Accelerating Mixture-of-Experts Inference with a Lightweight Pipelined Gating Network

By Vaibhav Gurunathan

Background

Goal: Want to run inference with an ML model on a GPU faster

- You want to pipeline results between different layers

Layer1 * Layer1_Weights → RELU = Layer2

Layer2 * Layer2_Weights → RELU = Layer3

(Layer1 * Layer1_Weights → RELU) gets fused into matmul_relu op

Previously: Done before for standard models

- Doesn’t have a standard name

Problem

Problem: What if we don’t know the weights at compile time?

- Mixture of Experts Models use different experts based on the input
- Use a gating network to figure out what input requires what expert
- Generally use multiple experts per input
- Different experts per layer

Research Goal

Goal: Is it possible to use a lightweight gating network to properly preload experts to use for the inference process.

Hope: Decrease the time to first token by overlapping expert loading with computation

Approach: Train a small gating model to speculatively route inputs before the full gating decision

- Evaluate if pipelined expert preloading can reduce latency without hurting model accuracy
- Long-term: Build a framework for predictive, pipelined MOE inference that can generalize to large models

Motivation

- A lot of standard ML models are MOE models now
  - Example: LLAMA, Mixtral,
    - Probably GPT4, Grok, etc.
- Now inference is the big problem to solve
- Time to first token is a really big metric for lots of systems

Key Questions

How can we streamline the pipelining process for MOE models?

- Can this pipelining be done while leveraging existing MOE model speedups?
- Will this decrease the time for inference?
- Is this a realistic solution?

Infrastructure

- Going to be using an MOE model
  - Not sure which model to use right now
  - Depends on GPU I have access to
- Going to limit this to a proof of concept by only using a small MOE model
- GPU thread programming
  - Going to be coding a CUDA kernel

Reach goals

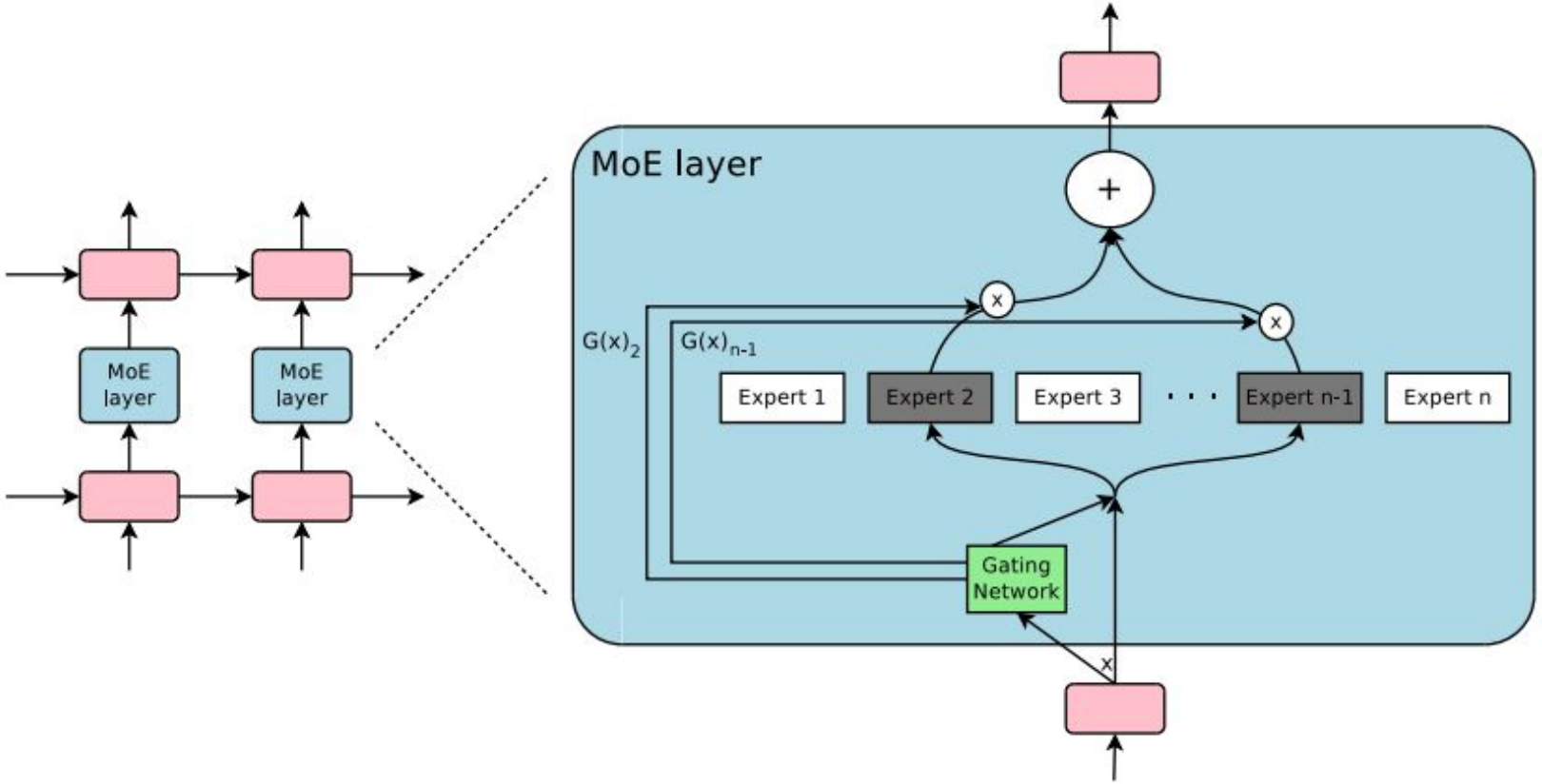
1. What kind of lightweight gating network is the most efficient?
2. How many operations speedups exactly will be achieved?
3. How accurate does the trained network need to be to actually see a speedup?

Help Probably Needed

1. Going to need access to a GPU that I can code with CUDA
    a. Great Lakes exists - just need to specify it is for a class

Potentially more when I start coding

System Architecture



System Architecture

Input → Lightweight Gating Network

↓

Predicted Experts per Layer

↓

Preload Weights → Pipelined MatMul + Activation

↓

Next Layer Computation → Output

Methodology

1. Do some calculations about how many operations can be performed based on time differences
2. Determine how large of an ML model can be created
    a. Train this model to predict routing network
3. Write or find code that does MOE with Cuda operations with the pretrained routing network
4. Implement the pipelined approach with Cuda
5. Get the results

Evaluation metrics

1. Time to First Token
2. Overall Inference Latency
3. GPU Utilization / Kernel Overlap Percentage
4. Accuracy of Gating Predictions
5. Throughput Speedup (%) vs Baseline

Key Milestones

1. Get MOE standard model working with Cuda
2. Train the lightweight gating network
3. Get the high level algorithm to run per thread
4. Implement this algorithm in Cuda
5. Test this on models and get evaluation metrics
6. Repeat from step 2 with different setups / heuristics

Key Papers

- MetaShuffling: Accelerating Llama 4 MoE Inference
  - Shows that smart scheduling can reduce inference latency
  - My work extends this idea with predictive gating + pipelined expert loading
- HAP: Hybrid Adaptive Parallelism for Efficient Mixture-of-Experts Inference
  - Dynamically decides between data parallelism and model parallelism based on input workload and GPU characteristics

Conclusion

- MOE models are efficient but limited by dynamic expert loading
- A predictive gating network can help pre-load experts and enable layer-to-layer pipelining
- This approach aims to reduce inference latency while maintaining accuracy
- If successful, it can influence future compiler/runtime designs for dynamic neural architectures