ECE5545/CS5775 A3

Naman Makkar

TOTAL POINTS

15.8 / 16

QUESTION 1

1 A3-Report 2 / 2

✓ **+ 2 pts** *Complete*

QUESTION 2

2 A3-Coding-Completeness 4 / 4

✓ **+ 4 pts** *ALl complete.*

- **- 1 pts** 1dconv_cpu
- **- 1 pts** 1dconv_gpu
- **- 1 pts** dwspconv2d_gpu
- **- 1 pts** gemm_gpu

QUESTION 3

3 A3-Coding-Correctness 4 / 4

✓ **+ 4 pts** *Correct*

- **- 1 pts** 1dconv_cpu
- **- 1 pts** 1dconv_gpu
- **- 1 pts** dwspconv2d_gpu
- **- 1 pts** gemm_gpu

QUESTION 4

4 A3-Leader-baord 5.8 / 6

- **+ 6 pts** Score
- **+ 5 pts** Click here to replace this description.
- **+ 4 pts** Click here to replace this description.
- **+ 3 pts** Click here to replace this description.
- **+ 2 pts** Click here to replace this description.
- **+ 1 pts** Click here to replace this description.
- **+ 0 pts** Click here to replace this description.

+ 5.8 *Point adjustment*

Machine Learning Hardware Assignment 3 - **USING 1 LATE DAY**

Naman Makkar (nbm49)

April 2023

1 Introduction

The aim of this assignment was to make use of the TVM compiler to optimize DNN primitives like 1D Convolution, Matrix Multiplication and Depthwise Separable 2D Convolution on both CPU and GPU.

2 1D Convolution CPU Scheduler

It was observed that the optimized implementation of Conv 1D on CPU achieved a runtime of **0.29348 ms** on Google Colab hardware whereas the original implementation achieved a runtime of **153.1 s** on the same hardware.

This was achieved by making use of input padding, parallelism and vectorization, loop unrolling, and shared memory usage. Padding is utilized to yield an output that matches the shape of the input reducing the dependence of the convolution operation on if-else statements and speeds up the computation by allowing for greater parallelism and vectorization. Loops are split and reordered into an outer and inner loop and the inner loop of the reduction axis is unrolled in order to reduce the loop overhead. Shared memory usage is utilized and the data for the padded matrix A and filter W is cached in local memory when accessed by the output matrix B for computation.

3 1D Convolution GPU Scheduler

It was observed that the optimized implementation of Conv 1D on GPU achieved a runtime of **0.018239 ms** on the T4 GPU on Google Colab.

The optimization was achieved by making use of padding, blocking, threading and parallelism. The input is padded identical to the padding utilized for 1D Conv on CPU. Loop splitting and parallelism is utilized for the caclulation of both the padding and the final convolved output matrix B. This process of loop splitting and parallelism is carried out by first splitting the tensors to be

computed into outer and inner loops, distributing the computation of the matrices on blocks and threads of the GPU with the outer loops being computed on blocks of the GPU and the inner loops being computer on threads of the GPU. This is logical since blocks are a collection of threads and the most intensive computation of the outer loops is carried out on the blocks while that of the inner loops is carried out on the threads in order to reduce the loop overhead.

4 GEMM GPU Scheduler

It was observed that the optimized implementation of GEMM on GPU achieved a runtime of **6.358848 ms** on the T4 GPU on Google Colab, compared to the original implementation which achieved a runtime of **7.63 s** on the T4 GPU on Google Colab.

The optimization made use of loop unrolling, blocking and thread tiling in addition to shared memory usage. The computation of the matrix multiplication was divided into smaller submatrices using tiling into blocks of size 32 x 32 with each block further divided into inner dimensions of 32 x 32. The outer dimensions are computed on the blocks of the GPU, additionally the reduction axis was split into an outer and inner loop after which a loop reordering was carried out followed by further parallelization by binding the split tensors to the blocks and threads of the GPU. Finally, the input matrices were cached in local memory while ensuring that the caching of A and B from global to local memory was carried out within the outer loop of the reduction axis.

5 Depthwise Separable 2D Convolution GPU Scheduler

The optimized implementation of Depthwise Separable 2D Convolution achieved a runtime of **0.103679 ms** on the Google Colab T4 GPU.

The optimization makes use of padding, tiling, loop reordering and parallelism. Padding is carried out around the height and width dimensions of the input and the convolution is calculated using the sum of the element-wise product of the padded input and the filter along the 2 reduction axes, each of size K. The output tensor dimensions are split into outer and inner loops and loop reordering is carried out in order to parallelize the computation of the outer loops on the blocks of the GPU whereas the inner loops are parallelized on the threads of the GPU. Similarly, the padded input tensor is also split into an outer and inner loop, followed by loop reordering and the computation is parallelized across blocks and threads of the GPU. In order to maximise the parallelism of the computations all three axes (x, y and z) of the GPU blocks and threads are utilised.

1 A3-Report **2 / 2**

✓ **+ 2 pts** *Complete*

Machine Learning Hardware Assignment 3 - **USING 1 LATE DAY**

Naman Makkar (nbm49)

April 2023

1 Introduction

The aim of this assignment was to make use of the TVM compiler to optimize DNN primitives like 1D Convolution, Matrix Multiplication and Depthwise Separable 2D Convolution on both CPU and GPU.

2 1D Convolution CPU Scheduler

It was observed that the optimized implementation of Conv 1D on CPU achieved a runtime of **0.29348 ms** on Google Colab hardware whereas the original implementation achieved a runtime of **153.1 s** on the same hardware.

This was achieved by making use of input padding, parallelism and vectorization, loop unrolling, and shared memory usage. Padding is utilized to yield an output that matches the shape of the input reducing the dependence of the convolution operation on if-else statements and speeds up the computation by allowing for greater parallelism and vectorization. Loops are split and reordered into an outer and inner loop and the inner loop of the reduction axis is unrolled in order to reduce the loop overhead. Shared memory usage is utilized and the data for the padded matrix A and filter W is cached in local memory when accessed by the output matrix B for computation.

3 1D Convolution GPU Scheduler

It was observed that the optimized implementation of Conv 1D on GPU achieved a runtime of **0.018239 ms** on the T4 GPU on Google Colab.

The optimization was achieved by making use of padding, blocking, threading and parallelism. The input is padded identical to the padding utilized for 1D Conv on CPU. Loop splitting and parallelism is utilized for the caclulation of both the padding and the final convolved output matrix B. This process of loop splitting and parallelism is carried out by first splitting the tensors to be

computed into outer and inner loops, distributing the computation of the matrices on blocks and threads of the GPU with the outer loops being computed on blocks of the GPU and the inner loops being computer on threads of the GPU. This is logical since blocks are a collection of threads and the most intensive computation of the outer loops is carried out on the blocks while that of the inner loops is carried out on the threads in order to reduce the loop overhead.

4 GEMM GPU Scheduler

It was observed that the optimized implementation of GEMM on GPU achieved a runtime of **6.358848 ms** on the T4 GPU on Google Colab, compared to the original implementation which achieved a runtime of **7.63 s** on the T4 GPU on Google Colab.

The optimization made use of loop unrolling, blocking and thread tiling in addition to shared memory usage. The computation of the matrix multiplication was divided into smaller submatrices using tiling into blocks of size 32 x 32 with each block further divided into inner dimensions of 32 x 32. The outer dimensions are computed on the blocks of the GPU, additionally the reduction axis was split into an outer and inner loop after which a loop reordering was carried out followed by further parallelization by binding the split tensors to the blocks and threads of the GPU. Finally, the input matrices were cached in local memory while ensuring that the caching of A and B from global to local memory was carried out within the outer loop of the reduction axis.

5 Depthwise Separable 2D Convolution GPU Scheduler

The optimized implementation of Depthwise Separable 2D Convolution achieved a runtime of **0.103679 ms** on the Google Colab T4 GPU.

The optimization makes use of padding, tiling, loop reordering and parallelism. Padding is carried out around the height and width dimensions of the input and the convolution is calculated using the sum of the element-wise product of the padded input and the filter along the 2 reduction axes, each of size K. The output tensor dimensions are split into outer and inner loops and loop reordering is carried out in order to parallelize the computation of the outer loops on the blocks of the GPU whereas the inner loops are parallelized on the threads of the GPU. Similarly, the padded input tensor is also split into an outer and inner loop, followed by loop reordering and the computation is parallelized across blocks and threads of the GPU. In order to maximise the parallelism of the computations all three axes (x, y and z) of the GPU blocks and threads are utilised.

2 A3-Coding-Completeness **4 / 4**

- ✓ **+ 4 pts** *All complete.*
  - **- 1 pts** 1dconv_cpu
  - **- 1 pts** 1dconv_gpu
  - **- 1 pts** dwspconv2d_gpu
  - **- 1 pts** gemm_gpu

Machine Learning Hardware Assignment 3 -
USING 1 LATE DAY

Naman Makkar (nbm49)

April 2023

1 Introduction

The aim of this assignment was to make use of the TVM compiler to optimize DNN primitives like 1D Convolution, Matrix Multiplication and Depthwise Separable 2D Convolution on both CPU and GPU.

2 1D Convolution CPU Scheduler

It was observed that the optimized implementation of Conv 1D on CPU achieved a runtime of **0.29348 ms** on Google Colab hardware whereas the original implementation achieved a runtime of **153.1 s** on the same hardware.

This was achieved by making use of input padding, parallelism and vectorization, loop unrolling, and shared memory usage. Padding is utilized to yield an output that matches the shape of the input reducing the dependence of the convolution operation on if-else statements and speeds up the computation by allowing for greater parallelism and vectorization. Loops are split and reordered into an outer and inner loop and the inner loop of the reduction axis is unrolled in order to reduce the loop overhead. Shared memory usage is utilized and the data for the padded matrix A and filter W is cached in local memory when accessed by the output matrix B for computation.

3 1D Convolution GPU Scheduler

It was observed that the optimized implementation of Conv 1D on GPU achieved a runtime of **0.018239 ms** on the T4 GPU on Google Colab.

The optimization was achieved by making use of padding, blocking, threading and parallelism. The input is padded identical to the padding utilized for 1D Conv on CPU. Loop splitting and parallelism is utilized for the caclulation of both the padding and the final convolved output matrix B. This process of loop splitting and parallelism is carried out by first splitting the tensors to be

computed into outer and inner loops, distributing the computation of the matrices on blocks and threads of the GPU with the outer loops being computed on blocks of the GPU and the inner loops being computer on threads of the GPU. This is logical since blocks are a collection of threads and the most intensive computation of the outer loops is carried out on the blocks while that of the inner loops is carried out on the threads in order to reduce the loop overhead.

4 GEMM GPU Scheduler

It was observed that the optimized implementation of GEMM on GPU achieved a runtime of **6.358848 ms** on the T4 GPU on Google Colab, compared to the original implementation which achieved a runtime of **7.63 s** on the T4 GPU on Google Colab.

The optimization made use of loop unrolling, blocking and thread tiling in addition to shared memory usage. The computation of the matrix multiplication was divided into smaller submatrices using tiling into blocks of size 32 x 32 with each block further divided into inner dimensions of 32 x 32. The outer dimensions are computed on the blocks of the GPU, additionally the reduction axis was split into an outer and inner loop after which a loop reordering was carried out followed by further parallelization by binding the split tensors to the blocks and threads of the GPU. Finally, the input matrices were cached in local memory while ensuring that the caching of A and B from global to local memory was carried out within the outer loop of the reduction axis.

5 Depthwise Separable 2D Convolution GPU Scheduler

The optimized implementation of Depthwise Separable 2D Convolution achieved a runtime of **0.103679 ms** on the Google Colab T4 GPU.

The optimization makes use of padding, tiling, loop reordering and parallelism. Padding is carried out around the height and width dimensions of the input and the convolution is calculated using the sum of the element-wise product of the padded input and the filter along the 2 reduction axes, each of size K. The output tensor dimensions are split into outer and inner loops and loop reordering is carried out in order to parallelize the computation of the outer loops on the blocks of the GPU whereas the inner loops are parallelized on the threads of the GPU. Similarly, the padded input tensor is also split into an outer and inner loop, followed by loop reordering and the computation is parallelized across blocks and threads of the GPU. In order to maximise the parallelism of the computations all three axes (x, y and z) of the GPU blocks and threads are utilised.

3 A3-Coding-Correctness **4 / 4**

✓ **+ 4 pts** *Correct*

- **- 1 pts** 1dconv_cpu
- **- 1 pts** 1dconv_gpu
- **- 1 pts** dwspconv2d_gpu
- **- 1 pts** gemm_gpu

Machine Learning Hardware Assignment 3 - **USING 1 LATE DAY**

Naman Makkar (nbm49)

April 2023

1 Introduction

The aim of this assignment was to make use of the TVM compiler to optimize DNN primitives like 1D Convolution, Matrix Multiplication and Depthwise Separable 2D Convolution on both CPU and GPU.

2 1D Convolution CPU Scheduler

It was observed that the optimized implementation of Conv 1D on CPU achieved a runtime of **0.29348 ms** on Google Colab hardware whereas the original implementation achieved a runtime of **153.1 s** on the same hardware.

This was achieved by making use of input padding, parallelism and vectorization, loop unrolling, and shared memory usage. Padding is utilized to yield an output that matches the shape of the input reducing the dependence of the convolution operation on if-else statements and speeds up the computation by allowing for greater parallelism and vectorization. Loops are split and reordered into an outer and inner loop and the inner loop of the reduction axis is unrolled in order to reduce the loop overhead. Shared memory usage is utilized and the data for the padded matrix A and filter W is cached in local memory when accessed by the output matrix B for computation.

3 1D Convolution GPU Scheduler

It was observed that the optimized implementation of Conv 1D on GPU achieved a runtime of **0.018239 ms** on the T4 GPU on Google Colab.

The optimization was achieved by making use of padding, blocking, threading and parallelism. The input is padded identical to the padding utilized for 1D Conv on CPU. Loop splitting and parallelism is utilized for the caclulation of both the padding and the final convolved output matrix B. This process of loop splitting and parallelism is carried out by first splitting the tensors to be

computed into outer and inner loops, distributing the computation of the matrices on blocks and threads of the GPU with the outer loops being computed on blocks of the GPU and the inner loops being computer on threads of the GPU. This is logical since blocks are a collection of threads and the most intensive computation of the outer loops is carried out on the blocks while that of the inner loops is carried out on the threads in order to reduce the loop overhead.

4 GEMM GPU Scheduler

It was observed that the optimized implementation of GEMM on GPU achieved a runtime of **6.358848 ms** on the T4 GPU on Google Colab, compared to the original implementation which achieved a runtime of **7.63 s** on the T4 GPU on Google Colab.

The optimization made use of loop unrolling, blocking and thread tiling in addition to shared memory usage. The computation of the matrix multiplication was divided into smaller submatrices using tiling into blocks of size 32 x 32 with each block further divided into inner dimensions of 32 x 32. The outer dimensions are computed on the blocks of the GPU, additionally the reduction axis was split into an outer and inner loop after which a loop reordering was carried out followed by further parallelization by binding the split tensors to the blocks and threads of the GPU. Finally, the input matrices were cached in local memory while ensuring that the caching of A and B from global to local memory was carried out within the outer loop of the reduction axis.

5 Depthwise Separable 2D Convolution GPU Scheduler

The optimized implementation of Depthwise Separable 2D Convolution achieved a runtime of **0.103679 ms** on the Google Colab T4 GPU.

The optimization makes use of padding, tiling, loop reordering and parallelism. Padding is carried out around the height and width dimensions of the input and the convolution is calculated using the sum of the element-wise product of the padded input and the filter along the 2 reduction axes, each of size K. The output tensor dimensions are split into outer and inner loops and loop reordering is carried out in order to parallelize the computation of the outer loops on the blocks of the GPU whereas the inner loops are parallelized on the threads of the GPU. Similarly, the padded input tensor is also split into an outer and inner loop, followed by loop reordering and the computation is parallelized across blocks and threads of the GPU. In order to maximise the parallelism of the computations all three axes (x, y and z) of the GPU blocks and threads are utilised.

4 A3-Leader-baord 5.8 / 6

- **+ 6 pts** Score
- **+ 5 pts** Click here to replace this description.
- **+ 4 pts** Click here to replace this description.
- **+ 3 pts** Click here to replace this description.
- **+ 2 pts** Click here to replace this description.
- **+ 1 pts** Click here to replace this description.
- **+ 0 pts** Click here to replace this description.

+ 5.8 *Point adjustment*