Matthew Katz
Hugo Latendresse

Parallelizing Mixture of Experts Transformers in FlexFlow for Low-Latency Inference

15-618 Final Project

Milestone Report

https://hugolatendresse.github.io/15618-final-project/

Progress

We spent the last weeks getting on-boarded with FlexFlow, since our ultimate goal is to add an MoE model capability to that framework. This onboarding took much more time than we initially estimated.

The first two weeks were spent trying to install FlexFlow and successfully run the existing examples (see Flexflow Installation section below). After multiple meetings with Professor Jia and Catalyst Lab member Gabriele Oliaro, we were finally able to successfully install on November 26th. While we were waiting for responses, we researched the Mixtral architecture and refined our strategies for parallelizing the Mixture of Experts mechanism.

At the moment, we have a work-in-progress regarding the implementation of MoE models (HuggingFace's MixtralForCausalLM architecture) in FlexFlow, both at the C++ level where we define the different layers of the model, and at the CUDA level where we define the operations specific to MoE. Our draft covers the entirety of the architecture. However, there are bugs right now, and we are actively working on making the code work. We are using the M4-ai/TinyMistral-6x248M model for development purposes. Once we get it working, we will try to benchmark using mistralai/Mixtral-8x7B-v0.1.

FlexFlow Installation

We tried to install FlexFlow using different methods, on different platforms, and tested builds using different models. We followed the installation documentation as best as we could but ran into difficulties each time.

We tried to pip install, build from source, and build while running the FlexFlow docker container locally on one of our ThinkPad machines running Ubuntu with an Nvidia GPU. We ran into difficulties installing and initializing the CUDA drivers and getting the FlexFlow installation process to recognize them.

We then tried the same with the PSC machines, which already have CUDA support. We tried all three methods again, but each time we ran into different dependency issues for each installation method. Upon trying to manually install missing dependencies, new issues would arise. Moreover, it seemed that FlexFlow did not recognize the CUDA installation on the PSC machines even after manually loading the CUDA module and setting the corresponding environment variables pointing to the CUDA driver and library locations. We tried several different ways to configure the dependencies, with and without the use of Singularity.

We then decided to spin up our own (GPU-enabled) AWS instances. We initially ran into the same type of dependency issues we experienced locally and with the PSC. We finally were able to complete the installation successfully by building a docker image, and were able to run a Python training example. However, C++ is preferable for development (for debugging), so we needed to ensure that the C++ inference examples worked. We ran into obscure error messages when we tried to run the inference examples with a LLaMA model, which is currently supported by FlexFlow.

The installation issues cost us more than two weeks of progress. In the end, we were able to schedule a meeting with Gabriele on November 26th. We spent more than an hour troubleshooting the installation process with him. With his help, we were able to get the LLaMA-3-1B C++ inference example working on a g6.4xlarge EC2 instance on AWS.

Next Step

Despite the hurdles, we are still confident we will be able to implement MoE models (incremental decoding only) in time for the post session. It is hard to perfectly predict how long it will take to get our implementation working, but we are making good progress debugging one issue after the other. Our immediate goal is to make TinyMistral work on a node with a single GPU. Once our implementation works, we will use the rest of the time we have to experiment with different parallelization strategies (see next section) across multiple GPUs.

Parallelization Strategies

Tensor and Pipeline Parallelism

FlexFlow supports tensor parallelism and pipeline parallelism. We hope that a successful implementation of our MoE model will be able to benefit from those two capabilities. It is worth noting that our MoE implementation is similar to FlexFlow's existing LlaMA implementation, except for the fact the regular MLP layers are replaced by MoE MLP layers.

The parallelization of MLPs reduces to parallelizing matrix multiplications with interspersed non-linear activations, and all popular Deep Learning frameworks already feature kernels to support this – including FlexFlow. We will use the pre-existing implementations of other models as much as we can rather than reinvent the wheel, especially for this baseline strategy.

Conceptually, passing a single token into the MoE layer involves the following steps:

1. Pass the tensor containing the token to the router.
2. The router outputs the indices of the top *k* experts (for TinyMistral and Mixtral-8x7B, *k* = *2*) and the corresponding softmax scores
3. Pass the tensor to the *k* experts and run the computation for each, sequentially
4. Multiply the output activations by the experts' softmax score
5. Add the *k* activations together

The difficulty lies in enabling the processing of many tokens, because each token in a sequence may be sent to a different selection of experts. We need to construct an input tensor for each expert consisting of the correct tokens. After computation and multiplication by softmax scores, we then need to gather the transformed copy of each token (e.g. if token *x* is passed to two experts, the experts in total will output two transformed *x*), add them together, and place them in the original sequence order in the final output tensor.

This baseline implementation is complex and may take more time than we had originally planned. We will focus on implementing this correctly before moving on to our next strategy.

Stretch Goal: Expert Parallelism

If time permits, we will explore expert parallelism. There are no data dependencies between experts, so after the Router layer, each expert computation can be done in parallel. This can be accomplished by mapping experts to different GPU devices. The theoretical maximum speedup over the baseline strategy would be linear with the number of devices.

Implementing this strategy would entail:

1. Getting access to a machine with multiple GPUs (doable with an AWS EC2 instance, but ideally we would use PSC, if we can figure out the installation on PSC)
2. Scattering the correct tokens to each GPU
3. Running the computation for each expert (potentially with the same code we wrote for the baseline strategy)
4. Gathering the output activations and permuting them in the manner described in the baseline strategy

A key part in expert parallelism is to decide how to allocate the experts to the different GPUs. The most naive way would be to allocate them randomly at compile time. A more sophisticated way would be to analyze which experts are the busiest and allocate the experts to the different GPUs to maintain good workload balance.

Poster Session and Deliverables

We will benchmark the per-token latency and throughput of our final implementation against the Hugging Face Transformers package. That library may already contain some optimizations, so we are not sure we will have enough time to optimize our FlexFlow optimization to beat it, but we will see.

For the poster session, we will create a poster explaining how FlexFlow works, describing the architecture of our chosen MoE model, and showing how we parallelized it. We initially planned to show token-by-token text generation in real time (something like the chatGPT user-interface), but we now doubt we will have enough time to set up such an interface.

Updated List of Goals

PLAN TO ACHIEVE

- Implement the key components of our baseline MoE model, namely a MoE MLP layer consisting of routers (gate functional units) and experts (FFNs). Incremental decoding only.
- Complete the implementation of a full MoE transformer by incorporating our work with existing FlexFlow capabilities for the traditional parts of MoE transformers (self-attention, etc.)
- Write other CUDA and C++ code to make our baseline model work with the FlexFlow API (inference only).
- Successfully serve a TinyMistral MoE model with FlexFlow.
- Benchmark per-token latency and throughput of our baseline model using FlexFlow vs the Hugging Face Transformers package.
- Create a poster explaining how FlexFlow works, describing the architecture of our chosen MoE model, and showing our process in parallelizing it.

HOPE TO ACHIEVE

- Implement Expert Parallelism.
- Beat Hugging Face Transformers in terms of per-token latency. We only "hope" to achieve that because that library already makes use of parallelization and is probably optimized to some extent, and because we are not using speculative decoding.
- Profile our implementation and identify the bottlenecks.
- Iterate on our implementation to achieve better speedups.
- Benchmark our implementation against other accelerators like vLLM and FasterTransformer.
- Have a Pull Request ready to be merged on the main branch of FlexFlow. We think it is likely this will be completed after the timeline of the 15-618 project.

Updated Schedule

Task	Initial Target	Revised Target	Status
Finalize the project proposal	11/11 - 11/17		Done
Familiarize ourselves with the FlexFlow repo	11/11 - 11/17		Done
Walk through FlexFlow's developer guide	11/11 - 11/17		Done
Meet with a member of Prof Jia's research team	11/11 - 11/17		Done
Successfully install FlexFlow on an EC2 instance	11/11 - 11/17		Done
Confirm the choice of baseline model	11/18 - 11/24		Done
Finalize our initial strategy to parallelize the baseline model	11/18 - 11/24		Done
Complete milestone report	11/18 - 11/24		Done
Complete MoE transformer code in C++ (and corresponding ops in CUDA)	11/18 - 11/24	12/2 - 12/5	In Progress
Successfully run incremental decoding on (M4-ai/TinyMistral-6x248M)	11/18 - 11/24	12/2 - 12/5	In Progress
Benchmark our implementation with Hugging Face Transformer inference	11/25 - 12/1	12/6 - 12/8	
Successfully install FlexFlow on the PSC machines	12/2 - 12/8	12/6 - 12/8	
Run and benchmark on a different MoE model, such as Mixtral-8x7B	12/2 - 12/8	12/8 - 12/11	
Implement expert parallelism (if time permits)	11/25 - 12/1	12/8 - 12/11	
Run benchmarking on the PSC machines	12/2 - 12/8	12/8 - 12/11	
Complete poster	12/2 - 12/8	12/8 - 12/11	
Complete final report	12/8 - 12/15	12/12 - 12/15	

Other details can be found on our project website
https://hugolatendresse.github.io/15618-final-project/