ECE 408 Applied Parallel Programming
Fall 2019 Report

1 Milestone 2

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

Solution
[*CUDA memcpy HtoD*] 30.05%
volta_scudnn_128x64_relu_interior_nn_v1 17.81%
volta_gcgemm_64x32_nt 17.10%
void fft2d_c2r_32x32 8.56%
volta_sgemm_128x128_tn 7.79%
void op_generic_tensor_kernel 6.50%
void fft2d_r2c_32x32 5.72%

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

Solution
cudaStreamCreateWithFlags 43.17%
cudaMemGetInfo 31.31%
cudaFree 21.65%

(c) Include an explanation of the difference between kernels and API calls.

Solution
Kernel calls use less time than API calls. API calls in this piece of code will call kernel. So they will use up more time.

(d) Show output of rai running MXNet on the CPU.

Solution
Loading fashion-mnist data... done
Loading model... done
New Inference
EvalMetric: 'accuracy': 0.8154
17.40user 4.58system 0:09.00elapsed 244%CPU (0avgtext+0avgdata 6046604maxresident)k
0inputs+2824outputs (0major+1603908minor)pagefaults 0swaps

(e) List program run time

Solution
user 17.46
system 4.65
real 0:09.07 elapsed

(f) Show output of rai running MXNet on the CPU.

Solution
Loading fashion-mnist data... done
Loading model... done
New Inference
EvalMetric: 'accuracy': 0.8154
5.11user 3.30system 0:04.68elapsed 179%CPU (0avgtext+0avgdata 2970344maxresident)k
0inputs+1712outputs (0major+732652minor)pagefaults 0swaps

(g) List program run time

Solution
user 5.11
system 3.30
real 0:04.68 elapsed

(h) List whole program execution time

Solution
user 100.89
system 8.12
real 1:31.66 elapsed

(i) List Op Times

Solution
Op Time: 12.051926
Op Time: 75.994165

2 Milestone 3

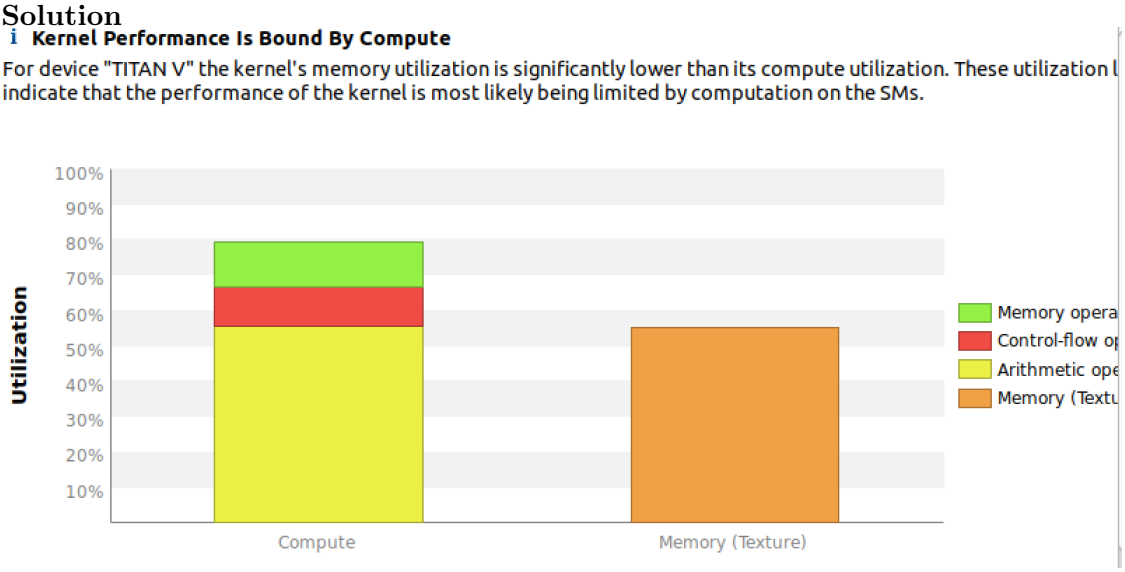
(j) Correctness and time with 3 different dataset sizes

Solution

Size	*User*	*System*	*Real*	*Correctness*
100	4.94	3.11	0 : 04.36	0.76
1000	7.14	3.68	0 : 07.61	0.767
10000	5.35	3.03	0 : 05.07	0.7653

(k) nvprof profiling

Solution



This kernel exhibits low compute throughput and memory bandwidth utilization, which are below 60%. As we can see from the graph, arithmetic and memory operation take up most of utilization. So its performance is most likely limited by the latency of arithmetic or memory operations.

Occupancy Is Not Limiting Kernel Performance

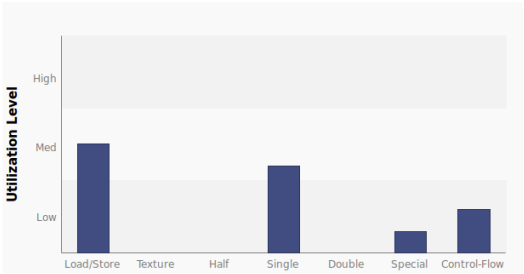
The kernel's block size, register usage, and shared memory usage allow it to fully utilize all warps on the GPU. More...

Variable	Achieved	Theoretical	Device Limit	Grid Size: [100,12,25] (30000 blocks)Block Size: [16,16,1] (256 threads)
Occupancy Per SM				
Active Blocks		8	32	0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32
Active Warps	53.8	64	64	0 4 8 12 16 20 24 28 32 36 40 44 48 52 56 60 64
Active Threads		2048	2048	0 256 512 768 1024 1280 1536 1792 2048
Occupancy	84.1%	100%	100%	0% 15% 30% 45% 60% 75% 90% 100%
Warps				
Threads/Block		256	1024	0 128 256 384 512 640 768 896 1024
Warps/Block		8	32	0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32
Block Limit		8	32	0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32
Registers				
Registers/Thread		32	65536	0 8192 16384 24576 32768 40960 49152 57344 65536
Registers/Block		8192	65536	0 8k 16k 24k 32k 40k 48k 56k 64k
Block Limit		8	32	0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32
Shared Memory				
Shared Memory/Block		0	98304	0 16k 32k 48k 64k 80k 96k
Block Limit		0	32	0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32

It shows that the kernel's block size, register usage, and shared memory usage allow it to fully utilize all warps on the GPU. But we are only using part of them due to control divergence.

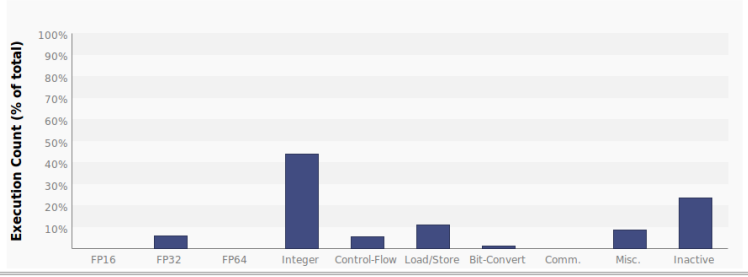
Results

Control-Flow - Direct and indirect branches, jumps, and calls.



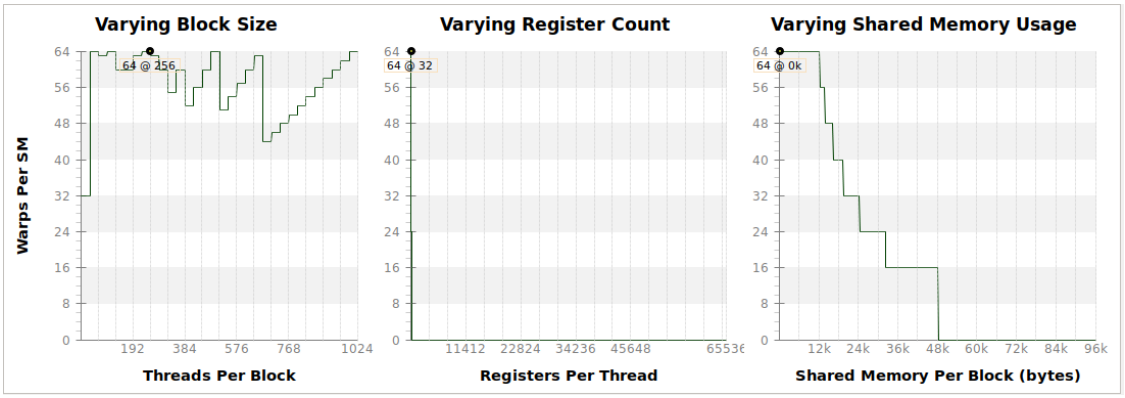
Instruction Execution Counts

The following chart shows the mix of instructions executed by the kernel. The instructions are grouped into classes and for each class the chart shows the percentage of thread execution cycles that were devoted to executing instructions in that class. The "Inactive" result shows the thread executions that did not execute any instruction because the thread was predicated or inactive due to divergence.



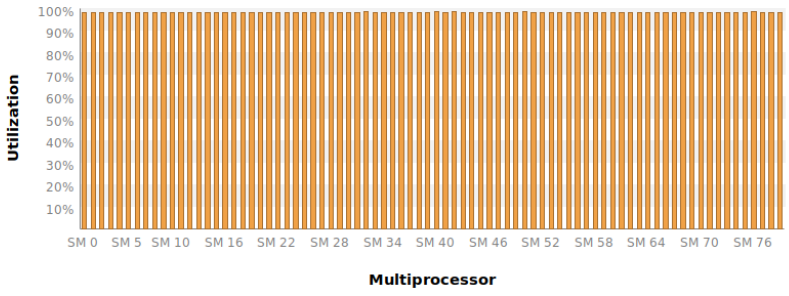
Floating-Point Operation Counts

The following chart shows the mix of floating-point operations executed by the kernel. The operations are grouped into classes and for each class the chart shows the percentage of thread execution cycles that were devoted to executing operations in that class. The results do not sum to 100% because non-floating-point operations executed by the kernel are not shown in this chart.



Multiprocessor Utilization

The kernel's blocks are distributed across the GPU's multiprocessors for execution. Depending on the number of blocks and the execution duration of each block some multiprocessors may be more highly utilized than others during execution of the kernel. The following chart shows the utilization of each multiprocessor during execution of the kernel. More...



Global Memory Alignment and Access Pattern

Memory bandwidth is used most efficiently when each global memory load and store has proper alignment and access pattern. The analysis is per assembly instruction.

Optimization: Select each entry below to open the source code to a global load or store within the kernel with an inefficient alignment or access pattern. For each load or store improve the alignment and access pattern of the memory access. More...

Line / File	new-forward.cuh - /mxnet/src/operator/custom
45	Global Load L2 Transactions/Access = 4.8, Ideal Transactions/Access = 3.4 [4752000 L2 transactions for 990000 total executions]
45	Global Load L2 Transactions/Access = 5.3, Ideal Transactions/Access = 3.4 [5247000 L2 transactions for 990000 total executions]
45	Global Load L2 Transactions/Access = 5.3, Ideal Transactions/Access = 3.4 [5247000 L2 transactions for 990000 total executions]
45	Global Load L2 Transactions/Access = 4.8, Ideal Transactions/Access = 3.4 [4752000 L2 transactions for 990000 total executions]
45	Global Load L2 Transactions/Access = 4.8, Ideal Transactions/Access = 3.4 [4752000 L2 transactions for 990000 total executions]
47	Global Store L2 Transactions/Access = 4.8, Ideal Transactions/Access = 3.4 [950400 L2 transactions for 198000 total executions]

Memory Bandwidth And Utilization

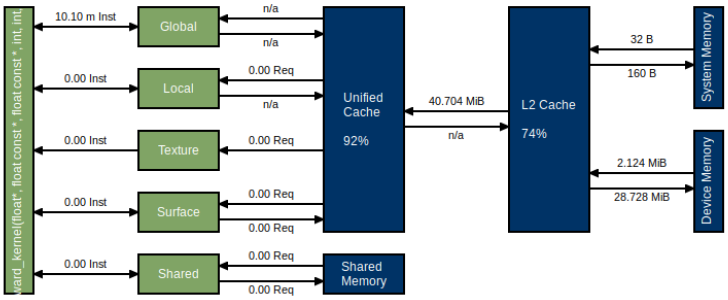
The following table shows the memory bandwidth used by this kernel for the various types of memory on the device. The table also shows the utilization of each memory type relative to the maximum throughput supported by the memory. More...

	Transactions	Bandwidth	Utilization
Shared Memory			
Shared Loads	0	0 B/s	
Shared Stores	0	0 B/s	
Shared Total	0	0 B/s	Idle Low Medium High Max
L2 Cache			
Reads	1348082	191.289 GB/s	
Writes	950429	134.863 GB/s	
Total	2298511	326.153 GB/s	Idle Low Medium High Max
Unified Cache			
Local Loads	0	0 B/s	
Local Stores	0	0 B/s	
Global Loads	29686667	4,212.462 GB/s	
Global Stores	950400	134.859 GB/s	
Texture Reads	12630813	7,169.12 GB/s	
Unified Total	43267880	11,516.441 GB/s	Idle Low Medium High Max
Device Memory			
Reads	69584	9.874 GB/s	
Writes	941356	133.576 GB/s	
Total	1010940	143.45 GB/s	Idle Low Medium High Max
System Memory [PCIe configuration: Gen3 x16, 8 Gbit/s]			
Reads	1	141.897 kB/s	Idle Low Medium High Max
Writes	5	709.487 kB/s	Idle Low Medium High Max

As the table shows, our kernel do not use any shared memory. The utilization of L2 cache and device memory is low relative to the maximum throughput supported by the corresponding memory.

Memory Statistics

The following chart shows a summary view of the memory hierarchy of the CUDA programming model. The green nodes in the diagram depict logical memory space whereas blue nodes depicts actual hardware unit on the chip. For the various caches the reported percentage number states the cache hit rate; that is the ratio of requests that could be served with data locally available to the cache over all requests made.
The links between the nodes in the diagram depict the data paths between the SMs to the memory spaces into the memory system. Different metrics are shown per data path. The data paths from the SMs to the memory spaces report the total number of memory instructions executed, it includes both read and write operations. The data path between memory spaces and "Unified Cache" or "Shared Memory" reports the total amount of memory requests made (read or write). All other data paths report the total amount of transferred memory in bytes.



Low Warp Execution Efficiency

Warp execution efficiency is the average percentage of active threads in each executed warp. Increasing warp execution efficiency will increase utilization of the GPU's compute resources. The warp execution efficiency for these kernels is 84.8% if predicated instructions are not taken into account. The kernel's not predicated off warp execution efficiency of 76.6% is less than 100% due to divergent branches and predicated instructions.

Optimization: Reduce the amount of intra-warp divergence and predication in the kernel. More...

Divergent Branches

Compute resource are used most efficiently when all threads in a warp have the same branching behavior. When this does not occur the branch is said to be divergent. Divergent branches lower warp execution efficiency which leads to inefficient use of the GPU's compute resources.

Optimization: Select each entry below to open the source code to a divergent branch within the kernel. For each branch reduce the amount of intra-warp divergence. More...

Line / File	new-forward.cuh - /mxnet/src/operator/custom
41	Divergence = 16.5% [39600 divergent executions out of 240000 total executions]

The report indicates that divergent executions in our kernel account for 16.5% of total excecutions. Therefore, divergent branches lower warp execution efficiency, which leads to inefficient use of the GPU's compute resources.