ECE 408/CS 483 Final Project

Team: elegant_and_easygoing_boys
Team members:
Licheng Luo (ll6)
Zhengqi Fang (zf4)
Ruian Pan (ruianp2)
Affiliation: on campus

Milestone 2

All kernels that collectively consume more than 90% of the program time

volta_scudnn_128x64_relu_interior_nn_v1
volta_gcgemm_64x32_nt
fft2d_c2r_32x32
volta_sgemm_128x128_tn
op_generic_tensor_kernel
fft2d_r2c_32x32
cudnn::detail::pooling_fw_4d_kernel

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

Kernels are the codes that run on GPU and do the parallel computations.
API calls are the calls to the CUDA's APIs, which are defined by CUDA(NVIDIA). They are usually used to do the initializations such as memory allocations and transfer.

Output of RAI running MXNet on the CPU (m1.1)

EvalMetric: {'accuracy': 0.8154}

Run time

User	20.89
System	7.39
Elapsed	0:10.28

Output of RAI running MXNet on the GPU (m1.2)

EvalMetric: {'accuracy': 0.8154}
Run time

User	5.10
System	2.69
Elapsed	0:05.01

CPU Implementation

Correctness: 0.7653 Model: ece408
Run time (m2.1)

User	90.33
System	10.19
Elapsed	1:19.22

Op Times

Op Time: 13.540072
Op Time: 60.894102

Milestone 3

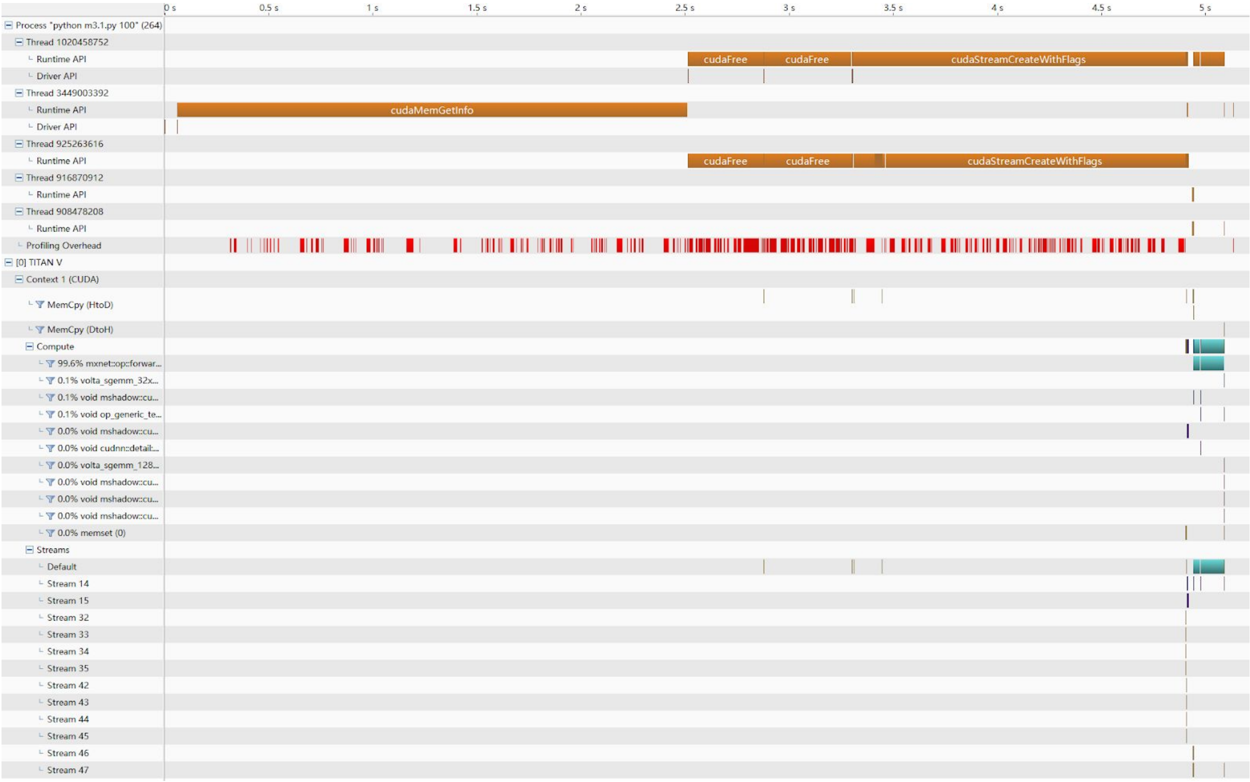
Result

	Data size 100	Data size 1000	Data size 10000
Correctness	0.76	0.767	0.7653

<table>
<tr>
<td>Timing</td>
<td><table><tr><td>User</td><td>System</td><td>Elapsed</td></tr><tr><td>5.03</td><td>2.88</td><td>0:04.53</td></tr></table></td>
<td><table><tr><td>User</td><td>System</td><td>Elapsed</td></tr><tr><td>6.55</td><td>3.32</td><td>0:06.55</td></tr></table></td>
<td><table><tr><td>User</td><td>System</td><td>Elapsed</td></tr><tr><td>25.01</td><td>9.50</td><td>0:31.01</td></tr></table></td>
</tr>
</table>

Nvprof and NVVP performance analysis

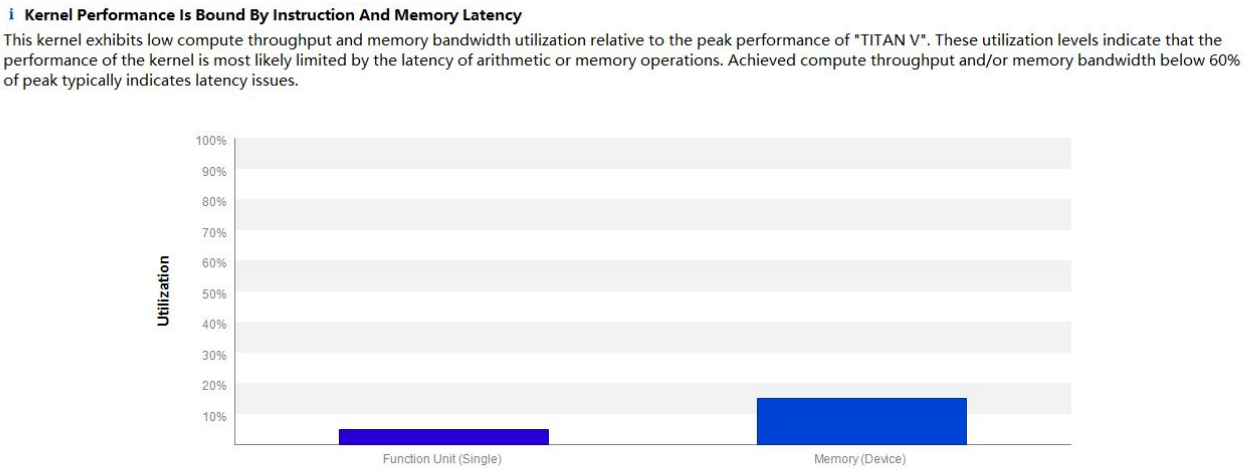
NVVP trace (data size: 100)



Note that 99.6% of the Compute is the forward_kernel.

Analysis

We did a detailed analysis on kernel 1, and NVVP showed that the kernel performance is bound by instruction and memory latency, as the screenshot shown below:

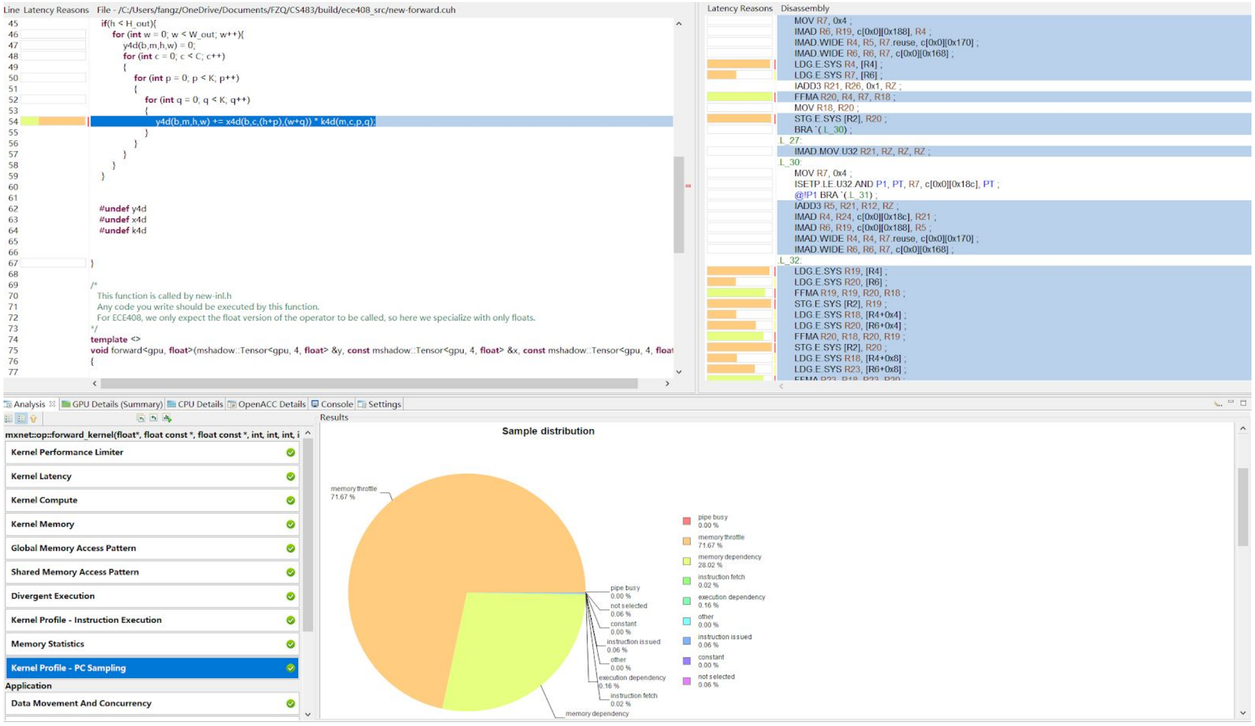


i Kernel Performance Is Bound By Instruction And Memory Latency

This kernel exhibits low compute throughput and memory bandwidth utilization relative to the peak performance of "TITAN V". These utilization levels indicate that the performance of the kernel is most likely limited by the latency of arithmetic or memory operations. Achieved compute throughput and/or memory bandwidth below 60% of peak typically indicates latency issues.

We believe this is because we don't use any shared memory to do the tiling, which makes the memory bandwidth really small. In the future, we are going to try to utilize the shared memory to load the input in order to improve the bandwidth.

We also ran the PC sampling analysis. We can see from below that the main bottleneck is the memory throttle, where a large number of memory operations are pending.



Line Latency Reasons File - /C:/Users/fangz/OneDrive/Documents/FZQ/CS483/build/ece408_src/new-forward.cuh

```
    if(h < H_out){
        for (int w = 0; w < W_out; w++){
            y4d(b,m,h,w) = 0;
            for (int c = 0; c < C; c++)
            {
                for (int p = 0; p < K; p++)
                {
                    for (int q = 0; q < K; q++)
                    {
                        y4d(b,m,h,w) += x4d(b,c,(h+p),(w+q)) * k4d(m,c,p,q);
                    }
                }
            }
        }
    }


    #undef y4d
    #undef x4d
    #undef k4d


}

/*
   This function is called by new-inl.h
   Any code you write should be executed by this function.
   For ECE408, we only expect the float version of the operator to be called, so here we specialize with only floats.
*/
template <>
void forward<gpu, float>(mshadow::Tensor<gpu, 4, float> &y, const mshadow::Tensor<gpu, 4, float> &x, const mshadow::Tensor<gpu, 4, float
{
```

Latency Reasons Disassembly

```
    MOV R7, 0x4 ;
    IMAD R6, R19, c[0x0][0x188], R4 ;
    IMAD.WIDE R4, R5, R7.reuse, c[0x0][0x170] ;
    IMAD.WIDE R6, R6, R7, c[0x0][0x168] ;
    LDG.E.SYS R4, [R4] ;
    LDG.E.SYS R7, [R6] ;
    IADD3 R21, R26, 0x1, RZ ;
    FFMA R20, R4, R7, R18 ;
    MOV R18, R20 ;
    STG.E.SYS [R2], R20 ;
    BRA `(.L_30) ;
.L_27:
    IMAD.MOV.U32 R21, RZ, RZ, RZ ;
.L_30:
    MOV R7, 0x4 ;
    ISETP.LE.U32.AND P1, PT, R7, c[0x0][0x18c], PT ;
    @!P1 BRA `(.L_31) ;
    IADD3 R5, R21, R12, RZ ;
    IMAD R4, R24, c[0x0][0x18c], R21 ;
    IMAD R6, R19, c[0x0][0x188], R5 ;
    IMAD.WIDE R4, R4, R7.reuse, c[0x0][0x170] ;
    IMAD.WIDE R6, R6, R7, c[0x0][0x168] ;
.L_32:
    LDG.E.SYS R19, [R4] ;
    LDG.E.SYS R20, [R6] ;
    FFMA R19, R19, R20, R18 ;
    STG.E.SYS [R2], R19 ;
    LDG.E.SYS R18, [R4+0x4] ;
    LDG.E.SYS R20, [R6+0x4] ;
    FFMA R20, R18, R20, R19 ;
    STG.E.SYS [R2], R20 ;
    LDG.E.SYS R18, [R4+0x8] ;
    LDG.E.SYS R23, [R6+0x8] ;
    FFMA R23, R18, R23, R20 ;
```

mxnet::op::forward_kernel(float*, float const *, float const *, int, int, int, i

- Kernel Performance Limiter ✓
- Kernel Latency ✓
- Kernel Compute ✓
- Kernel Memory ✓
- Global Memory Access Pattern ✓
- Shared Memory Access Pattern ✓
- Divergent Execution ✓
- Kernel Profile - Instruction Execution ✓
- Memory Statistics ✓
- Kernel Profile - PC Sampling ✓

Application

- Data Movement And Concurrency ✓

Sample distribution