Team Name: hwu10clan
Team members: Bryan Hwu / Abhinav Shah
NetIds: hwu10 / ashah101
ECE 408
Final Project Report

Baseline Results

M1.1

Elapsed time for the whole python program: 7.57 seconds.

M1.2

Elapsed time for the whole python program: 2.56 seconds.

M1.3

Table of Most Time Consuming Kernels Sorted by Total Time

Kernel Call	Total Time	Calls	Avg Time
cudnn::detail::implicit_convolve_sgemm	50.440ms	1	50.440ms
sgemm_sm35_ldg_tn_128x8x256x16x32	39.215ms	1	39.215ms
void cudnn::detail::activation_fw_4d_kernel	19.390ms	2	9.6952ms
cudnn::detail::pooling_fw_4d_kernel	14.502ms	1	14.502ms
[CUDA memcpy HtoD]	6.3152ms	13	485.78us
sgemm_sm35_ldg_tn_64x16x128x8x32	3.6539ms	1	3.6539ms
mshadow::cuda::SoftmaxKernel	1.1189ms	1	1.1189ms
mshadow::cuda::MapPlanKernel	755.15us	12	62.929us
mshadow::cuda::MapPlanKernel	436.09us	2	218.04us
sgemm_sm35_ldg_tn_32x16x64x8x16	391.26us	1	391.26us
mshadow::cuda::MapPlanKernel	23.327us	1	23.327us
[CUDA memcpy DtoH]	9.8870us	1	9.8870us

There is a marked drop in the average time per call after SoftmaxKernel. Any kernel from the cudnn seems to be related to the convolution. The most obvious kernel would be the most time intensive, implicit_convolve_sgemm. Kernels involved in activation and pooling seem to be taking up a lot of time as well.

Please find the full profile generated in the Appendix.

M2.1

Table of Correctness and Time:

Model	Size	Correctness	Time
ece408-high	10,000	.8562	9.076
Ece408-high	10,000	.629	9.412313

M3.1

Table of Correctness and Time:

Model	Size	Correctness	Time
ece408-high	10,000	.8562	0.448293 sec.

The full nvprof profile for 3.1 is in the Appendix.

Optimization Approach and Results

First Steps

Before we began to optimize our existing code, we decided to take a holistic view of how our forward path of the convolutional layer was designed. We quickly realized that the input and output image dimensions were surprisingly small, being 28x28 and 24x24. Because of this, we determined that we could easily map each thread to an output pixel. Then each thread block could be given an input image set X and be tasked with generating all 50 output maps in Y. We then re-designed our kernel as such and launched B number of blocks to take care of every X in the batch. With this baseline implementation, we managed to reach 222 ms before any sort of optimization.

Optimizing Filter Bank Optimization

Next, we saw that in our kernel, each thread is iterating over each output image and performing the 5x5 convolution. Thus there was a total of 1,250 iterations occuring for each thread. Any

access or calculation in the innermost loop would be extremely costly. Because of that, we decided to pass in the filter banks as constant memory because we never modify it. To do so, we had to take in the GPU tensor input to the Host code, cudaMemCpy it to a host variable, then perform a cudaMemCpyToSymbol to the global constant variable. This optimization brought us down to 167 ms.

Shared memory Optimization

In addition to the filter banks, we were also accessing the input X array in every iteration of the kernel. Thus we first loaded X into shared memory and performed all our arithmetic in shared memory. This allowed us to break 100 ms and reach 76 ms.

Improved indexing

Next, we noticed that we were calculating the index of shared memory every single time when iterating through the for loop. This was costly because we had to perform a GPU multiplication and add in each iteration, which is extremely costly. We then found a way to calculate the starting index of the convolution and use the filter bank iterating variables to offset from that starting index. We then reached around 50 ms.

Create multiple output images

Then we noticed that every thread accesses the same X index in every iteration. Thus to save on index calculation arithmetic and reduce the number of outer iterations on the number of output images, we calculated the X starting indices of 7 input images. Because of this we reduced the number of outer iterations each thread performs from 50 to 7. We then manually performed the last iteration after the loops. We determined how many indices to calculate by trying to max out the number of thread block registers we could use. We tried calculating 10 indices, but that proved to slow down code because we were over using registers. We reached 19 ms with this optimization.

Taking Advantage of Constants

One of our final optimizations was that we were told that the input images in each X, output images in each Y, filter bank count and size, and input and output image dimensions were all constant. We then hardcoded all these variables using numerical values. Then we eliminated any unnecessary variables to bring down our register count. Because of this, we brought our total competition time to 11.5. Finally, we used our reduced register count to calculate more input image indices and bring our output image iteration count down. We reached 11.0 as our final optimized time.

Additional Resources

As we optimized our kernel, we used Nvidia Visual Profiler (NVVP) to help us figure out the computational and time intensive aspects of our code and thus where to focus our attention. Whenever we made an optimization, we checked NVVP to see exactly what sort of benefits it was giving us. For us, the Kernel Memory and Kernel Profile -- Instruction Execution tools were the most helpful. The first showed us the total counts for local, shared, and global memory accesses and bandwidth. The second gave us percentages of how efficient the warps were in terms of control divergence. See Appendix D for a side by side comparison of our baseline implementation and the final optimized one using the two tools described above.

The full nvprof profile for the final optimized code is in the Appendix C.

Team Member Contributions

Bryan and Abhi worked together for both milestones 1, 2, and 3, and the final optimization.

References

None

Suggestions

None

Appendix

A. Full profile generated for M1.2

==312== Profiling application: python m1.2.py
==312== Profiling result:
Time(%) Time Calls Avg Min Max Name
37.02% 50.440ms 1 50.440ms 50.440ms 50.440ms void cudnn::detail::implicit_convolve_sgemm<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, cudnn::detail::implicit_convolve_sgemm<float, int=1024, int=5, int=5, int=3, int=3, int=3, int=1, bool=1, bool=0, bool=1>*, float const *, kernel_conv_params, int, float, float, int, float const *, float const *, int, int)
28.78% 39.215ms 1 39.215ms 39.215ms 39.215ms sgemm_sm35_ldg_tn_128x8x256x16x32
14.23% 19.390ms 2 9.6952ms 461.72us 18.929ms void cudnn::detail::activation_fw_4d_kernel<float, float, int=128, int=1, int=4, cudnn::detail::tanh_func<float>>(cudnnTensorStruct, float const *, cudnn::detail::activation_fw_4d_kernel<float, float, int=128, int=1, int=4, cudnn::detail::tanh_func<float>>, cudnnTensorStruct*, float, cudnnTensorStruct*, int, cudnnTensorStruct*)
10.64% 14.502ms 1 14.502ms 14.502ms 14.502ms void cudnn::detail::pooling_fw_4d_kernel<float, float, cudnn::detail::maxpooling_func<float, cudnnNanPropagation_t=0>, int=0>(cudnnTensorStruct, float const *, cudnn::detail::pooling_fw_4d_kernel<float, float, cudnn::detail::maxpooling_func<float, cudnnNanPropagation_t=0>, int=0>, cudnnTensorStruct*, cudnnPoolingStruct, float, cudnnPoolingStruct, int, cudnn::reduced_divisor, float)
4.63% 6.3152ms 13 485.78us 1.5360us 4.3623ms [CUDA memcpy HtoD]
2.68% 3.6539ms 1 3.6539ms 3.6539ms 3.6539ms sgemm_sm35_ldg_tn_64x16x128x8x32
0.82% 1.1189ms 1 1.1189ms 1.1189ms 1.1189ms 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)
0.55% 755.15us 12 62.929us 2.1120us 381.14us void mshadow::cuda::MapPlanKernel<mshadow::sv::saveto, int=8, 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)
0.32% 436.09us 2 218.04us 16.639us 419.45us void mshadow::cuda::MapPlanKernel<mshadow::sv::plusto, int=8, mshadow::expr::Plan<mshadow::Tensor<mshadow::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 int, mshadow::Shape<int=2>, int=2)
  0.29% 391.26us   1 391.26us 391.26us 391.26us sgemm_sm35_ldg_tn_32x16x64x8x16
  0.02% 23.327us   1 23.327us 23.327us 23.327us void mshadow::cuda::MapPlanKernel<mshadow::sv::saveto, int=8, mshadow::expr::Plan<mshadow::Tensor<mshadow::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)
  0.01% 9.8870us   1 9.8870us 9.8870us 9.8870us [CUDA memcpy DtoH]
==312== API calls:
Time(%)   Time   Calls   Avg   Min   Max Name
 46.64% 1.86710s   18 103.73ms 18.616us 933.18ms cudaStreamCreateWithFlags
 28.53% 1.14222s   10 114.22ms   859ns 326.63ms cudaFree
 20.87% 835.55ms   24 34.815ms 236.10us 828.42ms cudaMemGetInfo
  3.23% 129.11ms   25 5.1645ms 5.1540us 83.821ms cudaStreamSynchronize
  0.30% 12.182ms   8 1.5228ms 15.746us 4.4472ms cudaMemcpy2DAsync
  0.22% 8.6119ms   42 205.05us 10.526us 1.7504ms cudaMalloc
  0.09% 3.7837ms   4 945.92us 34.946us 3.6375ms cudaStreamCreate
  0.04% 1.4046ms   4 351.14us 337.85us 387.55us cuDeviceTotalMem
  0.02% 897.40us   352 2.5490us   261ns 79.948us cuDeviceGetAttribute
  0.02% 608.57us   23 26.459us 12.646us 122.94us cudaLaunch
  0.01% 557.58us   114 4.8910us 1.0100us 66.520us cudaEventCreateWithFlags
  0.01% 477.79us   6 79.631us 31.799us 148.17us cudaMemcpy
  0.01% 382.47us   2 191.23us 24.555us 357.91us cudaStreamCreateWithPriority
  0.00% 124.22us   4 31.055us 23.786us 49.725us cuDeviceGetName
  0.00% 87.637us   110   796ns   491ns 2.3920us cudaDeviceGetAttribute
  0.00% 85.506us   32 2.6720us   598ns 8.5110us cudaSetDevice
  0.00% 66.441us   147   451ns   273ns 1.1030us cudaSetupArgument
  0.00% 27.211us   23 1.1830us   471ns 2.6150us cudaConfigureCall
  0.00% 22.945us   10 2.2940us   875ns 8.1350us cudaGetDevice
  0.00% 11.906us   1 11.906us 11.906us 11.906us cudaBindTexture
  0.00% 10.073us   16   629ns   461ns   800ns cudaPeekAtLastError
  0.00% 6.2590us   1 6.2590us 6.2590us 6.2590us cudaStreamGetPriority
  0.00% 5.0270us   6   837ns   256ns 1.6000us cuDeviceGetCount
  0.00% 4.6230us   2 2.3110us 1.4360us 3.1870us cudaStreamWaitEvent
  0.00% 4.2160us   2 2.1080us 2.0150us 2.2010us cudaDeviceGetStreamPriorityRange
  0.00% 3.6920us   6   615ns   459ns   954ns cuDeviceGet
  0.00% 3.6270us   2 1.8130us 1.3450us 2.2820us cudaEventRecord
  0.00% 3.2860us   6   547ns   347ns   840ns cudaGetLastError
  0.00% 3.0640us   3 1.0210us   875ns 1.2380us cuInit

```
 0.00%  2.1870us         1  2.1870us  2.1870us  2.1870us  cudaUnbindTexture
 0.00%  1.8480us         3     616ns     572ns     675ns  cuDriverGetVersion
 0.00%  1.3100us         1  1.3100us  1.3100us  1.3100us  cudaGetDeviceCount
```

B. Full profile generated for M3.1

```
==319== NVPROF is profiling process 319, command: python m3.1.py ece408-high 10000
Loading model... done
Op Time: 0.448293
Correctness: 0.8562 Model: ece408-high
==319== Profiling application: python m3.1.py ece408-high 10000
==319== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
80.21%  428.68ms         1  428.68ms  428.68ms  428.68ms  mxnet::op::forward_kernel(float*,
float const *, float const *, int, int, int, int, int, int)
  7.40%  39.527ms         1  39.527ms  39.527ms  39.527ms
sgemm_sm35_ldg_tn_128x8x256x16x32
3.67%  19.590ms         1  19.590ms  19.590ms  19.590ms  void
mshadow::cuda::MapPlanLargeKernel<mshadow::sv::saveto, int=8, int=1024,
mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=4, float>, float>,
mshadow::expr::Plan<mshadow::expr::BinaryMapExp<mshadow::op::mul,
mshadow::expr::ScalarExp<float>, mshadow::Tensor<mshadow::gpu, int=4, float>, float, int=1>,
float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=4, int)
  3.63%  19.390ms         2  9.6950ms  461.75us  18.928ms  void
cudnn::detail::activation_fw_4d_kernel<float, float, int=128, int=1, int=4,
cudnn::detail::tanh_func<float>>(cudnnTensorStruct, float const *,
cudnn::detail::activation_fw_4d_kernel<float, float, int=128, int=1, int=4,
cudnn::detail::tanh_func<float>>, cudnnTensorStruct*, float, cudnnTensorStruct*, int,
cudnnTensorStruct*)
  2.71%  14.501ms         1  14.501ms  14.501ms  14.501ms  void
cudnn::detail::pooling_fw_4d_kernel<float, float, cudnn::detail::maxpooling_func<float,
cudnnNanPropagation_t=0>, int=0>(cudnnTensorStruct, float const *,
cudnn::detail::pooling_fw_4d_kernel<float, float, cudnn::detail::maxpooling_func<float,
cudnnNanPropagation_t=0>, int=0>, cudnnTensorStruct*, cudnnPoolingStruct, float,
cudnnPoolingStruct, int, cudnn::reduced_divisor, float)
  1.18%  6.3054ms        13  485.03us  1.5360us  4.3642ms  [CUDA memcpy HtoD]
  0.70%  3.7285ms         1  3.7285ms  3.7285ms  3.7285ms
sgemm_sm35_ldg_tn_64x16x128x8x32
0.21%  1.1218ms         1  1.1218ms  1.1218ms  1.1218ms  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)
  0.14%  755.57us        12  62.963us  2.1120us  381.27us  void
mshadow::cuda::MapPlanKernel<mshadow::sv::saveto, int=8,
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)
0.08% 438.07us 2 219.04us 17.504us 420.57us void mshadow::cuda::MapPlanKernel<mshadow::sv::plusto, int=8, mshadow::expr::Plan<mshadow::Tensor<mshadow::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 int, mshadow::Shape<int=2>, int=2)
0.07% 388.73us 1 388.73us 388.73us 388.73us sgemm_sm35_ldg_tn_32x16x64x8x16
0.00% 23.551us 1 23.551us 23.551us 23.551us void mshadow::cuda::MapPlanKernel<mshadow::sv::saveto, int=8, mshadow::expr::Plan<mshadow::Tensor<mshadow::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)
0.00% 9.4720us 1 9.4720us 9.4720us 9.4720us [CUDA memcpy DtoH]
==319== API calls:
Time(%) Time Calls Avg Min Max Name
42.64% 1.90041s 18 105.58ms 16.957us 949.87ms cudaStreamCreateWithFlags
26.05% 1.16128s 10 116.13ms 1.1300us 331.62ms cudaFree
18.91% 842.87ms 23 36.646ms 235.87us 836.09ms cudaMemGetInfo
10.06% 448.24ms 1 448.24ms 448.24ms 448.24ms cudaDeviceSynchronize
1.77% 78.739ms 25 3.1496ms 5.9080us 42.536ms cudaStreamSynchronize
0.29% 13.103ms 8 1.6379ms 11.467us 4.5714ms cudaMemcpy2DAsync
0.16% 6.9359ms 41 169.17us 10.669us 1.1498ms cudaMalloc
0.03% 1.3969ms 4 349.22us 338.13us 380.77us cuDeviceTotalMem
0.03% 1.3795ms 4 344.87us 50.154us 1.1607ms cudaStreamCreate
0.02% 883.95us 352 2.5110us 246ns 75.887us cuDeviceGetAttribute
0.02% 678.20us 114 5.9490us 618ns 131.31us cudaEventCreateWithFlags
0.01% 586.77us 24 24.448us 11.171us 58.823us cudaLaunch
0.01% 336.63us 6 56.105us 23.955us 84.961us cudaMemcpy
0.00% 122.90us 4 30.724us 28.621us 32.386us cuDeviceGetName
0.00% 81.193us 30 2.7060us 690ns 9.0620us cudaSetDevice
0.00% 65.077us 104 625ns 421ns 1.7070us cudaDeviceGetAttribute
0.00% 60.591us 145 417ns 253ns 1.2260us cudaSetupArgument
0.00% 35.493us 2 17.746us 17.473us 18.020us cudaStreamCreateWithPriority
0.00% 28.424us 24 1.1840us 429ns 2.3220us cudaConfigureCall
0.00% 20.979us 10 2.0970us 1.1430us 6.2420us cudaGetDevice
0.00% 9.2550us 17 544ns 335ns 1.0720us cudaPeekAtLastError
0.00% 6.4140us 1 6.4140us 6.4140us 6.4140us cudaStreamGetPriority
0.00% 4.8720us 6 812ns 451ns 1.9560us cuDeviceGetCount
0.00% 4.5240us 2 2.2620us 1.3180us 3.2060us cudaEventRecord

0.00%	4.0900us	6	681ns	477ns	1.1790us	cuDeviceGet
0.00%	4.0830us	2	2.0410us	1.3900us	2.6930us	cudaStreamWaitEvent
0.00%	2.7940us	2	1.3970us	1.3060us	1.4880us	cudaDeviceGetStreamPriorityRange
0.00%	2.7600us	3	920ns	907ns	938ns	cuInit
0.00%	2.5440us	5	508ns	361ns	604ns	cudaGetLastError
0.00%	2.1640us	3	721ns	603ns	887ns	cuDriverGetVersion
0.00%	1.5280us	1	1.5280us	1.5280us	1.5280us	cudaGetDeviceCount

C. Full profile generated for Final Code

```
==316== NVPROF is profiling process 316, command: python final.py ece408-high 10000
Loading model... done
Op Time: 0.011880
Correctness: 0.8562 Model: ece408-high
==316== Profiling application: python final.py ece408-high 10000
==316== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
33.21%  39.292ms         1  39.292ms  39.292ms  39.292ms
sgemm_sm35_ldg_tn_128x8x256x16x32
16.55%  19.576ms         1  19.576ms  19.576ms  19.576ms  void
mshadow::cuda::MapPlanLargeKernel<mshadow::sv::saveto, int=8, int=1024,
mshadow::expr::Plan<mshadow::Tensor<mshadow::gpu, int=4, float>, float>,
mshadow::expr::Plan<mshadow::expr::BinaryMapExp<mshadow::op::mul,
mshadow::expr::ScalarExp<float>, mshadow::Tensor<mshadow::gpu, int=4, float>, float, int=1>,
float>>(mshadow::gpu, unsigned int, mshadow::Shape<int=2>, int=4, int)
 16.39%  19.387ms         2  9.6933ms  461.72us  18.925ms  void
cudnn::detail::activation_fw_4d_kernel<float, float, int=128, int=1, int=4,
cudnn::detail::tanh_func<float>>(cudnnTensorStruct, float const *,
cudnn::detail::activation_fw_4d_kernel<float, float, int=128, int=1, int=4,
cudnn::detail::tanh_func<float>>, cudnnTensorStruct*, float, cudnnTensorStruct*, int,
cudnnTensorStruct*)
12.25%  14.494ms         1  14.494ms  14.494ms  14.494ms  void
cudnn::detail::pooling_fw_4d_kernel<float, float, cudnn::detail::maxpooling_func<float,
cudnnNanPropagation_t=0>, int=0>(cudnnTensorStruct, float const *,
cudnn::detail::pooling_fw_4d_kernel<float, float, cudnn::detail::maxpooling_func<float,
cudnnNanPropagation_t=0>, int=0>, cudnnTensorStruct*, cudnnPoolingStruct, float,
cudnnPoolingStruct, int, cudnn::reduced_divisor, float)
  9.92%  11.734ms         1  11.734ms  11.734ms  11.734ms  mxnet::op::forward_kernel(float*,
float const *)
  6.29%  7.4470ms        14  531.93us  1.5680us  5.4959ms  [CUDA memcpy HtoD]
  3.08%  3.6483ms         1  3.6483ms  3.6483ms  3.6483ms
sgemm_sm35_ldg_tn_64x16x128x8x32
  0.94%  1.1169ms         1  1.1169ms  1.1169ms  1.1169ms  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)
0.64%  754.80us        12  62.900us  2.1120us  380.95us  void
mshadow::cuda::MapPlanKernel<mshadow::sv::saveto, int=8,
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)
0.37% 436.31us 2 218.16us 16.608us 419.71us void mshadow::cuda::MapPlanKernel<mshadow::sv::plusto, int=8, mshadow::expr::Plan<mshadow::Tensor<mshadow::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 int, mshadow::Shape<int=2>, int=2)
0.33% 391.16us 1 391.16us 391.16us 391.16us sgemm_sm35_ldg_tn_32x16x64x8x16
0.02% 23.712us 1 23.712us 23.712us 23.712us void mshadow::cuda::MapPlanKernel<mshadow::sv::saveto, int=8, mshadow::expr::Plan<mshadow::Tensor<mshadow::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)
0.01% 14.847us 2 7.4230us 5.2800us 9.5670us [CUDA memcpy DtoH]
==316== API calls:
Time(%) Time Calls Avg Min Max Name
45.65% 1.93756s 18 107.64ms 17.414us 968.43ms cudaStreamCreateWithFlags
28.38% 1.20468s 10 120.47ms 685ns 340.95ms cudaFree
22.74% 965.16ms 23 41.964ms 237.04us 958.41ms cudaMemGetInfo
1.85% 78.695ms 25 3.1478ms 4.9690us 42.518ms cudaStreamSynchronize
0.74% 31.336ms 3 10.445ms 4.2060us 19.586ms cudaDeviceSynchronize
0.35% 14.956ms 8 1.8695ms 10.893us 5.6761ms cudaMemcpy2DAsync
0.15% 6.3479ms 41 154.83us 9.1110us 1.1175ms cudaMalloc
0.03% 1.3678ms 4 341.94us 338.56us 350.82us cuDeviceTotalMem
0.02% 871.98us 352 2.4770us 246ns 66.053us cuDeviceGetAttribute
0.02% 862.31us 4 215.58us 29.504us 753.14us cudaStreamCreate
0.02% 843.07us 114 7.3950us 624ns 304.38us cudaEventCreateWithFlags
0.01% 481.46us 24 20.061us 9.0600us 51.086us cudaLaunch
0.01% 479.85us 7 68.550us 25.914us 118.75us cudaMemcpy
0.00% 103.22us 4 25.805us 18.246us 31.785us cuDeviceGetName
0.00% 74.192us 30 2.4730us 634ns 6.7390us cudaSetDevice
0.00% 73.163us 104 703ns 414ns 2.0170us cudaDeviceGetAttribute
0.00% 63.685us 138 461ns 251ns 1.4570us cudaSetupArgument
0.00% 40.182us 2 20.091us 19.384us 20.798us cudaStreamCreateWithPriority
0.00% 29.428us 24 1.2260us 378ns 3.8310us cudaConfigureCall
0.00% 26.410us 10 2.6410us 1.3490us 6.4340us cudaGetDevice
0.00% 15.055us 1 15.055us 15.055us 15.055us cudaMemcpyToSymbol
0.00% 8.9010us 17 523ns 326ns 735ns cudaPeekAtLastError
0.00% 5.8390us 6 973ns 336ns 2.4070us cuDeviceGetCount
0.00% 4.5790us 1 4.5790us 4.5790us 4.5790us cudaStreamGetPriority

0.00%	3.9490us	2	1.9740us	1.4400us	2.5090us	cudaStreamWaitEvent
0.00%	3.7070us	2	1.8530us	1.3770us	2.3300us	cudaEventRecord
0.00%	3.6170us	6	602ns	408ns	910ns	cuDeviceGet
0.00%	3.0600us	2	1.5300us	1.1530us	1.9070us	cudaDeviceGetStreamPriorityRange
0.00%	2.9700us	3	990ns	965ns	1.0050us	cuInit
0.00%	2.7500us	5	550ns	361ns	813ns	cudaGetLastError
0.00%	2.4080us	3	802ns	757ns	884ns	cuDriverGetVersion
0.00%	2.2770us	1	2.2770us	2.2770us	2.2770us	cudaGetDeviceCount

D. Side by Side Comparison of Baseline and Final Optimized Implementations Via NVVP

Memory Bandwidth And Utilization

The following table shows the memory bandwidth used by this kernel for the various types of memory on the device. The table also shows the utilization of each memory type relative to the maximum throughput supported by the memory. More...

	Transactions	Bandwidth	Utilization
L1/Shared Memory			
Local Loads	0	0 B/s	
Local Stores	0	0 B/s	
Shared Loads	0	0 B/s	
Shared Stores	0	0 B/s	
Global Loads	688500000	206.087 GB/s	
Global Stores	9000000	5.251 GB/s	
Atomic	0	0 B/s	
L1/Shared Total	697500000	211.338 GB/s	Idle Low Medium High Max
L2 Cache			
L1 Reads	1413000000	206.087 GB/s	
L1 Writes	36000000	5.251 GB/s	
Texture Reads	0	0 B/s	
Noncoherent Reads	0	0 B/s	
Atomic	0	0 B/s	
Total	1449000000	211.338 GB/s	Idle Low Medium High Max

Figure 1: Baseline Memory Bandwidth and Utilization

Memory Bandwidth And Utilization

The following table shows the memory bandwidth used by this kernel for the various types of memory on the device. The table also shows the utilization of each memory type relative to the maximum throughput supported by the memory. More...

	Transactions	Bandwidth	Utilization
L1/Shared Memory			
Local Loads	0	0 B/s	
Local Stores	0	0 B/s	
Shared Loads	6910000	141.328 GB/s	
Shared Stores	250000	5.113 GB/s	
Global Loads	370000	2.505 GB/s	
Global Stores	9000000	92.037 GB/s	
Atomic	0	0 B/s	
L1/Shared Total	16530000	240.983 GB/s	Idle Low Medium High Max
L2 Cache			
L1 Reads	980000	2.505 GB/s	
L1 Writes	36000000	92.037 GB/s	
Texture Reads	0	0 B/s	
Noncoherent Reads	0	0 B/s	
Atomic	0	0 B/s	
Total	36980000	94.542 GB/s	Idle Low Medium High Max

Figure 2: Optimized Memory Bandwidth and Utilization

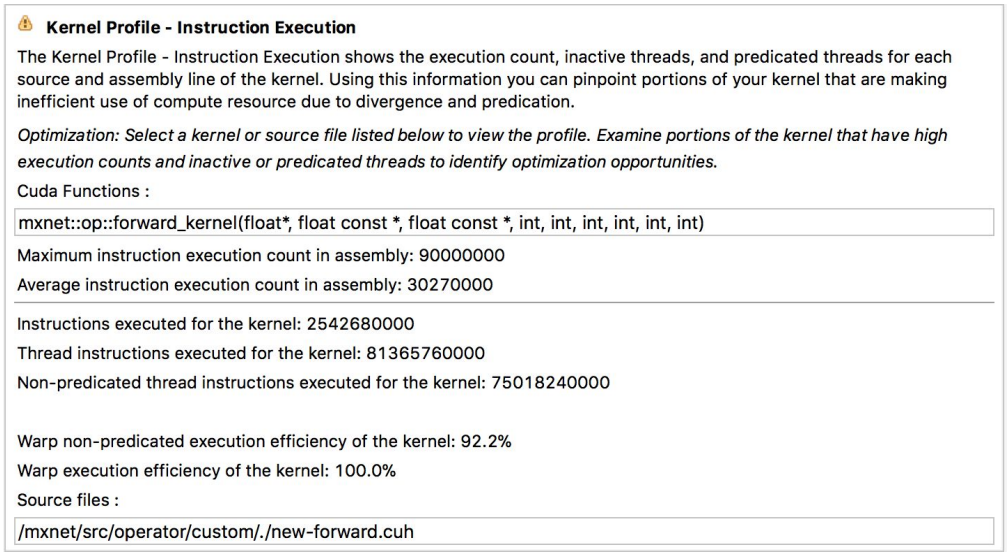

Figure 3: Baseline Divergence (divergence 7.8%)

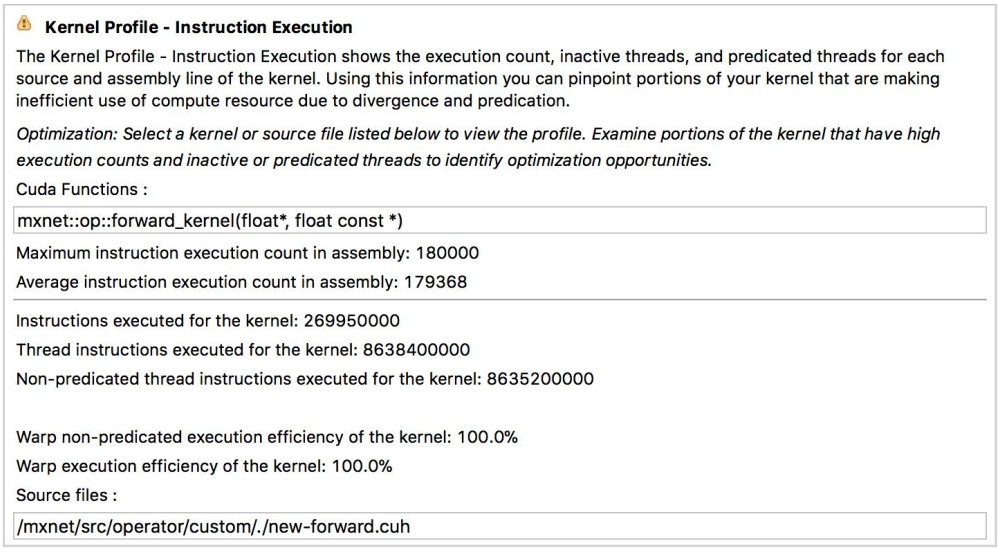

Figure 4: Optimized Divergence (No divergence)