Final Project – Text Generation

Project due: 2025. 12. 19. 11:59 PM

CHAMP

SNU THUNDER Research Group

Project Goal

- Optimize LFM2-8B-A1B inference
  - You are given a multi-thread CPU code.
  - Parallelize and optimize the code across four nodes.
  - You can use x86 intrinsics, Pthread, OpenMP, MPI and CUDA.
  - You cannot use external libraries.

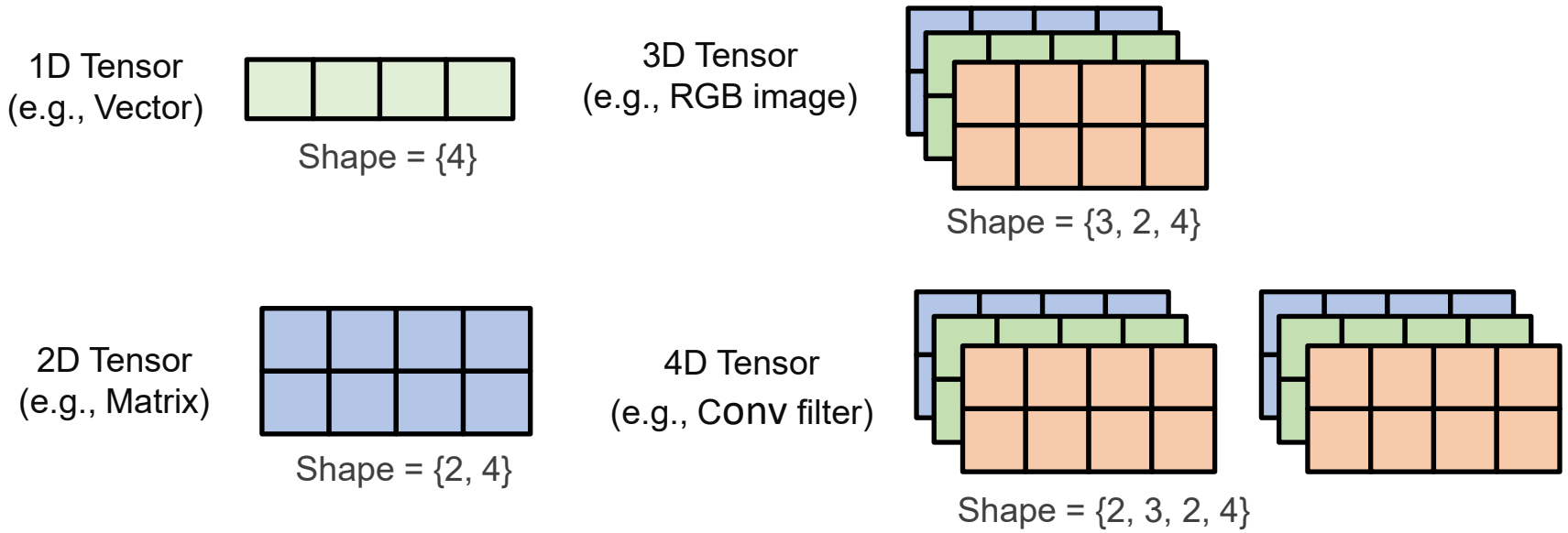
Background

- You don’t have to understand all background details.
  - You can start optimizing the code by focusing solely on its calculation and memory access patterns.

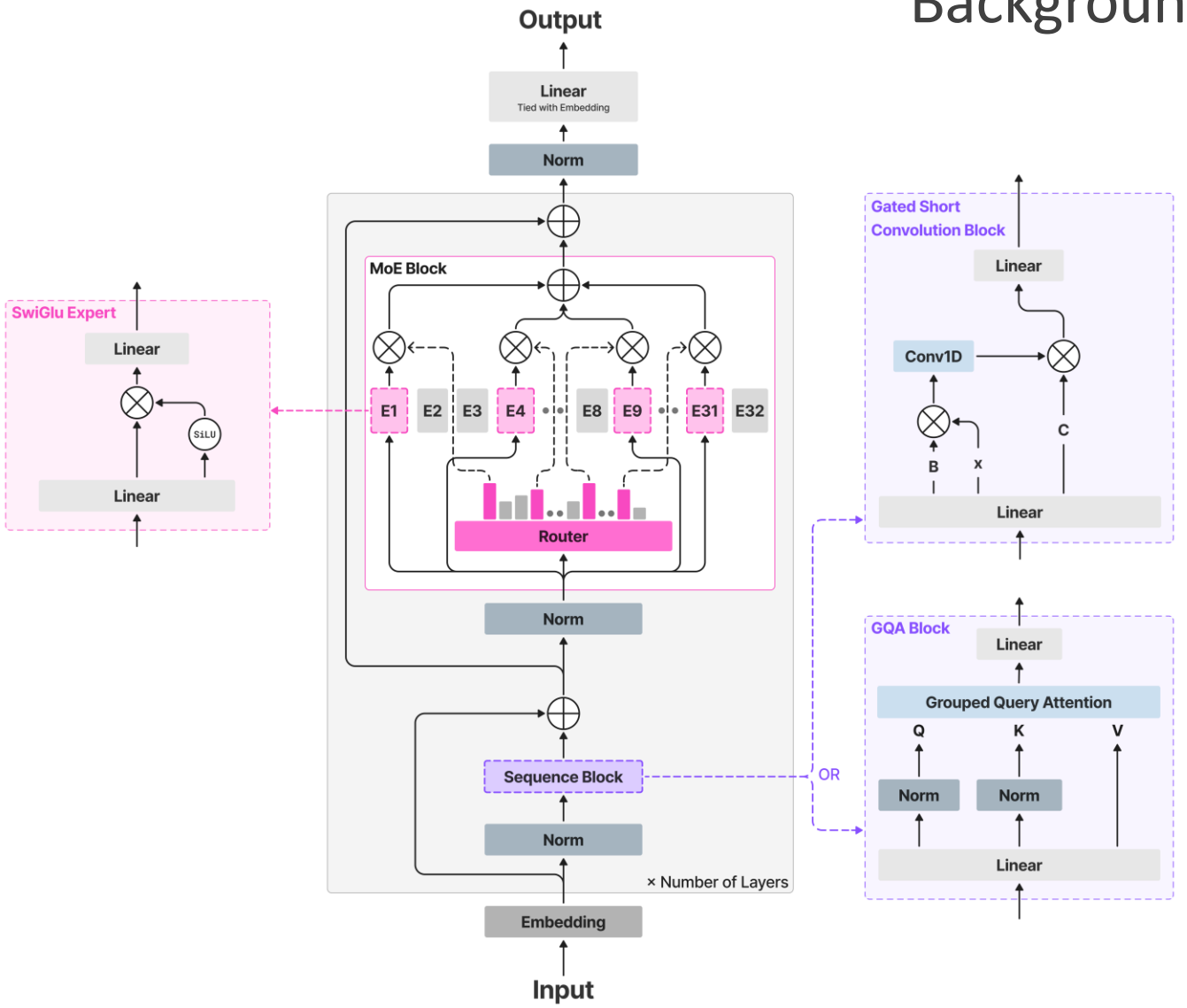
- Having this knowledge you can try much broader range of optimization techniques.

Background - Tensor

- In this project, data is primarily handled in units called tensors.
  - The operations implemented in the provided skeleton code take tensors as both input and output.
  - Definition: `include/tensor.h`, Implementation: `src/tensor.cu`



Background - Model



https://www.liquid.ai/blog/lfm2-8b-a1b-an-efficient-on-device-mixture-of-experts

Background - Model

- Embedding
  - Map a token to embedding vector

- RMSNorm
  - Normalizes the hidden state by its root mean square
  - $\mathrm{RMSNorm}(x_i) = \frac{x_i}{\mathrm{RMS}(\mathbf{x})} \cdot \gamma_i = \frac{x_i}{\sqrt{\frac{1}{D}\sum_{j=1}^{D} x_j^2 + \epsilon}} \cdot \gamma_i$

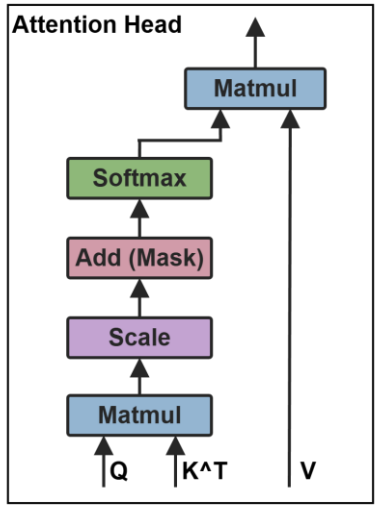
Background - Model

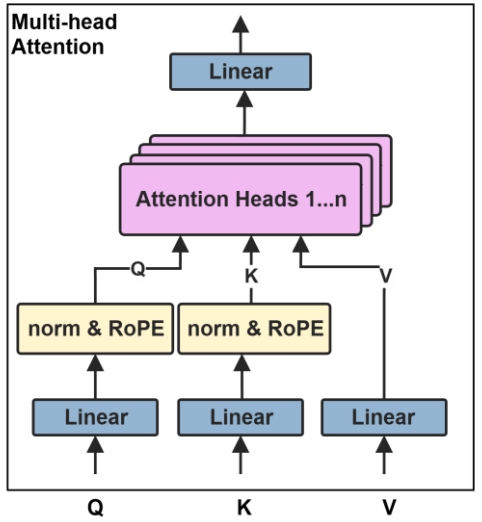
- Rotary Positional Embedding (RoPE)
  - Encodes the absolute position of tokens by applying a rotation matrix to the Query and Key vectors in the self-attention calculation.

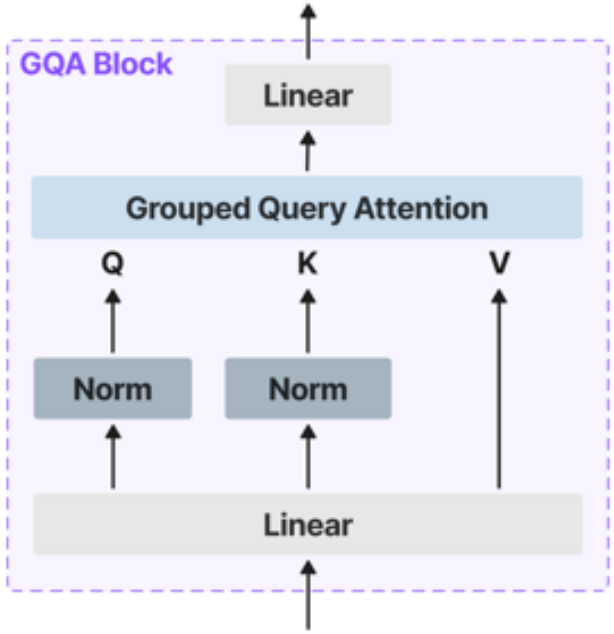
```
float q1 = q.at(b, h, s, d);                  // first half
float q2 = q.at(b, h, s, d + half_dim);       // second half

// q_rotated = q * cos + rotate_half(q) * sin
// rotate_half(q) = [-q2, q1]
q.at(b, h, s, d) = q1 * cos.at(s, d) + (-q2) * sin.at(s, d);
q.at(b, h, s, d + half_dim) = q2 * cos.at(s, d + half_dim) + q1 * sin.at(s, d + half_dim);
```

Background - Model

- Attention Block
  - Computes attention scores from the scaled dot product of query and key vectors, applying a causal mask to prevent attention to certain elements (e.g., future tokens), and then uses the resulting probabilities to weight the value vectors.

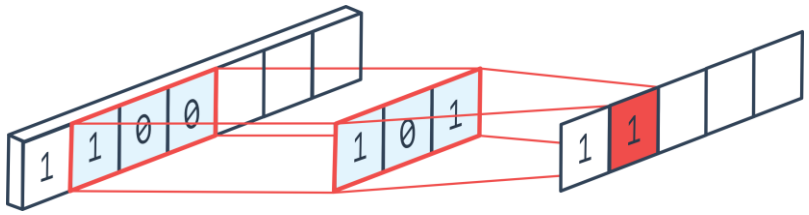


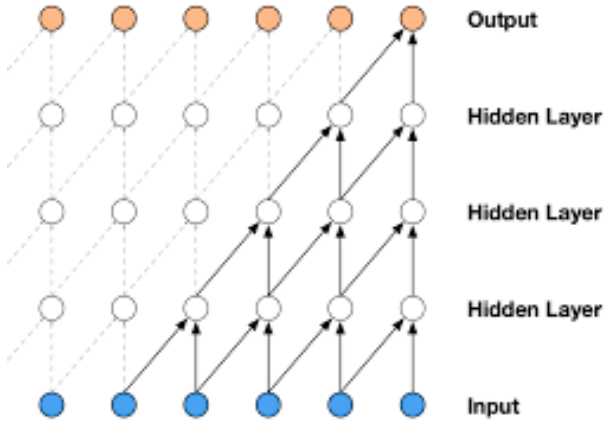


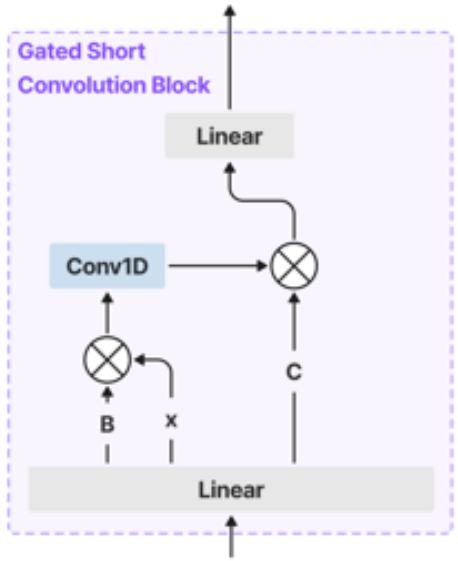


Background - Model

- Convolution Block
  - The ShortConv block computes the output by first projecting the input into three streams (B, C, and x), it then applies a causal 1D convolution to the gated input (B multiplied by x_gate) to efficiently capture local dependencies without looking forward in the sequence and finally multiplies this convolved output by the C stream (gating) to produce the result.

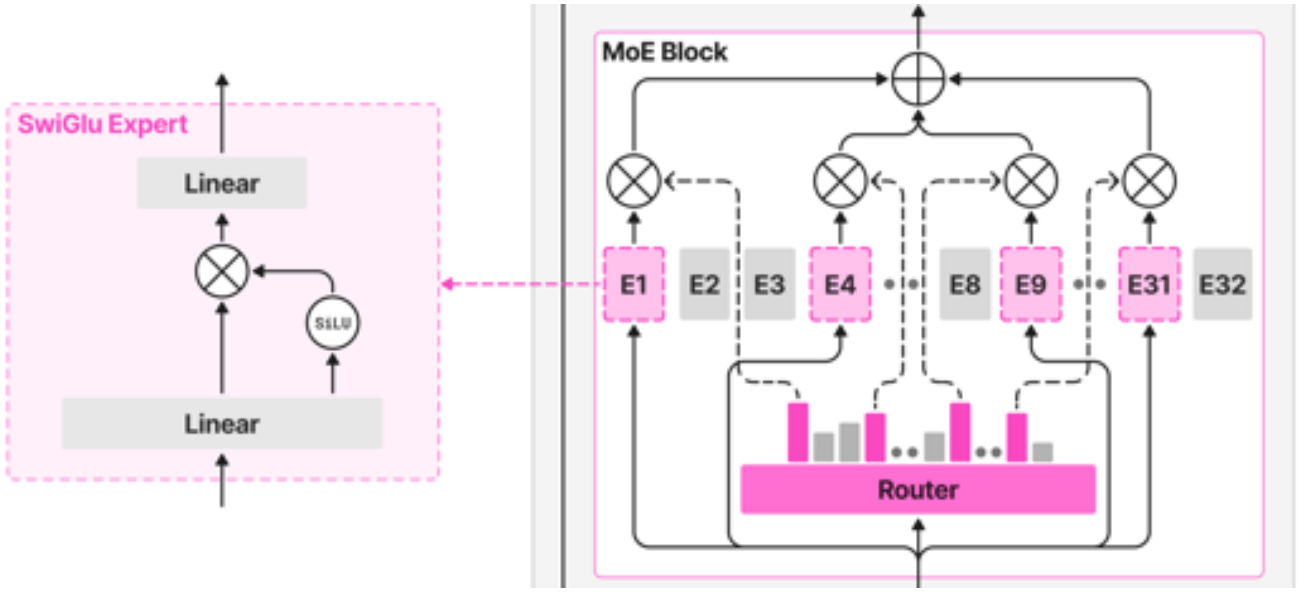






Background - Model

- Sparse Mixture-of-Experts (MoE) Block
  - The MoE Block leverages a Router to selectively send each token's input to a small set of SiLU Experts (Top-4) and then weighted sums their outputs.



Background - Models

- Seeing the code may be slightly easier to understand than listening to a verbal explanation.
  - Definition: `include/layer.h, include/model.h` Implementation: `src/layer.cu, src/model.cu`

- Please refer to the following link for the PyTorch implementation of this model
  - https://github.com/huggingface/transformers/tree/main/src/transformers/models/lfm2_moe

Skeleton Code - Basic block

- The skeleton codes is located at /shpc/skeleton/final-project/tests.

- Files you can modify and should submit
  - attn.cu, conv.cu, moe.cu
  - run.sh: You may edit this script to add or adjust program execution options as needed.
  - The project will be evaluated using the command: ./run.sh -v
  - Before submission, make sure to modify run.sh using the best-performing configuration (e.g., with the -n option)

Skeleton Code – Full Model

- The skeleton codes is located at /shpc/skeleton/final-project.

- Files you must not modify
  - inputs.bin, answers.bin: Input and ground truth data files
  - Makefile: Core files that must remain unchanged

- Files you can modify and should submit (FP32 version and FP16 version (optional))
  - tensor.h, tensor.cu, layer.h, layer.cu
  - model.h, model.cu, model_loader.h, model_loader.cpp
  - main.cpp
  - run.sh: You may edit this script to add or adjust program execution options as needed.
  - Before submission, make sure to modify run.sh using the best-performing configuration (e.g., with the -n option)
    - You can select 16, 32, 64 or 128 (at least 16)

Constraints

- Modifications to the program logic or model architecture are Not Allowed.

- Examples of Allowed Modifications
  - Changing memory layout, Reordering loop structures, Adding padding data or operations, Applying operator fusion, etc

- Examples of Disallowed Modifications
  - Performing model inference outside the generate function
  - Replacing the model with a different one or using a different algorithm that produces the same output.
  - Skipping model loading based on known expert activations is not allowed, as it assumes unrealistic deployment conditions.

- If you are unsure whether a modification is allowed, please consult the TA.

Grading

- Report (10 points): Briefly describe the optimizations you applied.
  - Include the following: Performance measured by yourself with screenshot
- Basic block (Conv, Attn, MoE) implementation (30 points, 10 points each)
  - Single basic block implementation in single GPU.
  - Full points awarded for performance exceeding a certain threshold. (linear)
    - Conv: 10000 token/s, Attn: 5000 token/s, MoE: 250 token/s
- Full model performance (60 points)
  - It is assigned through a **relative evaluation** based on the final performance achieved across all submissions. The scoring utilizes a **log scale** to differentiate results effectively. Crucially, all computations and parameters used for this evaluation **must be in FP32**.

Grading (Cont.)

- FP16 (using Tensor Core) implementation (Bonus points)
  - This bonus is worth 10% of your FP32 speed score.
  - Earn the full 10% bonus if the FP16 version speed doubles the speed of your own FP32 version.
  - If the FP16 version performance does not exceed 2x your FP32 speed, you can still earn the full 10% bonus by providing a detailed analysis of why the target speedup was not achieved.
  - Only the top-1 token for every input needs to be correct, not just within the 1e-3 validity tolerance.

Submission

- Deadline: 12/19 11:59:59 PM
  - You cannot use your grace day on final project.

- Submit your codes with shpc-submit script.

Comments

- It is crucial to understand the characteristics of each operation and the dependencies between them.
- We recommend starting only after thoroughly understanding the provided skeleton code.
- Always base your optimizations on evidence/data.
- The fundamental first step before optimization is to identify where the bottleneck lies.
  - [X] Professor said Kernel fusion is good.
  - [O] The number of kernel launches is too high. Let's reduce the kernel launch overhead through kernel fusion.
  - [X] The Select operation looks the most complex; maybe we should optimize it?
  - [O] Measuring the execution time reveals that the Select operation takes the longest.

Comments (Cont.)

- Potential Optimizations
  - Process multiple inputs as a batch.
  - Optimize the matrix multiplication operation, similar to past assignments.
  - Determine the best model parallelism strategy (DP, TP, PP, EP) learned in class by considering the combined effects of the node's computation performance and communication performance.
  - It might be beneficial to execute some operations on the CPU.
  - These are general suggestions and may not constitute the correct answer for every scenario.

Q&A

Updates