ECE 408

Project Milestone3 Report

Fall 2018

Team: whopper

Chengyi Zhang (chengyi5)

Ziyan Feng (fziyan2)

Younan Deng (younand2)

Table of Content

Chapter 1 Milestone 1

1.1 List of all kernels that collectively consume more than 90% of the program time.

Kernel	Time(%)
[CUDA memcpy HtoD]	36.52%
volta_scudnn_128x32_relu_interior_nn_v1	22.81%
cudnn::detail::implicit_convolve_sgemm	20.90%
volta_sgemm_128x128_tn	7.31%
cudnn::detail::activation_fw_4d_kernel	7.43%
Total	94.97%

1.2 List of all CUDA API calls collectively consume more than 90% of the program time.

CUDA API	Time(%)
cudaStreamCreateWithFlags	37.91%
cudaMemGetInfo	34.36%
cudaFree	22.25%
Total	94.52%

1.3 Explanation of the difference between kernels and API calls

CUDA C extends C by allowing the programmer to define C functions, called kernels. The code in kernels is executed N times in parallel by N different CUDA threads. Each thread that executes the kernel is given a unique thread ID that is accessible within the kernel through the built-in threadIdx variable.

For API calls, some of them provide C functions that execute on the host to allocate and free device memory, transfer data between host memory and device memory, manage systems with multiple devices.

The code in kernels is executed on the device in parallel, using many threads. The API calls are executed on the host, not in parallel.

1.4 Output of rai running MXNet on the CPU

```
* Running /usr/bin/time python m1.1.py
Loading fashion-mnist data... done
Loading model... done
New Inference
EvalMetric: {'accuracy': 0.8177}
20.59user 5.72system 0:14.27elapsed 184%CPU (0avgtext+0avgdata
5954972maxresident)k
0inputs+2856outputs (0major+1573257minor)pagefaults 0swaps
```

1.5 List program run time of CPU

```
20.59 seconds used by user.
5.72 seconds used by system.
```

1.6 Show output of rai running MXNet on the GPU

```
* Running /usr/bin/time python m1.2.py
Loading fashion-mnist data... done
Loading model... done
New Inference
EvalMetric: {'accuracy': 0.8177}
4.07user 2.63system 0:04.62elapsed 145%CPU (0avgtext+0avgdata
2849972maxresident)k
0inputs+4568outputs (0major+706744minor)pagefaults 0swaps
```

1.7 List program run time of GPU

4.07 seconds used by user.
2.63 seconds used by system.

Chapter 2 Milestone 2

2.1 Our CPU implementation

```
void forward(mshadow::Tensor<cpu, 4, DType> &y, const
mshadow::Tensor<cpu, 4, DType> &x, const mshadow::Tensor<cpu, 4,
DType> &k){
    const int B = x.shape_[0];
    const int M = y.shape_[1];
    const int C = x.shape_[1];
    const int K = k.shape_[3];
    const int W_out = y.shape_[3];
    const int H_out = y.shape_[2];
    for(int b = 0; b < B; b++) {
        for(int m = 0; m < M; m++){
            for(int h=0;h<H_out;h++){
                for(int w=0;w<W_out;w++){
                    y[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++){
                  y[b][m][h][w]+=x[b][c][h+p][w+q]*k[m][c][p][q];
                            }
                        }
                    }
                }
            }
        }
    }
}
```

2.2 Output of rai running MXNet on the CPU

```
✱ Running /usr/bin/time python m2.1.py
Loading fashion-mnist data... done
Loading model... done
New Inference
Op Time: 22.301848Op Time: 101.531430
Correctness: 0.8171 Model: ece408
134.08user 4.66system 2:08.18elapsed 108%CPU (0avgtext+0avgdata
5951284maxresident)k
0inputs+2856outputs (0major+2266850minor)pagefaults 0swaps
✱ The build folder has been uploaded to
http://s3.amazonaws.com/files.rai-project.com/userdata/build-5bcb9431336f254b8e39bb6
e.tar.gz. The data will be present for only a short duration of time.
✱ Server has ended your request.
```

2.3 Whole program execution time

134.08seconds used by user.
4.66 seconds used by system.

2.4 Op Times

Op Time of first convolution layer: 22.301848
Op Time of second convolution layer: 101.531430

2.5 Correctness Check

```
Loading fashion-mnist data... done
Loading model... done
New Inference
Op Time: 21.720908
Op Time: 103.334474
Correctness: 0.8171 Model: ece408
135.41user 4.37system 2:09.47elapsed 107%CPU (0avgtext+0avgdata
5950332maxresident)k
0inputs+2856outputs (0major+2267099mino
r)pagefaults 0swaps
```

The correctness is same as the value provided in the github readme.

Chapter 3 Milestone 3

3.1 Our GPU implementation

```
__global__ void forward_kernel(float *y, const float *x, const float *k, const int B, const int
M, const int C, const int H, const int W, const int K, const int W_SIZE)
{
    const int H_out = H - K + 1;
    const int W_out = W - K + 1;
    (void)H_out; // silence declared but never referenced warning. remove this line when
you start working
    (void)W_out; // silence declared but never referenced warning. remove this line when
you start working

// An example use of these macros:
// float a = y4d(0,0,0,0)
// y4d(0,0,0,0) = a
#define y4d(i3, i2, i1, i0) y[(i3) * (M * H_out * W_out) + (i2) * (H_out * W_out) + (i1) *
(W_out) + i0]
#define x4d(i3, i2, i1, i0) x[(i3) * (C * H * W) + (i2) * (H * W) + (i1) * (W) + i0]
#define k4d(i3, i2, i1, i0) k[(i3) * (C * K * K) + (i2) * (K * K) + (i1) * (K) + i0]
    int b = blockIdx.x;
    int m = blockIdx.y;
    int hbase = (blockIdx.z / W_SIZE) * TILE_SIZE;
    int wbase = (blockIdx.z % W_SIZE) * TILE_SIZE;
    int h = threadIdx.y + hbase;
    int w = threadIdx.x + wbase;
    double acc = 0.0;
    for(int c = 0; c < C; c++){
        for(int p = 0; p < K; p++){
                for(int q = 0; q < K; q++){
                        if(h + p < H && w + q < W){
                         acc += x4d(b,c,h + p,w + q) * k4d(m,c,p,q);
                        }
                }
            }
    }
    if(b < B && m < M && h < H_out && w < W_out){
            y4d(b,m,h,w) = acc;
    }
#undef y4d
#undef x4d
#undef k4d
}
```

3.2 Output of rai

```
✱ Running /usr/bin/time python m3.1.py 10000
Loading fashion-mnist data... done
Loading model... done
New Inference
Op Time: 0.091147
Op Time: 0.224074
Correctness: 0.8171 Model: ece408
4.44user 2.71system 0:04.90elapsed 145%CPU (0avgtext+0avgdata
2835852maxresident)k
0inputs+4600outputs (0major+705432minor)pagefaults 0swaps
✱ Running /usr/bin/time python m3.1.py 1000
Loading fashion-mnist data... done
Loading model... done
New Inference
Op Time: 0.009761
Op Time: 0.025157
Correctness: 0.827 Model: ece408
4.01user 2.63system 0:04.16elapsed 159%CPU (0avgtext
+0avgdata 2644380maxresident)k
0inputs+0outputs (0major+606791minor)pagefaults 0swaps
✱ Running /usr/bin/time python m3.1.py 100
Loading fashion-mnist data... done
Loading model... done
New Inference
Op Time: 0.000951
Op Time: 0.002517
Correctness: 0.85 Model: ece408
3.97user 2.53system 0:04.05elapsed 160%CPU (0avgtext+0avgdata 2627356
maxresident)k
0inputs+0outputs (0major+602473minor)pagefaults 0swaps
```

3.3 Whole program execution time and correcteness

3.3.1 Datasize = 10000

4.44 seconds for user, 2.71 seconds for system. Correctness is 0.8171

3.3.2 Datasize = 1000

4.01 seconds for user, 2.63 seconds for system. Correctness is 0.827

3.3.3 Datasize = 100

3.97 seconds for user, 2.53 seconds for system. Correctness is 0.85

3.4 Op Times

3.4.1 Datasize = 10000

Op Time of first convolution layer: 0.091147
Op Time of second convolution layer: 0.224074

3.4.2 Datasize = 1000

Op Time of first convolution layer: 0.009761
Op Time of second convolution layer: 0.025157

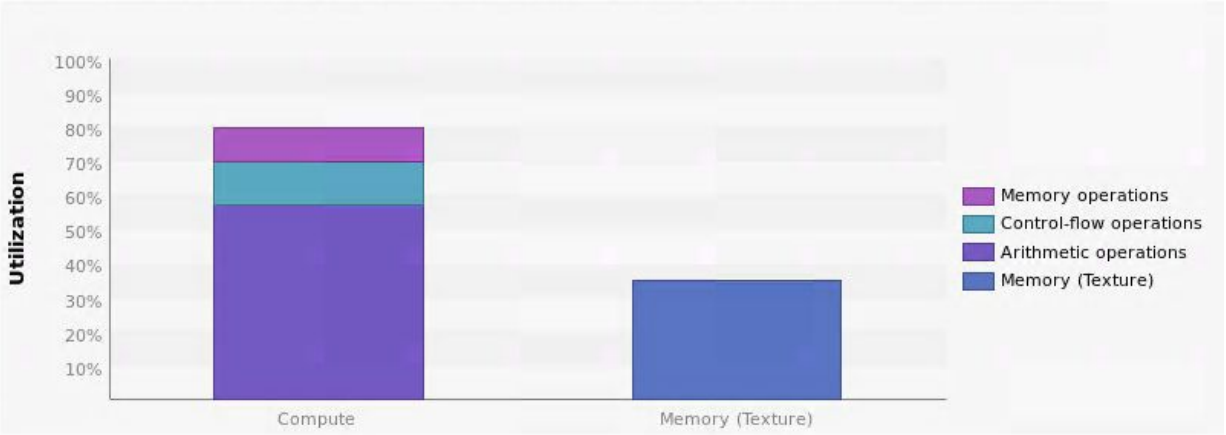
3.4.2 Datasize = 100

Op Time of first convolution layer:0.000951
Op Time of second convolution layer: 0.002517

3.5 Nvvp analysis (Datasize = 100)

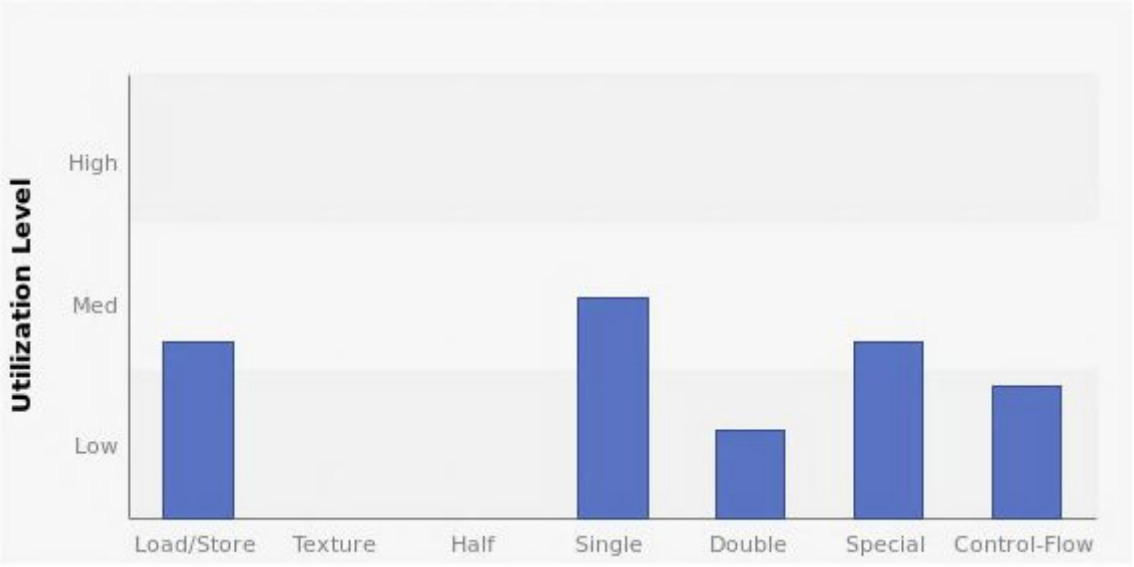
3.5.1 Limiting factor of Kernel performance

From Nvvp we can see in the chart as follows:



The utilization levels indicate that the performance of the kernel is most likely being limited by computation on the SMs.

3.5.2 Kernel compute analysis



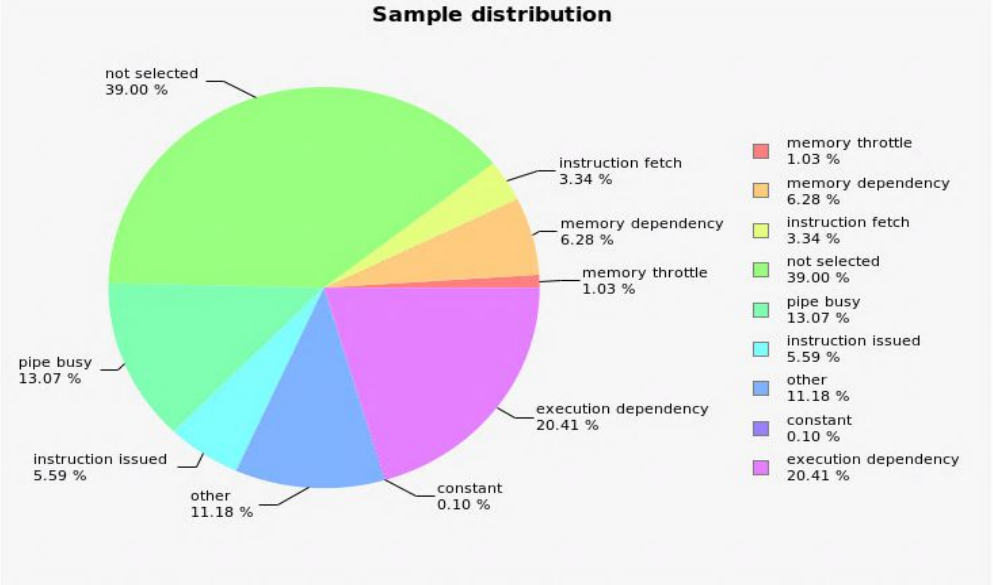
The chart above shows that the kernel's performance is not limited by ant function unit, because every function unit is not overused.

3.5.3 Kernel memory analysis

	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	3999144	56.625 GB/s	
Writes	275416	3.9 GB/s	
Total	4274560	60.524 GB/s	Idle Low Medium High Max
Unified Cache			
Local Loads	0	0 B/s	
Local Stores	0	0 B/s	
Global Loads	235850856	3,339.465 GB/s	
Global Stores	275400	3.899 GB/s	
Texture Reads	87149610	4,935.883 GB/s	
Unified Total	323275866	8,279.247 GB/s	Idle Low Medium High Max
Device Memory			
Reads	2003202	28.364 GB/s	
Writes	263463	3.73 GB/s	
Total	2266665	32.094 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	70.796 kB/s	Idle Low Medium High Max

The chart above shows that the utilization of each memory type doesn't exceed the maximum throughput supported by the memory.

3.5.4 Kernel profiling - PC samples



The pie chart above shows that the primary stall reason in this kernel is “not selected”, which means that some warps are ready to issue, but they don’t issue. Some other warps are issued instead.

From the analysis of source code, we find that

```
if(h + p < H && w + q < W)
                    acc += x4d(b,c,h + p,w + q) * k4d(m,c,p,q);
```

In this line of our code, many warps are “not selected”, which means that they are "ready" to be issued, but for some reason the warp scheduler chose to select instruction(s) from another warp to issue in the clock cycle that was sampled.

Also, 20.41% fo the warps have execution dependency with other warps(i.e., an input required by the instruction is not yet available), so they are not issued either.

For the latency reasons of this line of code, memory dependency accounts for 41.7%, and execution dependency accounts of 31.2%. The execution dependency has been explained. The memory dependency is because a load/store can’t be made because the required resources are not available or are fully utilized.