Georgia Institute of Technology

ECE 8803: Hardware-Software Co-Design for Machine Learning Systems (HML)

Spring 2025

Lab 1B

Due: Sunday, February 2, 2025 @ 11:59 pm EST

Contact for any queries: Ritik Raj (ritik.raj@gatech.edu)
Adopted from previous version created by Abhimanyu Bambhaniya

Instructions

Please read the following instructions carefully.

- The lab is distributed in 2 parts (Each worth 8 points).
- You will need to modify the code in various files, generate output csv and submit them independently.
- Rename the zip according to the format:
- **LastName_FirstName_GTID_ECE_8803_HML_sp25_lab1B.zip**
  Note: Any other format will result in 1 point deduction (12.5% of the lab weightage)
- Submit the generated csv file separately. **DO NOT MODIFY IT.**
- It is encouraged for you to discuss homework problems amongst each other, but any copying is strictly prohibited and will be subject to Georgia Tech Honor Code.
- Late homework is not accepted unless arranged otherwise and in advance.
- Comment on your codes.
- For all problem, please post queries on piazza.

Lab Layout

Part B.1: Operator Fusion – 4 points

- Implementing operator fusion
- Identifying various fusion operations opportunities among different operations.

Part B.2: Dataflow aware roofline – 3 points.

- A stationary
- B stationary
- C stationary
- Bonus – 1 point

Part B.3: Quantization of data – 1 point

- Data fetch quantization
- Quantization with different compute units.

Lab Setup

1. Please download the zip file and extract the contents to access the code base.
2. Copy all python files after you have completed lab1A. [Note: Keep a separate copy for lab1A submission]
    a. Files: unit.py, system.py, operator_base.py, operator.py, analyse_model.py, plot_rooflines.py.
3. Start the lab.

Lab Submission

- Submission date: Feb 2nd
- Submission format: 1 zip + all csv.
    - Keep all csv in zip also.

Lab details

Part B.1: Operator Fusion.

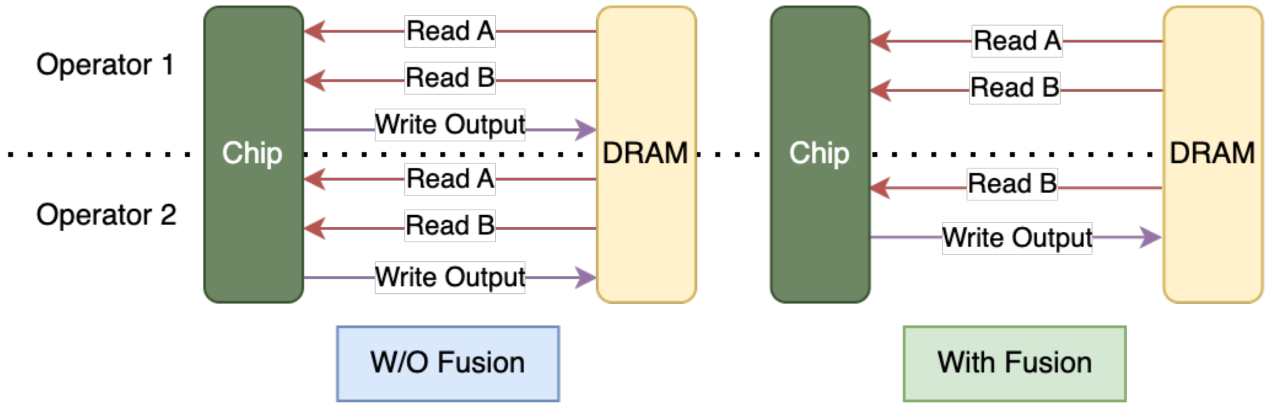
B.1.i) Implement operator fusion

In typical implementation that we did in part A, each operation consists of 3 memory accesses. Both input operators are read in from the external and output of the operation is written back.

A special feature of DNNs is that output of an operation generally feeds the input of next operation. We can take use of this feature by keeping the output on chip, this technique is called operator fusion.

In the operator fusion, the output of 1st operator feeds in directly to next operator. In terms of our simulator this would **reduce the number of memory operations**.

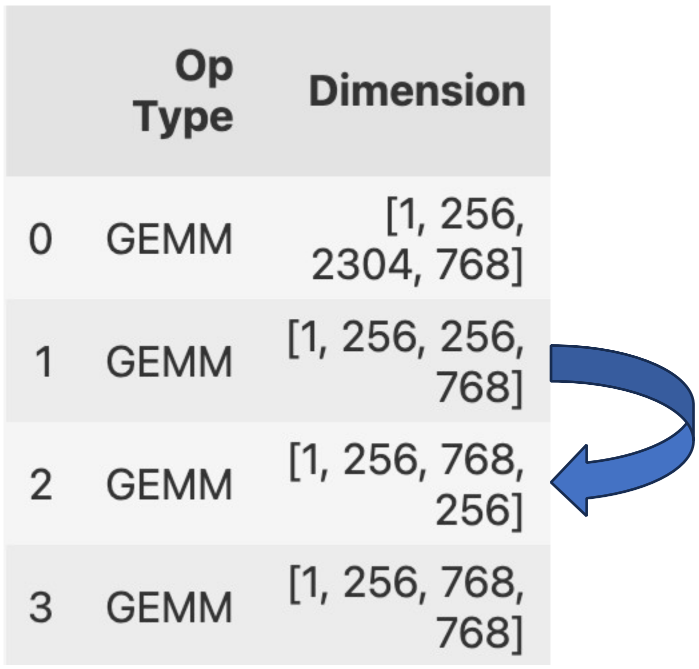
Once an operator is fused, we don't write its output back to external memory, and input A to next operator is not to be fetched.



We would do this in the following way:

Currently, the models are called without operator fusion. Let's take Alexnet as an example:

```
alexnet_on_a100_df = analysis_model(bert(1), A100_GPU)
display(alexnet_on_a100_df)
```



	Op Type	Dimension
0	GEMM	[1, 256, 2304, 768]
1	GEMM	[1, 256, 256, 768]
2	GEMM	[1, 256, 768, 256]
3	GEMM	[1, 256, 768, 768]

We will call fusion between GEMM1, GEMM2. The output of GEMM1 is sent to GEMM2.

```
alexnet_on_a100_df = analysis_model(bert(1), A100_GPU, fusion = [(1,2)])
display(alexnet_on_a100_df)
```

In the above example, we fused operators 1 and 2 (note that operators 1 and 2 correspond to 2nd and 3rd GEMM layers in the above example since we start indexing from 0). To do multiple fusions say we want to fuse operators a, b and c together and also d and e together. To do that, the syntax should be fusion = [(a, b, c), (d,e)]

Your Tasks:

1. Modify the analysis_model() function in analye_model.py to accept fusion as input parameter.
2. Modify appropriate function in operator_base.py to change memory fetches and modified memory time when fusion is done.
3. After completing the code modifications, run cells in lab1B.ipynb till TODO B1.i and generate output_b1.csv.

B.1.ii) Identify possible speedup with fusion.

1. We have instantiated an attention model in lab1B.ipynb part TODO B1. ii), first run this model on A100_GPU.

2. You are allowed to perform any number of legal pairwise fusions. What is the minimum number of operator fusions, and which specific pairs help speed up the network. Identify the minimum number of operator fusion that can speed up this network.

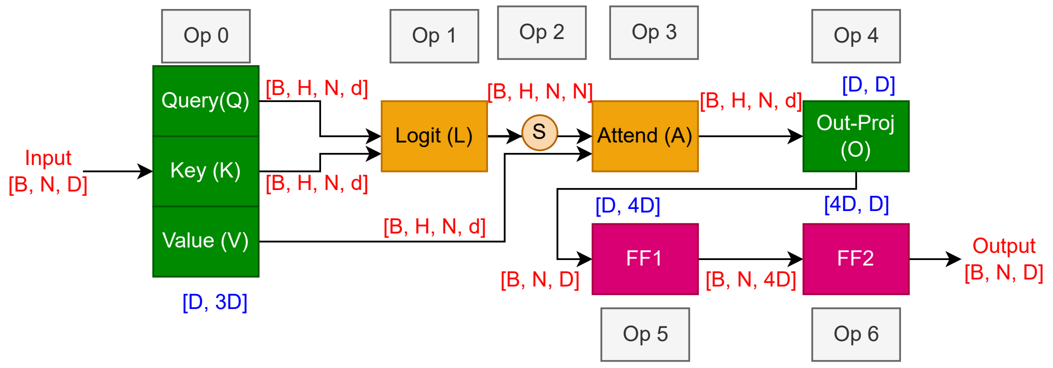


Figure: A typical transformer layer. We show 6 matrix multiplication operations and a SoftMax operation in the figure. All weights dimensions are shown in blue color, and activations are shown in red color. Note: we ignore the add operations for simplicity.

You are allowed to perform any number of legal pairwise fusions. Legal pairwise fusion is fusing any 2 operations that are connected in the graph by arrow.

3. Generate output_b2.csv with configuration obtained in the previous part.
4. Comment on the type of operations that benefit from operator fusion. Why do you think, operator fusion is able to help in bringing the runtime down.

Part B.2: Dataflow aware roofline

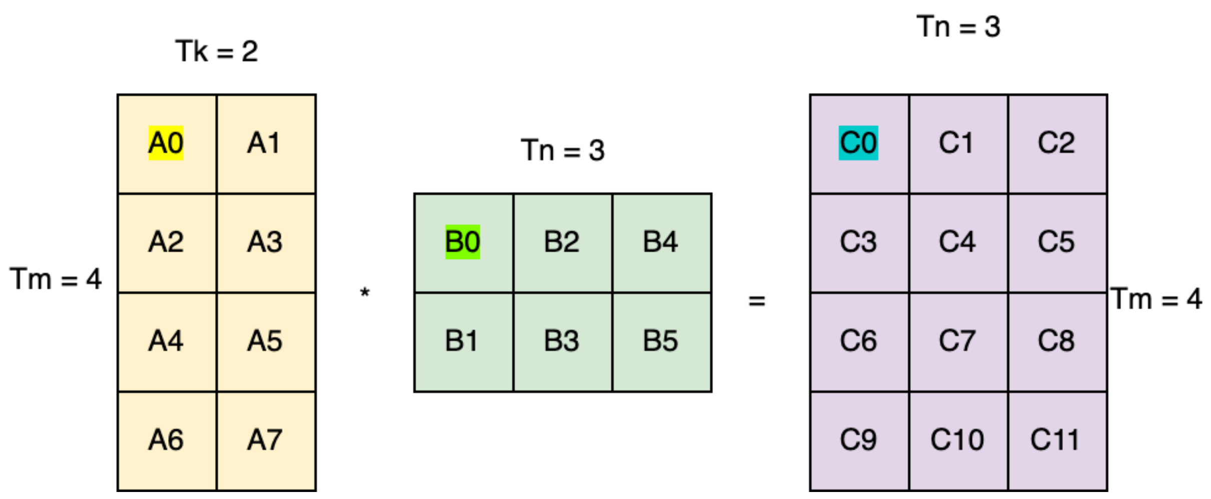
In part A of the lab, we have assumed that all data of an operation, i.e. Input A, B and output can fit completely in the on-chip memory. But with large models, this might not always be true.

In this case, we use technique called tiling, where each input is broken are split down into multiple "tiles". Using this technique, the output is computed in tiles.

Note: Each tile in A, B, C will be a matrix itself.

Example: Large matrix A [4096 x 4096], B [4096 x 1536] are divided into tiles of 4x2 and 2x3. The output will have 12 tiles of dimension 4x3.

Dim of A_i: [1024 x 2048], B_i: [2048 x 512], C_i: [1024 x 512].



C0 = A0 *B0 + A1*B1
C1 = A0 *B2 + A1*B3
C2 = A0 *B4 + A1*B5
C3 = A2* B0 + A3*B1
C4 = A2*B2 + A3*B3
C5 = A2*B4 + A3*B5
C6 = A4* B0 + A5*B1
C7 = A4*B2 + A5*B3
C8 = A4*B4 + A5*B5
C9 = A6* B0 + A7*B1
C10 = A6*B2 + A7*B3
C11 = A6*B4 + A7*B5

Assume that you can only fit 1 tile of each A_i, B_i, C_i on chip.

This requires that some of the tiles be re-fetched from external memory to chip.
We will be understanding 3 different tiling strategy:

B.2.i) A stationary.

Below we have provided a brief snapshot of how tile A0 is kept stationary on chip and Bi and Ci are read, calculated, and written back.

Reads	Reads	Read + Write
A0	B0	C0
	B2	C1
	B4	C2

In total we do 1+3+3 = 7 tile reads and 3 tile writes to fully use A0 tile.

B.2.ii) B stationary

Below we have provided a brief snapshot of how tile B0 is kept stationary on chip and Ai and Ci are read, calculated, and written back.

Reads	Reads	Read + Write
A0	B0	C0
A2		C3
A4		C6
A6		C9

In total we do 4+1+4 = 9 tile reads and 4 tile writes to fully use B0 tile.

B.2.iii) C stationary

Below we have provided a brief snapshot of how tile C0 is kept stationary on chip and Ai and Bi are read and used.

Reads	Reads	Read + Write
A0	B0	C0
A1	B1	

In total we do 2+2+1 = 5 tile reads and 1 tile write to fully calculate C0 tile.

Your job is to calculate the total number of memory access in each case.

In the ideal case (Part A) total number of memory fetches = □Ai + □Bi + □Ci. i.e. each tile is only read once.

In this part, you would be calculating the number of memory fetches. This would be of the form.

$$f(T_m, T_n, T_k) * \sum Ai \quad + \quad g(T_m, T_n, T_k) * \sum Bi \quad + \quad h(T_m, T_n, T_k) * \sum Ci$$

For each tiling strategy, calculate the corresponding value of *f, g, & h .*
Hint: For each tiling strategy, one of the three functions will have the value 1.

Your Tasks:

1. Derive the number of memory fetches for all tiling strategy described above.

2. Tiling will be defined as shown below. Tm, Tk, Tn represents value as shown in the figure above.

```
model_b2_a_tiling = analysis_model(model_b2, A100_GPU, tiling = 'A', Tm = 4, Tk = 2, Tn = 3)
```

3. Modify appropriate function in operator_base.py and anlyse_model.py to change memory fetches and modified memory time.
4. After completing the code modifications, run cells in lab1B.ipynb till TODO B1.iii and generate all output_b3*.csv.

B.2.iv) [Bonus] Find the best tiling strategy - 1 points.

Find the tiling strategy that would require minimum fewer number of memory access compared to these three strategies.

Explain the strategy by giving example in form a table of reads and write for Ai, Bi, Ci. Also derive the *f , g, & h* for the best strategy.

Part B.3: Quantization of data

In part A of the lab, we have assumed the data is in FP32 format.
Modern accelerators have capabilities to handle multiple different data types. Various datatype possible = bf16, int8, fp32, fp64.

B.3.i) Change in Data access

We would be passing the *data_format* to analysis_model function, which can further pass to appropriate functions.

```
alexnet_on_a100_df = analysis_model(bert(1), A100_GPU, fusion = None, data_format='int8')
display(alexnet_on_a100_df)
```

Write the code to change the memory time according to the data format of the model.

1. Modify analysis_model function and appropriate function in operator_base.py.
2. Generate output_b4.csv in lab1b.ipynb.
3. We have instantiated a network in lab1B.ipynb part TODO B.3.i), run this model on A100_GPU with all 4 possible data format.

B.3.ii) Change in compute ops

Now that we have changed how data is stored in memory. With quantization we can also have different amount compute ops available.

Please refer to the below table and change the system.ops_per_sec for respective data formats.

BF16	Int8	FP32	FP64
X	X*2	X/2	X/4

Write the code the change the compute time according to the data format of the model.

1. Modify appropriate function in operator_base.py.
2. Generate output_b5.csv in lab1b.ipynb.
3. We have instantiated an network in lab1B.ipynb part TODO B.3.ii), run this model on A100_GPU with all 4 possible data format.