Project Proposal: "Compiling Neural Networks with Dynamic Resource Adaptation"

Patrick Coppock
pcoppock@andrew.cmu.edu

Yuttapichai (Guide) Kerdcharoen
ykerdcha@andrew.cmu.edu

https://pnxguide.github.io/15745_project/

1 Project Description

1.1 Motivation

Our project will examine neural network (NN) compilation. While NN compilers may optimize within GPU kernels, we (along with many traditional NN frameworks) only optimize across kernels. Inter-kernel optimization simply schedules kernel launches and memory copies to improve the run time (or throughput) of the NN. Furthermore, such compilation doesn't resemble that done in the front and back ends of traditional CPU compilers—where high-level, human-readable source code is transformed into low-level machine code—but is nevertheless analogous to the scheduling performed in the middle end of modern, optimizing CPU compilers.

NNs are compiled either for training to fit parameters or for inference to compute with a trained NN. Training is a batch job, while inference is usually more interactive and may serve a bursty request pattern. In our project, we will focus on inference. Sharing a device between multiple bursty jobs with different priorities requires the low-priority jobs to be able to respond to varying resource availability. For example, one high-priority job might suddenly receive many more requests, resulting in a need for more memory. Lower priority jobs will need to be able to give up memory, while continuing to utilize the GPU.

The largest inference jobs are so called "large language models" (LLM), such as OpenAI's GPT and Meta's OPT. These models require billions of weights and are trained on hundreds of cutting-edge GPUs. The state of the art in LLM inference compilation depends heavily on offloading of weights, activations, or the transformer key-value (KV) cache to CPU DRAM memory (or even disk storage). How many batches to process through a layer at a time and how much of what to offload where are both important questions that affect how memory is copied and kernels launched. These questions are traditionally answered before inferencing starts, resulting in inelastic GPU memory use throughout inferencing.

Can we compile inference applications to elastically use GPU resources? Our project will examine the feasibility of placing offloading-based optimization in the application runtime, allowing for elastic GPU resource usage.

1.2 Related Work

As we already mentioned, state-of-the-art LLM inference compilation offloads memory objects from GPU memory to CPU memory and storage. "High-throughput Generative Inference of Large Language Models with a Single GPU" proposes FlexGen, which does just this. The authors notice that for LLMs, it makes more sense to offload the batch transformer key-value (KV) cache and activations than to offload the layer weights. Therefore, they propose combining batches into blocks. All batches in a block are run through a layer at a time.

The main contribution of FlexGen is a cost model for the above algorithm. The model relates inference throughput to the bandwidths of each memory, the sizes of each memory, and the block size (number of batches). FlexGen's policy search minimizes the modeled cost by tuning both the degree of offloading of each operand type (weights, activations, and KV cache) onto each memory and the block size. The optimized inputs are then used to guide inference.

FlexGen is published on GitHub, but—to the best of our knowledge—the cost model and policy search aren't implemented in the code base.

1.3 Proposed Method

While FlexGen searches policies offline, we suspect it could be extended to search policies online, throughout inferencing. This could lead to a straightforward solution for memory elasticity: drop batches from the current

block as memory becomes scarce and research policies at the start of each block to determine whether its size may be increased.

A more extensive solution could tune other model inputs throughout block inference. Offloading can be tweaked throughout block inference; the only limit is that batches cannot be added to the block, in the middle of an inference pipeline (as their KV cache and activations depend on earlier layers).

1.4 Project Goals

Initial Milestone (75% Goal)

- Implement FlexGen policy search, running it once at the beginning of an inference job (static compilation).
- Evaluate by reproducing FlexGen results.

Conservative Goal (100% Goal)

- Complete the initial milestone.
- Throughout block inference, drop batches as memory becomes scarce.
- At the beginning of block inference, increase block size as able.
- Evaluate by qualitatively demonstrating model flexibility in synthetic environment.

Stretch Goal A (125% Goal)

- Complete the conservative goal.
- Improve evaluation with a realistic environment (multiple tenants processing independent traces).

Stretch Goal B (125% Goal)

- Complete the conservative goal.
- Run full policy search throughout block inference, tweaking policy as allowed.
- Evaluate similar to the conservative goal (or Stretch Goal A) and compare to the results of other completed goals.

Stretch Goal C (125% Goal)

- Complete the conservative goal.
- Explore the use of suboptimal policies that conservatively choose block size to reduce incidence of batch drops in a memory-constrained environment.
- Evaluate similar to the conservative goal (or Stretch Goal A) and compare to the results of other completed goals.

2 Logistics

2.1 Schedule

Week 1

- Implement the cost model from FlexGen
- Implement the policy search from FlexGen

Week 2

- Integrate both the cost model and the policy search into the neural network compiler to attain the static compiler optimization
- Reproduce FlexGen results with the OPT-175B model

Week 3

- Include memory utilization at runtime into the cost model and policy search
- Implement dynamic compiler optimization by resizing block sizes and dropping batches based on the new cost model and policy search

Week 4

- Simulate a synthetic environment to verify the elasticity of the neural network based on our dynamic optimization
- Evaluate our week 3's optimization with a synthetic environment

Week 5

- Explore stretch goals A, B, or C

Week 6

- Polish the project
- Prepare the final report and poster

2.2 Milestone

We should *at least* attain the neural network compilation with dynamic resource adaptation based on the cost model and the policy search from FlexGen. This corresponds to our conservative goal, which we have scheduled to occur in the fourth of the six weeks allotted for this project.

2.3 Literature Search

- **FlexGen:** https://github.com/FMInference/FlexGen/
- **"High-throughput Generative Inference of Large Language Models with a Single GPU":** https://arxiv.org/pdf/2303.06865.pdf

2.4 Resources Needed

For our project, we will need access to an NVIDIA GPU on a host with hundreds of gigabytes of CPU memory. FlexGen was evaluated on an NVIDIA T4 GPU with 16GB of GPU memory and 208GB of CPU memory. One team member currently has access to a machine with two NVIDIA A30 GPUs (24GB of memory each) and 256GB of CPU memory.

In the event that this machine becomes unavailable, we will transition to a Google Cloud Platform instance with the same type as that used by the creators of FlexGen.

2.5 Getting Started

Our team was formed late in the semester. While we have not written any code or otherwise tested our ideas, we have familiarized ourselves with related work and set up a shared code repository. We have no blockers at this point and are able immediately to start on the initial milestone.