15745 Proposal: Hardware-aware Compilation of Tensor Programs for Low-Precision Computing

Jinqi Chen, Zhibo Chen
jinqic@andrew.cmu.edu, zhiboc@andrew.cmu.edu

https://www.andrew.cmu.edu/user/zhiboc/15745-fa24/

1 Project Description:

1.1 Background:

With the explosion of large language models (LLMs) like GPT, the machine learning field has begun to realize the potential of hugely large models. For example, GPT-4 is estimated to contain over 175 billion parameters, LLaMA 2-70B by Meta contains 70 billion parameters, and BLOOM, a multilingual model, contains about 176 billion parameters.

Quite a few challenges emerge for such large models. First, they often require high memory footprints, and can lead to excessive resource consumption and energy use during training or inference. Second, their significantly heavy computation often leads to long training time and thus affects the length of deployment cycles. Last but not least, they have high requirements for the hardware's memory capability, and are hard to be deployed on resource-constrained, edge devices such as mobile devices.

Low-precision computing has emerged as an essential tool for accelerating large language models (LLMs). In fact, latest NVIDIA GPUs, such as Hopper, Ampere etc., follow the trend by supporting for low-precision formats such as FP16, INT8, TF32, and BFLOAT16 etc., and introduce Tensor Cores specifically designed to accelerate these formats. AMD GPUs, such as MI200 series etc., also provide support for FP16, BFLOAT16, and INT8 etc. The emerging hardware support has brought opportunities to design tensor computations that can leverage the benefits of lowered memory requirements and computational complexity.

1.2 Related Work:

We focus on a related paper called **Ladder**, published in OSDI 2024. Ladder is able to support a variety of low-bit precision custom data types through a general type system called *tType*. The algorithm designers can use commonly used data type (i.e. FP16) or define a custom data type (i.e. MXFP8, NF4) as *tType*, and define DNN computations on this type. Then, Ladder takes the DNN model as input and convert it to a *tTile*-based data flow graph (*tTile-graph*).

There are several key challenges with supporting various data precision formats on different hardware accelerators. Among them, one significant issue is that it is difficult to align fine-grained low-bit data access with the coarse-grained memory system. For example, NVIDIA GPU's shared memory bank size is 4 bytes in width, thus simply loading and storing 8-bit data elements can easily lead to bandwidth waste. Another issue is that operations on different data types typically require different data layout optimization to align with the memory system, and existing optimizations such as swizzling are only designed for a few specific data types and not generalizable. A good data layout should be able to be propagated to adjacent operators to avoid explicit layout conversion costs, and data layout in a specific memory layer should consider both the memory feature (DRAM, L2 cache, shared memory, register) and the upper-layer access pattern.

Ladder introduces four transformation primitives for data storage and access: *slice*, *map*, *pad*, and *convert*. It then introduces a scheduling policy, in which lower-layer memory provides preferred data access granularity as a hint, and upper-layer decides the optimal compute granularity by aligning with the data access granularity. It thus compile the *tTile-graph* and generate executable code for the given hardware accelerator.

1.3 Project Goals:

By inspecting their code, we notice that Ladder supports the compiler scheduling policies for some tensor operations, but they are quite limited. Built on top of TVM, it supports two kinds of IR: TE (tensor expression) and TIR (tensor IR). On TE, the supported ops are element-wise op, reduce op, and WMMA tensor cores. On TIR, the supported ops are element-wise op, SIMT, MMA tensor cores, and MMA pad computations. We discovered that there are other types of ops that are interesting to support, like GEMM (and their variants), WGMMA (and other tensor cores), etc. Moreover, the support for TE and TIR are quite diverged, and it would be interesting to find out a generic way to support ops in both IR forms and for different types of ops.

Initial Milestone (75% Goal):

- Implement a new draft scheduler under the current framework. (or re-implement a current scheduler)
- Understand how current scheduler works by changing memory layouts and access patterns. Know clearly what optimization contributes to performance gain on hardware accelerators.

Conservative Goal (100% Goal):

- Implement a working scheduler and evaluate its performance on various kernels.

Stretch Goal A (125% Goal):

- Support fancier tensor core operations, like WGMMA.

Stretch Goal B (125% Goal):

- Come up with a generic approach to make scheduling policies more generic.

1.4 Metrics:

We would like to measure runtime for kernels on a GeForce RTX 4090 GPU as benchmarks. We prioritize the kernels tested by authors of the paper for comparison, and would use other kernels when necessary.

2 Logistics:

2.1 Schedule:

Here is a rough schedule over six weeks:

Week 1: Get familiar with the codebase. Reproduce the results, and understand key optimizations.

Week 2: Draft ideas for a scheduling policy. Implement part of the code.

Week 3: Implement all of the scheduling policy code. Get ready for measurements.

Week 4: Measure performance on our implementation and think about improvements.

Week 5: Investigate stretch goals A, B.

Week 6: Write the report, polish the project, and prepare for final poster session.

2.2 Milestone:

We should at least be able to reproduce and re-implement a scheduling policy based on ideas from the paper. This corresponds to our conservative goal, which we scheduled about four out of six weeks to accomplish.

2.3 Literature Search:

- **Ladder**: https://www.usenix.org/system/files/osdi24-wang-lei.pdf
- **TensorIR**: https://arxiv.org/pdf/2207.04296
- **TVM**: https://arxiv.org/pdf/1802.04799

2.4 Resources Needed:

We have access to GeForce RTX 4090 GPU and H100 GPU, which are enough for single kernel measurements.

2.5 Getting Started:

We have get started in reading the paper and the codebase, and picking up background knowledge. We are in the progress of thinking about scheduling policies for the compiler.