	
Registers/Thread		32	65536	
Registers/Block		32768	65536	
Block Limit		2	32	

### Shared Memory

		0	98304	
Shared Memory/Block		0	98304	
Block Limit		0	32	

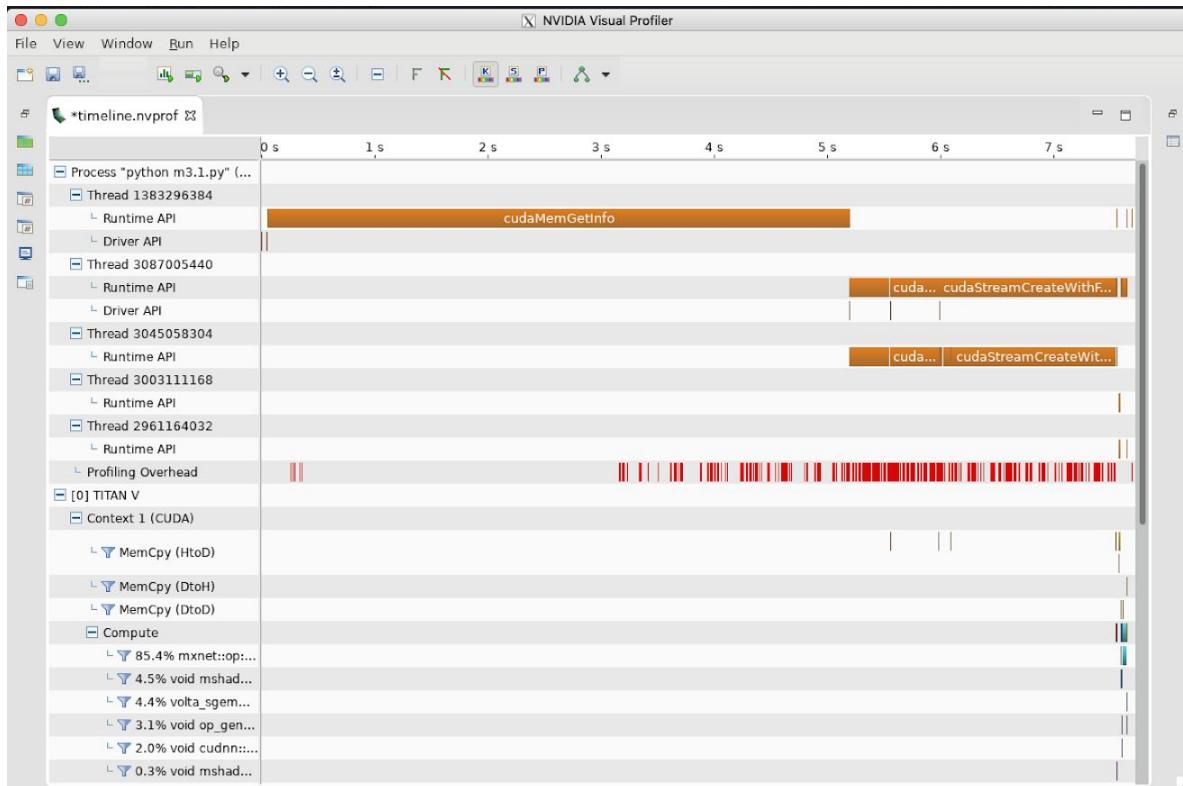
## Optimization #2: Shared Memory Convolution

- **Description**

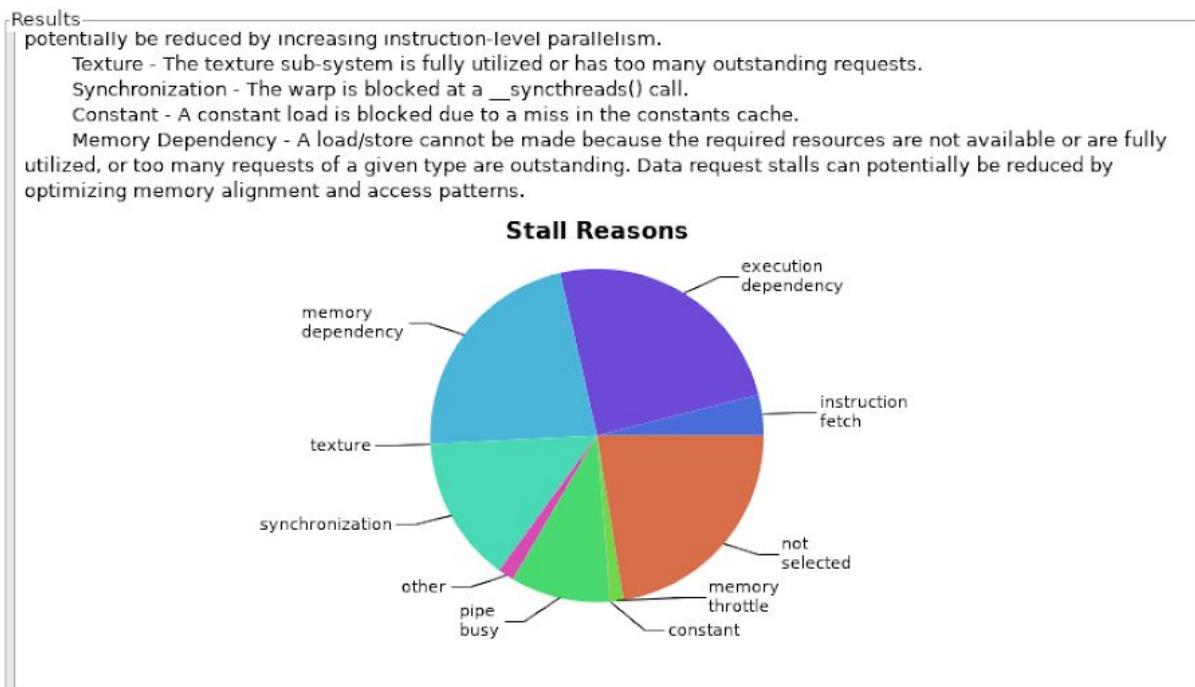
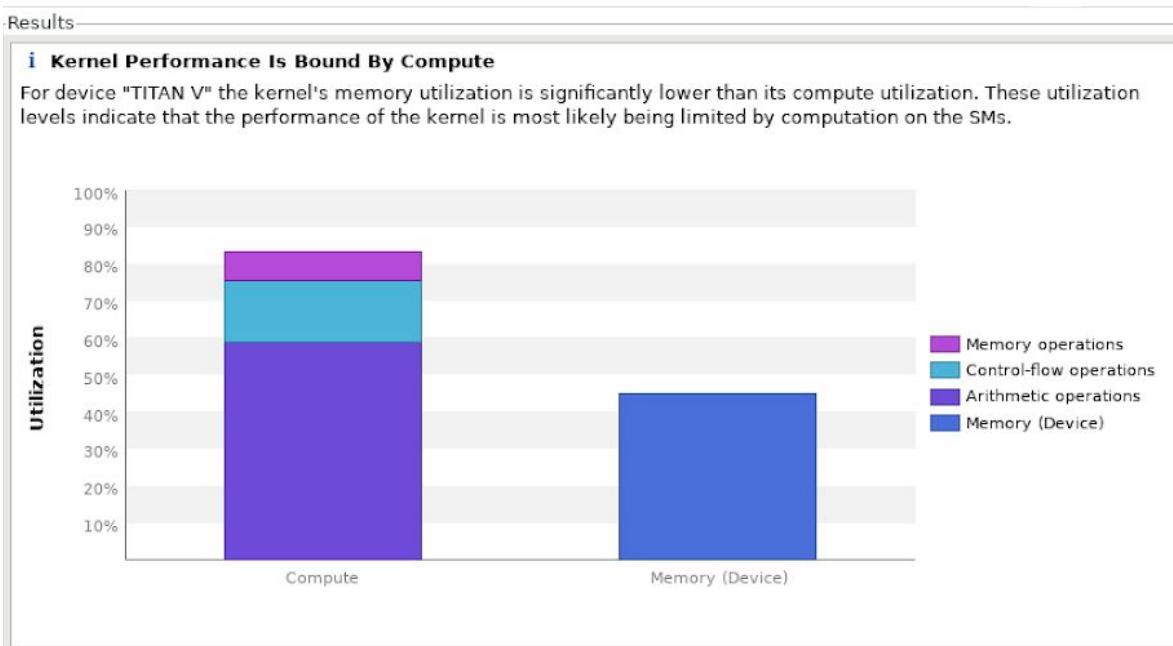
We changed the size of our blocks (i.e. the amount of threads per block) from 32 to 16 so that our tiles are smaller (thus reducing the amount of memory accesses per tile). We perform tiling using strategy 1 from the lecture notes (i.e. our block size covers the output size and we load the input over multiple steps). In our kernel we instantiate an array in shared memory called `X_Shared`, of size  $(\text{TILE\_SIZE} + K - 1)^2$  that holds our input values for `x`. This reduces our global memory accesses significantly. More specifically, we have global memory accesses that are converted into shared memory

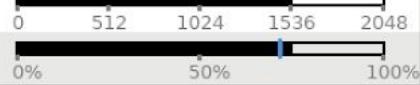
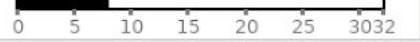
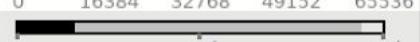
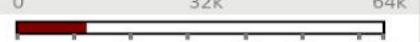
accesses, where  $T$  is `TILE_SIZE`. This gives us a total reduction of  $\frac{T^2 K^2}{(T + K - 1)^2}$ .

- **Timeline Overview**

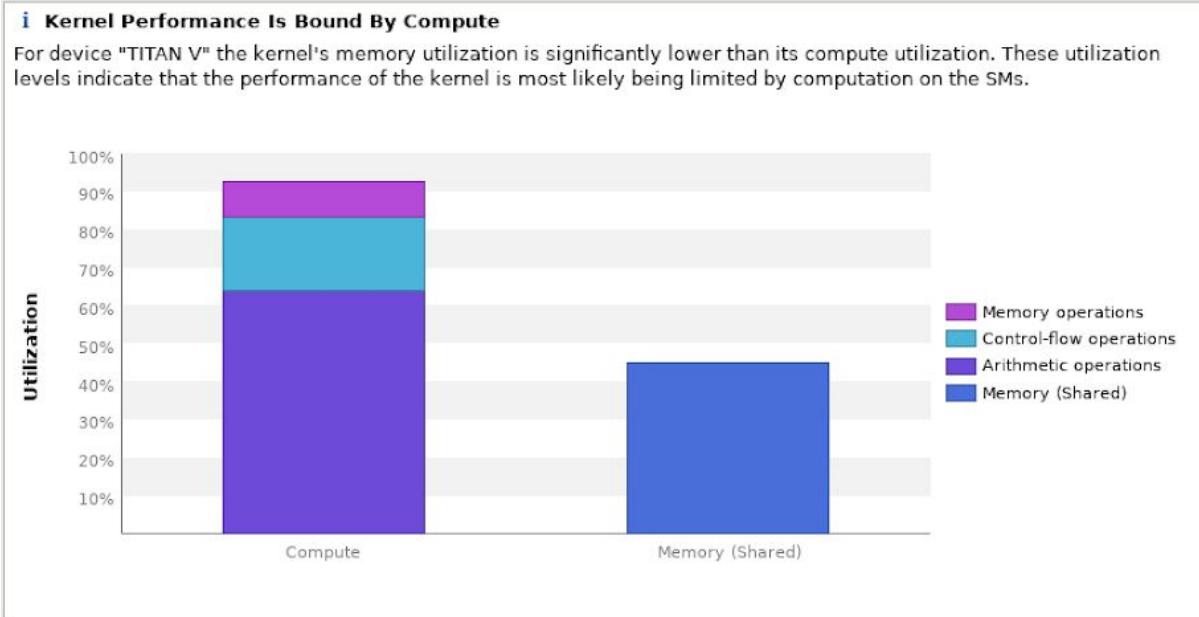


- Forward Layer 1

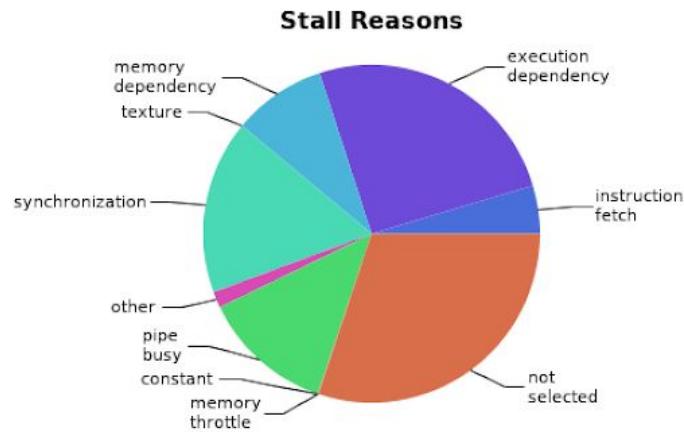


Results				
ACTIVE BLOCKS		0	32	
Active Warps	45.64	48	64	
Active Threads		1536	2048	
Occupancy	71.3%	75%	100%	
Warp				
Threads/Block		256	1024	
Warps/Block		8	32	
Block Limit		8	32	
Registers				
Registers/Thread		39	65536	
Registers/Block		10240	65536	
Block Limit		6	32	
Shared Memory				
Shared Memory/Block		1700	98304	
Block Limit		54	32	

- Forward Layer 2



**Results**  
 potentially be reduced by increasing instruction-level parallelism.  
 Texture - The texture sub-system is fully utilized or has too many outstanding requests.  
 Synchronization - The warp is blocked at a `_syncthreads()` call.  
 Constant - A constant load is blocked due to a miss in the constants cache.  
 Memory Dependency - A load/store cannot be made because the required resources are not available or are fully utilized, or too many requests of a given type are outstanding. Data request stalls can potentially be reduced by optimizing memory alignment and access patterns.



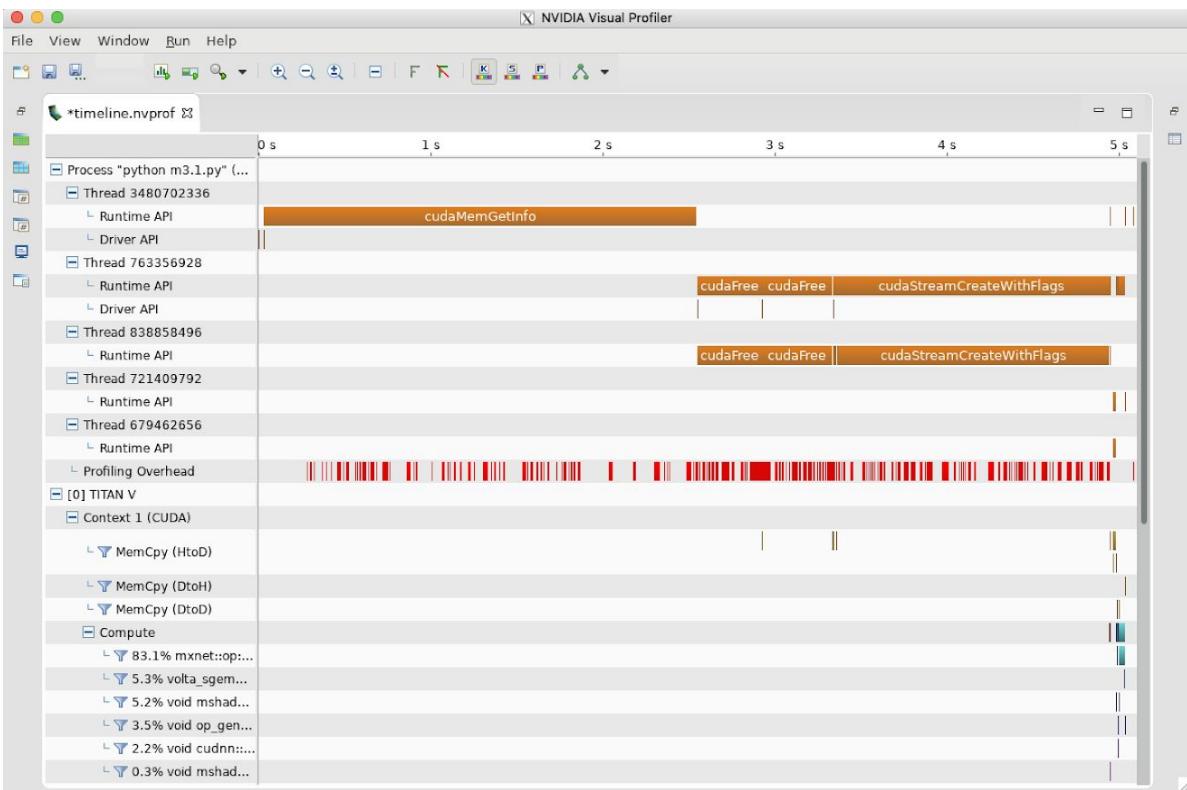
ACTIVE BLOCKS			
Active Warps	47.54	48	64
Active Threads		1536	2048
Occupancy	74.3%	75%	100%
<b>Warp</b>			
Threads/Block		256	1024
Warps/Block		8	32
Block Limit		8	32
<b>Registers</b>			
Registers/Thread		39	65536
Registers/Block		10240	65536
Block Limit		6	32
<b>Shared Memory</b>			
Shared Memory/Block		1700	98304
Block Limit		54	32

## Optimization #3: Tuning with Restrict and Loop Unrolling

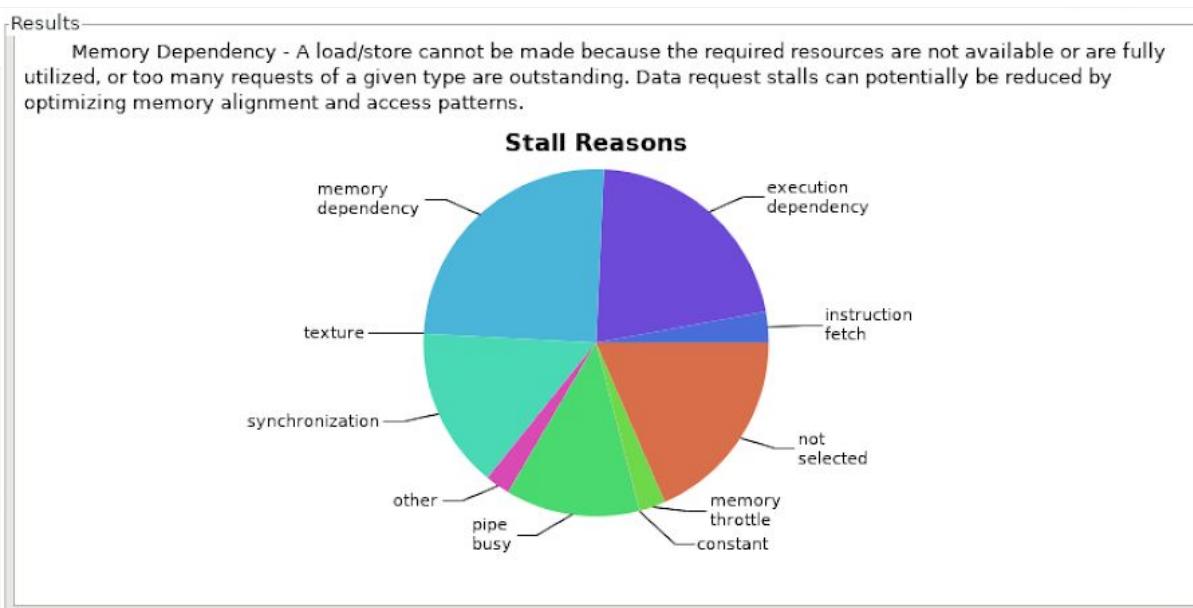
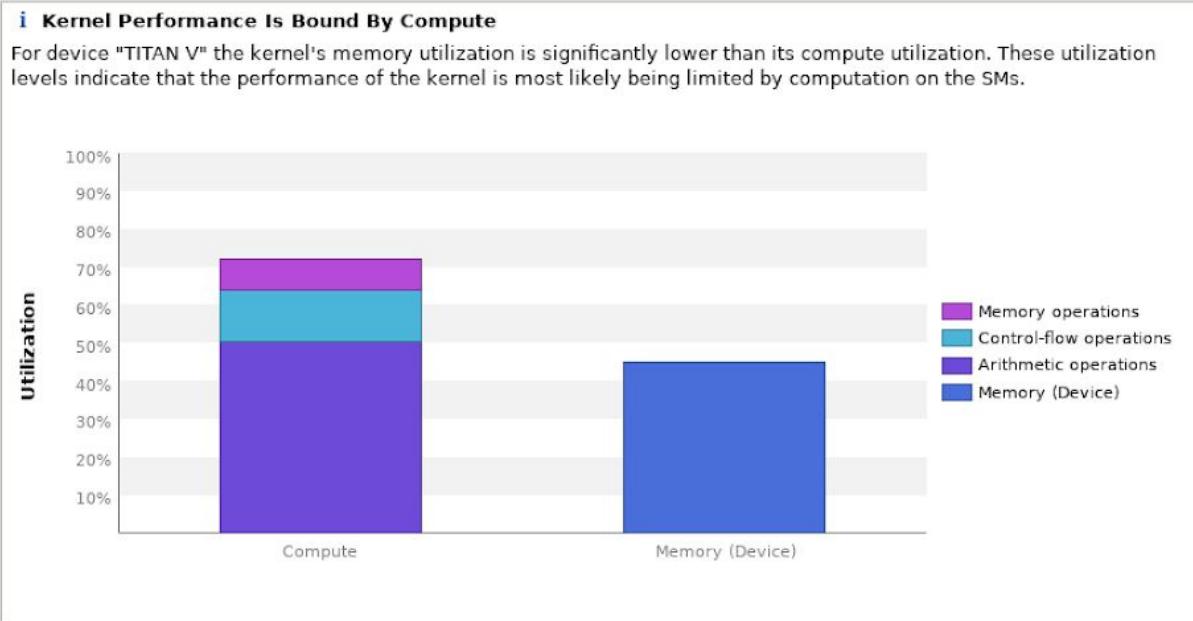
- **Description**

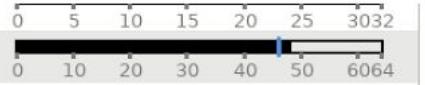
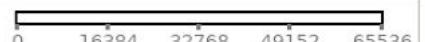
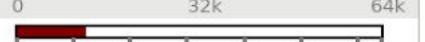
We tune our implementation by adding the *restrict* keyword to our arrays. That is, we add `__restrict__` for our x, y, and w arrays. According to an article from Nvidia (<https://devblogs.nvidia.com/cuda-pro-tip-optimize-pointer-aliasing/>), adding this keyword to our arrays “greatly helps the compiler optimize code”. Along with this, in CUDA, using the restrict keyword caches our values so that we have faster data accesses. We also perform loop unrolling in our kernel. We unroll so that we use less threads but each thread ends up performing more work. In our code, we unroll five times when calculating our acc value.

- **Timeline Overview**

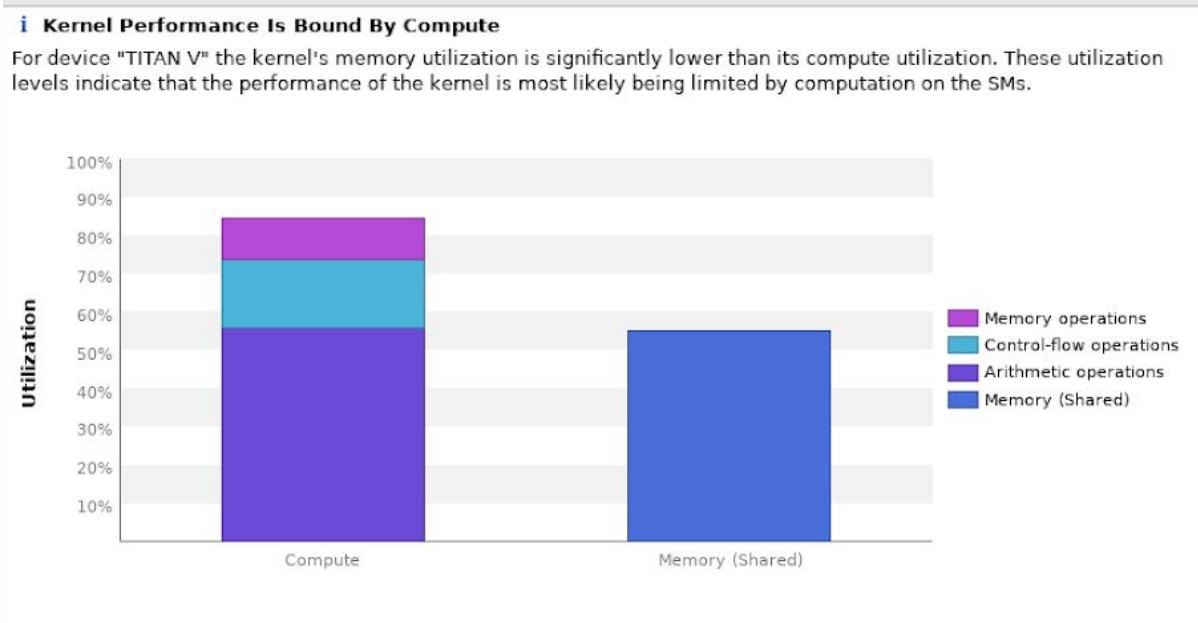


- Forward Layer 1



Results				
ACTIVE BLOCKS		0	32	
Active Warps	45.64	48	64	
Active Threads		1536	2048	
Occupancy	71.3%	75%	100%	
WarpS				
Threads/Block		256	1024	
Warps/Block		8	32	
Block Limit		8	32	
Registers				
Registers/Thread		39	65536	
Registers/Block		10240	65536	
Block Limit		6	32	
Shared Memory				
Shared Memory/Block		1700	98304	
Block Limit		54	32	

- Forward Layer 2



## Results

potentially be reduced by increasing instruction-level parallelism.

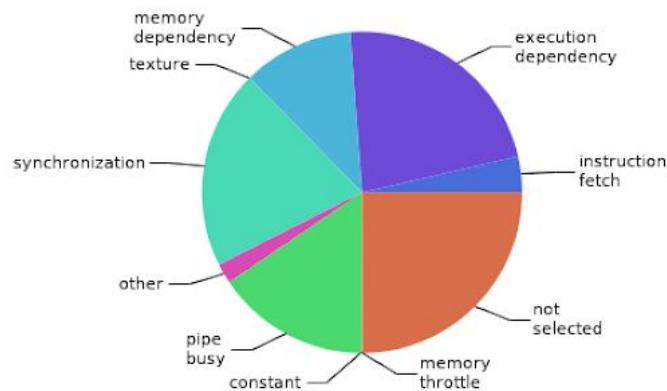
Texture - The texture sub-system is fully utilized or has too many outstanding requests.

Synchronization - The warp is blocked at a `_syncthreads()` call.

Constant - A constant load is blocked due to a miss in the constants cache.

Memory Dependency - A load/store cannot be made because the required resources are not available or are fully utilized, or too many requests of a given type are outstanding. Data request stalls can potentially be reduced by optimizing memory alignment and access patterns.

### Stall Reasons



## Results

ACTIVE BLOCKS			
Active Warps	47.47	48	64
Active Threads		1536	2048
Occupancy	74.2%	75%	100%
Warp			
Threads/Block		256	1024
Warps/Block		8	32
Block Limit		8	32
Registers			
Registers/Thread		39	65536
Registers/Block		10240	65536
Block Limit		6	32
Shared Memory			
Shared Memory/Block		1700	98304
Block Limit		54	32

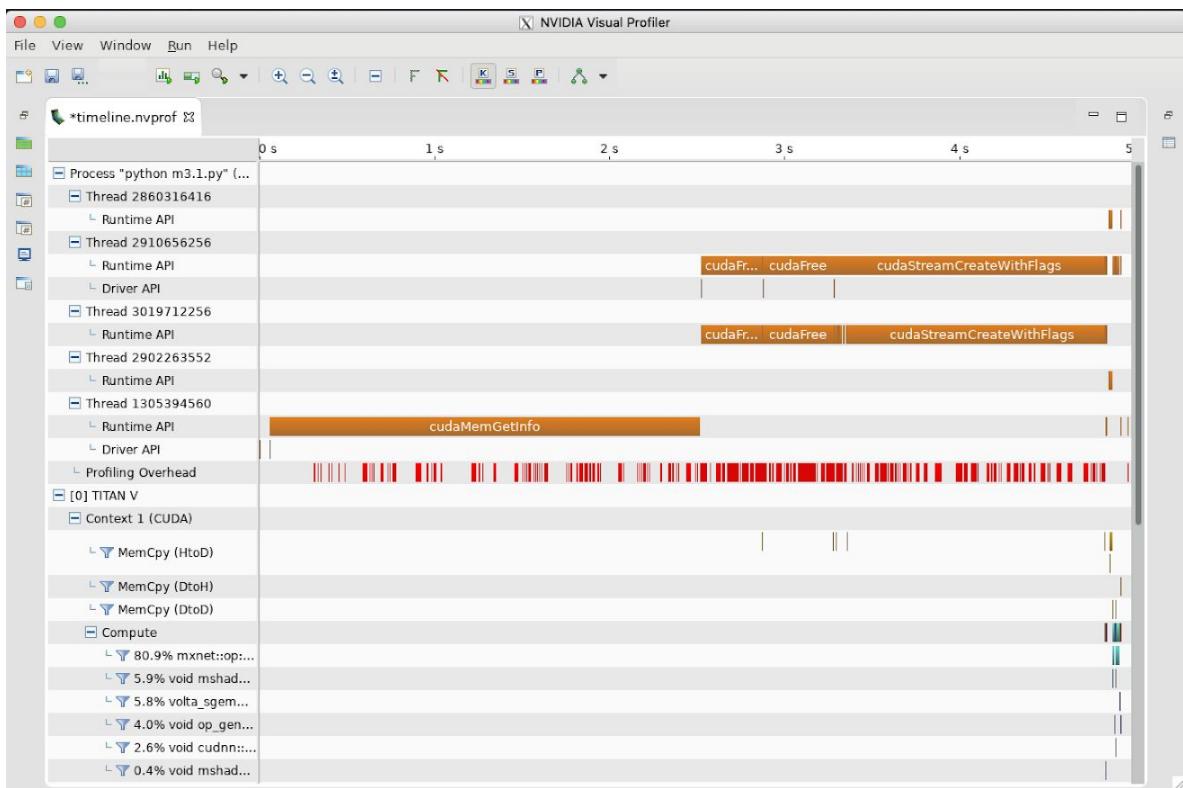
# FINAL MILESTONE

## Optimization #4: Unroll + Shared-memory Matrix Multiply

- **Description**

For this optimization, we performed matrix multiplication that is similar to our MP3 assignment. This optimization requires creating a new matrix, which we call `x_unrolled`. We call our unrolling kernel  $B$  times to unroll our  $X$  into an array suitable for multiplication. Furthermore, we call our `matrixMultiply` kernel after every unroll kernel. We load `x_unrolled` and weights into shared memory (using tiling) and then we compute our outputs from the partial products of the tiles.

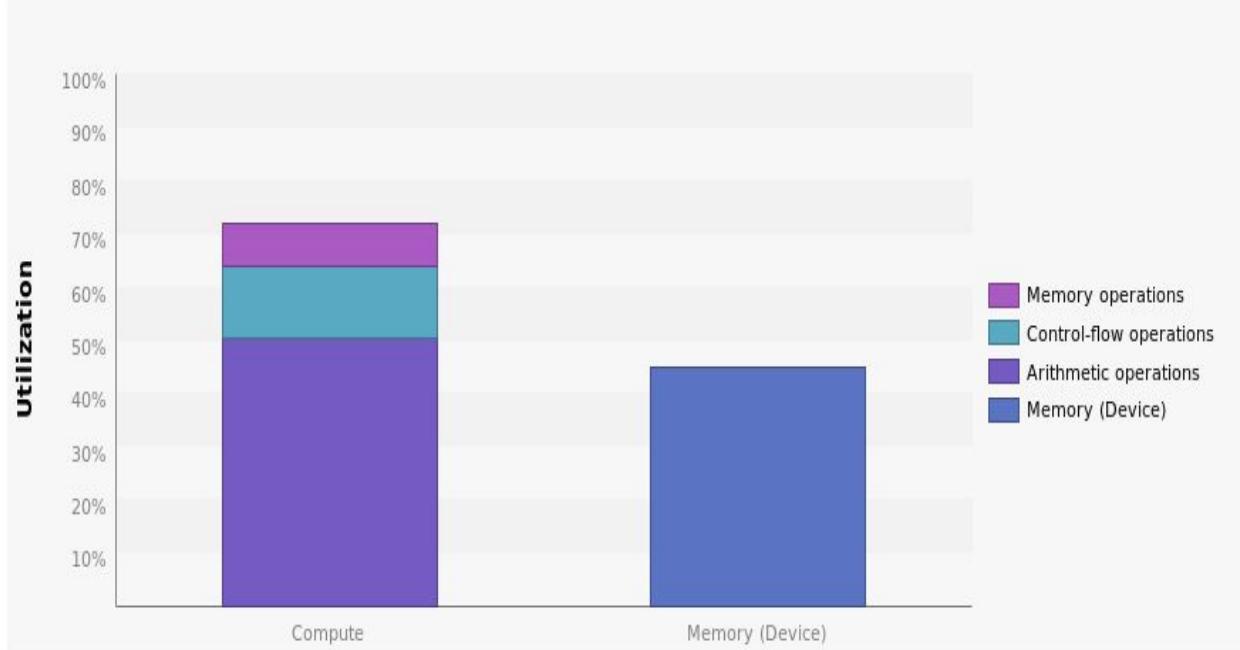
- **Timeline Overview**



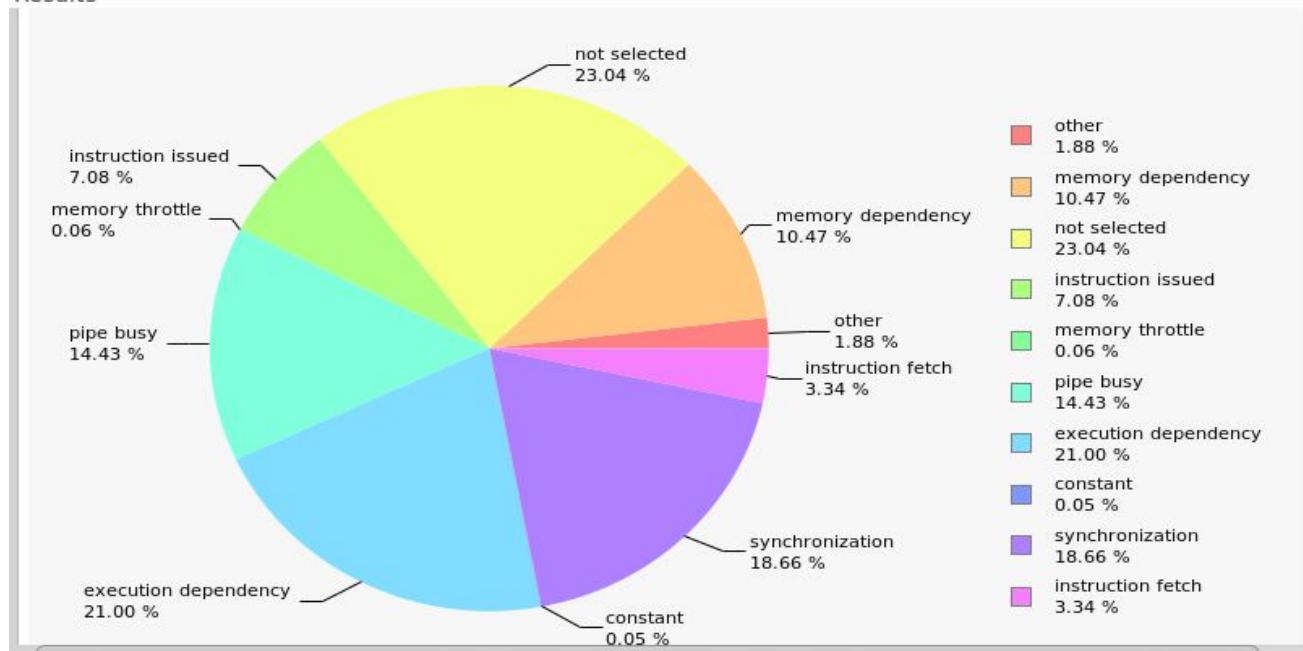
- Forward Layer 1

### i Kernel Performance Is Bound By Compute

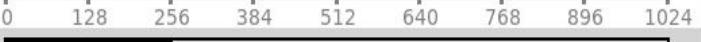
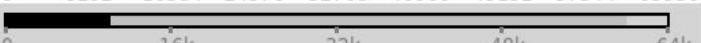
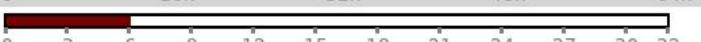
For device "TITAN V" the kernel's memory utilization is significantly lower than its compute utilization. These utilization levels indicate that the performance of the kernel is most likely being limited by computation on the SMs.



### Results



## Results

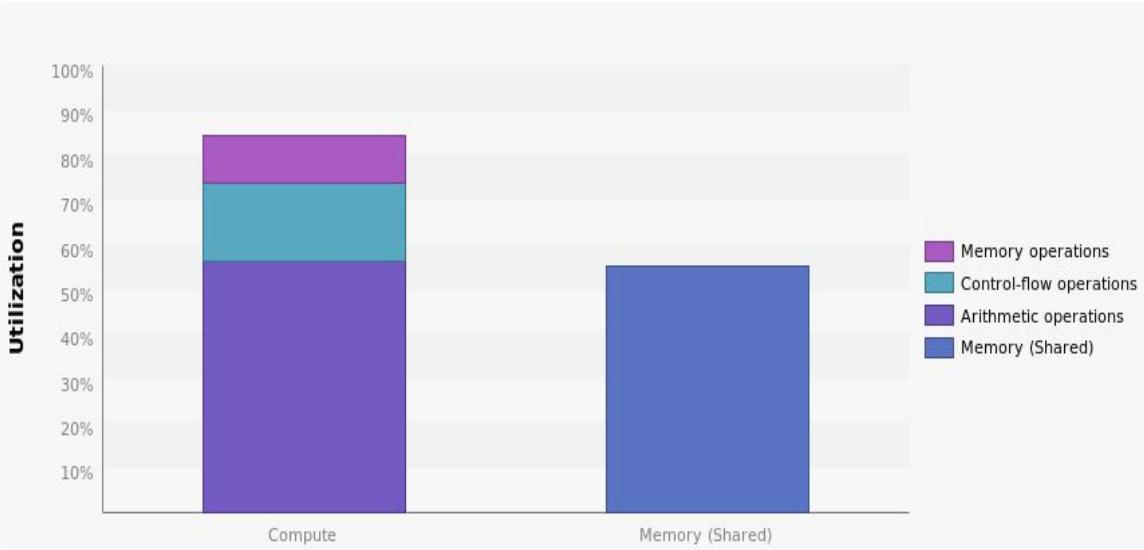
Active Threads		1536	2048	
Occupancy	74.3%	75%	100%	
<b>Warp</b>				
Threads/Block		256	1024	
Warps/Block		8	32	
Block Limit		8	32	
<b>Registers</b>				
Registers/Thread		39	65536	
Registers/Block		10240	65536	
Block Limit		6	32	
<b>Shared Memory</b>				
Shared Memory/Block		1700	98304	
Block Limit		54	32	

- **Forward Layer 2**

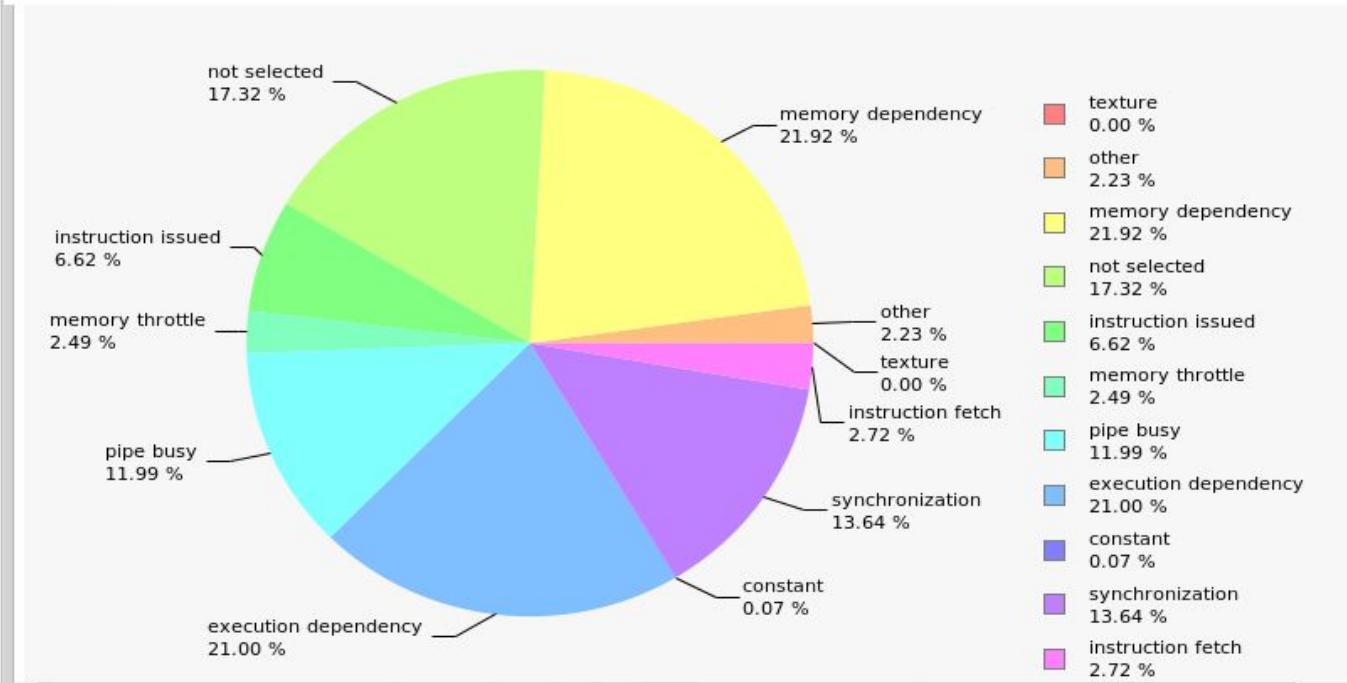
## Results

### **i Kernel Performance Is Bound By Compute**

For device "TITAN V" the kernel's memory utilization is significantly lower than its compute utilization. These utilization levels indicate that the performance of the kernel is most likely being limited by computation on the SMs.



## Results



## Results

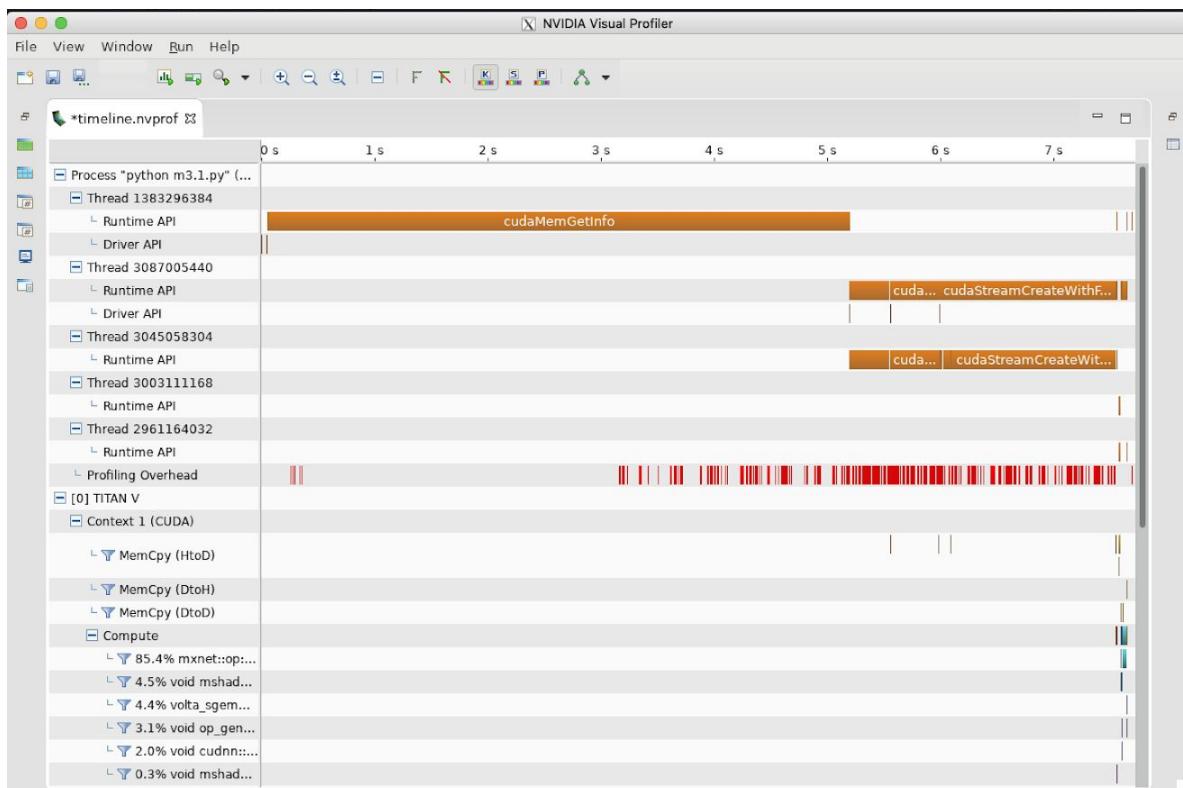
Active Threads		1536	2048	
Occupancy	74.3%	75%	100%	
<b>Warp</b>				
Threads/Block		256	1024	
Warps/Block		8	32	
Block Limit		8	32	
<b>Registers</b>				
Registers/Thread		39	65536	
Registers/Block		10240	65536	
Block Limit		6	32	
<b>Shared Memory</b>				
Shared Memory/Block		1700	98304	
Block Limit		54	32	

## Optimization #5: Kernel Fusion For Unrolling & Matrix-Multiply

- **Description**

We used kernel fusion to further optimize unrolling and general matrix multiplication. Originally, we did unrolling and matrix multiplication separately. By doing this, we need to use a global array to hold intermediate unrolling matrix, which involves global memory reads and writes of high latency. Therefore, we use kernel fusion by combining unrolling and matrix multiplication into one step to avoid use of global memory. We now will unroll the input for one tile before title matrix multiplication, instead of unrolling input all at once. This optimization works well and we got the right correctness of 83.97 % and gave us about 10 times speed up compared to the original one. This could mainly be due to use of faster shared memory to replace use of slow global memory.

- **Timeline Overview**

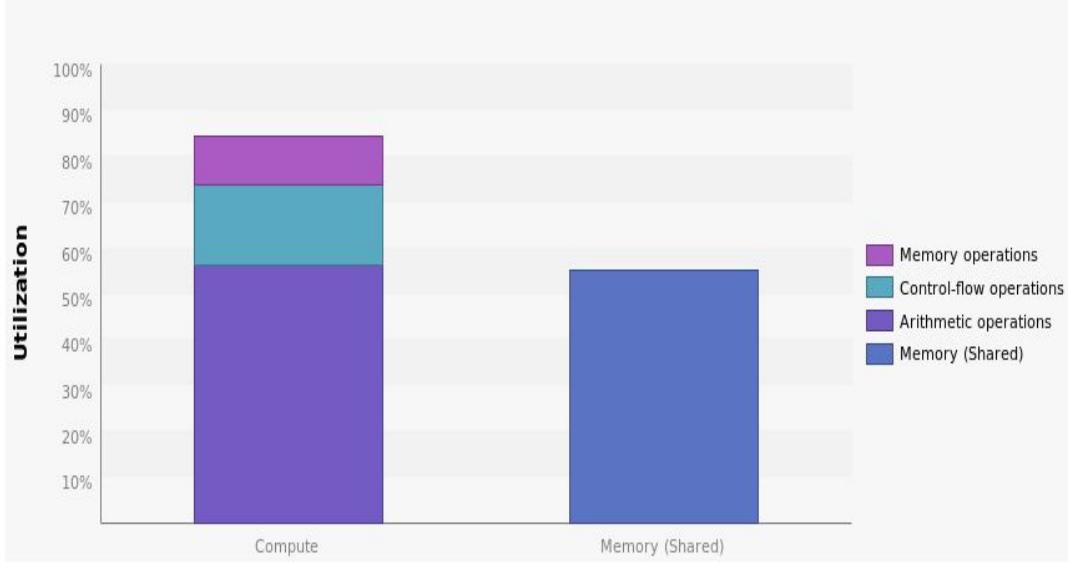


## • Forward Layer 1

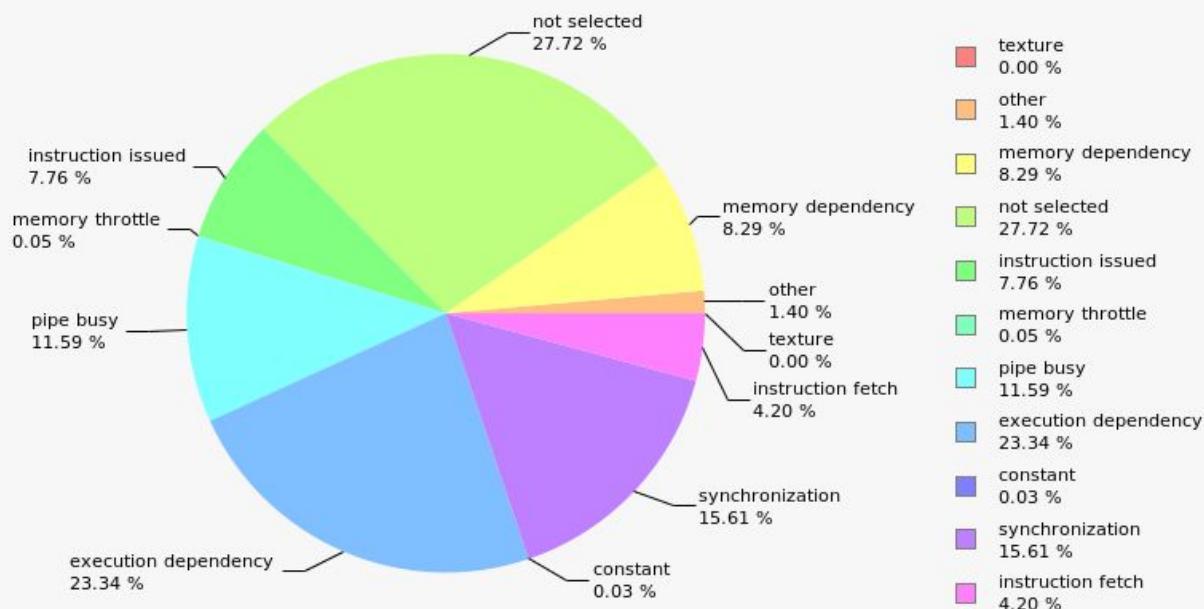
Results

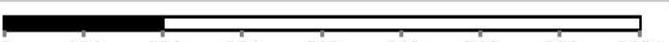
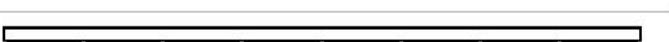
### **i Kernel Performance Is Bound By Compute**

For device "TITAN V" the kernel's memory utilization is significantly lower than its compute utilization. These utilization levels indicate that the performance of the kernel is most likely being limited by computation on the SMs.

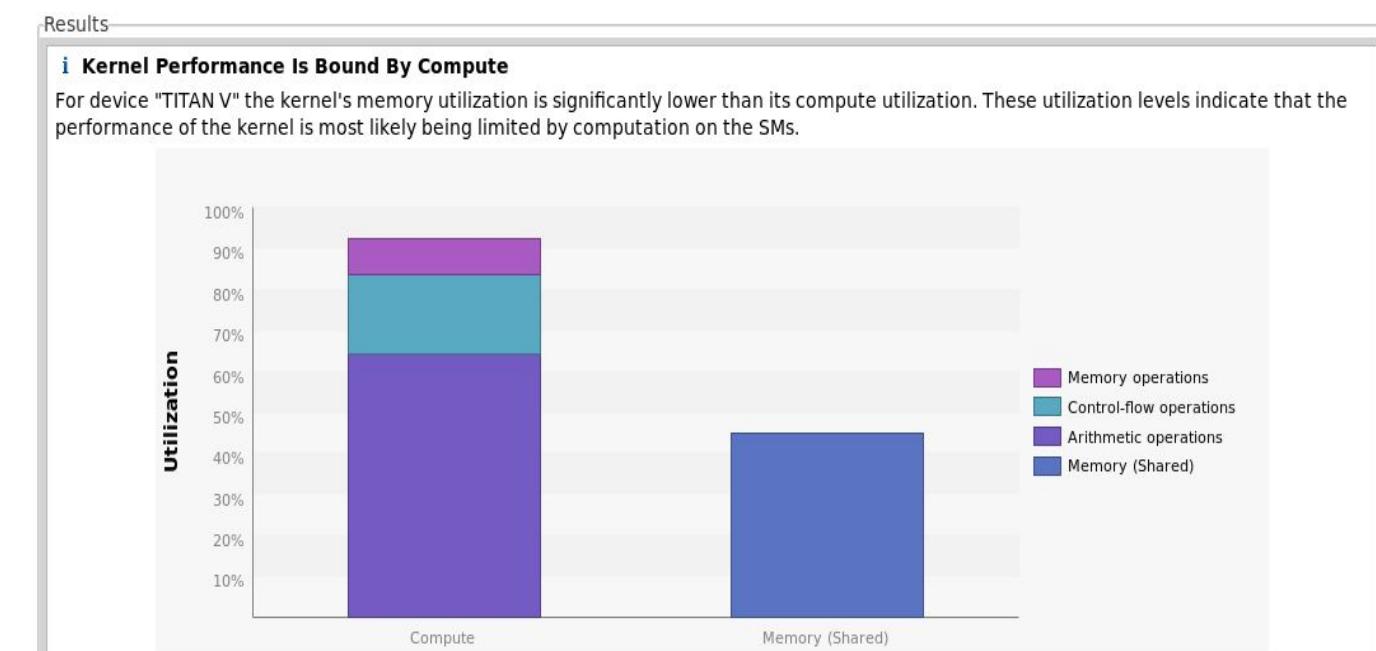


Results

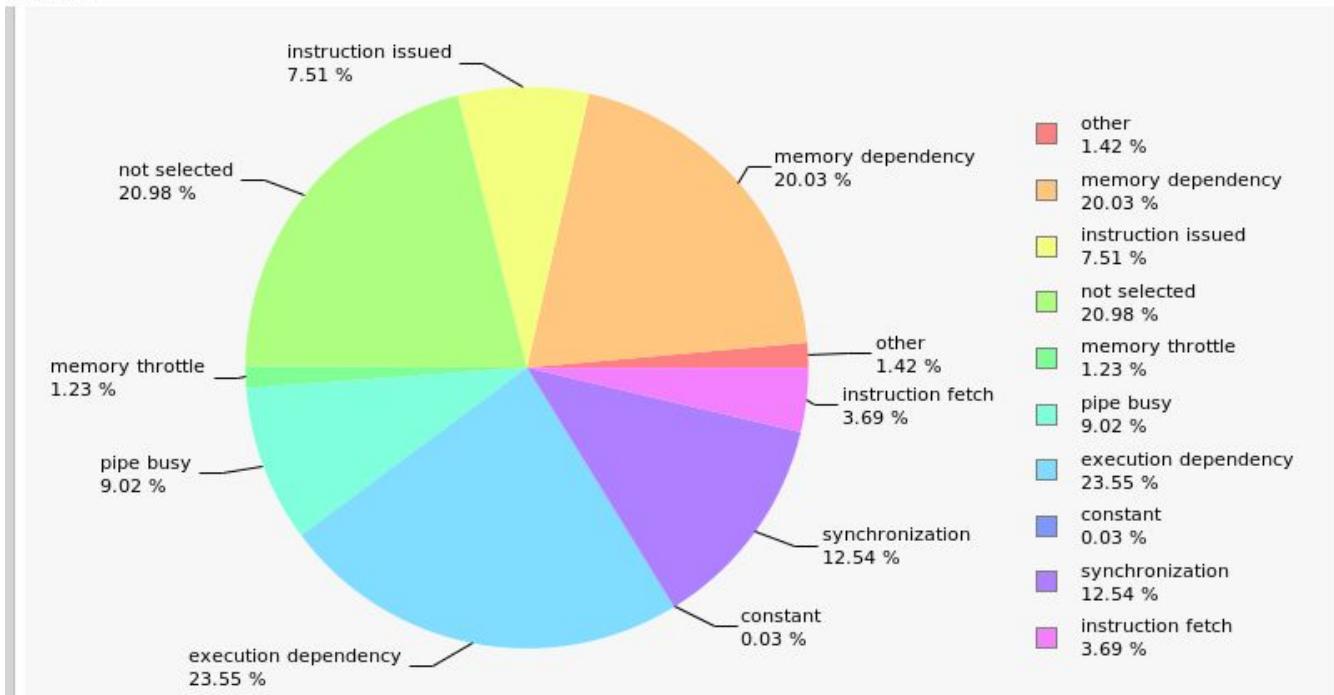


Results			
Active Threads	1536	2048	
Occupancy	71.3%	75%	100% 
Warps			
Threads/Block	256	1024	
Warps/Block	8	32	
Block Limit	8	32	
Registers			
Registers/Thread	39	65536	
Registers/Block	10240	65536	
Block Limit	6	32	
Shared Memory			
Shared Memory/Block	1700	98304	
Block Limit	54	32	

- Forward Layer 2



## Results



## Results

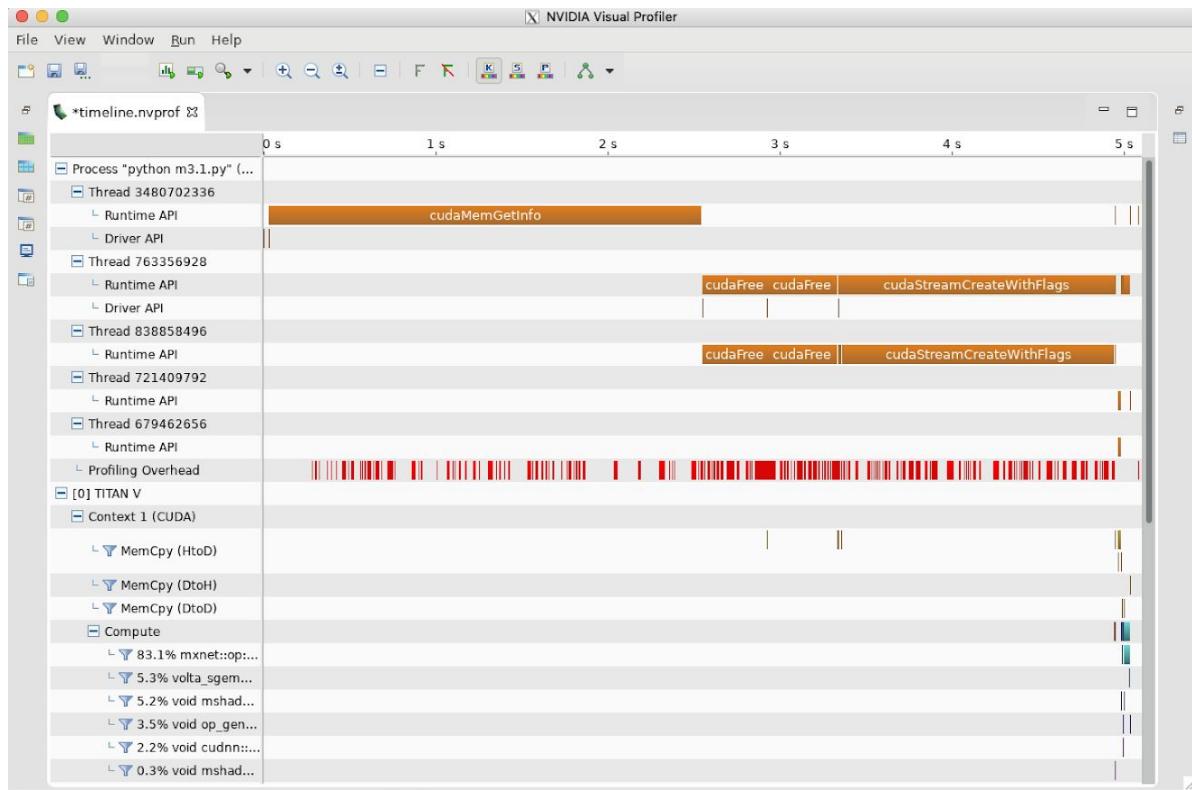
Active Threads		1536	2048	
Occupancy	71.3%	75%	100%	
<b>Warps</b>				
Threads/Block		256	1024	
Warps/Block		8	32	
Block Limit		8	32	
<b>Registers</b>				
Registers/Thread		39	65536	
Registers/Block		10240	65536	
Block Limit		6	32	
<b>Shared Memory</b>				
Shared Memory/Block		1700	98304	
Block Limit		54	32	

## Optimization #6: Multiple Kernel Implementations for Different Layer Sizes

- **Description**

This optimization combines the matrix multiplication kernel, the original kernel, and the fusion kernel into a single file. To do this, we check the parameters before calling our kernel. If our batch size is greater than or equal to 1000, then we copy the weights to constant memory and perform our original kernel, since it runs larger batches faster. Next, we check if the product of our batch size, output features, and filters ( $B \times M \times C$ ) is larger than 10000, we copy our weights to constant memory again, albeit in a different way since we end up executing our fused matrix multiplication kernel in this scenario. Finally, in all other cases, we run our unrolled kernel and matrix multiplication kernel that utilizes shared memory. These cases use the matrix multiplication and unrolling kernels because it ensures that our layer is small enough such that we can call B kernels efficiently.

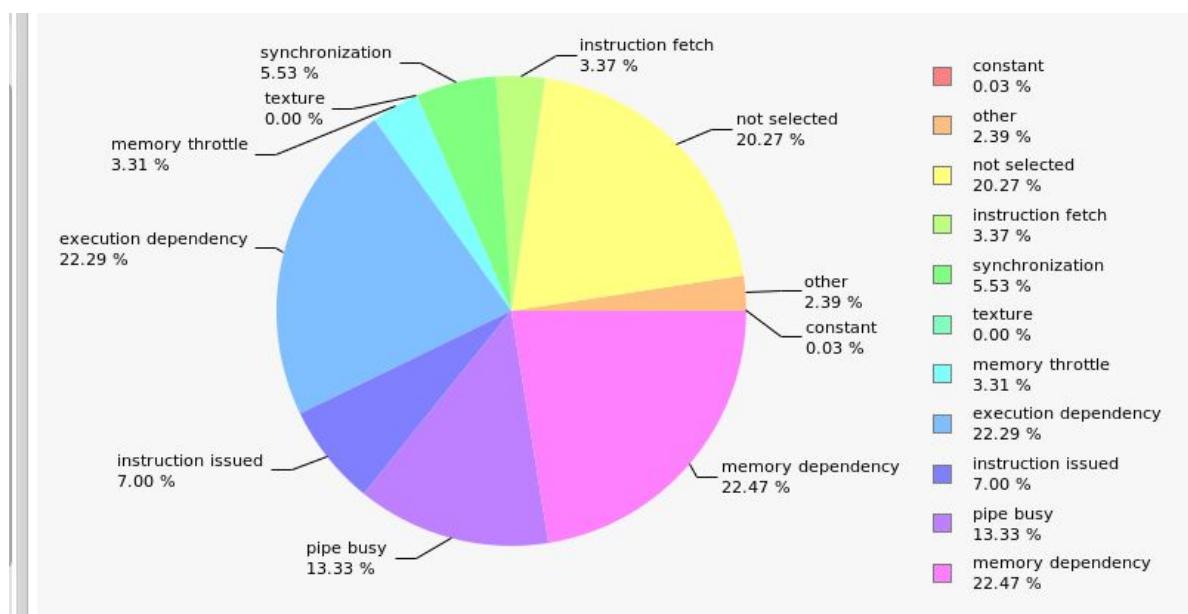
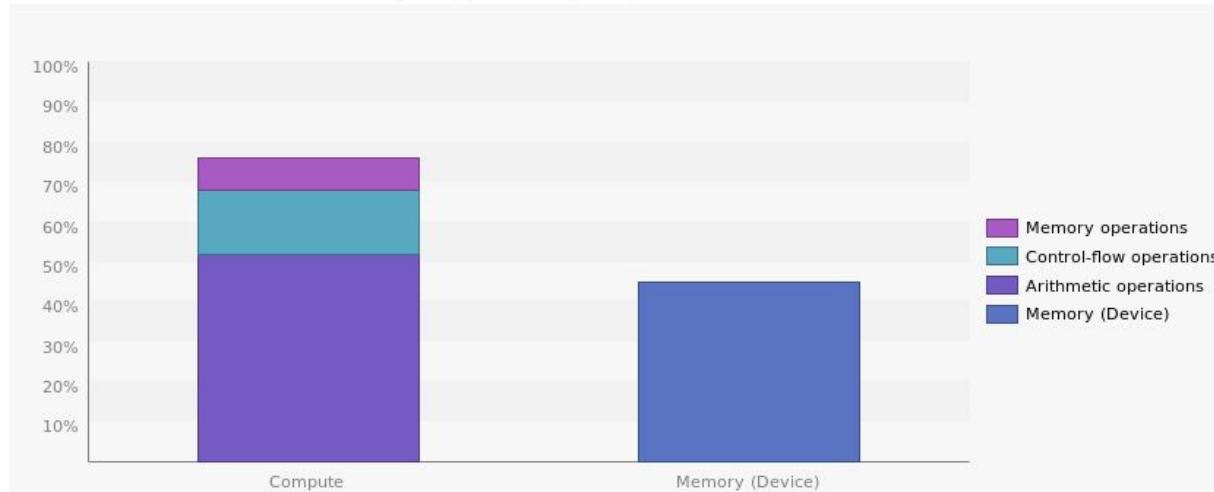
- **Timeline Overview**



## • Forward Layer 1

### Performance Is Bound By Compute

"TITAN V" the kernel's memory utilization is significantly lower than its compute utilization. These utilization levels indicate that performance of the kernel is most likely being limited by computation on the SMs.

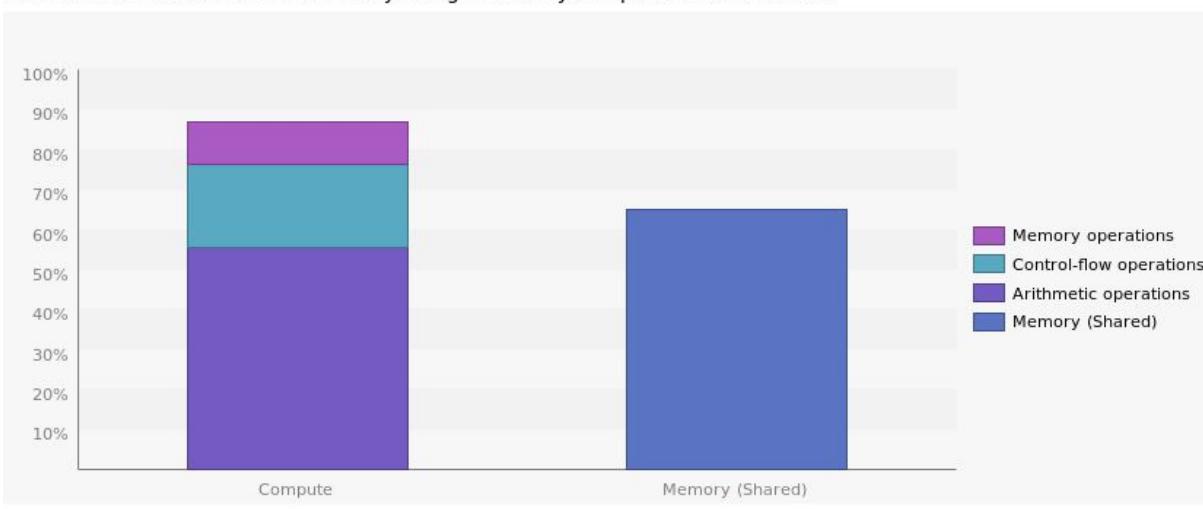


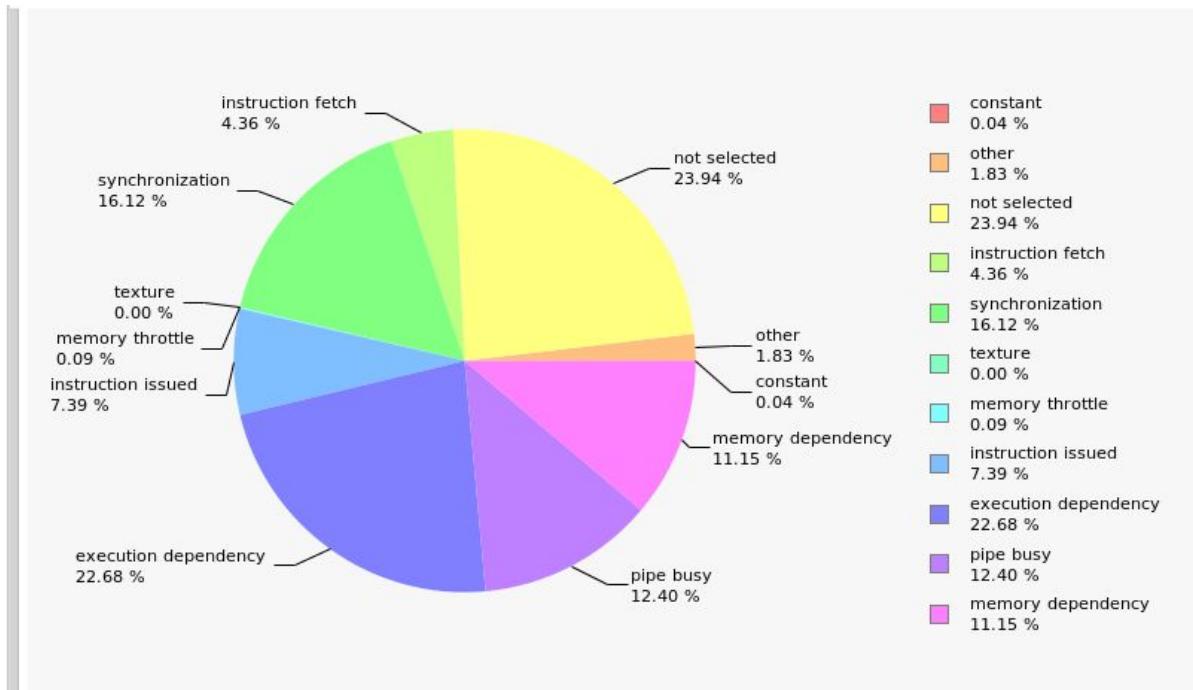
Results			
Active Warps	45.13	48	64
Active Threads		1536	2048
Occupancy	70.5%	75%	100%
Warp			
Threads/Block		256	1024
Warps/Block		8	32
Block Limit		8	32
Registers			
Registers/Thread		40	65536
Registers/Block		10240	65536
Block Limit		6	32
Shared Memory			
Shared Memory/Block		1600	98304
Block Limit		54	32

- Forward Layer 2

#### Performance Is Bound By Compute

"TITAN V" the kernel's memory utilization is significantly lower than its compute utilization. These utilization levels indicate that performance of the kernel is most likely being limited by computation on the SMs.





Results			
Active Warps	47.4	48	64
Active Threads		1536	2048
Occupancy	74.1%	75%	100%
Warps			
Threads/Block		256	1024
Warps/Block		8	32
Block Limit		8	32
Registers			
Registers/Thread		40	65536
Registers/Block		10240	65536
Block Limit		6	32
Shared Memory			
Shared Memory/Block		1600	98304
Block Limit		54	32

## Run Times For Final 3 Optimizations:

### Optimization #4: Unroll + Shared-memory Matrix Multiply

```
* Running /usr/bin/time nvprof -o timeline.nvprof python final.py
Loading fashion-mnist data...
done
Loading model...
==287== NVPROF is profiling process 287, command: python final.py
done
New Inference
Op Time: 0.131282
Op Time: 0.148911
Correctness: 0.8397 Model: ece408
==287== Generated result file: /build/timeline.nvprof
6.07user 4.12system 0:12.02elapsed 84%CPU (0avgtext+0avgdata 4717260maxresident)k
0inputs+17280outputs (0major+1115957minor)pagefaults 0swaps
```

### Optimization #5: Kernel Fusion For Unrolling & Matrix-Multiply

```
* Running /usr/bin/time nvprof -o timeline.nvprof python final.py
Loading fashion-mnist data...
done
Loading model...
==287== NVPROF is profiling process 287, command: python final.py
done
New Inference
Op Time: 0.012452
Op Time: 0.014506
Correctness: 0.8397 Model: ece408
==287== Generated result file: /build/timeline.nvprof
7.77user 4.93system 0:16.54elapsed 76%CPU (0avgtext+0avgdata 4712152maxresident)k
0inputs+6008outputs (0major+1114023minor)pagefaults 0swaps
```

## Optimization #6: Multiple Kernel Implementations for Different Layer Sizes

```
* Running /usr/bin/time nvprof -o timeline.nvprof python final.py
Loading fashion-mnist data...
done
Loading model...
==287== NVPROF is profiling process 287, command: python final.py
done
New Inference
Op Time: 0.005850
Op Time: 0.027449
Correctness: 0.8397 Model: ece408
==287== Generated result file: /build/timeline.nvprof
5.46user 3.86system 0:06.29elapsed 148%CPU (0avgtext+0avgdata 4715624maxresident)k
0inputs+5960outputs (0major+1114876minor)pagefaults 0swaps
```